Universitat Autònoma de Barcelona

Towards a better understanding of Bitcoin: from system analyses to new protocol designs

SERGI DELGADO SEGURA

Universitat Autònoma
de Barcelona

Towards a better understanding of Bitcoin: from system analyses to new protocol designs

Author:
SERGI DELGADO SEGURA

Advisors:
DR. JORDI HERRERA JOANCOMARTÍ
DR. GUILLERMO NAVARRO ARRIBAS

Departament d'Enginyeria de la Informació i de les Comunicacions
UNIVERSITAT AUTÒNOMA DE BARCELONA

A dissertation submitted to the Universitat Autònoma de Barcelona in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in Computer Science.

JULY 2018

I certify that I have read this thesis entitled "Towards a better understanding of Bitcoin: from system analyses to new protocol designs" and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Cerdanyola del Vallès, July 2018

Dr. Jordi Herrera Joancomartí	Dr. Guillermo Navarro Arribas
(Advisor)	(Advisor)

Als meus pares

Abstract

Bitcoin has kicked off one of the biggest paradigm shifts of the last century regarding how we understand and use money. The birth of criptocurrencies lays the foundations of a new financial system, where the need of trusted third parties, or central authorities, has been replaced by cryptography and an open flow of information between all the actors of the system. By sharing all the information regarding the operations of the system, all users can eventually agree in a distributed ledger, known as blockchain. Such a paradigm shift, however, poses some threads that, if not properly handled, may compromise the security of the system.

In this thesis we have studied two of the core components of Bitcoin: its P2P network, and the set of unspent Bitcoins. Such analysis aimed to spot the strengths and weaknesses of the system in order to design solutions for them. The outcomes of our analyses have been, on the one side, characterizing the cryptocurrency P2P networks and, on the other side, spotting one of the current Bitcoin scalability problems: the unprofitable coins.

Moreover, after analysing the system and obtaining a deep understanding of it, the thesis has focused on designing protocols to extend Bitcoin's functionality in different payment scenarios. First, we have designed a solution to reduce the likelihood of a merchant of being deceived when accepting zero-confirmation transactions. Finally, we have designed a fair protocol for data trading using Bitcoin, where the exchange between data and coins is performed atomically.

Resum

Bitcoin ha donat peu a un dels majors canvis de paradigma de l'últim segle respecte a com entenem i utilitzem els diners. El naixement de les criptomonedes ha obert la porta a un sistema econòmic distribuït on la necessitat de terceres parts de confiança, o d'entitats centrals, ha estat substituïda per la criptografia i el flux obert d'informació entre tots els actors del sistema, construint d'aquesta forma un registre de transaccions comú conegut com a blockchain. Aquest canvi de paradigma, però, comporta certes implicacions que, de no ser tractades adientment, poden comprometre la seguretat del sistema.

En aquesta tesis ens hem centrat en analitzar dos dels grans components de Bitcoin: la seva xarxa P2P i el conjunt de monedes en circulació. Amb aquest anàlisis es pretén identificar els punts forts i les febleses de Bitcoin, amb l'objectiu de proposar solucions i/o millores per aquestes. Aquests anàlisis ens han permès, per una banda, caracteritzar les xarxes P2P de criptomonedes, i, per altra banda, identificar un dels actuals problemes d'escalabilitat de Bitcoin: les monedes no rentables.

D'altra banda, i un cop assolit un coneixement suficient del sistema, la tesis s'ha centrat en el disseny de protocols per estendre la funcionalitat de Bitcoin en diferents escenaris de pagament. A més a més, s'ha proposat una solució per reduir la probabilitat de ser estafat a l'utilitzar transaccions sense confirmar. I finalment, s'ha dissenyat un protocol de compra-venta de dades utilitzant Bitcoin, eliminant la necessitat inherent de confiança per part del comprador.

ACKNOWLEDGEMENTS

En primer lloc vull donar les gràcies als meus directors, en Jordi Herrera i en Guillermo Navarro, per confiar en mi i donar-me l'oportunitat de realitzar aquesta tesi, pel seu suport incondicional, i per tot el que m'han ensenyat durant aquests tres anys de tesi. M'agradaria estendre aquest agraïment a la meva companya Cristina Pérez, amb qui he estat treballant els darrers dos anys, i sense qui aquesta tesi tampoc hagués estat possible.

I would also like to express my most sincere gratitude to Andrew Miller for accepting me as a visiting scholar at the UIUC and for providing me guidance throughout the months I stayed there. It has been a pleasure having the opportunity to work with him. I would also like to thank the guys at UIUC: Surya, Kevin, Hanyun, Tom, Deepak, Riccardo and Zane, you really made me enjoy my stay in Illinois.

A tots els membres del grup de recerca SeNDA, i a la resta de docents del departament, que m'heu guiat des del meu pas com a estudiant fins a la finalització de la tesi.

Als meus companys de *penúries*, a en Carlos, la Sara, l'Iván, en Roger i a la resta dels anomenats *pifos*, per tots el grans moments viscuts durant aquests anys, pel suport moral, i per la quantitat de problemes resolts entre pissarres i dinars.

Al Jose i l'Andreu, per ser-hi, sempre, i per fer aquest camí infinitament més divertit. A la Clàudia i a la Núria, perquè amb vosaltres vaig poder tornar a gaudir de la muntanya que tant m'estimo, i que m'ha permès desconnectar més m'ha fet falta.

Als de casa, en especial al meus pares, per ensenyar-me a buscar la millor versió de mi, i per recolzar-me de forma incondicional. I finalment a la Laia, per acompanyar-me durant el camí, per ser al meu costat i ajudar-me, per aguantar les meves xerrades sobre temes inintel·ligibles, per tot.

Gràcies.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

This thesis aims to fill some of the gaps found in the literature regarding Bitcoin. At the time of starting the thesis, some of the insights of how Bitcoin works where mainly unknown especially in the research community. Such situation has drastically changed in the recent years, with an increasing dedication from the research community towards analysing all the bits Bitcoin is made of, helping turning the system into a more robust one. However, both the number of users and the market capitalization of Bitcoin have drastically increased over the last years, making the system more attractive for malicious users to tamper with. Therefore, more effort need to be devoted to ensure the security of the system.

1.1 Motivation

Bitcoin was born during the worst financial crisis since the Great Depression of 1929. The current economic system and the role played especially by banks helped

to magnify the catastrophic outcome of such a crisis, from which some countries are still struggling to emerge after almost a decade. Some hints given by the author of Bitcoin[1], Satoshi Nakamoto, makes us believe that the main goal of his system was indeed to provide an alternative to banks, giving people full control over their money without the need to trust in a system that has proven to be flawed throughout the recent history. Bitcoin introduced the concept of blockchain, an append only ledger to keep track of transactions, by putting together several other concepts from the late '90s and early 2000s [80], such as digital cash [27, 28], linked timestamping [10, 54, 55], merkle trees [76], proof-of-work [6, 43, 60], etc. After the whitepaper describing Bitcoin was made public (November 2008) and the first version of the code was finally released (January 2009), it draw the attention of people from different areas: from decentralization maximalists, cryptographers and researchers to enthusiasts who believed a better economic system was possible. The system started as not much than a toy to play with for most, but as a promise of something bigger from people who was willing to invest his time to make the system succeed. The development of Bitcoin during the following years was made by a large open source community of developers, with close attention from researches who made contributions towards strengthening the security of the system. Several hundreds new cryptocurrencies were created over the years, some tweaking Bitcoin (ie: Litecoin or Dogecoin) and others using the concept of blockchain to create their own distinct system (CryptoNote or ZCash). Nowadays Bitcoin, and the hundreds of other cryptocurrencies that have been created so far, have set the foundations towards a greater decentralization, first in terms of economics, but also in governance and trust. We are, however, in the early years of such a new fascinating technology, and therefore, open issues need to be dealt with and some questions need to be answered in order to make Bitcoin an actual alternative to fiat: from scalability issues, such as the number of transactions per second the system can process, or the space required to store the blockchain, to potential decentralization issues both in mining power (i.e. is

[1]Text found in the coinbase of the genesis block: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"

there someone controlling and outstanding portion of the hash power?) and in connectivity (i.e: do super nodes or bridge nodes exist?) or even usage issues, such as fees being abnormally high, etc.

1.2 Objectives

In such a scenario, and focusing on Bitcoin, the work included in this thesis aims to extend the knowledge and requirements of the system and to define solutions towards a more robust, fair, and optimal system. The objective of the thesis is twofold:

- To extend the current knowledge of the system and to analyse the issues it may have, in order to design possible solutions.
- To provide protocols and solutions to extend the functionality of the system, and to fill the gaps the system may have.

1.3 Contributions

This thesis has followed two main lines of work:

- Analysing the Bitcoin system and spotting potential problems.
- Designing protocols and solutions to extend Bitcoin's functionality.

On the one hand, and regarding the analysis performed in the thesis, the study has focused mainly in two parts. First, the Peer-to-Peer network the Bitcoin systems builds on top of, and secondly, the set of unspent transactions outputs required by the system to create new transactions. In this line of work, we have made the following contributions:

1. Pérez-Solà, C., Delgado-Segura, S., Navarro-Arribas, G., and Herrera-Joancomartí, J. Another coin bites the dust: An analysis of dust in UTXO based cryptocurrencies, submitted to *Royal Society Open Science*.

2. Delgado-Segura, S., Pérez-Solà, C., Navarro-Arribas, G., and Herrera-Joancomartí, J. Analysis of the Bitcoin UTXO set, *The 5th Workshop on Bitcoin and Blockchain Research (BITCOIN'18)*, 2018.

3. Delgado-Segura, S., Pérez-Solà, C., Herrera-Joancomartí, J., Navarro-Arribas, G., and Borrell, J. Cryptocurrency networks: a new P2P paradigm, *Mobile Information Systems*, 2018.

4. Delgado-Segura, S., Pérez-Solà, C., Navarro-Arribas, G., Herrera-Joancomartí, J., and Borrell, J. Survey of network based attacks to the Bitcoin P2P network,*Actas de la XIV Reunión Española de Criptología y Seguridad de la Información (RECSI 2016)*, 2016.

On the other hand, we have designed a new type of transaction within Bitcoin, and two additional protocols that build on top of such a new transaction type. The first of the protocols aims to give a solution to one of the typical user experience problems when using Bitcoin: transaction confirmation times. Our solution takes advantage of zero-confirmation transactions while reducing the odds of the merchant being deceived. The second proposed solution aims to remove the trust users have to put in merchants when performing Bitcoin transactions. Bitcoin transactions are final once they have been included in the blockchain, and no reimbursement is possible if the recipient of the transaction does not agree on doing so. Therefore, when creating a transactions customers have to trust that merchants will deliver the product / service. Our protocol proposes a solution to deal with such an inherent problem by providing an atomic way of exchanging data for bitcoins. The following contributions have been made in the current line of work:

5. Pérez-Solá, C., Delgado-Segura, S., Navarro-Arribas, G. and Herrera-Joancomartí, J. Double-spending Prevention for Bitcoin zero-confirmation transactions, submitted to *International Journal of Information Security*.

6. Delgado-Segura, S., Pérez-Solà, C., Navarro-Arribas, G., and Herrera-Joancomartí, J. A fair protocol for data trading based on Bitcoin transactions, *Future Generation Computer Systems*, 2017, ISSN 0167-739X.

7. Delgado-Segura, S., Pérez-Solà, C., Herrera-Joancomartí, J., and Navarro-Arribas, G. Bitcoin Private Key Locked Transactions, submitted to *Information Processing Letters*.

8. Delgado-Segura, S., Tanas, C. and Herrera-Joancomartí, J. (2016). Reputation and Reward: Two Sides of the Same Bitcoin, *Sensors*, 2016, 16(6), p. 776.

9. Tanas, C., Delgado-Segura, S. and Herrera-Joancomartí, J. An Integrated Reward and Reputation Mechanism for MCS Preserving Users Privacy, *International Workshop on Data Privacy Management*, 2015, pp. 83-99.

1.4 Thesis outline

The rest of the thesis is structured as follows:

First, Chapter 2 introduces some of the preliminary concepts the thesis builds on, along with a review of the current state of the art in such topics. From that point on, the thesis is split into two parts, a first analytical one (chapters 3-4), in which some of the parts of Bitcoin are analysed and reviewed, and a more practical one (chapters 5-7), in which some proposals and protocols over Bitcoin are designed.

Later, Chapter 3 analyses the Bitcoin P2P network especially focusing in its security properties in order to characterize the requirements of cryptocurrency P2P network.

Afterwards, Chapter 4 analyses the Bitcoin set of unspent transaction outputs, introducing its design and showing statistical data about its content. The analysis pays especial attention to dust and unprofitable outputs.

Beginning the second part of the thesis, Chapter 5 presents private key locked transactions (PKLT), a new type of transactions where a private key is atomically exchanged within a Bitcoin transaction.

Chapter 6 introduces the design of a double-spending prevention mechanism for zero-confirmations transactions based on PKLT, ECDSA and the Bitcoin scripting language.

Later, Chapter 7 introduces a fair protocol for data trading based on bitcoin transactions which also uses PKLT as a building block.

Finally, Chapter 8 concludes the thesis and gives some guidelines for future work.

CHAPTER 2

BITCOIN AND BLOCKCHAIN

This chapter introduces the two main topics the thesis fits into, namely Bitcoin and blockchain, which will be dealt with throughout the rest of the thesis. First we will give a brief overview on both concepts along with their origin and purpose. Afterwards, we will jump into their foundations and what makes them an interesting research topic. Finally, we will introduce some of the topics that will be further analysed in later chapters, such as the Bitcoin P2P network (Chapter 3), the UTXO set (Chapter 4) and double-spending transactions (Chapter 6).

2.1 The origin of cryptocurrencies

Bitcoin [79] is the first decentralized cryptocurrency to have ever existed and succeeded. It was released as an open source code project in 2009 by someone under the pseudonym of Satoshi Nakamoto and his creation laid the foundations for the design and deployment of a plethora of different cryptocurrencies, which at

the time of writing surpasses 1500. Bitcoin introduced the concept of blockchain, an append only distributed data structure shared among the users of the system that acts as a ledger in most of the deployed cryptocurrencies. Entries in such a ledger are known as transactions, and are used to keep track of the value transfer between different actors in the system. Transactions encapsulate the transfer of value within the currency and require a proof of ownership of the coins by the sender, in most cases fulfilled by providing a digital signature, and the knowledge of a destination point, normally bound to an address. Transactions are not confirmed until included in the blockchain. Such inclusion is not performed one by one but grouped and encapsulated in blocks, the data structure the blockchain is made of. The process of creating blocks is known as mining and it is performed by miners. Each block contains a hash pointer to the previous block, creating in that way a chain of blocks, which gives name to the technology.

2.2 Bitcoin transactions

Bitcoin transactions are, along with blocks, one of the two entities the Bitcoin protocol builds on top of. Transactions are used to transfer coins between different users of the system. A transaction is basically formed by two data structures: the **inputs**, containing references to the coins that will be spent, and the **outputs**, containing information regarding the coins that will be created. The actual structure of a transaction can be found depicted in Figure 2.1. When a transaction is created, a coin is spent for each one of its inputs, and a new coin is created for each of its outputs.

Coins are also referred to as **unspent transaction outputs (UTXOs)** that are, as the name already suggests, non-spent outputs from previously created transactions. Coins consist of two main fields, the value and the output script. The value of a coin is a pretty straightforward concept that defines how many bitcoins the coin is worth, whereas the output script defines the conditions under which the coin can be spent. Such conditions are encoded as **scripts** (using the Bitcoin scripting language), and the most common one links the coin to a certain

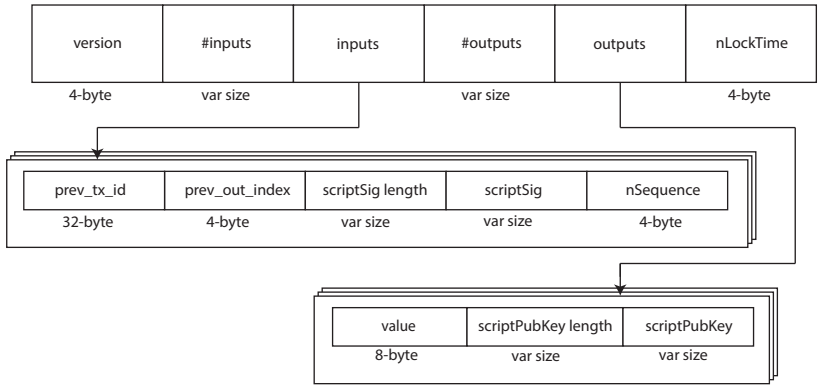

Figure 2.1: Generic transaction structure.

Bitcoin address. A coin can be referenced by its outpoint, a two-field structure that uniquely identifies it. The outpoint consists of the id (`prev_tx_id`) and the output index (`prev_out_index`) of the transaction that created the coin.

Coins can be redeemed by referencing them in the inputs of a transaction, i.e. by including the coin outpoint. Moreover, a proof of ownership of the coins should be provided for each coin the transactions tries to redeem, otherwise anyone will be able to spend any coin. Such proof is encoded in the input script field of each input which contains, in the most common case, a **signature**.

Regarding signatures and Bitcoin addresses, Bitcoin uses the Ecliptic Curve Digital Signature Algorithm (ECDSA), with the secp256k1 elliptic curve, as the cryptographic algorithm under the hood [63]. Users in the system are required to hold at least one key pair ($K = \{sk, pk\}$) to send and receive transactions. The secret key (sk) is used to perform digital signatures over transactions when required, whereas the public key (pk) is used as an endpoint to send coins to. A Bitcoin address is a simplification of a public key in the form of Base58 encoded 160-bit hash, and it is the most common type of Bitcoin transaction end point as of today.

2.2.1 The Bitcoin scripting language

Bitcoin uses a stack based non-Turing-complete scripting language with no loops known as **Script** to encode both input and output scripts. Two different clauses can be found in Script, operation codes (**opcodes**) that define some functionality, such as addition, subtraction, signature validation, etc, and regular data, that is used as input of the aforementioned opcodes. Scripts work in pairs, when an output script is created certain locking conditions are specified in the `scriptPubKey` field. Such conditions state how the output can be redeemed. When a new transaction is created each input must provide proof of fulfilment of the conditions in the form of a script, placed in the `scriptSig` field, and pass a correctness validation. The correctness of a script is validated by evaluating both parts of the script, namely the `scriptPubKey` of the UTXO and the `scriptSig` of the input, one after the other, and analysing the final result. To do so both scripts are pushed into the stack starting from the `scriptPubKey`. If the script evaluates to true, the correctness is verified, otherwise the script is invalid making the transaction also invalid. At the time of writing, five type of standard scripts can be found within Bitcoin[1].

Pay-to-PubKey (P2PK) is a fairly simple type of script that allows to send coins to a given public key. The input script for P2PK scripts contains only a signature, whereas the output script contains a public key and an opcode to validate signatures.

```
ScriptPubKey: <pubKey> OP_CHECKSIG
ScriptSig:    <sig>
```

When both scripts are put together `OP_CHECKSIG` validates if the provided public key can validate the provided signature. If the validation succeeds, the script is valid, otherwise it fails. P2PK scripts where broadly used during the

[1] Excluding the recently defined SegWit scripts.

early years of Bitcoin, however, nowadays their use has been drastically reduced in favour of Pay-to-PubkeyHash scripts.

Pay-to-PubkeyHash (P2PKH) is a more secure type of script than P2PK that pays to a public key hash (Bitcoin address) instead of a public key.

```
ScriptPubKey: OP_DUP OP_HASH160 <hash160(pubKey)> OP_EQUAL
              OP_CHECKSIG
ScriptSig:    <sig> <pubKey>
```

Regarding the script structure, the input script contains a signature and a public key, whereas the output script contains several opcodes and a simplified form of a Bitcoin address (`ripemd160` hash). When executed, the script duplicates the public key and the `ripemd160` hash is computed over it, then the resulting hash is compared with the provided one, and finally, the signature is validated with the public key. If all the execution succeeds, the script is valid, otherwise it fails, invalidating the transaction.

Pay-to-multisig (P2MS), also known as bare multisig, or M-of-N multisig, is a type of script that allows multiple actors to be involved in the fulfilment of the requirements to spend a certain coin. The script is designed to allow up to N users to take part in the transactions signature, where at least M are required. The script structure looks as follows:

```
ScriptPubKey: OP_N <pk_1> <pk_2> ... <pk_N> OP_CHECKMULTISIG
ScriptSig:    OP_0 <sig_1> <sig_2> ... <sigm>
```

Any combination of M signatures will verify to true, whereas if less than M signatures are provided, the script will fail. It is worth noting that signatures need to be provided in the same order the public keys where specified in the output script. Moreover, only scripts up to 20-of-20 will be accepted by the network,

whereas only up to 3-of-3 are considered standard.

Pay-to-ScriptHash (P2SH) is a smarter type of script that hides the conditions to be fulfilled behind a hash. P2SH scripts allow, for instance, the creation of standard P2MS scripts up to 20-of-20, the creation of hash puzzles [19], etc. Moreover, being the output script hidden behind a hash, no other user but the creator of the coin should know how it can be spent. Finally, P2SH scripts generate smaller output scripts, which are beneficial for coin storage and pass most of the fees to be paid to the redeemer of a coin instead its creator, reducing the costs of creating them. We will see the importance of optimizing the storage of coins and the importance of fees later on in Chapter 4.

```
ScriptPubKey: OP_HASH160 <hash160(redeemScript)> OP_EQUAL
ScriptSig:    <data> <redeemScript>
```

P2SH require a two step verification. First the provided redeemScript is hashed and compared with the hash provided in the coin referred by the input script. If the validation holds then the data is evaluated over the redeemScript. If both evaluations pass, the script is valid, otherwise the script fails.

OP_Return, or null data scripts, are the only kind of standard scripts that do not generate any coin, making it impossible to redeem from an output of such a type. They are intended for storing arbitrary data in the blockchain up to a limit.

```
ScriptPubKey: OP_Return <0 to 83 bytes of data>
```

However, some conditions do apply: Every transaction can only have a single OP_Return output, and such output value must be 0.

2.2.2 Time-locked transactions

At this point, we have introduced the most common way of locking transactions, namely **signature locked transactions**, which require a signature performed with a specific private key to be redeemed. However, other types of locks can be enforced by Bitcoin transactions in order to create more complex redeem conditions.

Time-locked transactions are those that require a certain time in the future to be reached in order to be redeemed. Any transaction spending from a time-locked transaction whose time lock has not been fulfilled will be dropped and therefore, not rellayed to any other peer in the network. Depending on whether the future time is absolute to Bitcoin, or relative to the transaction publishing time, two types of time-locks can be found. On the one hand, absolute time-locks, those based on the `CheckLockTimeVerify` opcode [103], establish a fixed date in the future from when the transaction can be redeemed. Down below an example of such time-lock (locked until 2022/12/13), along with a standard signature, is provided:

```
ScriptPubKey: <2022/12/13> OP_CHECKLOCKTIMEVERIFY OP_DROP
              <pubKey> OP_CHECKSIG
ScriptSig:    <sig>
```

On the other hand, relative time-locks can be included in transaction through the `CheckSequenceVerify` opcode. Relative time-locks establish an amount of time to be elapsed before any future transaction spending from the locked one can redeem it. Such elapsed time is counted from the transaction publishing time (i.e. when a block is created including such a transaction). An example of such time-lock (locked for 25 days), together with a traditional signature, can be found as follows:

```
ScriptPubKey: <25d> OP_CHECKSEQUENCEVERIFY OP_DROP
              <pubKey> OP_CHECKSIG
ScriptSig:    <sig>
```

Notice that in both examples the `ScriptSig`, included in the transaction that will spent the output, does not contain any time reference since the transaction creation time is used to check the time-locks. Moreover, both examples include a traditional signature lock. The reason behind this second lock is to restrict the redeemer to a single person, otherwise anyone will be able to spend the output once the requested time has been reached. Figure 2.2 depicts a general time-locked transaction and can be seen as a representation of any of the two introduced types.

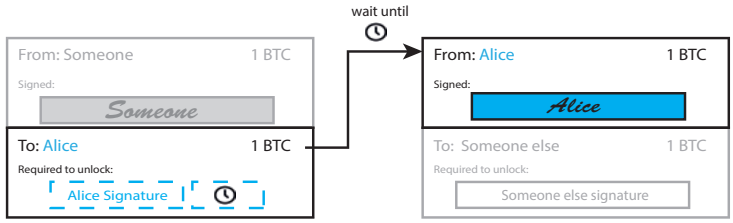

Figure 2.2: Time-locked transaction abstraction.

2.2.3 Hash-locked transactions

Hash locks are yet another type of locking conditions used within Bitcoin transactions, where the knowledge of the preimage of a hash is required to redeem an output.

```
ScriptPubKey: OP_SHA256 <value_hash> OP_EQUAL
ScriptSig:    <value>
```

Figure 2.3 depicts an abstraction of a hash-locked transaction. Hash-locks can be used along with time-locks and signature-locks to create smarter scripts, like in some well known Bitcoin scaling proposals such as the Lightning Network [88], or the Duplex Micropayment Channels [35].

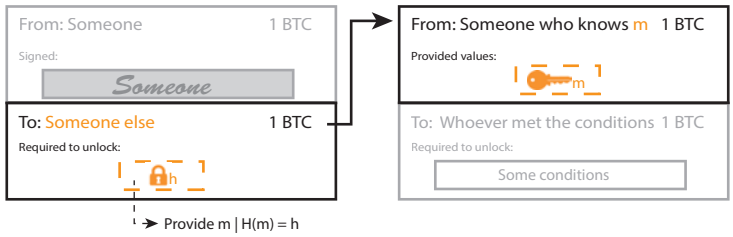

Figure 2.3: Hash-locked transaction abstraction.

2.3 Blocks and mining

Transactions are included in the blockchain at time intervals rather than in a flow fashion. Such addition is performed by collecting all new transactions of the system, compiling them together in a data structure called block and including the block at the top of the blockchain. Every time a block containing a specific transaction is included in the blockchain such transaction is said to be a **confirmed transaction**, since it can be checked for double-spending prevention from that point on.

Blocks are data structures that mainly contain a set of transactions that have been performed in the system (see Figure 2.4). In order to achieve the append-only property, the inclusion of a block in the blockchain was designed to be a hard problem, so adding blocks to the blockchain is both time and work consuming. Furthermore, every block is indexed using its hash value and every new block contains the hash value of the previous one (see `prev_block_hash` field in Figure 2.4). Such mechanism ensures that the modification of a block

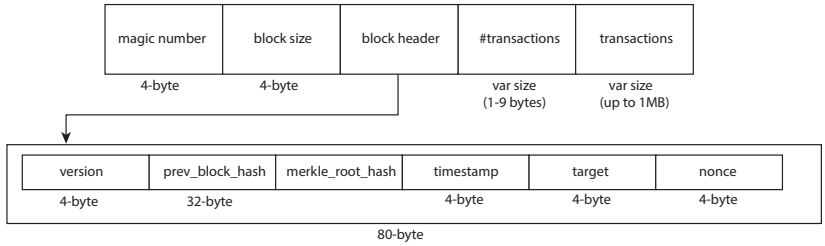

Figure 2.4: Generic block structure.

from an arbitrary height in the chain would imply to modify all remaining blocks up to the top in order to match all the hash values.

Adding a block to the blockchain is known as **mining**, a process that is also distributed and that can be performed by any user of the Bitcoin network using specific-purpose software (and hardware). The mining process uses a hashcash proof-of-work system, first proposed by Adam Back as an anti-spam mechanism [6]. The proof-of-work consists in finding a hash of the new block with a value lower than a predefined target[2]. This process is performed by brute force, varying the nonce value of the block and computing the hash over the later until the conditions hold. Once the value has been found, the new block becomes the top of the blockchain and all miners should discard their previous work and start working on the new top until a newer block is found.

Mining new blocks is a key task in Bitcoin, which helps to confirm the transactions of the system. Being it a hard task, and also a crucial one, miners should be properly rewarded. In Bitcoin, miners are rewarded with two mechanisms. The first one provides the newly created coins for each mined block. Every new block includes a special transaction, called **generation transaction** or coinbase transaction, that has no input. Coinbase transactions are the only kind of transactions that have no input, and they are the way Bitcoin uses to mint

[2]Notice that the value of the target determines the difficulty of the mining process. Bitcoin adjusts the target value, every 2016 blocks, depending on the hash power of the miners in order to set the throughput of new blocks to 1 every 10 minutes (on average).

new coins. A miner looking for a new block will generate a coinbase transaction bound to himself as a reward for his work, using any output script from the ones showed in Section 2.2.1. The second rewarding mechanism are transactions fees. Every transaction created in Bitcoin may include fees, which are an inclusion incentive for miners. The higher the fees, the more likely a transaction will be confirmed within the next block. Transaction fees are encoded in transactions as the difference between the sum of all input amounts and the sum of all output amounts. Therefore, any non-claimed bitcoins will be considered fees. Transaction fees are rewarded to miners also in the coinbase transaction. The value of the coin (or coins) generated from the coinbase transaction of a block must be at most the block reward (R) plus the sum of all fees, otherwise the block is invalid. The block reward follows a logarithmic distribution determined by the block height. It started at 50฿halves every 210000 blocks (approximately 4 years). At the time of writing $R = 12.5$฿.

2.3.1 Blockchain forks

Being proof-of-work a competitive approach for mining, more than one valid block may be found in a short time frame and all data broadcast to the network as the new blockchain head. In such situation, nodes may receive several valid blocks and decide which one to pick, while discarding the rest. Three rules apply here. First, the blockchain with the most work wins, being work calculated as:

$$\sum_{i=1}^{n} \frac{2^{256}}{target_i}$$

Where n represents the height of the latest block and $target_i$ represents the target difficulty of block i. Secondly, the order in which the blocks were received, the first one wins. And finally, a tie breaking rule which compares the memory address pointer of each block, where the higher wins. The three rules are evaluated in the aforementioned order, and the first one not ending up in a tie will decide which block will be picked by the node.

2.3.2 The append-only property

The link between blocks, the inherent difficulty of the mining process and the consensus rules regarding which chain is valid, are what gives blockchain its append only property. In order to remove (or modify) a transaction from the chain, the block including that transaction needs to be modified. Modifying a block will change its hash and therefore break the hash pointer with the next block. In order to make such a change final, all the hash pointer up to the head need to be updated and all the blocks re-mined, while the rest of the network continues mining at the the current head. Therefore, once consensus has been reached, the odds of successfully tampering with data on the blockchain decreases over time.

While performing such a modification is theoretically possible, it will require the devotion of a substantial amount of mining power to be achieved, which will be seen as an attack to the network and will undermine the trust users have on the system. Such attacks are known as 51% attacks and have been widely studied in the literature [45, 81].

2.4 The underlying P2P network

Bitcoin is built on top of a P2P network used by the currency to spread transactions and blocks among peers. Transactions are broadcast between peers, aiming to reach the whole network to eventually be included in a block. When a transaction is received by a node from one of its peers, the node verifies mainly three things: first, that the transaction has not been already seen, secondly, that it claims non-spent coins, and finally, that the transaction is correct. If all conditions hold, the node stores the transaction in a list of unconfirmed transactions, known as **memory pool of transactions** or **mempool** for short, and further relays such transaction to the rest of its peers. Otherwise, the transaction is discarded[3].

[3] It is worth noting that if the transaction refers to any unknown coin, it is stored in a separate list of orphans transactions. Such transactions will be accepted by the node but not relayed any further.

Transactions stored in the mempool are known as unconfirmed transactions, or zero-confirmation transactions.

2.4.1 Unconfirmed transactions

Transactions will leave the mempool under two circumstances: when they are included in a block, or when they are dropped from it. Dropping reasons include transaction being in conflict with other transactions included in the blockchain by a new block, transaction being replaced by a newer version of the transaction, eviction of transaction due to node's mempool becoming full, or transaction timing out after 14 days on it.

Transactions in the mempool can be replaced by a newer version of them if they have been created signaling such property. Such a technique is known as replace-by-fee (RBF) [56] and was designed to allow transaction creators to increase the fee of an already broadcast transaction in order to boost the odds of its inclusion in a block and avoid delays in the transaction confirmation. Transactions can signal such a property by specifying a value lesser than `0xfffffffd` in the `nSequence` field (see Figure 2.1).

2.4.2 Double-spending transactions

Zero-confirmation transactions, in contrast to transactions included in the blockchain, are not covered by the double-spending protection mechanism Bitcoin provides by design. Zero-confirmation transactions are therefore a potential target of double-spending attacks, where conflicting transactions are sent to different nodes on the network. For instance, suppose two transactions (tx_1 and tx_2) that spend the same output from a previous transaction (tx_0) are created by an attacker A. tx_1 is used to pay for some goods to a merchant B, while tx_2 is used to return the funds to the attacker. In this scenario, if A can make B believe that tx_1 is the only transaction spending from tx_0's output, but tx_2 finally ends up included in a block, the attack is successful. Figure 2.5 depicts the aforementioned example.

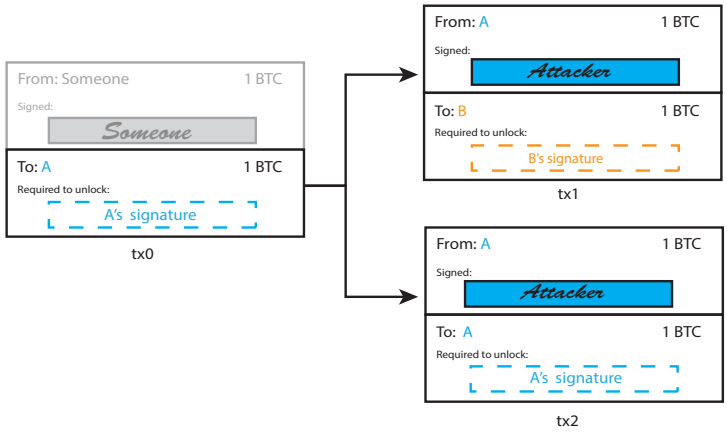

Figure 2.5: Double-spending transactions.

2.5 Double-spending attacks

Double spending attacks on zero-confirmation transactions in Bitcoin have been previously analysed in the literature by Karame et al. [64, 65]. The authors show how, with some reasonable assumptions and without the need of special computation nor much network overhead, an attacker has a great probability of succeeding with a double spending attack. Moreover, the authors also show how basic countermeasures such as waiting a few seconds before accepting a payment or adding observers that report back to the payee are not enough on their own to avoid these type of attacks. Furthermore, they proposed an additional countermeasure, consisting of modifying the protocol rules so that nodes forward double spending transactions instead of dropping them. By doing so,all nodes may be notified of the double spending attempts. However, this mechanism facilitates denial of service attacks and, while nodes will indeed be able to see both transactions, they do not have the means to distinguish which of the two is the original one.

Regarding mitigation of double spending attacks, Decker et al. [7] proposed

some other countermeasures that can reduce the merchant's likelihood of being deceived by an attacker, which are: requiring the merchant to be connected to a large random sample of nodes of the network and not accepting incoming connections. Therefore, the attacker cannot send transactions directly to the merchant neither identify the merchant's neighbours.

Other research studies have indeed demonstrated that this kind of attacks were possible, and not only was the attacker able to identify the merchant's neighbors but also forced them to be a set of nodes controlled by the attacker [15, 16, 68].

CHAPTER 3

CHARACTERIZING CRYPTOCURRENCY P2P NETWORKS

In this chapter we analyse the Bitcoin P2P network to characterize general P2P cryptocurrency networks [37]. Two main reasons made us choose Bitcoin as the subject of analysis. On the one hand, far beyond the economic impact, being the largest cryptocurrency also conveys technical implications: the volume of information flowing through its network, as well as its size and heterogeneity, surpasses any other deployed cryptocurrency. On the other hand, being Bitcoin the first open source cryptocurrency proposed, several other new cryptocurrencies have been developed as a software fork of the Bitcoin reference implementation, using the exact same code for the creation of their underlying P2P network.

3.1 Description of the Bitcoin P2P network

As slightly introduced in Section 2, Bitcoin is built on top of a P2P network used by the currency to spread transactions and blocks among peers. The initial

definition of such a P2P network was coded in the first release of Bitcoin, and later on cloned in multiple new cryptocurrencies that derive from the Bitcoin implementation. In such new cryptocurrencies, the network configuration has been implemented almost identically. For instance, as described in [41], Litecoin, Dogecoin, Dash and Peercoin have exactly the same network message types of Bitcoin, being the resulting networks for those cryptocurrencies very similar and in some cases identical to Bitcoin's.

The Bitcoin network has largely changed from its deployment in 2009, where the only available implementation was the reference client, until nowadays, where the network is made of very heterogeneous peers whose hardware capabilities and software implementations largely differ from each other. Furthermore, even specific purpose protocols have been created aiming to optimize particular tasks of the Bitcoin ecosystem.

In order to describe the existing Bitcoin network, we first identify some of the properties that characterize Bitcoin's peers. After that, we review the most common peer configurations, using the properties described before. Finally, we describe the composition of the current Bitcoin network.

3.1.1 Properties describing Bitcoin nodes

In this section we focus on describing the main properties that define a Bitcoin node: the knowledge about the blockchain, its main functionalities, its connectivity, and the protocols it uses to communicate with other nodes. Figure 3.1 summarizes such classification.

Depending on the **knowledge** peers have **about the blockchain**, their storage requirements largely differ going from a few megabytes to dozens of gigabytes.

On the one hand, *full blockchain* peers store a complete and up-to-date version of the blockchain, which allows them to perform full validation of blocks [1]. On the other hand, *pruned blockchain* peers store an up-to-date version of the blockchain

[1] On May 2018, the total size of block headers and transactions consists of 167GB of data.

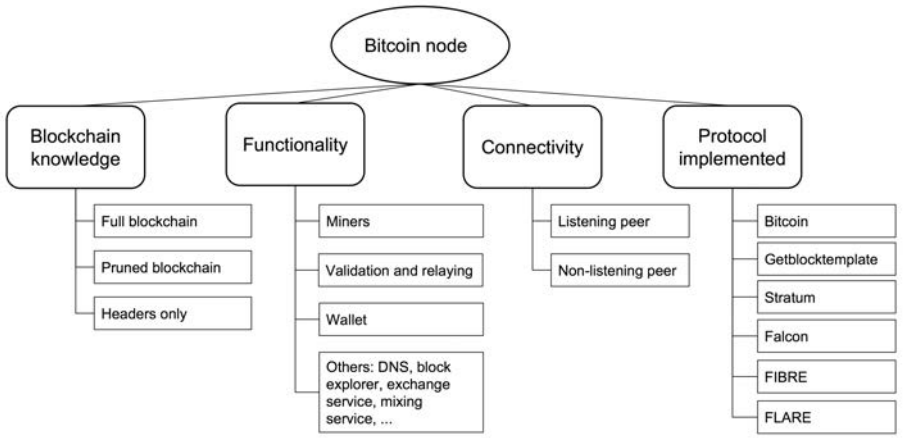

Figure 3.1: Bitcoin node classification.

with complete blockchain data accounting for at least the last $550MB$[2]. Even not holding a full copy of the blockchain, `pruned` nodes can validate new blocks as well as relay them. However, they cannot relay blocks older than the time frame they hold data from. Both `full` and `pruned` nodes store a copy of the UTXO set, allowing them to validate and relay transactions. The UTXO set will be deeply analysed later on in Chapter 4.

Simplified Payment Verification (SPV) clients hold an up-to-date version of the blockchain headers. Additionally, such clients may store data from some transactions of interest. SPV clients are usually deployed in mobile devices such as smartphones, where having the full blockchain is generally unaffordable.

SPV clients are said to be lightweight clients because they minimize the resources needed to accomplish their functionality. However, there exist other lightweight clients that are not based on SPV. The current alternatives are centralized approaches, where clients connect to a set of predefined servers that

[2]The amount of data to be stored can be set when configuring the node.

relay them the information they need in order to work as wallets. This approach requires to trust the servers. The specific amount of data about the blockchain or cryptographic keys stored by these clients depends on each implementation. Some of them publish their source code for public review, while others do not. The protocol is also specific. Some examples of these kinds of wallets are Mycelium [78] or Copay [30].

Peers can also be classified on the basis of their **functionality**. There are three functionalities needed for the Bitcoin system to work. *Mining* is the computationally expensive task of trying to create blocks. New blocks are appended to the end of the blockchain, thus making the public ledger grow. Peers that perform mining are known as miners. Groups of miners are known as pools and usually operate as client-server architectures, with the pool operator providing a pool mining server where pool mining clients connect to and retrieve their portion of work. Some peers perform *validation and relaying* of the transactions and blocks they receive, that is, they relay to other peers valid transaction and block data, together with network data. Some peers also have a *wallet* functionality, they store a set of key pairs, keep track of the amount of bitcoins deposited on addresses associated to those keys, and are able to create transactions that spend those bitcoins. These functionalities do not necessarily exclude each other, thus, a peer may perform more than one functionality at the same time. Additionally, although not strictly necessary for Bitcoin to work, some peers may provide other functionalities. For instance, they may provide a *DNS* service, that offers information about existing peers; a *block explorer service*, where users can query for transaction and block data through a graphical interface; an *exchange* service, where users can buy or sell bitcoins in exchange for other currencies; *mixing services*, where users are able to obfuscate the history of their coins, etc.

Depending on their **connectivity**, peers can be classified in *listening* peers or *non-listening* peers. Listening peers are nodes that accept incoming connections, while non-listening nodes are those not doing so. Although most Bitcoin full implementations listen for incoming connections, some network configurations do not allow establishing such connections (e.g., peers behind NAT, firewalls, etc.).

Since its foundations, Bitcoin has grown far beyond using a single **protocol**, giving place for lots of other protocols to arise. We will use the term *Bitcoin* protocol to refer to the network protocol used by the current reference implementation, the Bitcoin Core client. Other protocols that currently exist on the Bitcoin system are mainly targeted to optimize pooled mining and speed up data propagation. *Getblocktemplate* [32, 33] is the most recent Bitcoin pooled mining protocol (supersedes the previous mining protocol, *getwork*), where the full block data is sent to miners. This allows miners to change the content of the block by themselves, thus gaining autonomy with respect to the pool servers. *Stratum* is a protocol first designed for lightweight clients and later extended to handle pooled mining. When used by mining pools, pool operators only send the block header to miners, thus optimizing the network throughput and storage requirements of miners but giving them less autonomy to decide what to include in the blocks. *Falcon* [9] is a backbone of nodes intended to make Bitcoin data propagation faster. Peers can connect to Falcon using either the Bitcoin protocol or a specially designed network protocol that relays packets as received (instead of waiting for all packets of a full block to be received before starting to relay that block). Similarly, *FIBRE* (Fast Internet Bitcoin Relay Engine) [47] is a protocol that uses UDP with Forward Error Correction to decrease the delays produced by packet loss. It also introduces the use of data compression techniques to reduce the amount of data sent over the network. Finally, the Lightning Network [88] was design as a layer two solution for Bitcoin scalability issues, and has been recently deployed within Bitcoin. In contrast to the aforementioned protocols, the Lightening Network works as an overlay of the Bitcoin network, allowing the creation of off-chain payment channels. In this context, *FLARE* [89] was proposed as an approach of routing for the Lightning Network.

3.1.2 Network overview

In terms of topology, the Bitcoin network can be characterized by splitting nodes in three broad categories, as depicted in Figure 3.2:

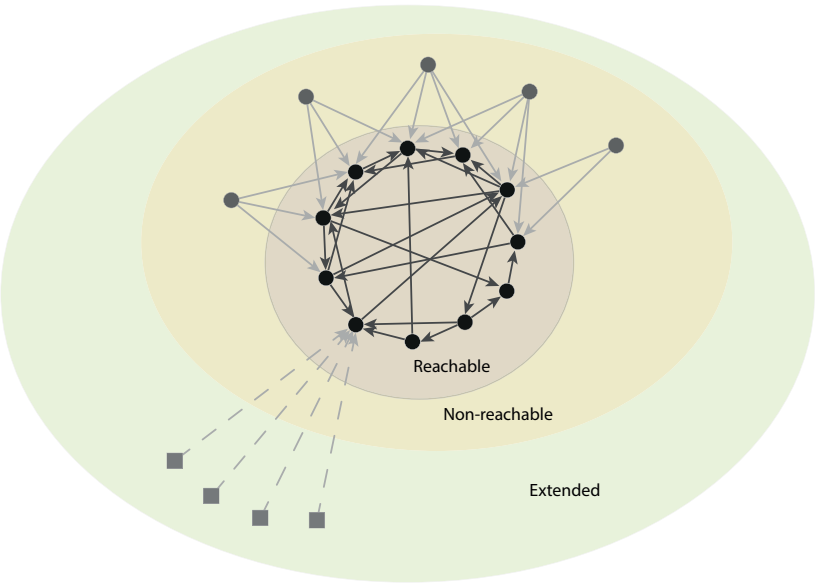

Figure 3.2: Abstraction of the defined network subsets: nodes in the *reachable* network accept incoming connections; nodes in *non-reachable* network just create outgoing connections; nodes in the *extended* network do not implement the Bitcoin protocol (for instance, miner clients using the stratum protocol).

- The *reachable* Bitcoin network is composed of all listening nodes that speak the Bitcoin protocol. The size of the reachable Bitcoin network is estimated to be around 10 000 nodes [114].

- The *non-reachable* Bitcoin network is made of nodes that speak the Bitcoin protocol, regardless of whether they are listening for incoming connections. The size of the non-reachable Bitcoin network is estimated to be 10 times bigger than that of the reachable Bitcoin network.

- The *extended* network comprises all nodes in the Bitcoin ecosystem, even those not implementing the Bitcoin protocol. This network includes, for instance, pooled miners communicating with the pool server using only the

stratum protocol. To our best knowledge, there are no estimations on the number of nodes that belong to the extended network.

Both the *reachable* and *non-reachable* Bitcoin networks are P2P networks: they are distributed systems build without mediation of a centralized server or authority, they can adapt to changes in the network and their participants autonomously, their nodes contribute with storage, computing power and bandwidth to the network.

Notice, however, that such topology is purely theoretical, and it is based on the connection capabilities of each type of node. Some studies have shown partial results of analysis of the topology for the reachable network [77], as well as thechniques to infer such topology [53]. However, to the best of our knowledge, no up-to-date snapshot of the topology is known.

In this section, we have provided a detailed description of the Bitcoin network by first describing the main properties that define a Bitcoin peer, then identifying the most common Bitcoin peers, and finally providing an overview of the network. Having described the Bitcoin network, the next section provides its characterization as a P2P network.

3.2 Network characterization

In this section we present an analysis of the Bitcoin P2P network using the taxonomy defined by Lua *et al.* [71] for the comparison of different P2P overlay network proposals. Such an analysis aims to characterize the new P2P network paradigm that cryptocurrency networks represent. Following the same taxonomy, we will be able to stress the differences of such new networks in comparison with the existing ones. The following analysis is performed aiming only at the Bitcoin reachable network, following the classification established in Section 3.1, since it is the only full P2P part of the Bitcoin network[3].

[3]Notice that, even the non-reachable network is also P2P, their nodes are forced to connect to nodes in the reachable network only.

3.2.1 Decentralization

Decentralization assesses to what extend the network presents a distributed nature or, on the contrary, its configuration shows some centralized characteristics. In that sense, the Bitcoin network is a non-structured P2P overlay with some similarities with Gnutella [66]. With a flat topology of peers, in the Bitcoin network every peer is a server or client and the system does not provide centralized services neither information about the network topology.

3.2.2 Architecture

The architecture describes the organization of the overlay system with respect to its operation. As we already mentioned, the Bitcoin network presents a flat architecture with no layers nor special peers. The network is formed by peers joining the network following some determined basic rules, where randomness is an essential component. Such random behaviour in the network creation intends to generate an unpredictable and uniform network topology, unknown to its users. As we will see in Section 3.3, such lack of knowledge about topology is needed for security reasons.

3.2.3 Lookup protocol

One of the main problems in P2P networks, specially those used for content distribution, is the lookup query protocol adopted by the overlay to find the desired content. However, although Bitcoin network can be regarded as a content distribution network (where content are transactions and blocks), the information flowing in the network is completely replicated in every node. Hence, there is no need for such a lookup protocol, since information is always available at one hop at most. However, information propagation has to be performed in order to synchronize all peers of the network with the same data. Such propagation is performed through a controlled flooding protocol.

Mainly, the controlled flooding protocol works on a push basis, propagating the data as it is generated. In order to reduce the amount of traffic while maintaining

a low latency propagation, each node selects (for every piece of information to be forwarded) a small subset of his connected nodes and sends the information to them. Instead of being directly sent, data availability is announced to the selected peers, and in case a peer lacks some of the announced information, it will be requested back to the announcer. Two type of data structures are propagated through the network following the aforementioned approach: *transactions* and *blocks*.

- Transaction propagation: transactions are the basic data structure flowing though the Bitcoin network, and the one most usually seen. Every single node, independently of its type, can take part in a transaction by simply using a wallet. Transactions flow though the network aiming to reach every single node to, eventually, be included in a block.

- Block propagation: blocks are the data structure the blockchain is made of, and include some of the transactions that have been created during the block mining process. Unlike transactions, blocks require a tremendous hashrate to be generated, which virtually limits their creation to mining pools. Moreover, the block generation throughput is set by design to 6 blocks per hour on average, periodically adjusting the block mining difficulty according to the total network hashrate.

Nonetheless, a pull data synchronization mechanism is also performed in the network, and while having a quite specific use, it is fundamental for proper operation of the system. Its main purpose is to synchronize the blockchain of outdated nodes, that have been off-line when data was originally propagated. Outdated nodes request an on demand synchronization to their peers during the bootstrapping phase, obtaining all the missing blocks in their local blockchain. Such request does not refer to specific blocks but to all blocks above the last block the enquirer is aware of. Beside blocks, on demand propagation of other type of data, such as transactions, is not set by default. Only nodes that have build a full index of transactions along the blockchain, like *block explorer services*, can

provide this type of data, since regular nodes only keep track of transactions bounded to their addresses.

3.2.4 System parameters

Different P2P network overlays require a set of system parameters for the overlay system to operate. For instance, structured P2P networks require to store information on the distribution of peers in the network in order to improve routing performance. However, the Bitcoin P2P network, in line with other unstructured P2P overlays, does not require any special system parameters for the normal behaviour of the network. Every single node can join the network with no prior knowledge of it. Apart from that, some default parameters are used by nodes, such as the maximum connection limit set to 125, the minimum relay fee for transactions set to 1 sat/byte, or whether the node will relay non-standard transaction or not, among others. However such values are not a restriction and each node can set them to match their needs.

3.2.5 Routing performance

Differing from traditional P2P networks, Bitcoin does not follow a multi-hop routing scheme. Peers in the network store a replica of all the information that has been flowing through the system up to the date, namely the blockchain. In that way, no queries are forwarded between peers, since all information should be found at one hop. Therefore, data is guaranteed to be located if the network is synchronized, and no routing protocol is needed nor used, apart from the synchronization protocol.

3.2.6 Routing state

Despite being a content distribution network, the routing state of Bitcoin cannot be directly defined due to the randomness and dynamism of its topology, and to the fact that it is not known. Moreover, as we have pointed out before, no multi-hop routing is performed since data can be found at one hop at most.

3.2.7 Peers join and leave

How to build the network is a classic problem P2P networks have to deal with. From building the network from its roots, to how nodes deal with peers disconnection, P2P networks need to be highly adaptable to avoid partitioning. In order to deal with this problem, and also provide a fair and secure way to choose the peers to connect to, the Bitcoin network performs a particular network discovery mechanism.

By default, all nodes maintain up to 125 connections with other peers. Each node will start 8 of those connections with other peers (namely outgoing connections), and will accept up to 117 from potential peers (namely incoming connections). Despite the name, all connections are bidirectional. In order to pick the outgoing connections, every single node will look for a subset of nodes it stores in a local database (the `addrman`). The `addrman` is formed by two different tables: *tried* and *new*. *Tried* table contains addresses from peers the node has already connected to, and *new* table contains addresses the node has only heard about. Additionally, when the node tries to establish a connection to the network for the first time, it queries a well known list of DNS nodes that will provide a set of on-line potential peers[4].

Nodes try to always maintain their 8 outgoing connections, selecting new peers from the database if any of the established connections is dropped. Peers are stored and selected from the database following a pseudo-random procedure that gives the network high dynamism and keeps its structure unknown. Peers information can be obtained by a node following two ways. First of all, a node can request such data to its neighbours, in order to fill up its database, or can receive such information spontaneously from one of its peers without any kind of request. In both cases, up to 2500 peers addresses from the neighbour's *addrman* (both from `tied` and `new`) are sent to the requester. Such addresses are stored in the requester's *new* table. On the other hand, a message containing a single address can be sent to a node when a node wants to start a connection with a potential

[4]Further information about how peers are stored and selected can be found in [57].

peer. By sending its address, the sender notifies the receiver that he has been picked as a peer, and, if the later has room for more incoming connections, the communication is established. Peer addresses received in that way are stored in *tried* table. All addresses are stored in the database along with a timestamp, that helps the node to evaluate the freshness of such address when selecting a peer.

3.2.8 Reliability and fault resiliency

Reliability and fault resiliency analyse how robust the overlay system is when subjected to faults. Typically, such robustness measurements are related to non intentional failures, for instance by a massive disconnection of peers of the network or an increasing volume of information being transferred through the network, but do not include intentional attacks that would be categorized inside the security properties of the network.

Bitcoin implements a distributed consensus protocol resilient to Byzantine faults. That is, the protocol is resistant to arbitrary faults produced in the participating peers, from software errors to adversary attacks. The main idea behind this protocol is to use a proof-of-work system to build the public ledger where transactions are stored. Appending new information to the public ledger requires a huge amount of computer power, thus preventing attackers to monopolize ledger expansion and censuring transactions. In a similar way, changing the content of the blockchain is also computationally expensive, as we have already introduced in Chapter 2.3 Additionally, the blockchain is replicated on all full blockchain nodes, contributing to the fault resiliency of the system and providing high availability of the ledger data.

Assuming that categorization, the Bitcoin P2P network has been designed with a high level of reliability due to the redundancy that implies the storage of all the relevant information in every peer of the network. With this approach, the high inefficiency level in terms of storage space is translated into a high resilience of the network since the availability of a single node in the network contains the information to keep the system alive. Moreover, the proof-of-work system

allows peers to (eventually) reach a consensus, even in presence of attackers trying to subvert the system. As a drawback, the consensus protocol is rather slow, with transactions needing 11 minutes (median confirmation time for May 5th, 2018 [20]) to confirm, and expensive, requiring the consumption of lots of energy for each mined block.

3.2.9 Security

Security in P2P networks has always been a broad topic since multiple security threats can be identified in different P2P implementations. The interested reader can refer to Wallachs's survey [110] for an introduction to the topic of security in general P2P networks; to Bellovin's paper [12] for a description focused on the security issues affecting specific P2P protocols such as Napster and Gnutella; and to [105] for an introduction to security problems in P2PSIP communications.

However, in P2P cryptocurrency networks, security takes a different twist. At first sight, one could believe that the threats P2P cryptocurrency networks face are a subset of the threats found in standard P2P networks. However, as we will see in detail in next section, most of the threats encountered in general P2P networks do not apply directly to P2P cryptocurrency networks, due to the cryptographical mechanisms used by the currencies and the degree of security offered by their protocols.

Additionally, one can also believe that multiple new threats will also arise in cryptocurrencies due to the sensitivity they have as money transfer networks. However, as we will see in the next section, this is not the case.

In the next section (Section 3.3), we provide a detailed review of the most common security threats identified for typical P2P networks and discuss to what extent they affect the Bitcoin network.

3.3 Security concerns in P2P networks

Security in P2P networks has been extensively studied in the literature. In this section we provide a broad overview of the main security problems that arise in

P2P networks, we review how each of the security problems may affect the Bitcoin network and, from those which affect it, we explain the specific countermeasures Bitcoin provides in order to defend from them.

The list of reviewed attacks goes over the most typical types of attacks and security flaws found in common P2P networks. It is clear that specific networks and applications might present specialized attacks but in most cases they can be seen as an specification of the attacks presented here.

So as to provide a clear picture of how common P2P attacks affect Bitcoin, we first review the three attacks that have been shown to be clearly applicable to Bitcoin. After that, we include a list of attacks identified for common P2P networks but that do not have such a high impact on Bitcoin, reviewing why the attacks do not apply to the specific Bitcoin network and detailing the particular cases where those attacks (or some variation) may somehow relate to Bitcoin.

In favor of a clear and concise presentation, we have not explicitly covered some recent attacks, such as [24], which do not directly affect or involve the Bitcoin network. Network related attacks such as [4] are also out of the scope of the study, since they rely on BGP hijacking.

3.3.1 DoS Flooding

Denial of service (DoS) attacks are possible in most P2P scenarios and are especially relevant, for example, in P2P streaming applications [23, 97, 111]. Given their dynamic nature, P2P networks are usually more resilient against generic DoS attacks than more static networks. Targeted DoS attacks to specific parts of the P2P network (a given node) or services are usually more important.

There exist several potential DoS flooding attacks in Bitcoin, but the system has countermeasures in place. *Transaction flooding* is prevented by not relaying invalid transactions and imposing fees to valid transactions. On the one hand, transactions are signed by the senders in order to prove they are authorized to transfer those bitcoins. If the signatures of a transaction are not correct, the transaction is considered invalid and it is not relayed to the network. On

the other hand, and due to the increased of use of Bitcoin, most miners do not accept transaction without fees, making a *transaction flooding* attack expensive. *Block flooding* is prevented by only relaying valid blocks, which must contain a valid proof-of-work. In order for a block to contain a valid proof-of-work, its hash must be lower than a given target. Obtaining a block with such a hash is a computationally expensive task, thus performing DoS attacks with block data is infeasible. *Network data flooding* is easier than the previous two cases, because it is indeed possible to create valid network messages without paying fees nor spending computation cycles. However, Bitcoin has a banning protocol: peers may ban other peers for one entire day if their banning score surpasses a certain threshold. Such a score is increased due to sending duplicate version messages, sending large messages, as well as sending invalid blocks. Given the nature of Bitcoin, *cpu usage DoS* is possible by trying to make peers spend lots of time validating a transaction or a block. In order to prevent this kind of attacks, Bitcoin tries to catch errors before starting to validate a transaction, limits the number of signature operations per transaction and per block, and limits the size of the scripts. Finally, previous versions of the Bitcoin client were also susceptible to *continuous hard disk read attacks*, where an attacker repeatedly sent double-spend transactions that passed the initial checks and required to retrieve data from disk in order to be fully validated. This attack is now prevented by checking that the inputs of the transaction that is being validated are in the UTXO set before retrieving any information from disk.

3.3.2 Eclipse attacks

An eclipse attack occurs when an attacker creates (or has control of) a large number of distinct nodes that populate the whole neighbourhood of the victim node [25]. The attacker can then *eclipse* the view of the network that has the victim. Common solution for sybil attacks are usually insufficient to defense against eclipse attacks [95].

In a cryptocurrency network, isolating a node from the rest of the network

may enable two other attacks to the eclipsed peer. First, an eclipsed peer may undergo a censorship attack because the victim's transactions must pass through the attacker's nodes in order to reach the network. Therefore, the attacker may decide not to forward these transactions, thus censoring the victim. Secondly, if the eclipsed victim is a miner, the attacker can drop or delay the propagation of the new blocks found by the rest of the network. As a consequence, the victim wastes computation time trying to mine on top of old blocks.

Bitcoin has many defense mechanisms to prevent eclipse attacks, some of which were added recently, after a study pointed out some of the flaws of the implementation at the time [57]: the client restricts the amount of outgoing connections to addresses in the same network, randomizes the address selection procedure, and maintains a big list of peers, among others.

3.3.3 User profiling

In some P2P networks, it is easy to record all the activities of a given node, allowing attackers to easily create identifying profiles of users and their activities. This is relevant in anonymous systems or systems that want to guarantee a certain degree of anonymity [22, 105].

Bitcoin provides pseudonymity by allowing users to receive payments to their addresses, which are not initially linkable to their identities. The use of new addresses for each transaction in the system is intended to provide unlinkability between the different actions a single user performs through Bitcoin. Therefore, user profiling in Bitcoin usually consists in attacking the unlinkability between the different addresses a single user has. Three different approaches have been taken to perform address clustering: using network layer data [67], performing analysis over the transaction graph [1, 85, 92], and analysing Bloom filters [49]. The idea of using network layer data to cluster addresses is straightforward: if an attacker is able to connect to all the peers of the network, the first node that sends him a given transaction should be the creator of that transaction. Therefore, if the attacker first receives two different transactions from the same

peer, he can infer that the source addresses of both transactions belong to the same user. However, as simple as the attack may seem conceptually, it is not that easy to perform in practice. It is not trivial to connect to all nodes of the network, since most of them do not accept incoming connections. Moreover, some peers anonymize their connections using Tor. Finally, collected data is very noisy and, therefore, it is not easy to make strong claims when analysing it. Regarding transaction graph analysis, there exist mixing services that are able to effectively break the relationship between an address and its past. Finally, concerning to the usage of bloom filters, users must be very careful when choosing the parameters of the filter and when generating different filters that match the same set of addresses and public keys. In that sense, new protocols are being designed to allow lightweight clients to retrieve their transactions of interest while maintaining privacy [86].

Bitcoin's scalability problems have triggered the search for new solutions that would allow to increase the transaction throughput of the network. Several proposals provide mechanisms to create off-chain payment channels, so that secure transactions between Bitcoin users may be performed without needing to include all the transactions into the blockchain. In turn, these solutions may also entail privacy problems that are yet to be carefully studied [58].

3.3.4 Other attacks

After analysing the three main attacks that have threatened the Bitcoin network over the last years, we summarize other common P2P attacks that have a lesser impact on Bitcoin. We will show how some of those attacks could be used as a preliminary phase to achieve one of the three previously introduced ones, while others are not harmful for the Bitcoin network due to its design.

3.3.4.1 ID attacks

Two different sub-attacks can be identified in this category:

ID mapping attack: when a node changes its own identifier with malicious purposes. As an example, in DHT-based P2P networks a node can gain control over given resources by changing its own identifier [26]. These kind of attacks are more difficult in networks where the identifier is derived from a public key [96].

ID Collision Attack: similarly to previous attacks, here the attack is considered to happen when there are duplicated identifiers. The problem is usually prevented by ensuring the uniqueness of identifiers [105].

There is no clear concept of a peer identifier in Bitcoin. Two different properties could be considered identifiers in Bitcoin, depending on the exact entity one wants to identify: IPs and Bitcoin addresses. IPs allow to identify peers, whereas addresses are linked to users. A malicious peer may benefit from a change of IP if it is banned for misbehaviour. Being able to change the IP allows a peer to effectively reset its banning score (refer to Section 3.3.1 for details.). Regarding the second kind of identifiers, Bitcoin addresses, the recommended behaviour for users is indeed to change them frequently. In fact, the suggestion is not to reuse addresses but to create a new address for each transaction made in the system, which helps protecting users privacy.

3.3.4.2 Sybil attack

A sybil attack is a well know attack in P2P networks, where a malicious user creates multiple identities in order to control the system or parts of the system [42]. This has been very extensively studied in the literature in the context of several P2P technologies [69, 82].

Sybil attacks may be a problem in Bitcoin if they are able to eclipse all the connections from a peer (see Section 3.3.2 for details of eclipse attacks). However, besides its extension to an eclipse attack, a peer with multiple identities cannot harm the system regarding the main content of the network: blocks and transactions. The former cannot be counterfeit without the corresponding proof-of-work and the generation of the later entails an associated fee (in a similar way that described in *flooding attacks* in Section 3.3.1). Nevertheless, if lots of sybil nodes start performing a huge amount of connections to the existing network,

they may monopolize all available incoming connection slots, and the system decentralization could be reduced.

3.3.4.3 Fake bootstrapping

Network access in P2P environments starts by connecting to one or multiple nodes of the network. This first contacted node is known as the bootstrap node. A malicious bootstrap node can influence the view of the network for the new user [25]. Several solutions already exist for this problem in general P2P networks, such as not relaying in a single bootstrap node, use of cached peers for subsequent connections, random address probing, using external mechanisms, specific bootstrapping services, or using network layer solutions (e.g. use of an special multicast group for bootstrapping) [31, 40, 105].

Bitcoin deals with bootstraping issues by defining a local peers database on every single node that is queried following a pseudo-random protocol to obtain a subset of potential peers (see Section 3.2.7 for details). In that way, Bitcoin applies most of the solutions for the fake bootstrapping protocol, such as *not relaying in a single bootstrap node* by establishing 8 outgoing connections on every bootstrap, *use cached peers for subsequent connections* by using peers stored in *tried* table, *random address probing* by using a pseudo-random protocol to store and retrieve peers addresses from the `addrman`, and *using external mechanisms* by querying a list of well known DNS nodes or even using a list of hardcoded nodes, if the DNS cannot be reached.

3.3.4.4 Unauthorized resource access

P2P networks often use some sort of private data that has to be protected from unauthorized access. Common solutions are those typically employed for distributed access control [106, 115].

Bitcoin is based on public key cryptography, where private keys are needed to authorize payments. Therefore, private keys must be kept secret and two methods are usually employed: encryption and offline storage. By using encryption, private

keys remain secure even if an attacker is able to retrieve the key file as long as the encryption key remains secret. As for offline storage, different approaches can be followed, from the usage of dedicated hardware devices to paper wallets. Notice that, unlike other uses of public key cryptography where private keys need to be online (for instance in the handshake process in TLS), Bitcoin network operation does not involve private information since validations are performed using public information. For that reason, offline storage of private keys does not impact the network performance.

3.3.4.5 Malicious Resource Management

A malicious node can deny existence of a given resource under its responsibility, or claim to have a resource it does not have. This is specially relevant in content distribution applications, and common solutions are replication of resources [51] or use of error correcting codes to reconstruct missing parts of the resource [112].

Bitcoin network is protected against malicious resource management by, on the one hand, the high amount of data redundancy information of the network and, on the other hand, the multiple neighbours a node of the network is connected to. As long as nodes establish connections with multiple peers, if a given neighbour denies the existence of a certain resource, the node can learn it from his other neighbours. Moreover, if a neighbour says he has some resource he actually does not have, peers will notice when they try to retrieve it, since transactions and blocks are identified by their hash.

3.3.4.6 Free-Riding

A free-rider in a P2P network is a node that attempts to benefit from the resources of the network (provided by other users) without offering their own resources in exchange [59, 84]. Depending on the application, this might not be an issue or even might not be considered a security problem. It is usually described in content distribution applications and the main solutions proposed rely on incentive or penalty based mechanisms [46].

Bitcoin is sustained by an equilibrium of economic incentives. Miners are remunerated for their work by obtaining a reward for each block they successfully mine. Additionally, transaction senders (and, although indirectly, also transaction recipients) may include a fee to their transactions, which is also collected by the miner of the block that contains the transaction. As a consequence, miners are encouraged not only to create blocks but also to include transactions on those blocks. There is, however, a set of nodes whose role is important in ensuring the decentralization of the network and that do not directly receive economic incentives for their work: full clients. While these clients store the blockchain and perform validation and relaying of transaction and blocks, they do not get a direct economic reward in return for their work.

3.3.4.7 Man-in-the Middle (MITM)

In the context of P2P networks, a MITM attack is usually considered a routing attack, similarly to classical network MITM attacks. P2P networks which require multi-hop routing, will need to include measures similar to onion routing in order to secure connections between all nodes along the path [52, 62].

MITM attacks in Bitcoin are not a problem for transaction and block integrity, because transactions are cryptographically signed and blocks must contain a valid proof of work. Transaction malleability may be a problem in very specific scenarios (refer to Section 3.3.4.10 for a detailed explanation). Censorship is not a problem either, because a single peer maintains different connections. An attacker must be in the middle of all of them to hide information to the peer (thus resorting in *eclipse attacks*).

3.3.4.8 Replay attack

A replay attack is produced when a legitimated transmission is delayed or lately replayed with malicious purposes. This is a very common network attack that can affect P2P networks in several ways, but it is usually solved at a protocol level.

Replaying transactions or blocks that have been sent to the network does not have any effect on Bitcoin network. Nevertheless, delaying block propagation may be a beneficial strategy for miners [45]. By not immediately propagating a block the miner has just found, the miner can start working on top of this newly found block while making other miners lose time working on the previous block. This strategy is known as selfish mining and reduces the bound on the percentage of hashing power an attacker must have in order to successfully control the information appended to the ledger.

3.3.4.9 Routing disfunction

Routing disfuntion can be presented in different aspects. On the one hand, *incorrect routing* involves attacks where a node routes messages incorrectly (or drops them) [25]. These attacks might not be relevant in P2P networks that do not provide multi-hop routing. Due to the flooding mechanism used to propagate information through the network, the consequences of a single node dropping messages are negligible.

On the other hand, in a *fake routing update*, the attacker tries to corrupt a given route (equivalently to corrupt a routing table for a given node) [25, 72, 96]. As we have mentioned previously in Section 3.2, there are no routing tables in the Bitcoin network. The most similar information a peer stores is addresses from other peers. Note that no information about where peers are in the network nor their connections are stored by the Bitcoin client, just the address and a timestamp. Therefore, the attack that better resembles fake routing updates in Bitcoin is to send fake addresses. This kind of attacks are usually performed as a first step in eclipse attacks, already described in Section 3.3.2.

3.3.4.10 Tampering with message bodies

When using multi-hop routing, intermediate nodes can modify the content of the relaying packets. End-to-end integrity has to be provided in order to, at least, detect this type of attacks [105]. Tampering with the content of a block will change

its hash, what which also invalidate its proof-of-work, making it an infeasible attack on Bitcoin. However, transactions are a signed data structure with the signature cryptographically protecting its integrity. Therefore, an attacker cannot tamper with a transaction to its will, for instance, by changing its destination address. There is, however, a very specific situation where this kind of attack would be possible. Bitcoin transactions are malleable, that is, an attacker can change some part of the transaction while keeping the signature valid. This happens mainly because not all parts of the transaction are signed (e.g. the signatures themselves are not signed). Malleability is a problem for Bitcoin when a user is dealing with zero-confirmation transactions (refer to Chapter 2.4.2). If a transaction is part of a protocol where transactions are identified by their hash, like off-chain protocols, the attacker can take advantage of it. However, a solution for such a problem, called segregated witness [70], has recently been deployed within Bitcoin. Segregated witness teaks the transaction serialization format moving signatures to a new field called witness. The witness is not part of the fields hashed to compute the transaction identifier, what solves the signature malleability problem, and therefore reduces the impact of such attack.

3.4 Conclusions

In this chapter we have characterized P2P cryptocurrency networks by providing an analysis of the most relevant cryptocurrency nowadays: Bitcoin. By characterizing P2P cryptocurrency networks using well-known taxonomy in the field of P2P networks, we can conclude that such networks present a new paradigm due to the main properties that a cryptocurrency has to provide: reliability and security.

P2P cryptocurrency network reliability stands on top of a strong redundant mechanism regarding system information. As a result, the relevant information is replicated in all the peers of the network. With this approach, the availability of a single full node in the network contains the information to keep all the system alive. Notice that this approach turns out to a high inefficiency state regarding

storage space, so this strategy is not followed by any other P2P network paradigm. Furthermore, such approach also demands new synchronization mechanisms to provide all nodes with the same correct information.

Trough the security analysis, we have shown how most of the attacks to regular P2P network do not apply to cryptocurrency network by design. However, due to its use as a financial network, a few of them can pose high threads, this being the case of eclipse attacks, which can make users incur in monetary loses. Hence, some information about the network needs to be protected, as it is the case of the topology. Due to data being available at one hop and to the controlled flooding protocol, such information is not required to maintain the network synchronized. Notice that this topological secrecy property of P2P cryptocurrency networks is not so relevant in other P2P network paradigms and, for that reason, the mechanisms to achieve it are also particular of such environments.

CHAPTER 4

ANALYSIS OF THE BITCOIN UTXO SET

Bitcoin makes use of the Unspent Transaction Output (UTXO) set in order to keep track of transaction outputs that have not been yet spent and thus can be used as inputs of new transactions (see Chapter 2). Bitcoin full nodes keep a copy of the UTXO set in order to validate transactions and produce new ones without having to check for unspent outputs throughout the blockchain. This allows, for instance, the use of sthe o called pruned nodes (introduced in Bitcoin Core v0.11 [99]), which can operate without having to persistently store the whole blockchain.

The UTXO set is therefore a key component of Bitcoin. The format, content, and operation of this set has an important impact on Bitcoin nodes' operations. The size of the UTXO set directly impacts on the storage requirements of a Bitcoin node, and its efficiency directly determines the node validation speed.

We believe that a deep understanding of the Bitcoin UTXO set is needed to clearly understand the operation of Bitcoin, helping to find potential scalability and efficiency problems. To that end, we present STATUS (STatistical Analysis

Tool for UTXO Set), a tool to analyse the UTXO set of Bitcoin. To the best of our knowledge, there is no clear description in the literature of the UTXO set, its format, and how to actually analyse it. We provide, in this chapter, such description along with a deep analysis of the set, and the tools needed to perform it [39].

4.1 The UTXO set

The Unspent Transaction Output (UTXO) set is the subset of transaction outputs that have not been spent at a given point in time. Whenever a new transaction is created, funds are redeemed from existing UTXOs, and new ones are created. Basically, transactions consume UTXOs (in their inputs) and generate new ones (in their outputs). Therefore, transactions produce changes in the UTXO set. Regarding UTXOs, they can be identified by their **outpoint**, a two field data structure containing the transaction id and the output index of the transaction that created them. They store two data fields: the amount they are holding, and the locking script (`scriptPubKey`) that specifies the conditions under which they can be redeemed.

The main purpose of the UTXO set is speeding up the transaction validation process. When a new block is appended to the blockchain, full nodes update their vision of the UTXO set, removing the outputs that have been spent in the block and adding the newly generated ones. Being all unspent transactions outputs stored in the set, there is no need to scan throughout the blockchain to check for double-spends, but just check if the inputs of a transaction can be found in the set.

The format and the storage requirements of the UTXO set may differ depending on the specific implementation. However, their content must be consistent between all the implementations, since all of them need to be able to verify every transaction. For our analysis, we have chosen the Bitcoin Core implementation of the set, since we aim to extend our analysis to other cryptocurrencies based on Bitcoin in the future, which may use the same approach. However equivalent

results can be obtained by any other implementation that follows the UTXO set approach.

In the Bitcoin Core's implementation, the UTXO set is stored in the chainstate, a LevelDB database that provides persistent key-value storage. LevelDB [50] is used to store the chainstate since Bitcoin v0.8. Apart from the UTXO set, the chainstate stores two additional values: the block height at which the set is updated and an obfuscation key that is used to mask UTXO data [100, 109]. Such an obfuscation key is used to obtain a different file signature of the UTXO set file for every different wallet in order to avoid false-positives with antivirus software.

The format of the chainstate database changed in version v0.15 of the Bitcoin Core, and it is the one used at the time of writing. We will refer to the previous format as 0.14, although it has been used in versions from 0.8 to 0.14.

4.1.1 The UTXO Bitcoin Core 0.14 format

The chainstate database of Bitcoin Core v0.14 uses a per-transaction model: there exists a record in the database (i.e., a key-value pair) for each transaction that has at least one unspent output. Multiple UTXOs belonging to the same transaction are thus stored under the same key. The key of the record is the 32-byte transaction hash, preceded by the prefix `c`. This prefix is needed to distinguish transactions from other data that are also stored in the database, and is also used to discriminate v0.14 format from v0.15's.

The value of the record stores metadata about the transaction (version, height and whether it is coinbase or not) and a compressed representation of the UTXOs of the transaction [101].

Regarding the UTXOs, the encoding first identifies the indexes of the transaction outputs that are unspent and then includes information about them[1]. UTXOs are then encoded depending on their type. They store a script and a compact representation of the amount of bitcoins hold. There are six types of

[1]The encoding is optimized to favour the first two outputs.

outputs where a simplified script is stored, since they have a fixed structure with some variable data (refer to Chapter 2.2.1 for details), while any other output type needs to store the whole script. Such 6 types are P2PKH, P2SH and four different cases of P2PK scripts. For instance, P2PKH outputs only store the `hash160` of the address.

4.1.2 The UTXO Bitcoin Core 0.15 format

One of the main changes from Bitcoin Core's v0.15 release was a change in the internal representation of the chainstate in favor of a better performance, both in reading time and memory usage [74, 102].

Such new format uses a per-output model in contrast to the previously defined per-transaction model, that is, every entry in the chainstate now represents a single UTXO instead of a collection of all the UTXOs available for a given transaction. To achieve this, the key-value structure (known as *outpoint-coin* in the source code) has been modified. Keys encode both the 32-byte transaction hash and the index of the unspent output, preceded by the prefix `C`. Regarding *coins*, each one encodes a code that contains metadata about the block height and whether the transaction is coinbase or not (notice that the transaction version has been dropped), a compressed amount of bitcoins, and the output type and script encoded in the same way as in version 0.14.

Storing unspent outputs one by one instead of aggregated by transaction greatly simplifies the structure of the *coin* and reduces the UTXOs accessing time. By using the previous structure, when a transaction with more than one unspent output was accessed, all data needed to be decoded and all the unused outputs encoded and written back into the database. However, this new format has the downside of increasing the total size of the database [74].

4.1.3 *STATUS*: The UTXO analytic tool

We have created *STATUS (STatistical Analysis Tool for Utxo Set)*, an open source code tool that provides an easy way to access, decode, and analyse data from the

Bitcoin's UTXO set[2]. STATUS is coded in Python 2 and works for both the existing versions of Bitcoin Core's UTXO set, that is, the first defined format (versions 0.8 - 0.14) and the recently defined one (version 0.15). STATUS reads from a given chainstate folder and parses all the UTXO entries into a file. From the parsed file, STATUS allows you to perform two types of analysis: a UTXO based one and a transaction based one, both by decoding all the parsed information from the chainstate.

In the UTXO based analysis, apart from the data mentioned in Sections 4.1.1 and 4.1.2 that STATUS directly decodes, it also creates additional meta-data about each parsed entry, such as dust and unprofitable fee rate limit that will be deeply analysed in Section 4.2. Regarding transaction based analysis, STATUS aggregates all the parsed UTXOs that belong to the same transaction, providing additional meta-data such as total number of UTXOs from a given transaction, total unspent value of the transaction, etc. Finally, STATUS uses *numpy* and *matplotlib* Python's libraries to provide several statistical data analyses.

4.2 UTXO set analysis

In this section we analyse the UTXO set of the Bitcoin blockchain state at block 491,868, corresponding to the 26th of October 2017 at 13:13:38 using the STATUS tool. First, we provide a general view of the data included, regarding the total number of outputs and their size depending on the Bitcoin Core UTXO set format. We also analyse different output subsets within the UTXO set that could be interesting to measure in order to provide some hints whether a more efficient UTXO set codification could be used.

4.2.1 General view

Using STATUS, we can retrieve details related to the general numbers behind the UTXO set. Table 4.1 presents a summary of such basic facts of the analysed

[2]It can be found under a bigger Bitcoin Tools library at `https://github.com/sr-gi/bitcoin_tools/tree/v0.1/bitcoin_tools/analysis/status`.

data. There are 52 and a half million UTXOs in the set, belonging to more than 23 million different transactions. Although this gives an average of 2.26 UTXOs per transaction, the distribution is very skewed, with most of the transactions having just one unspent output.

	v0.14	v0.15
Num. of tx	23,241,914	
Num. of UTXOS	52,543,649	
Avg. num. of UTXOS per tx	2.26	
Std. dev. num. of UTXOS per tx	18.27	
Median num. of UTXOS per tx	1	
Size of the (serialized) UTXO set	2.02 GB	3.00 GB
Avg. size per record	93.45 B	61.46 B
Std. dev. size per record	443.20	7.65 B
Median size per record	62	61

Table 4.1: UTXO set summary.

Figure 4.1 shows a cumulative distribution function (cdf) of the number of UTXOs per transaction[3]. Note that 87.9% of the transactions have only 1 UTXO[4] and 94.97% have less than 3. The maximum number of UTXOs per transaction is 3,452 [5] which originally had 5,419 outputs.

Differences between both data formats (v0.14 and v0.15) are clear regarding the serialized UTXO set size (see Table 4.1). While the v0.14 format uses 2.02GB with an average size per record of 93.45 bytes (a total of $23,241,914$ records), the 0.15 format expands the information to 3.00GB which represents an average size per record of 61.46 bytes (with $52,543,649$ records). Such a difference is due to the way outputs are stored in both formats, as detailed in Section 4.1. However,

[3] All the analysis plots included in this section show cumulative distribution functions. Therefore, a point (x, y) in the plot shows the probability y that a given variable (depicted in the x axis label) will take a value less than or equal to x.

[4] Notice that such measure indicates that, although the average number of outputs in regular Bitcoin transactions is higher, the number of outputs that remain unspent is, mostly, only one.

[5] `https://blockchain.info/tx/d8505b78a4cddbd058372443bbce9ea74a313c27c586b7bbe8bc3825b7c7cbd7`

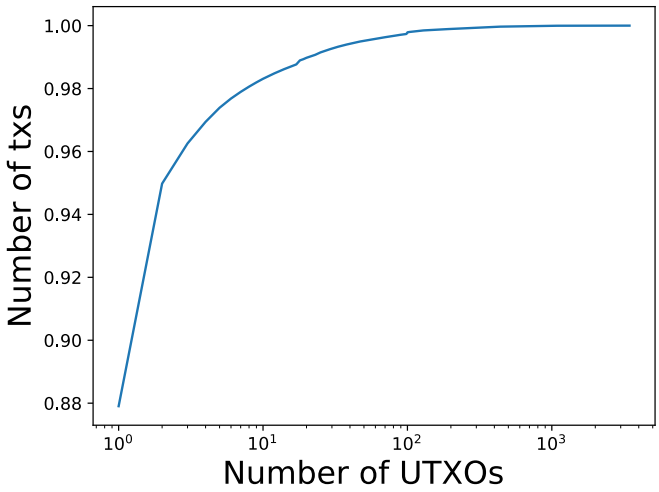

Figure 4.1: Number of UTXOs per transaction.

the median size per record of both versions is very similar, with most records being 59-64 bytes long. Such measurement is sound since both versions store the 32-byte transaction id and some identifier of the output, thus being the size difference for every record only significant when the transaction has more than a single UTXO. Whereas the number of records with less than 59 bytes is negligible (just 30 of them for v0.14 and 222 for v0.15), 83.25% of them in v0.14 and 99.0% in v0.15 are $\leq$ 63-byte long.

As a matter of fact, the smallest stored record in v0.14 is just 41-byte long [6] and contains a single non-standard UTXO with a 1-byte length script containing an invalid opcode. This UTXO is also one of the smallest records in v0.15, with 40 bytes (12 additional records have also the same size in v0.15). Section 4.2.4 provides an exploration of non-standard transactions in the UTXO set.

Another interesting information of the UTXO set that can be retrieved with

[6] https://blockchain.info/tx/8a68c461a2473653fe0add786f0ca6ebb99b257286166dfb00707be24716af3a

STATUS is the amount of UTXOs of each type, as detailed in Table 4.2. Notice that UTXOs are classified between the different standard types also providing a distinction between compressed and uncompressed keys for the P2PK type. As data show, more than 99% of the UTXOs are P2PKH and P2SH, being P2PKH the vast majority of them. In Section 4.2.4 we provide detailed information regarding the 0.8% of UTXOs classified as `others`.

Num. of utxos	52,543,649	100%
Pay-to-PubkeyHash (P2PKH)	43,079,604	81.99%
Pay-to-ScriptHash (P2SH)	8,987,799	17.11%
Pay-to-Pubkey (P2PK)	66,759	0.12%
Compressed	29,977	0.06% (44.90%)
Uncompressed	36,782	0.07% (55.10%)
Others	409,487	0.8%

Table 4.2: UTXO types.

Figure 4.2 provides information about the amount of satoshi hold by each UTXO, showing that 98.46% of the UTXOs hold less than one Bitcoin, with an average of 0.32฿ per UTXO.

4.2.2 Dust and unprofitable UTXOs

An interesting type of outputs included in the UTXO set are those whose economical value is small enough to represent a problem when they have to be spent. Dust is a well known definition of such type of outputs. According to the Bitcoin Core reference implementation [98], **a dust output** is the output of a transaction where the fee to redeem it is greater than 1/3 of its value. Besides this well known definition, we also define **an unprofitable output** as the output of a transaction that holds less value than the fee necessary to be spent, resulting in financial loses when used in a transaction.

In order to identify both types of outputs, it is important to recall that the amount of fees a transaction has to pay to be included in a new block depends

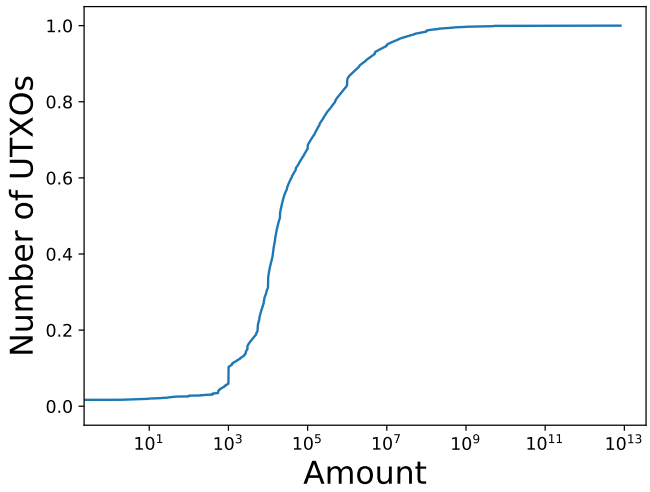

Figure 4.2: Amount of bitcoins per UTXO (in satoshis).

on two factors: the fee-per-byte rate that the network is expecting at the time of creating the transaction and the size of the transaction. The fee-per-byte rate, measured in satoshi, is a highly variable factor that depends on the transaction backlog (i.e. how many transactions are pending to be included in new blocks).

Since fees depend on the transaction size, in order to label the outputs in the UTXO set as a dust or unprofitable, we need an estimation of the size of data needed to spend such output. In order to identify the minimum information needed, we can consider an already standard transaction with its inputs and its outputs and enough fees to be relayed. Then, we define the **minimum-input of a UTXO** as the smallest size input that spends such UTXO. The size of such minimum-input, along with the value held in the output and the fee rate, will determine whether a UTXO may be included in the dust or unprofitable categories.

In order to measure the size of such minimum-input, we need to have in mind

the structure of a Bitcoin transaction (refer to Chapter 2.2). All transactions follow a standard structure containing some fixed length parameters, that determine a minimum transaction size, and some variable length parameters, depending on the transaction type. When a transaction is created, inputs are defined referring to some UTXOs. The sizes of such inputs depend on the type of the outputs they are claiming. Moreover, new outputs will be also generated in every new transaction, adding some additional size determined by the their type.

Depending on the UTXO type, its minimum-input size will be different. Such measure can be split in two parts: fixed size and variable size. Regarding the fixed size, three fields of a transaction have always the same size: `prev_tx_id`, `pev_out_index` and `nSequence`. Therefore, for every UTXO, its minimum-input will be at least 40-byte long independently of its type. On the other hand, the content and length of the fields `scriptSig` and `scriptSig length` depend on the UTXO type, specified in the field `scriptPubKey` of the UTXO.

The different types of outputs, with their corresponding size, can be classified as follows:

Pay-to-PubKey (P2PK) outputs: The minimum-input of this type of UTXO specifies just a digital signature to redeem the output and the `scriptSig` includes the following data:

```
PUSH  sig (1 byte) + sig (71 bytes)
```

Bitcoin uses DER encoded ECDSA signatures in the scripts of its transactions, which can be between 71 and 73 bytes long depending on their `r` and `s` components. Such variability comes from the randomness of the r parameter. Since we are defining the minimum possible input to be created, 71-byte signatures are considered. Hence, the `scriptSig` for a P2PK UTXO will be 72-bytes long and `scriptSig len` field will be 1-byte long, resulting in a minimum-input size of 73 bytes.

Pay-to-PubkeyHash (P2PKH) outputs: For this UTXO to be redeemed, both a signature (sig) and a public key (pk) are required in the `scriptSig`, as shown below:

```
PUSH  sig (1 byte) + sig (71 bytes) + PUSH pk (1 byte) +
pk (33-65 bytes)
```

Regarding the signatures, the same assumptions as for P2PK outputs applies, that is, 71-byte length can be considered. Regarding public keys used by Bitcoin, they can be either compressed or uncompressed, which will significantly vary their size:

- Uncompressed keys: such keys were used, by default, in the first versions of the Bitcoin Core client, and they are 65-byte long.
- Compressed keys: by 30th March 2012 (around block height 173480) Bitcoin core started using this more efficient type of keys, which are almost half size of the previous ones (33 bytes), and therefore make smaller scripts.

So, the size for the `scriptSig` varies from 106 to 138, and consequently, the `scriptSig length` field will be 1-byte long, resulting in a total minimum-input size between 107 and 139 bytes.

Pay-to-multisig (P2MS) outputs: the size of the minimum-input to redeem such a script highly varies depending on the number of signatures required, which ranges up to 20 (20-of-20 multisig)[7], so the `scriptSig` for redeeming such output is as follows:

```
OP_0  (1 byte) +  (PUSH  sig (1 byte) + sig (71 bytes)) *
required_signatures (1-20)
```

[7]Although the standard considers a maximum number of 3 signatures in a P2MS output, up to 20 are valid regarding the consensus rule [113] so they could potentially be found in the UTXO set.

Thus, the size of the `scriptSig` field will range between 73 and 1441 bytes, making the `scriptSig len` field range between 1 and 2 bytes, so the total minimum-input size will be between 74 and 1443.

Pay-to-ScriptHash (P2SH) outputs: unlike any previous output type, input size created from P2SH outputs can not be straightforwardly defined in advance. P2SH outputs hide the actual input script behind a hash in order to make smarter outputs, by making them smaller and thus allowing the payer to pay lower fees. However, the scripts held by those UTXOs give us no clue about how the minimum-input should be built.

Table 4.3 summarizes the sizes of the minimum-input for each UTXO type.

			scriptSig			
UTXO type	**Fixed size**	`scriptSig length`	**sig**	**pk**	**push data**	**Total size**
P2PK	40	1	71	-	1	113
P2PKH	40	1	71	33-65	2	147-179
P2MS	40	1-2	71-1420	-	2-21	114-1483
P2SH	40	?	?	?	?	40-?

Table 4.3: Minimum-input size summary.

Notice that the previous analysis does not take into account the new SegWit transaction format [70]. The minimum-input size for such type of outputs needs an extended analysis. However, at the time of performing the analysis, the total outputs in the UTXO set that correspond to a SegWit output is upper bounded by a 2.26% (see Section 4.2.1 and Section 4.2.4). Giving such small amount of data, the results presented here will not significantly change, therefore, we leave such analysis for further research.

Once we determined the amount of data of the minimum-input for each type of UTXO, based on a defined fee-per-byte rate, we can identify those outputs from the UTXO set that fall into both the dust and the unprofitable categories. To obtain such data, the following considerations have been taken. The minimum-input size for P2PK and P2MS outputs have been precisely computed since the information to determine the exact size of the minimum-input can be derived

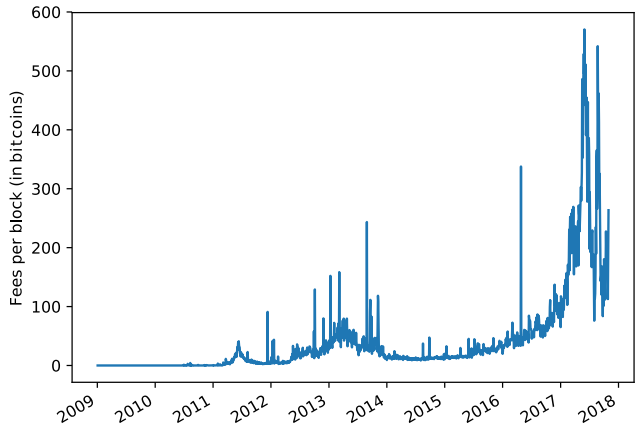

Figure 4.3: Evolution of fees (Source: Blockchain (`https://www.blockchain.info`).

from the output data itself. However, it is not possible to exactly determine such value for the P2PKH neither for the P2SH. In the first case, we have taken the following approach. For those outputs up to block 173480, we have considered uncompressed addressed, whereas for the newer ones, we have taken the most conservative approach assuming that all public keys from that point onwards are in compressed form (33 bytes), so reducing the number of UTXO that fall into both categories. For the P2SH, being not able to set a proper lower bound for the variable part, we have performed the analysis assuming only the fixed 40 bytes.

Finally, the last parameter to set is the fee-per-byte rate. As depicted in Figure 4.3, such rate is far from fixed and has high variability. Thus, in order to measure different possible scenarios, we have considered a wide fee-per-byte spectrum, ranging from 0 to for 350 satoshi/byte. The volume of both dust outputs and unprofitable outputs (blue and dotted orange lines respectively) in the UTXO

set are depicted in Figures 4.4, 4.5 and 4.6.

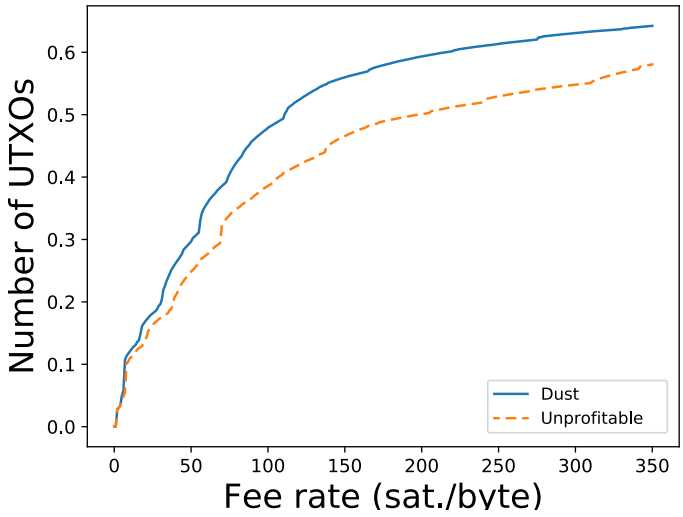

Figure 4.4: Percentage of dust/unprofitable UTXOs w.r.t. fee-per-byte.

Figure 4.4 shows the relative size of dust and unprofitable output sets within the total UTXO set. Notice that for a fee-per-byte as small as 80 satoshi/byte onwards, more than the 50% of UTXOs (26.29 million outputs) from the set can be considered dust, whereas the same 50% size for the unprofitable set is reached for 240 satoshi/byte onwards. Regarding the size of such data, Figure 4.5 shows how those UTXOs represent a relevant part of the total size from the set (more than the 50% for around 70 satoshi/byte onwards), while the same can be seen for unprofitable UTXOs for a rate of 200 satoshi/byte onwards. Finally, from an economic point of view, Figure 4.6 shows, as expected, how those dust and unprofitable UTXOs represent a negligible amount from the total value of the UTXO set, which is the total number of bitcoins in circulation.

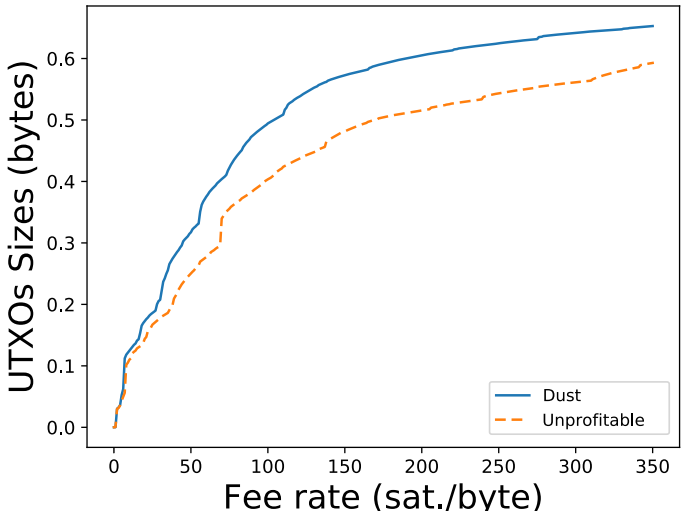

Figure 4.5: Percentage of occupied space w.r.t. fee-per-byte rate.

4.2.3 Height

Another interesting type of UTXOs are those that were created a long time ago. Although it is difficult to determine the average time in which a UTXO will be spent, some old UTXOs may belong to keys that are lost, meaning that they may never be spent.

Figure 4.7 depicts the height of the block where the transaction is included in a per-transaction (v0.14, blue line) and per UTXO (v0.15 orange line) fashion. Half of the stored UTXOs are older than January 2017 (block 449,896 corresponds to the median), whereas the other half are younger. This means that almost half of the current UTXO set is filled with UTXOs created in the last year (2017). On the other hand, there are still very old UTXOs: 2% of them are older than August 2012 (block height 194,635).

In Figure 4.8 we can see the evolution in time of the different types of outputs in the UTXO set. Notice that P2PKH and P2SH show a stable distribution in time.

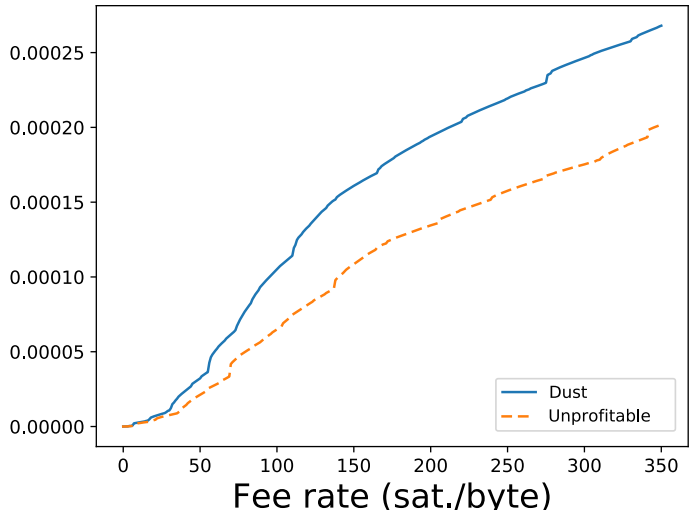

Figure 4.6: Percentage of economic value w.r.t. fee-per-byte rate.

Apart from that, outputs labelled as `others` are mainly from old transactions since 95% of them are older than March 2016 (block height 403,052). Finally, the chart also shows that P2PK outputs have an irregular behaviour. 50% of them were created before block 91,542, which is an expected result since P2PKH were developed afterwards as an improvement of P2PK. However, it is interesting to see that, after a long time with very few outputs of this type, around March 2017 and during 324 blocks, 15% of the P2PK outputs included in the UTXO set were created.

Figure 4.9 shows an already known fact that indicates that most of the bitcoins created at the beginning of the cryptocurrency are still pending to redeem. More precisely, 75% of the coinbase outputs in the UTXO set were created before block 274,946 (December 2013). In contrast, just 6% of the current UTXOs were created before that block (see Figure 4.7).

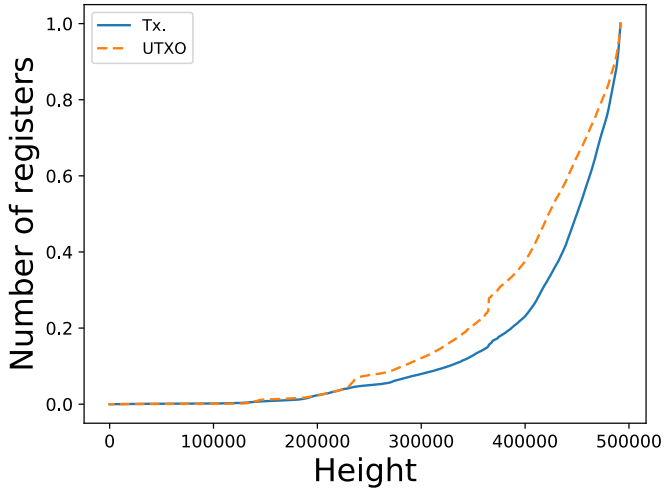

Figure 4.7: Height per transaction/UTXO.

4.2.4 Non-standard

As shown in Table 4.2, we have labelled as `others` 409,487 UTXOs from the set, since they do not fall into the main categories: P2PK, P2PKH and P2SH. A detailed analysis of such UTXOs, provided in Table 4.4, shows that almost all UTXOs correspond to a Pay-to-Multisig (P2MS) outputs being 1-2 and 1-3 set-ups the most popular ones. Notice that the UTXOs included are those with configuration up to three public keys, which is sound according to the fact that this is the upper bound for a multisignature output to be considered standard by the Bitcoin network transaction relaying policies. Finally, it is worth mentioning that there are 828 UTXOs with 1-1 configuration, a fact that does not make much sense since it is an output with a functionality equivalent to P2PK but with a larger script size and therefore higher fees may be needed to spent it.

Regarding the 1,169 outputs labelled as `others` in Table 4.4, 34.05% of them

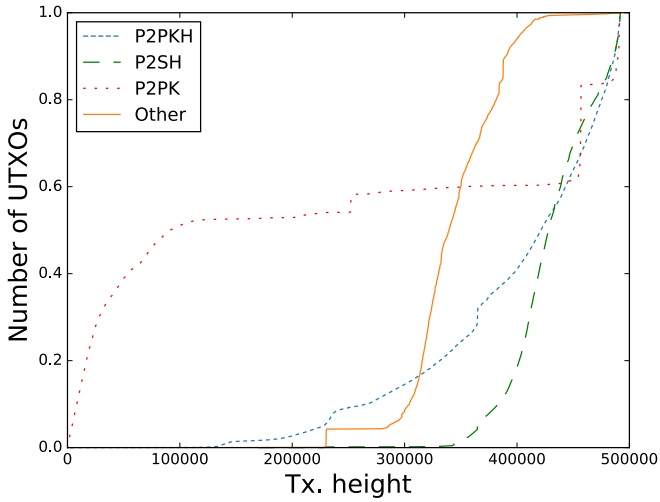

Figure 4.8: UTXO type evolution by height.

1-1	828	0.20%
1-2	199,904	48.81%
2-2	1,353	0.33%
1-3	206,096	50.33%
2-3	117	0.02%
3-3	20	0.005%
Others	1,169	0.28%

Table 4.4: Multisig analysis.

(398) are new native SegWit type outputs. More precisely, Pay-to-Witness-Public-Key-Hash (P2WPKH) account for 40 outputs and Pay-to-Witness-Script-Hash (P2WSH) accounts for a total of 358.

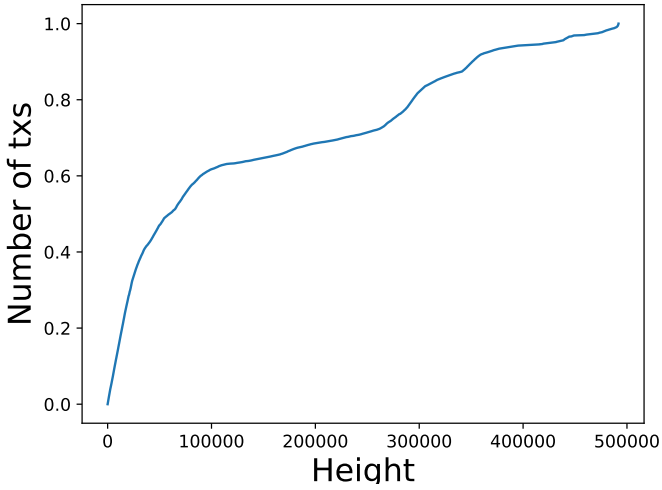

Figure 4.9: Coinbase evolution by height.

4.3 Conclusions

In this chapter we have introduced STATUS, a tool to analyse the UTXO set of Bitcoin (based on the Bitcoin Core implementation), and we have provided an analysis of such set, paying special attention to dust and unprofitable outputs. We have also provided a detailed description of the UTXO set format, including the new format introduced in Bitcoin Core v0.15. The use of this format compared to the previous one does not have an impact on the analysis we have presented in this chapter. The new version provides a more efficient access to the UTXO information at the expense of slightly higher storage requirements. Additionally, we have provided interesting data that shows the high percentage of `static information` (in the sense that is not going to be spent -dust and unprofitable-) included in the UTXO set that reduces the efficiency of the database in terms of space. Finally, it is worth mentioning that currently there is a very low percentage of SegWit UTXO, upper bounded by a 2.26% of the total outputs stored in the

UTXO set. As this will possibly increase in the future, the analysis of dust and unprofitable transactions will need to be revisited in further research in order to update the results with these new types of outputs.

CHAPTER 5

BITCOIN PRIVATE KEY LOCKED TRANSACTIONS

In this chapter we introduce and propose private key locked transactions (PKLT), a new type of Bitcoin transaction were the disclosure of a private key from an asymmetric key pair must be performed to spend a UTXO [36]. Note that standard scripts requiring a signature are already useful to prove the possession of a private key (because the private key is needed to create the signature). However, the disclosure of such private key will be useful as a building block of several applications, as we will see in later chapters.

5.1 Private key locked transactions

Two alternative but equivalent ways for implementing private key locked transactions will be introduced in this chapter. First, through the definition of a new opcode. Secondly, by using a well known ECDSA vulnerability. In our case, such vulnerability becomes a property of ECDSA, since it allows us to nicely implement the key disclosure through a Bitcoin transaction.

5.1.1 Designing a new Bitcoin opcode

Our first proposed design for PKLT is the implementation of a new crypto opcode that checks whether a public key and a private key belong to the same key pair: `OP_CHECKKEYPAIRVERIFY`.

With the use of this new opcode, a transaction output could be constructed so that, in order to be redeemed, the private key matching the specified public key has to be revealed. An example[1] of the `scriptPubKey` of such an output along with the `scriptSig` needed to spend it would be:

```
ScriptPubKey: <pubKeyA1> OP_CHECKKEYPAIRVERIFY OP_2DROP
              <pubKeyA2> OP_CHECKSIG
ScriptSig:    <sigA2> <privKeyA1>
```

The script will first check that the public and private keys belong to the same key pair. Note that, if the validation is successful, the stack values will remain untouched. Therefore, before checking the validity of the signature with `OP_CHECKSIG`, `privKeyA1` and `pubKeyA1` have to be removed from the stack (since they are not needed for signature validation). The execution of `OP_2DROP` removes them from the stack. Finally, `OP_CHECKSIG` validates the signature with the public key. If the signature is correct, the script terminates successfully.

Note that the execution of `OP_CHECKKEYPAIRVERIFY` would fail if the validation is unsuccessful and would leave the stack as it was before if the validation is successful. This ensures that the new opcode can be implemented as a soft fork modification of the Bitcoin Core protocol by reusing one of the currently unused `OP_NOPx` opcodes, in a similar way that it has been done in the past with the opcodes `OP_CHECKLOCKTIMEVERIFY` (`OP_NOP2`) and `OP_CHECKSEQUENCEVERIFY` (`OP_NOP3`).

[1] The provided script includes a digital signature condition, following the structure of the ones previously introduced in Chapter 2.2.2.

5.1.2 ECDSA vulnerability

An alternative approach to build transaction outputs, which locking conditions require to disclose a specific private key, can be taken by using a vulnerability in the ECDSA signature scheme.

ECDSA (Elliptic Curve Digital Signature Algorithm) is the crytographic algorithm used by Bitcoin to create and validate digital signatures. ECDSA has a set of system parameters: an elliptic curve field and equation `C`, a generator G of the elliptic curve `C`, and a prime q which corresponds to the order of G. The values for these parameters are defined to be `secp256k1` [91] for Bitcoin.

Let denote by $*$ the operation of multiplying an elliptic curve point by a scalar. Given a specific configuration of the parameters and a private key d, the ECDSA signature algorithm over the message m is defined as follows:

1. Randomly choose an integer k in $[1, q-1]$
2. $(x, y) = k * G$
3. $r = x \mod q$
4. $s = k^{-1}(m + rd) \mod q$
5. Output[2]: $sig(m) = (r, s)$

The ECDSA signature scheme is therefore probabilistic, that is, there exist many different valid signatures made with the same private key for the same message. The selection of a specific signature from the set of valid ones is determined by the election of the integer k.

There exists a well known ECDSA signature vulnerability (also present in the non-elliptic curve signature scheme of ElGamal and its popular variant, DSA [75]) by which an attacker that observes two signatures of different messages made with the same private key is able to extract the private key if the signer reuses the same k. Therefore, the selection of k is critical to the security of the system.

[2] A new integer k is chosen and the procedure is repeated if either s or r are 0.

Indeed, given two signatures that have been created using the same k and the same private key, $sig_1(m_1) = (r, s_1)$ and $sig_2(m_2) = (r, s_2)$ with $m_1 \neq m_2$, an attacker that obtains m_1, sig_1, m_2, sig_2 may derive the private key d:

1. Recall that, by the definition of the signature scheme:

$$s_1 = k^{-1}(m_1 + rd) \mod q \Rightarrow ks_1 = m_1 + rd \mod q$$
$$s_2 = k^{-1}(m_2 + rd) \mod q \Rightarrow ks_2 = m_2 + rd \mod q$$

   Note that, since r is deterministically generated from k and the fixed parameters of the scheme, the r of both signatures will be the same.

2. The attacker learns k by computing $k = \frac{m_2 - m_1}{s_2 - s_1}$

3. The attacker learns the private key d by computing $d = \frac{s_1 k - m_1}{r}$ or $d = \frac{s_2 k - m_2}{r}$

Moreover, the leakage of a private key can also be produced in situations where similar k values are generated [11, 83].

Some Bitcoin wallets adopted deterministic ECDSA after this vulnerability was found to affect some Bitcoin transactions [18, 94, 108].

It is woth mentioning that taking advantage of such vulnerability to disclose a private key in Bitcoin has been previously used for timestamping in data commitment schemes [29].

5.1.3 Private key disclose mechanism

Our proposed scheme makes use of the aforementioned ECDSA vulnerability to perform targeted private key disclosure within Bitcoin. The private key disclosure mechanism we propose allows the construction of transaction outputs that need to reveal a private key in order to be redeemed, in such a way that we ensure the revealed private key is the counterpart of a certain public key.

Let $\{PK, SK\}$ be an ECDSA key pair belonging to Bob (being $Addr(PK)$ the Bitcoin address associated to it) and sig_{prev} an existing signature made with

SK. Alice (that is interested in acquiring Bob's private key) needs to know the value of the previous signature sig_{prev} to be able to request, afterwards, a second signature made with the same k. The previous signature may appear in the blockchain as the input script of an existing transaction. For instance, if Bob has performed a transaction using any UTXO bound to $Addr(PK)$, therefore providing a signature performed with SK, sig_{prev} will be publicly available in the Bitcoin blockchain. Hence, any observer will know this value, and the signed message m will correspond to a transaction hash.

Once an existing previous signature sig_{prev} is known by Alice, she creates a transaction with an output that requires a second signature sig to be spent. However, instead of using the classical P2PKH script, she uses a special script that forces Bob (the redeemer) not only to prove he has the private key SK associated to the given address $Addr(PK)$ by creating a valid signature, but also to deliver a signature that has exactly the same k value that was used to create sig_{prev}. The output may also have a time lock that allows Alice to get back her bitcoins if Bob chooses not to reveal the private key.

Doing so accomplishes two purposes: first, Bob proves he knows the private key associated to the public key by generating a signature that correctly validates with that public key; secondly, Bob is implicitly revealing the private key. Note that Bob does not directly provide the private key, but provides information from which the private key can be derived.

Moreover, the operation is atomic, in the sense that Bob gets Alice's bitcoins (the amount deposited into the output) only when Alice gets Bob's private key (derived from the two signatures by exploiting the reuse of k).

Furthermore, unlike when revealing symmetric keys with hash locks, the private key disclosure mechanism allows to validate that the leaked secret key is correct, that is, it matches the specified public key.

Figure 5.1 shows an scheme of the Bitcoin transactions involved in the construction of a private key locked output. In this example, the input of transaction tx_2 contains the signature sig_{prev} made by Bob in the past.

Once the previous signature is known, Alice can construct the transaction tx_4,

that transfers some bitcoins of her property to Bob, only if Bob provides a valid signature that has the same r as the previous signature sig_{prev} that appeared on tx_2. Moreover, the output has an additional condition with a time lock allowing Alice to get a refund of her bitcoins if Bob decides not to collaborate and does not redeem tx_4's output.

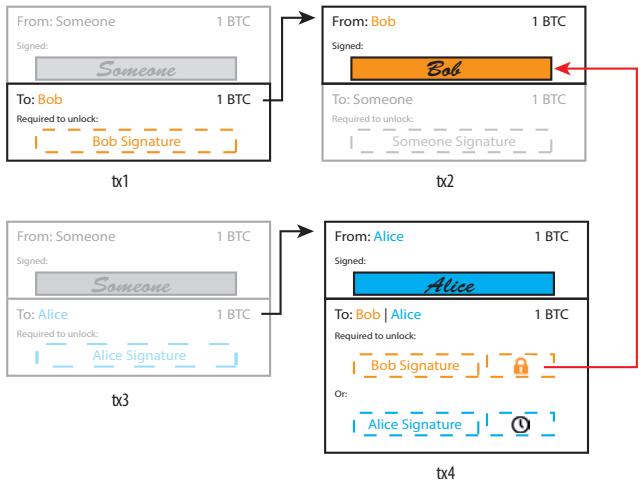

Figure 5.1: Transactions involved in the scheme.

In the next section, we describe how to construct the output of tx_4 taking into account Bitcoin's signature format and Bitcoin's scripting language.

5.1.4 Implementation

Although ECDSA signatures are made of two values, r and s, Bitcoin signatures are just a single hexadecimal value, which corresponds to the DER encoding of the two-element sequence of the two integers. Figure 5.2 describes the format of a Bitcoin signature, where z denotes the bytes representing r and s (that is, for each value, the 0x02 integer flag, the size, and the value itself) and ht denotes the hash type, a flag that indicates the parts of the transaction that are hashed

and signed.

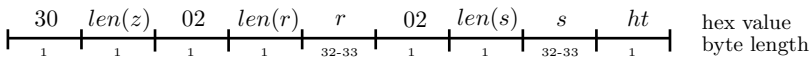

Figure 5.2: Bitcoin signature format.

Both r and s are 32-byte integers. However, when the first bit of any of the values is set (that is, the fist byte is > `0x7f`), an additional byte (`0x00`) is added in front of the value, thus making it 33 byte long. DER rules interpret this first bit as a sign, and therefore not adding `0x00` would cause the value to be interpreted as negative.

Recall that Alice was in possession of a previous signature sig_{prev} that Bob had made in the past, and that Alice wanted to construct a transaction output that can be redeemed by Bob only if he presents a new signature sig that uses the same k (i.e. has the same r component).

For the sake of simplicity, let us assume that we are dealing with 71-byte signatures, i.e., signatures where both r and s are 32 byte long (Figure 6.4a). Taking into account the format of signatures in Bitcoin, Alice can construct a signature mask sig_{mask}: a byte array that has ones on the positions where r is specified and zeros in the rest of positions. Figure 5.3b shows the construction of sig_{mask}. Alice can also construct a byte array r_{prev} (Figure 5.3c) that results from the bitwise AND operation between the previous signature sig_{prev} and the signature mask sig_{mask}.

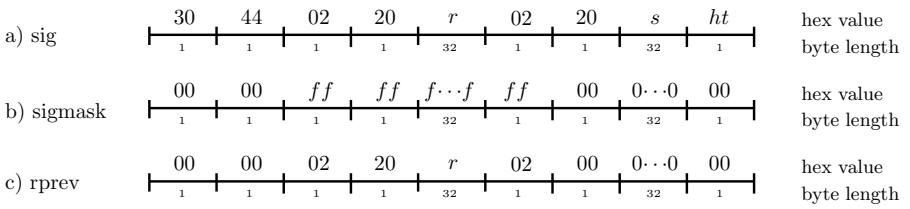

Figure 5.3: Values used in the proposed script.

Finally, Alice can create an output that requires a second signature sig with the same r as the previous signature sig_{prev} by using the values sig_{mask} and r_{prev} she has computed. The `ScriptPubKey` of the output (and its corresponding `ScriptSig`) would then be:

```
ScriptPubKey: OP_DUP <pubKey> OP_CHECKSIGVERIFY
              OP_SIZE <0x47> OP_EQUALVERIFY
              <sigmask> OP_AND <rprev> OP_EQUAL
ScriptSig:    <sig>
```

First, the script validates the signature against the specified public key. Then, the length of the signature is checked. Finally, a bitwise AND between the new signature and sig_{mask} is computed, and the result is compared with r_{prev}. If both values are equal (that is, both signatures have the same r and thus were made using the same k), the script terminates successfully; otherwise, the script terminates with a `false` value on the stack, making it fail.

Note that the only way to ensure that the script succeeds is by providing a valid signature that has exactly the same r as the previous signature. Therefore, although the redeem `ScriptSig` that spends the output does not include the private key directly, it is implicitly leaking its value. Also note that the `ScriptSig` needed to spend the output only requires one value: the new signature.

We have created a set of transactions in the Bitcoin testnet that exemplify the proposed protocol. Following the transaction naming used in Figure 5.1, the input[3] of tx_2 contains a signature made by Bob with his private key SK, together with the public key PK needed to validate it. This signature is public, and thus can be used as the previous signature sig_{prev} needed by our scheme. The private key SK is, therefore, the private key that is going to be disclosed by Bob by providing a second signature with the same k. With this previous signature, Alice can construct tx_4, whose output[4] contains the special script that requires a second signature from Bob with the same private key and the same

[3] https://www.blocktrail.com/tBTC/tx/7767a9eb2c8adda3ffce86c06689007a903b6f7e78dbc049ef0dbaf9eeebe075

[4] https://www.blocktrail.com/tBTC/tx/cb0174d950761ab1b28cff7bb6d0da63414305540e5f52cd71dbca213a5a910d

k. Unfortunately, the output of tx_4 is currently unspendable because it uses an `OP_AND` opcode that is disabled in current Bitcoin standard implementations.

Moreover, note that the transaction tx_5 that spends tx_4 may be vulnerable to double-spending. Indeed, once tx_5 is made public, any observer will be able to derive the private key from the information it contains. As a consequence, an attacker would be able to construct another transaction, tx'_5, that also spends tx_4 (and also reuses k) but that moves the amount of bitcoins to an address controlled by the attacker. In order for the attack to succeed, the attacker not only has to create and broadcast tx'_5, but also has to be able to make the network accept his transaction tx'_5 instead of the original tx_5.

This attack can be prevented by modifying the proposed script, so that tx_4 output contains a 2-out-of-2 multisignature script. Then, two signatures are needed from Bob to spend tx_4. One of the signatures will be required to have the same k than sig_{prev} (as in the original formulation of our protocol), whereas the second signature will not have any special conditions. This second signature will be validated against a second public key from Bob, whose corresponding private key will not be leaked by the protocol. In this way, an attacker capturing tx_5 will not be able to create tx'_5 double spending tx_4, since he will lack one of the private keys.

```
ScriptPubKey: OP_DUP OP_TOALTSTACK
              OP_2 <pubKey2> <pubKey> OP_2 OP_CHECKMULTISIGVERIFY
              OP_FROMALTSTACK
              OP_SIZE <0x47> OP_EQUALVERIFY
              <sigmask> OP_AND <rprev> OP_EQUAL
ScriptSig:    OP_0 <sig2> <sig>
```

An output containing such an script can be also found in the testnet[5].

[5] https://www.blocktrail.com/tBTC/tx/e27b236bb124f5eefd66aa7bcd97aabb1e4fe5c1dee509a1fbe50339400f6260

5.2 Conclusions

We have presented private key locked transaction for Bitcoin, a new transaction type where the redeemer is required to disclose a given private key corresponding to a ECDSA asymmetric key pair. This has been achieved by using an existing vulnerability of the ECDSA signature scheme. The private key can be obtained from two different signatures of the same key re-using a given parameter.

The exchange of a key for a payment in an atomic way provides a nice tool for interesting digital exchanges, where sold information is protected by a key until the payment is received. However, in such scenarios, the correction of the key should be verifiable before the payment. Our construction allows such verification since a private key is related to a publicly known public key, and then a correctness verification can be performed.

CHAPTER 6

DOUBLE-SPENDING PREVENTION FOR BITCOIN ZERO-CONFIRMATION TRANSACTIONS

In this chapter we describe a proposal to mitigate the double spending problem for Bitcoin zero-confirmation transactions [87], building on top of private key locked transaction introduced in the previous chapter. In our proposal, any single observer who identifies a double spending attempt may take part and punish the attacker. Moreover, our solution discourages the attacker to even attempt the double-spending, because doing so makes him risk losing an amount of bitcoins bigger than the double-spent amount. Our solution benefits fast-payment scenarios, like in-shop purchases or trading platforms, where the transfer bitcoin-product/service cannot wait until the transaction is confirmed in the blockchain.

6.1 Bitcoin transactions propagation

As was introduced in Chapters 2 and 3, Bitcoin transactions are propagated through a peer to peer network. Every node of the network broadcasts the transactions he generates and propagates transactions received from other network peers. In order to defeat denial of service attacks, every peer performs different validations on every received transaction before its propagation[1], like data format validation, digital signature verification, or correctness of the values involved in the transaction. Besides such verification, the node also validates that the received transaction does not spend output previously spent, neither by a transaction already in the blockchain nor by a transaction included in the mempool of the node. In case that some validation fails, the node drops the transaction and, therefore, the transaction is not propagated any further.

However, since Bitcoin Core 0.12, Bitcoin includes a replace-by-fee mechanism (RBF) that allows transactions to signal replaceability. Such mechanism was introduced to allow to increase the fee of an already broadcast transaction in order to boost its odds of inclusion in the blockchain. Therefore, if a transaction is flagged as replaceable with RBF, it can be replaced from the node's mempool by a newer transaction that spends the same outputs but includes a higher fee[2]. Moreover, such new transaction will also be further propagated throughout the network, since it is considered a valid transaction that replaces the previous one. Such feature ensures that when a transaction is tagged as a RBF, a double spending transaction of the same UTXO will be propagated further than a double spending of a regular (non-RBF) transaction.

6.2 Double-spending prevention mechanism

Our proposed scheme discourages users from performing double spending attacks in zero-confirmation transactions used in fast payment scenarios. Fast payment

[1] See [3] for all the validation details.

[2] Notice that this only affects transactions in the mempool, since transactions included in the blockchain are final and thus not replaceable.

scenarios are those where the merchant delivers the goods or services when seeing the payment transaction in the Bitcoin network, without waiting for the transaction to be confirmed. Examples of such scenarios are on-site shopping where the buyer cannot wait 10 minutes to leave the shop after purchase or in trading platforms where a timely transaction can save/earn you money.

In our scenario, we assume that the adversary is the buyer that pays for some goods to a merchant, and that may have incentives to try to double-spend the payment in order to finish the interaction with both the goods and the money.

We assume that the adversary can perform a double-spending attack by generating multiple transactions that spend from the same output and broadcast them selectively in the Bitcoin P2P network. Additionally, we also assume all peers of the network have the same capabilities, that is, they are able to generate and broadcast double spend transactions (if they know the private key needed to generate a signature).

In order to discourage double-spending attacks, we propose a mechanism to construct special transaction outputs. Such outputs can be spent with a single signature but have the property that if two different signatures for the same output are disclosed (for instance, in two different transactions spending the same output as a double-spending attack), the private key used to sign the transaction is revealed. This allows any observer to generate a third transaction spending the same output and sending the amount to an address controlled by himself.

To allow such construction, we propose a new Bitcoin script that we call **fixed-r pay-to-pubkey script (FR-P2PK)** which builds on top of the private key locked transactions introduced in Chapter 5. Indeed, if the sender generates another transaction that spends the same output and propagates it through the P2P network, the sender risks losing all the funds deposited in the address because any peer that captures both transactions will be able to derive the private key.

6.2.1 Basic prevention mechanism

Let Alice be a user that wants to take advantage of the proposed double-spending prevention mechanism and let $\{PK_a, SK_a\}$ be an ECDSA key pair belonging to Alice.

The double spending prevention mechanism is made of two phases: *initialization*, that is performed before the payment is made, and *fast-payment*, where the payment is executed.

6.2.1.1 Initialization

The initialization phase is performed beforehand. During this phase, Alice generates a *funding transaction* that transfers some funds from an output in her control to a FR-P2PK output also under her control. In order to do so, Alice chooses a random integer k and a public key PK_a (for which she knows the associated secret key SK_a), constructs the FR-P2PK output, and sends some funds to an output in her control (see Figure 6.1).

Alice broadcasts the funding transaction and waits for the transaction to be confirmed, upon which the initialization phase is considered terminated.

A single funding transaction may include multiple FR-P2PK outputs (with different public keys) in order to allow Alice to use the proposed prevention mechanism multiple times. Moreover, Alice may repeat the initialization phase if she runs out of unspent FR-P2PK outputs. Note that this phase is independent of any specific payments and that Alice alone participates in the procedure. Additionally, notice that Alice remains in control of all the funds deposited by the funding transaction and she is able to transfer them back to an standard output whenever she wants.

At some point in the future, Alice wants to send some amount of bitcoins to another user Bob. Alice does not want to wait for the confirmation of the transaction and Bob is not willing to accept the transaction without confirmation.

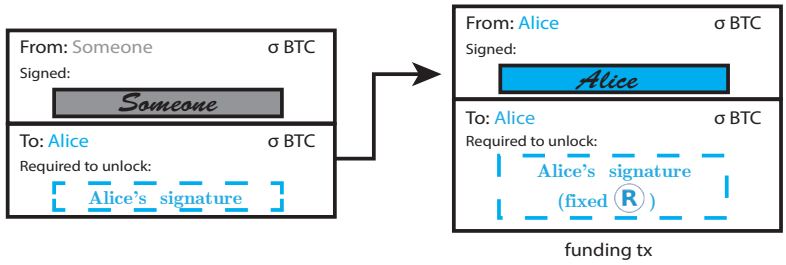

Figure 6.1: Creation of the funding transaction.

So they decide to use the proposed double spending prevention mechanism, executing the fast-payment phase.

6.2.1.2 Fast payment

Alice creates a *fast-payment transaction* that pays to Bob spending from the FR-P2PK output of the funding transaction. The input script in the fast-payment transaction forces Alice (the redeemer) not only to prove that she has the private key SK_a associated to the given public key PK_a by creating a valid signature, but also to deliver a signature that has been made using the specific k value that Alice chose during the initialization phase (see Figure 6.2). Alice broadcasts the fast-payment transaction to the Bitcoin P2P network.

Then, when Bob sees the fast-payment transaction in his mempool, he can validate that the output script of the funding transaction spent by the fast payment transaction is indeed a FR-P2PK script. If the validation is correct, Bob knows that if Alice tries to double spend the transaction she takes the risk of losing the Bitcoins of that output.

If Alice decides to try to double spend the fast-payment transaction (see

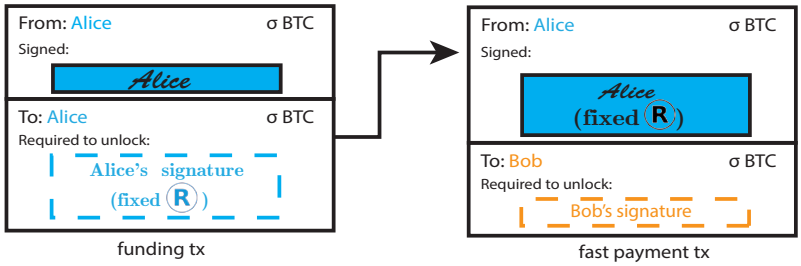

Figure 6.2: Fast payment transaction.

Figure 6.3, double-spending attempt), she needs to create a *double spending transaction* that also spends the FR-P2PK output of the funding transaction. This double spending transaction has to be valid, so it needs to include a (second) signature made with SK_a and the k value chosen on the initialization phase. Hence the moment the double spend transaction is created, there exist two different signatures made with the same private key SK_a using the same r. The signatures will be indeed different, since the signed content (i.e. the transactions) will also be different.

As a consequence, if Alice broadcasts the double-spending transaction, she risks losing her funds. This happens because any observer that receives both transactions (the fast-payment transaction and the double-spending transaction) will be able to derive Alice's secret key SK_a and, as a consequence, create a third transaction (the *penalty transaction*) that also spends the FR-P2PK output of the funding transaction but that sends the bitcoins to the observer. Note that this strategy may be performed simultaneously by any observer, ending with multiple penalty transactions, as it is depicted in Figure 6.3.

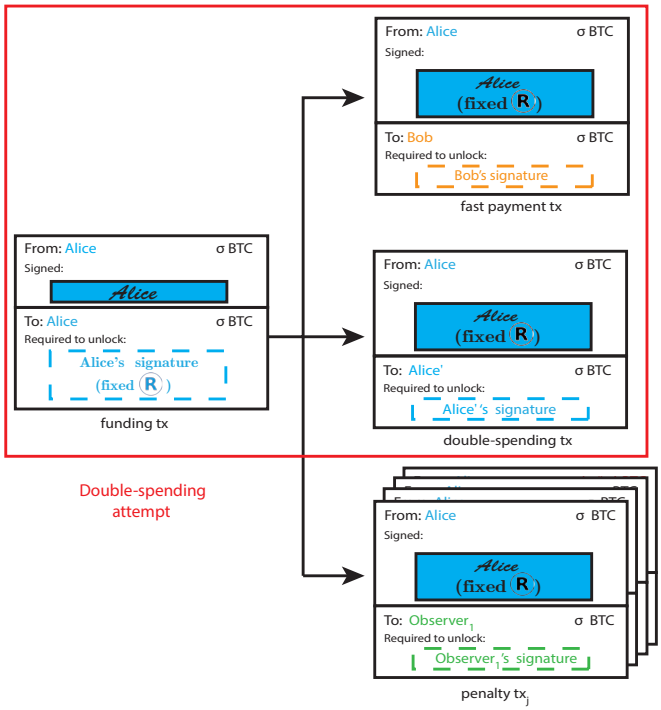

Figure 6.3: Transactions involved in the scheme.

6.2.2 Disincentive-based prevention mechanism

The basic prevention method described in the previous section has a clear drawback. Suppose that Alice buys something from Bob's shop and Bob accepts a fast transaction from Alice as a payment. When such transaction is received, Bob delivers the goods to Alice. However, once Alice has the goods, she may try to perform a double-spending attack. In case an observer sees both the fast-payment transaction and the double-spending transaction, he constructs the penalty transaction, and manages to get it accepted in the blockchain, Alice will therefore lose her funds but Bob will not receive the payment. In that case, Bob may have complied with his part of the agreement (e.g. delivered the goods) but will not

receive the agreed amount of bitcoins in exchange. Alice would have paid the agreed amount of bitcoins to a third party (the observer) instead of paying them to Bob, but she would remain in possession of the goods. The observer would obtain the total amount of the transaction. As a consequence, Alice will not lose anything by trying the double-spend (just the amount she was already willing to pay for it), and thus the proposed method may not be discouraging enough to prevent double spending attempts.

However, a minor modification to the method is enough to discourage Alice from attempting any double-spend: enforcing that the amount deposited to the FR-P2PK output of the funding transaction is higher than the paid amount by a certain factor λ. Recall that a Bitcoin UTXO must be spent in its totality (i.e. it is not possible to spend a part of a UTXO). Therefore, if the FR-P2PK output has an amount bigger than what Alice must pay to Bob, Alice proceeds to create the fast-payment transaction including two different outputs: one that pays to Bob the agreed amount, and the other that pays back to her the change. This has no consequences on the normal operation of the protocol, that is, if Alice is honest, Bob ends up with his payment and Alice gets her change back. However, because the entire FR-P2PK output is spent, if Alice tries to double-spend the fast payment she risks losing not only the amount paid to Bob, but the entire amount of the FR-P2PK output. It is worth mentioning that if Alice has more UTXOs associated with the Bitcoin address used in the double spend attempt, she risks losing them all since her private key will be compromised. Therefore, the penalty could be even higher. However, the technique aims to ensure that at least the agreed amount is hold by Alice.

As we will discuss in Section 6.4, by adjusting the λ factor Alice's penalty for double-spending can also be adjusted (and thus Bob's confidence on the fast payment).

Finally, note that the fast-payment transaction may also have multiple inputs spending different FR-P2PK outputs. This allows Alice to perform payments of different amounts and with different penalty levels without having to freeze a high amount of bitcoins into FR-P2PK outputs.

6.2.2.1 The role of the observers

The funding transaction is confirmed before starting the fast-payment phase, so any full node of the network is aware of its existence. Moreover, because it has an output with an easily identifiable script, the FR-P2PK script, any observer aware of the specification of our proposed mechanism is able to identify the transaction as a funding transaction belonging to our protocol. Therefore, such an observer will be able to monitor his mempool, looking for transactions that spend the FR-P2PK output. Once a transaction spending from the FR-P2PK output is seen, the observer is able to actively listen the network, searching for any other transaction spending the same output. If the observer is able to catch a double spending transaction, he should be able to construct a penalty transaction, moving the funds to an address controlled by himself. If the observer does not capture a double spending transaction, he may stop this active listening period and return to its normal behaviour when a transaction spending the FR-P2PK output is included in the blockchain.

In order to achieve the maximum level of propagation, and thus to spread awareness of the double-spending attempt, *fast-payment transactions* are flagged with replace-by-fee (RFB). By forcing a customer to flag a transaction in such a way a merchant enables the forwarding of future *double-spending transactions*, that could be detected by a higher number of nodes, increasing the odds of an observer receiving two instances of the transaction and, therefore, publishing a *punishment transaction*.

6.3 Implementation details

In this section, we describe how to construct the FR-P2PK output of the funding transaction as well as the inputs of the transactions that spend it, taking into account Bitcoin's signature format and scripting language.

First of all, notice that it would be possible to encapsulate the proposed FR-P2PK script into a standard P2SH output. However, doing so makes the funding transaction no longer recognizable as belonging to our protocol by external

observers. Therefore, an observer that is aware of the existence of our protocol would be able to detect that the mechanism is being used only after one of the transactions spending the encapsulated FR-P2PK is seen in the network. This transaction will include the FR-P2PK script in the `scriptSig` (input script). The moment the observer processes this script, he can start the active listening period in which he looks for other transactions spending from the same funding transaction output. Because timing is critical in our scenario, we argue that using directly a FR-P2PK output in the funding transaction is the best alternative.

Figures 6.4b-c show the construction of `<sigmask>` and `<r>`, respectively. Regarding the construction of the byte array `<r>`, on the one hand the integer r is derived uniquely from the randomly chosen k value (recall Step 3 of the ECDSA signature generation algorithm in Chapter 5.1.2). Note that any value of k may be used by the protocol, what matters is that it is fixed beforehand (that is, before the signature is made). In contrast to what we have already seen in previous chapters, the hash flag tag ht is set to `0x01`, which corresponds to `SIGHASH_ALL`. This flag signals that the signed content corresponds to the entire transaction (except the signature scripts themselves). By enforcing that signatures cover the entire transaction, we ensure that a double spending attempt will include a signature different from the one found in the fast-payment transaction, and thus that observing both transactions indeed allows to derive the private key.

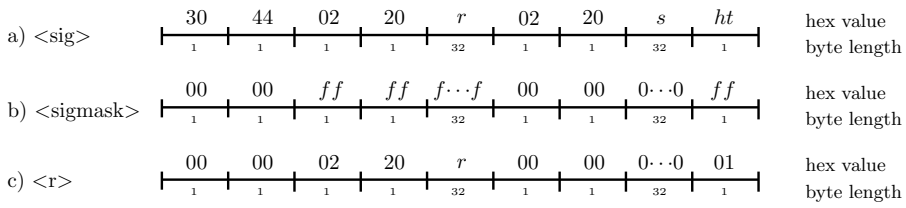

Figure 6.4: Values used in the proposed script.

6.4 Proposal analysis

In this section we provide an analysis of the possible outcomes of performing a payment with the proposed mechanism. The analysis measures the benefits of each party taking part in the system to show how it discourages double-spending attacks. Table 6.1 summarizes the notation used in this section.

Symbol	Meaning
τ_f	Fast-payment transaction
τ_d	Double-spending transaction
τ_{p_j}	Penalty transaction j
$Pr[\tau_x \in \mathscr{B}]$	Probability that transaction τ_x is included in the blockchain
σ	Payment amount
$\lambda \cdot \sigma$	Funding transaction output amount
γ	Value of goods

Table 6.1: Notation summary.

Our analysis makes the following assumptions. First of all, we assume that Alice always generates the fast payment transaction since it is the triggering action for the payment. Once the fast payment has been generated, we assume that Bob sees the payment and, at that time and acting honestly, he delivers the goods to Alice. Furthermore, to focus the analysis on the proposed mechanism, we assume that at least one of the transactions of the system τ_f, τ_d, or τ_{p_j} will be confirmed. Although transactions do include fees, these are intentionally excluded from the computation of the payoffs since they do not directly affect the result of our evaluation. The use of the replace-by-fee mechanism, that will allow the propagation of multiple instances of the transaction, forces double-spending transactions to include a higher fee than the original transaction. However, it could be argued that a higher fee will incentive miners to mine a double-spending transaction instead of a fast-payment transaction, affecting not only the payoff but also the odds of either succeed or fail in the attack. Nonetheless, if a miner

sees two instances of the transaction (fast-payment and double-spending) he would maximize his payoff as observer by generating a third transaction (penalty) that pays all the amount to himself (instead of choosing to include any of the previous ones depending on the fees). Moreover, notice that our analysis can be seen as an upper-bound on the attacker's gains since including fees in the computation only lowers the attacker's payoff, but never increases it.

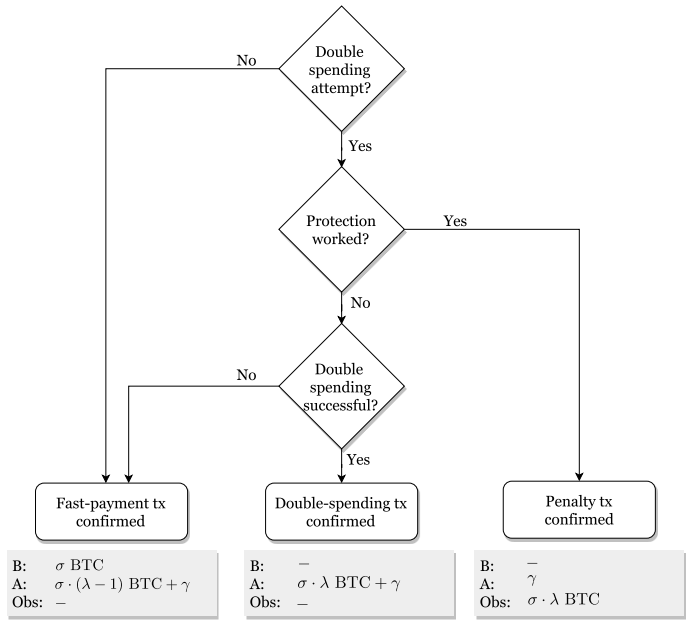

Figure 6.5: Flow chart showing the protocol's final states and the paths leading to them.

Of course, due to the double-spending protection of Bitcoin for on-chain transactions, at most one of these transactions gets into the blockchain, that is, the events $\tau_f \in \mathscr{B}$, $\tau_d \in \mathscr{B}$, and $\tau_{p_j} \in \mathscr{B}$ are mutually exclusive. Finally, notice that $Pr[\tau_f \in \mathscr{B}] + Pr[\tau_d \in \mathscr{B}] + \sum_j Pr[\tau_{p_j} \in \mathscr{B}] = 1$, since such probabilities depend on the distribution hash rate devoted to the interests of every set of users (Alice, Bob and the rest of the network, acting as observers) and we can assume that

such sets will be disjunct.

Taking into account these assumptions, the protocol we propose may end in three different states, as described by Figure 6.5. If the fast payment transaction τ_f gets confirmed, then Bob receives the payment for the goods, Alice receives the change (the amount deposited to the funding transaction minus the payment) and the goods, and the observer does not intervene. If the double spending transaction τ_d is confirmed, then Alice gets everything (the whole amount deposited in the funding transaction and the goods) and therefore, both Bob and the observers do not obtain anything. Finally, if one of the penalty transactions τ_{p_j} is confirmed, then Alice obtains the goods but loses the whole deposited amount that goes to the observer. Figure 6.5 also describes the possible paths that end up in each of the states.

We define the payoff $\mathscr{P}$ of any party participating in the protocol as the gains (or losses) obtained by deviating from the correct operation of the protocol. That is, the payoff of all parties (Alice, Bob, and the observers) will be 0 when no double spending is attempted (leftmost box in Figure 6.5)[3]. In that case, there will be an equilibrium, since Alice pays the specified price for some goods and obtains the goods in exchange; Bob delivers the goods and gets paid for them; and the observers do not intervene. On the contrary, if Alice tries to double spend the payment, the equilibrium may be altered and the payoff will reflect the gains or losses each party assumes.

Then, Bob's payoff function $\mathscr{P}_B$ is given by the following expression:

$$\begin{aligned}\mathscr{P}_B = Pr[\tau_f \in \mathscr{B}] \cdot (\sigma - \gamma) - \\ - Pr[\tau_d \in \mathscr{B}] \cdot \gamma - \\ - Pr[\tau_{p_j} \in \mathscr{B}] \cdot \gamma = \\ = Pr[\tau_f \in \mathscr{B}] \cdot \sigma - \gamma\end{aligned}$$

Note that, for fixed σ and γ, Bob's payoff only depends on $Pr[\tau_f \in \mathscr{B}]$. Recall that our mechanism tries to disincentivize Alice from double-spending the pay-

[3]Here we assume that the price of the goods is equal to the value of the goods. If the price paid is higher than the cost, B's payoff is positive and reflects the benefit obtained from the sale.

ment transaction, but does not directly benefit the merchant (regardless of the λ value used by the protocol)[4].

In a similar way, Alice's payoff $\mathscr{P}_A$ is given by:

$$\begin{aligned}\mathscr{P}_A = & Pr[\tau_f \in \mathscr{B}] \cdot (\gamma - \sigma) + \\ & + Pr[\tau_d \in \mathscr{B}] \cdot (\gamma) + \\ & + Pr[\tau_{p_j} \in \mathscr{B}] \cdot (\gamma - \sigma\lambda)\end{aligned}$$

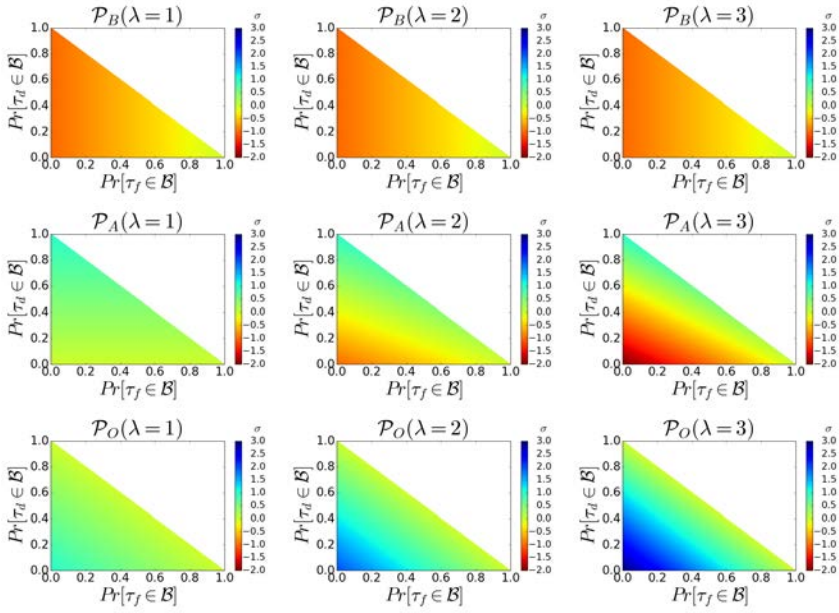

Figure 6.6: Parties payoffs for $\sigma = \gamma$.

Alice's maximum payoff is, therefore, γ, and is obtained when Alice's successfully double spends the transaction, thus keeping the goods γ without paying anything. However, Alice's minimum payoff (that is, maximum losses) depends

[4]Note, however, that Bob may also act as an observer himself, being able to create a penalty transaction and trying to gain the observer's payoff.

on λ, a parameter that can be adjusted in our protocol. Therefore, by adjusting λ, the protocol allows to tune Alice's losses, and so the risks she assumes by trying to perform a double spending attack. The bigger λ, the higher the risks Alice's faces on a double-spending attempt.

Finally, an observer's j payoff is given by the expression:

$$\mathscr{P}_{O_j} = Pr[\tau_{p_j} \in \mathscr{B}] \cdot (\sigma\lambda)$$

Figure 6.6 shows the evolution of the parties payoffs as a function of $Pr[\tau_f \in \mathscr{B}]$ and $Pr[\tau_d \in \mathscr{B}]$[5] for the case where $\sigma = \gamma$ (the value of goods is equal to the price it is paid for them), for different values of the parameter λ. The payoff dimension is measured based on the value σ. That means that a payoff of 3 implies a benefit of 3 times the value of σ while a payoff of -3 implies a lost of 3 times the value of σ. The payoff results are thus proportional to σ.

The graphics show that, as expected, when there is no double spending attempt ($Pr[\tau_f \in \mathscr{B}] = 1$) there is an equilibrium in the parties' payoffs, and in all graphics we obtain $\mathscr{P}_B = \mathscr{P}_A = \mathscr{P}_{O_j} = 0$ (green zone). Note that, as λ increases, Bob's payoff (first three graphics) remains exactly the same since his payoff is independent of the parameter λ. On the contrary, Alice's payoff (next three graphics) depends on λ. With $\lambda = 1$, Alice's payoff is always positive or zero: Alice does not lose anything by trying to double spend and may even gain something if the attack is successful. That situation is the basic prevention mechanism described in Section 6.2.1. However, by increasing λ the scenario changes radically for Alice: the probabilities range at which Alice gains something from the attack decrease fast and, at the same time, for some probability values she even starts to get a negative payoff (that is, she has to assume losses). Finally, notice that the observer's payoff (last three graphics) is never negative, and his gains increase with λ.

Bear in mind that our analysis does not assume any specific values on the probabilities $Pr[\tau_f \in \mathscr{B}]$, $Pr[\tau_d \in \mathscr{B}]$, and $Pr[\tau_{p_j} \in \mathscr{B}]$. However, as we have

[5] Since $Pr[\tau_f \in \mathscr{B}] + Pr[\tau_d \in \mathscr{B}] + \sum_j Pr[\tau_{p_j} \in \mathscr{B}] = 1$, fixing the first two probabilities uniquely determines the third operand.

already indicated, such probabilities depend on the hash distribution of the Bitcoin network among mining the transactions, τ_f, τ_d, and τ_{p_j}. For that reason, in case the hash rate devoted to τ_d with respect the rest is low, the graphics show that Alice's payoff, for values $\lambda > 1$, is moving in the red zone thus being negative (Alice is losing money). The greater the λ value, the bigger the red zone.

6.5 Conclusions

The speed at which payments in blockchain based cryptocurrencies can be performed is lower bounded by the block generation interval, which in Bitcoin is fixed to 10 minutes. In order to provide fast payments, one of the alternatives used in these scenarios is to rely on zero-confirmation transactions. Experimental analysis have shown that, in Bitcoin, most of the transactions propagated through the network reach 75% of the nodes in less than 8 seconds [34], which is two orders of magnitude faster than the block production interval. However, zero-confirmation transactions are not secured by the standard Bitcoin double-spending protection mechanism, since this mechanism is applied to transactions included in the blockchain and zero-confirmation transactions are not yet in blocks by definition.

In this chapter, we have presented a mechanism to secure fast payments within Bitcoin by reducing the risk of double-spending attacks in transactions with zero confirmations. The proposed mechanism discourages double spending attempts by creating a special type of outputs that enforce private key disclosure in case of double-spending attempt. Any Bitcoin network user may act as an observer and obtain a reward by detecting double-spending attempts. The reward the observer receives is equal to the price the attacker pays as punishment for having tried to double spend a transaction.

CHAPTER 7

A FAIR PROTOCOL FOR DATA TRADING BASED ON BITCOIN TRANSACTIONS

One of the major differences between Bitcoin and traditional payment systems, such as Visa or PayPal, is irreversibility. Bitcoin, in contrast to traditional payments systems, does not have any central authority who can reverse a payment once it has been issued. Therefore, once a transaction has been performed, no one but the person receiving the funds can reverse it. If a transaction is performed in exchange of goods or services and such are not finally provided, there is little the payer can do but undermine the reputation of the merchant/service provider. In that sense, some approaches have been presented to exchange digital goods for bitcoins, like hash locked transactions introduced in Chapter 2. In this Chapter, we will show how to build a fair exchange protocol for data trading, based on private key locked transactions, as a solution for the aforementioned problem [38]. Our protocol is fair since none of the participants have an advantageous position in the execution of the protocol. The protocol is also atomic, in the sense that it is either fully executed, ending the buyer with

the data and the seller with the payment, or no party incurs in any loss.

7.1 Fair exchange protocols

Fair exchange protocols can be used as a way to sign a contract between two parties, stating the conditions under which the exchange has to be carried. The signature is produced so that no party can gain advantage over the other.

Fair exchange protocols are usually divided into two party protocols and protocols requiring a trusted third party (TTP). Two party protocols provide a gradual exchange of messages or information between the two parties, to gradually decrease uncertainty and increase fairness in the transaction without the need for a TTP. First proposed in [21], the idea is for the two parties to exchange secrets bit by bit allowing them to verify the correctness of the received bits. This idea was also proposed in other approaches [44], more specifically for the signature of contracts. Probabilistic protocols for fair exchange were introduced in [13], where the goal is for the parties to end up with a given probability on the fairness (commitment to the contract by the two parties) at a given time (or step).

Regarding the use of a TTP, we usually distinguish between online and offline TTP. The online TTP acts as an intermediary between the two parties ensuring the fairness of the exchange [48, 116]. On the other hand, an offline TTP only acts in case of dispute and does not participate in the protocol if all parts act honestly, also called an *optimistic* fair exchange [5, 8, 117].

Authors refer to the notion of perfect fairness (also called strong fair-exchange) when a party cannot leave the protocol with a small advantage over the opponent [90]. Perfect fair exchange usually requires the use of TTP-based protocols, although there are several alternatives to implement it. Some of them relay in some penalty mechanism to be applied to the misbehaving parties [107]. However, it is important to note that from a practical perspective, this advantage could be small enough in order to be tolerated by both parties.

In [61] Bitcoin is used for the payment in an optimistic fair exchange (with a

TTP) with anonymity. Most notably, Bitcoin has been proposed for fair exchange as a mean of implementing a penalty mechanism [14]. The idea is that if a party leaves the protocol with more knowledge than the rest, those honest parties are compensated. The same idea is applied to multiparty computation in [2].

Penalty schemes to deal with misbehaving parties within Bitcoin have been also proposed in [93] as a generic penalty for parties issuing contradictory statements.

Zero knowledge proofs are used in [17, 73] as a way of excahnging data for a given secret. In this case the secret is a symmetric key used to encrypt some given data. A zero knowledge proof is used (externally to Bitcoin) to prove the validity of the encrypted data and the secret key, thus providing some sort of strong fair exchange. This is only feasible if a zero knowledge proof can be built to check the correctness of the data. A more generic solution is outlined in [104], where a symmetric key is used to encrypt chunks of data so that a subset can be revealed as a proof. As different keys are used for each chunk, revealing a subset does not ensure that the key of the other chunks is correct.

7.2 Private key locked transactions

Private key locked transactions (PKLT), introduced in Chapter 5, are a special case of Bitcoin transactions in which the transaction output can be redeemed by anyone who provides a private key corresponding to a public predefined key. In this chapter, we will use PKLT as a building block to perform atomic exchange between digital goods and a private key. From the two approached presented in Chapter 5, we will use the ECDSA vulnerability one with a slight modification.

7.2.1 ECDSA vulnerability

In contrast with the approach followed in the previous chapter, where the previous signature appeared in the blockchain as the input script of an existing transaction, in this chapter the signature sig_{prev} will be sent to Alice by an off-chain exchange of values. sig_{prev} may be transmitted confidentially (and thus only Alice and Bob

know its value). Following this approach, the signed message m does not need to correspond to a Bitcoin transaction hash (check Figure 7.1).

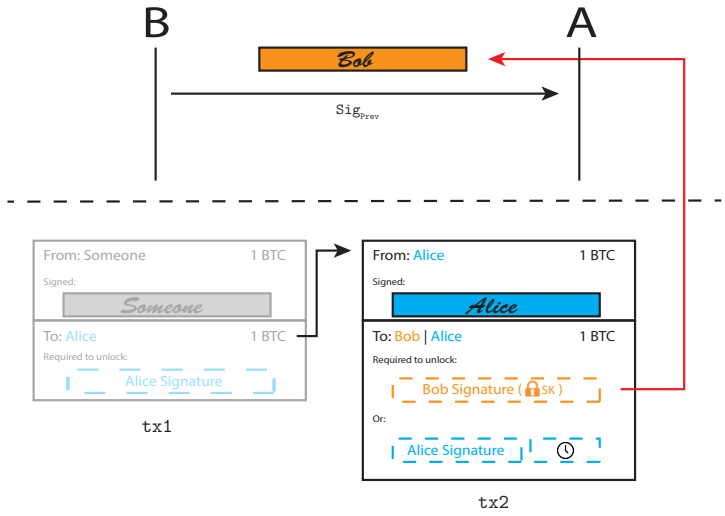

Figure 7.1: Transactions involved in the scheme.

7.3 The data trading protocol

The main property of our data trading protocol is its atomicity which provides its fairness for the participants of the protocol. The proposed protocol is run by two parties, the buyer, B, and the seller, S, and no additional party, like a TTP, is needed. Notice that, contrary to other fair exchange protocols [5, 8, 117], our proposal does not define a dispute mechanism, in which some proposals also need a TTP, thanks to the atomicity of the protocol. Furthermore, since our protocol is based on bitcoins, both parties need to be connected with the Bitcoin network to send/receive transactions from/to the blockchain.

In our scenario, the buyer, B, wants to buy some data D to the seller, S, and he is willing to pay x bitcoins for such data. In order to minimize any possible advantage of one of the parties, we consider that the data being sold can be divided in n different parts and each of those parts may have a meaning by itself. Notice that this scenario is not as restrictive as it would appear, since multiple data falls into this category. On the one hand, many multimedia data has the desired properties. For instance, movies or songs can be sliced and each slice may be recognized as part of the whole performance. The buyer may be interested in acquiring the full movie or song, and would be able to verify that it is indeed correct by just watching or listening to a few segments. Datasets consisting on multiple images may also be partitioned so that each individual image is one of the parts. The images can be individually verified and then the full dataset can be sold. Just to name a specific example, one may be interested in buying a dataset of images of historic monuments and may also be able to check the correctness of the dataset by verifying that a sample of the images are indeed pictures of monuments. On the other hand, when dealing with sensor data, some sensing values may provide evidence that the sensing is correct but the whole sensing data could be needed for specific purposes. For instance, a temperature sensor may record samples every hour. A buyer may be interested in acquiring the sensor data for a full month, for sensors in a given country. By checking a few samples, the buyer may verify they match the expected values and decide to buy the whole dataset in order to perform the desired analysis.

7.3.1 Protocol description

The full protocol, depicted in Figure 7.2, can be divided in three main parts: the *Data correctness proof*, in which a cut & choose protocol between B and S is performed in order to convince B that the acquired data is correct; the *Signature commitment*, used by B to obtain a previous signature performed by S with the private key used to encrypt the data; and the *Private Key Exchange*, used to exchange, atomically, the private key that allows to decrypt the sold information

for the agreed amount of bitcoins.

In the following paragraphs, we describe in detail each subprotocol. We denote by $\{PK, SK\}$ a public key pair and $E_{PK}(\cdot)$ the encryption function using the public key.

In the **Data correctness proof** subprotocol, the buyer B starts the protocol by requesting data to the seller S. In such first step, B will indicate to S the data he is willing to buy. Such request will include the conditions, $cond$, that the data being sold have to hold. Such conditions need to hold not only for the complete data being sold but also for each of the data chunks[1]. Upon reception, S generates a new key pair $\{PK, SK\}$ and sends B the following information (see Step 2 in Figure 7.2): the public key PK, the requested data D encrypted using PK, and the data price x. In order to allow B to prove the data correctness, S does not send the D as a whole bunch of encrypted data but split in n chunks which are encrypted individually (as shown in Figure 7.3), that is: $E_{PK}(D) = \{E_{PK}(D_0), E_{PK}(D_1), ..., E_{PK}(D_{n-1})\}$.

When B receives all the encrypted data, he requests a correctness proof to S consisting in a random subset of non-encrypted data from D. To that end, B selects the subset by randomly choosing a set of m pieces from the encrypted dataset, that is $i_j \ \ \forall j \in [0, m-1], \ \ i_j \in [0, n-1]$. B sends this information and S can build the correctness proof by choosing the unencrypted pieces of data that matches the received indexes, that is, $proof = \{D_{i_j} \ \ \forall j \in [0, m-1], \ \ i_j \in [0, n-1]\}$. S sends such correctness proof to B.

Once B has received the $proof$, he verifies the correctness of D by checking that the proof satisfies the conditions. Furthermore, B validates that the received data also matches with the subset of received encrypted data by recreating the data encryption using PK. Therefore, since the subset has been randomly chosen by B, the correctness of the full dataset can be proved with a given probability. Section 7.3.3 analyses in depth the impact of the parameters of the scheme on

[1] Notice that such conditions will be verified by a validation mechanism. Whether such mechanism is performed automatically or the validation needs a supervised environment is out of the scope of our protocol.

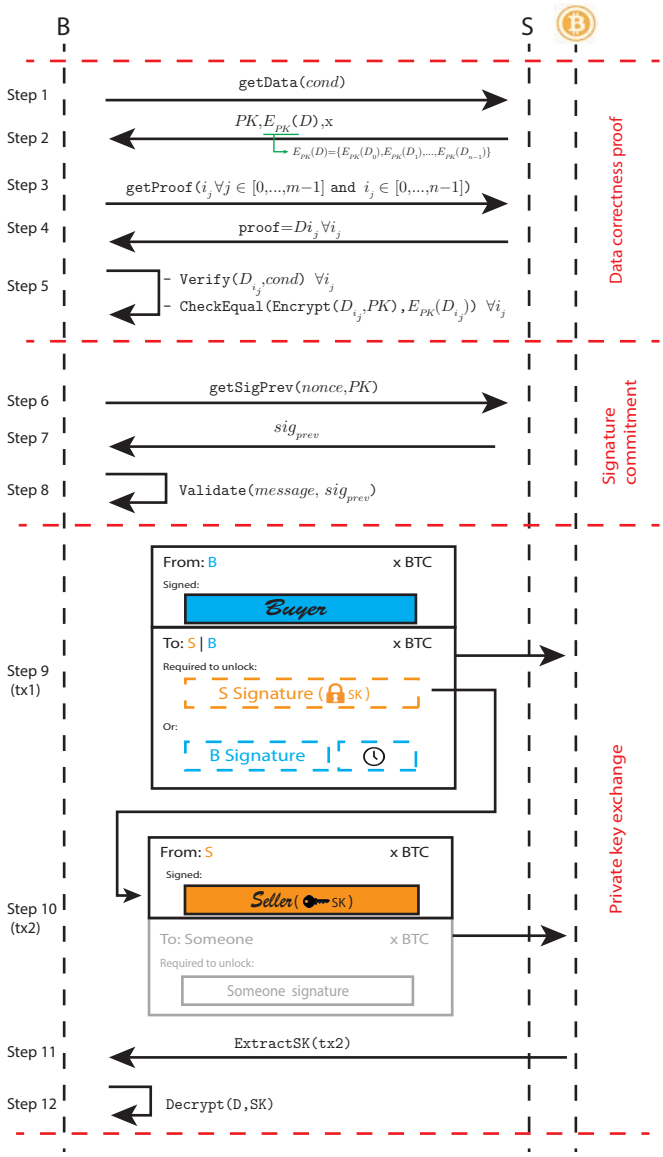

Figure 7.2: Fair data trading protocol.

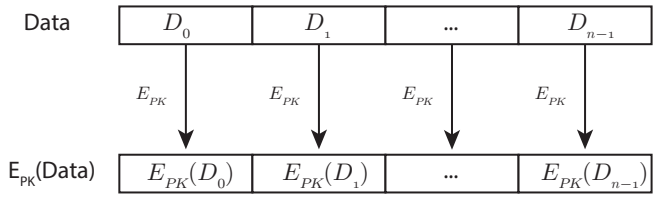

Figure 7.3: Split and encrypt procedure.

such probability.

Once the data correctness has been proven, the **Signature commitment** subprotocol is performed. B requests a signature sig_{prev} over a nonce message performed with the private key SK generated by S. S sends sig_{prev} and B validates that the signature is correct, using the public key PK that has received in Step 2 of the Data Correctness Proof subprotocol.

Finally, the **Private Key Exchange** subprotocol is performed. In such subprotocol, B builds a private key locked transaction, tx_1, to perform the atomic exchange between the private key, SK, and the bitcoin price x. Such private key locked transaction is built using the technique described in Chapter 5.1 and also adding another time constraint condition following the details of Chapter 2.2.2. Such time constraint is used for B to recover the amount of x bitcoins in case S decides not to reveal the private key by not spending the received transaction. B broadcasts the transaction tx_1 to the Bitcoin P2P network. Once tx_1 is included in a block, S can spend the output of such transaction with an input of a new transaction tx_2, in which S will provide the second signature with the same k of sig_{prev}. Once tx_2 appears on the blockchain, B will be able to recover the private key SK and decrypt the data $E_{PK}(D)$ he received in Step 2 to retrieve the purchased data.

7.3.2 Implementation details

In the protocol description provided in the previous section some implementation details have been deliberately omitted to allow a better understanding of the general protocol. In this section, we provide some comments regarding such specific details.

7.3.2.1 Privacy protection

First of all, sensitive information exchanged in the protocol should be protected from third parties. For instance, if an attacker could retrieve the information transmitted in Step 2 and in Step 7, later on, with the knowledge of tx_2 (which is publicly available in the blockchain), he would be able to decrypt the information and retrieve the original data D. To avoid such situation, information transmitted on Steps 2 and 7 could be encrypted using the public key of B, that could be sent to S in Step 1. Furthermore, in Step 4, some part of the data is transmitted in clear for validation purposes. In this case, an external attacker could also obtain such information. Again, such situation can be avoided by encrypting the information of Step 4 in the same way we just described for Steps 2 and 7.

7.3.2.2 Data encryption mechanism

As it is well known, public key cryptography is not suitable for encrypting large files due to its poor performance. Then, since the size of the data chunks that are encrypted and transmitted in Step 2 can vary depending of the traded data, we suggest to use digital envelopes to encrypt D. Digital envelopes [75] protect the message by using a two layer encryption in which the data itself is encrypted using symmetric encryption, and then the symmetric key is encrypted using public-key cryptography. Following such an approach, for each chunk i of data created from D, D_i, a symmetric key k_i is also generated. D_i will be encrypted using k_i, that is $C_i = E_{k_i}(D_i)$ and k_i will then be encrypted using PK, that is, $c_i = E_{PK}(k_i)$. Thus, each encrypted chunk of data D_i sent by S to B during Step 2 should be replaced by $\{C_i, c_i\}$, that is, $E_{PK}(D_i) \rightarrow \{C_i, c_i\}$. Furthermore,

when sending the correctness proof, S will include the corresponding symmetric encryption keys k_i $\forall i \in 0,...,m-1$. Finally, B will need to undo the digital envelope process in Step 5 in order to perform all the required verifications, and also in Step 12, when finally decrypts D.

7.3.2.3 Script building

The private key locked transaction used in the secret key exchange subprotocol, tx_1 also includes a time lock condition to allow B to refund his x bitcoins in case S decides not to follow the last step of the protocol. The details on how the `ScriptPubKey` of such transaction can be build are next provided[2] :

```
ScriptPubKey:    IF
                     OP_DUP <S pubKey> OP_CHECKSIGVERIFY
                     OP_SIZE <0x47> OP_EQUALVERIFY
                     <sigmask> OP_AND <rprev> OP_EQUAL
                 ELSE
                     <expiring time> OP_CHECKLOCKTIMEVERIFY
                     OP_DROP <B pubKey> OP_CHECKSIG
                 ENDIF
```

7.3.3 Protocol fairness discussion

The main objective of the proposed protocol is to achieve fairness in the sense that neither B nor S would have any advantage in the protocol. By advantage we mean that B cannot obtain the data without paying x bitcoins and S cannot obtain the bitcoins without revealing the data. The Data Correctness Proof subprotocol ensures that S cannot sell fake data. Without B verifying parts of the encrypted data, S could encrypt fake data and send it to B. B would not be aware of it until

[2] An example of such a transaction can be found in https://www.blocktrail.com/tBTC/tx/19f8799e074bf253ac1ed39aa25d97b7fd5d82d962d268723971dd84a7cd08f3

he decrypts the data, and it would be too late since S will already have received the payment.

In the following paragraphs, we will show how the buyer B is probabilistically protected against deception by using a cut-and-choose mechanism. Furthermore, the level of protection may be adjusted by fixing the ratio of chunks revealed on Step 2.

In the main steps of the Data Correctness Proof subprotocol:

1. S encrypts each of the n chunks of data with a public key PK and commits to the ciphered chunks by sending them to B.

2. B chooses a subset of m chunks and asks S to reveal the original data corresponding to them.

3. B validates the received chunks by checking both that the original data meets the specified conditions and that the encryption of the original data is equal to the committed values.

We say that a seller S successfully deceives a buyer B if the seller is able to include b corrupted chunks of data within the n traded chunks without the buyer noticing it after having validated the m revealed chunks (that is, after finishing the Data Correctness Proof subprotocol). The probability Ω of S successfully deceiving B is given by the following equation:

$$\Omega(m,n,b) = 1 - \sum_{i=1}^{\min\{b,m\}} \frac{\binom{b}{i}\binom{n-b}{m-i}}{\binom{n}{m}}$$

Indeed, $\binom{n}{m}$ counts the number of ways of choosing m elements from a set of n elements. We are interested in knowing how many of those ways include at least one corrupted chunk. We compute this value by counting the number of ways of selecting m elements with exactly i of them being corrupt and summing them up for all possible i values. $\binom{b}{i}\binom{n-b}{m-i}$ computes the number of ways of selecting exactly i corrupted chunks, that is, the number of ways of selecting i bad chunks from the set of b corrupted chunks, $\binom{b}{i}$, multiplied by the number of ways of selecting

the rest $m-i$ elements from the non-corrupted set $n-b$, $\binom{n-b}{m-i}$. The summation gives the probability of selecting at least one corrupted chunk within the m revealed, that is, the probability of detecting a fraud. Therefore, the probability of deception is the complement.

Figure 7.4 shows the probability of deception Ω for different ratios of chunks revealed, $\frac{m}{n}$, and different number of corrupted chunks included by the seller, b, for $n = 1000$. Note that, even when the checked chunks ratio is low, the probability of successfully deceiving a buyer is low whenever b is over a certain threshold. For instance, when 20% of the chunks are checked, the probability of deception is 0.8 if the seller includes just 0.1% of corrupted chunks ($b = 1$, red dot on Fig. 7.4). However, if the seller includes 1% of corrupted chunks ($b = 10$), the probability of successfully deceiving the buyer decreases to 0.106 (green dot on Fig. 7.4).

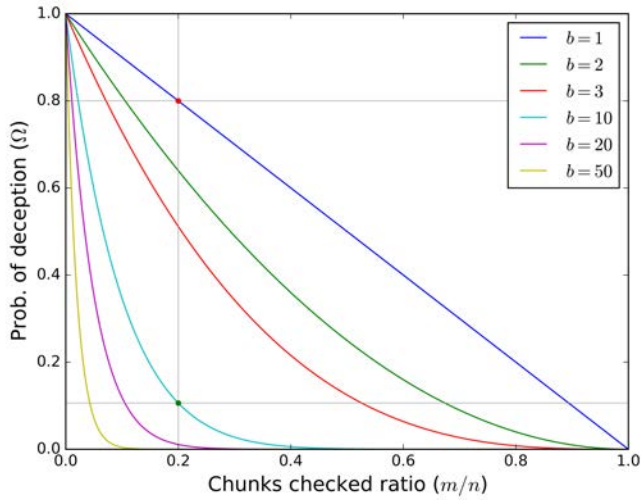

Figure 7.4: Probability of deception Ω. Grey lines highlight the values mentioned as numerical examples.

When using the proposed data selling protocol, the two parties (buyer and

seller) agree on the value of the parameter m. Therefore, the buyer can decide beforehand whether or not to buy a given dataset depending on the deception risk he is willing to assume. Buyers will be interested in using high m values, since these offer higher levels of protection. Of course, even honest sellers will prefer low m values, since if the client does not finally buy the data, m data chunks end up being revealed for free.

It is worth mention that a malicious buyer could try to get advantage of the protocol by executing it with different identities (trying to perform a sybil attack) buying the same product or even colluding with other buyers. Multiple executions of the Data correctness proof could potentially provide attackers with all the chucks of decrypted data. To avoid such situation, the ordering of the encrypted chunks $\{E_{PK}(D_0), E_{PK}(D_1), ..., E_{PK}(D_{n-1})\}$ used in the data correctness proof protocol is randomly chosen for each buyer, so the probability that the buyer (or a coalition) obtains all the chunks decrypted can be minimized by increasing the value n. Note that given two different encrypted sets of data, an attacker would not be able to know the correspondence between each data chunk in both sets.

As an example, let us consider a certain seller S that sells a dataset divided in $n = 1000$ chunks. The buyer may allow to accept, at most, 5% of corrupted chunks in his purchased data, and the seller does not want to reveal in the Step 2 more than the 5% of the chunks. With this configuration, if we analyse the probability Ω for $b = 50$ (the 5% of the 1000 chunks),

$$\Omega(50, 1000, 50) \approx 0.072$$

the buyer can be certain that the seller cannot cheat with a probability greater $1 - 0.072 = 0.928$. Of course, if the buyer does not have any trust in the seller, he could force the seller to reveal 10% of chunks instead of only 5% in Step 2. With this settings, the buyer knows that the probability the seller cheats is almost negligible, less than a 0.004.

7.4 Conclusions

In this chapter we have introduced a fair data trading protocol based on Bitcoin transactions. The protocol uses a new type of transactions, the private key locked transaction, which provides an atomic way of exchanging a private key for Bitcoins. Such key is used to encrypt all the sold data and will be traded, as a part of a Bitcoin smart contract, only when the two parties agree. The correctness of the data sold using the protocol is verifiable by the buyer before performing the transaction by checking a small random subset of data. By using such a cut-and-choose technique, deception is avoided with a high probability while only a small part of the information is learned by the buyer.

The protocol can be implemented by using private key locked transaction introduced in Chapter 5, and exchanging a few messages between the parties involved in the process, making it easy to deploy. Moreover, it lays on the security measures Bitcoin provides without introducing more complexity, and it is bound to the computational capabilities of the Bitcoin Scripting language.

CHAPTER 8

CONCLUSIONS

This final chapter summarizes the results of the thesis and concludes with some guidelines for future work.

8.1 Conclusions

In the first part of the thesis we have analysed some of the foundations of Bitcoin in order to identify potential flaws, as well as to gain further knowledge about the system to design several solution. Some of those solutions are presented in the last part of the thesis.

In Chapter 3 we have given a network characterization of Bitcoin focusing especially in its security. We reviewed the most harmful attacks to P2P networks and we have seen how Bitcoin has built-in countermeasures to deal with them. Some clear examples are liveness of the system via massive data replication, protection against DoS attacks both at high level (through digital signatures in case of transactions and trough mining and PoW in case of blocks), and low

level (through peer banning when malformed network messages are detected) and, most notably, protection against network partitioning and node eclipsing by using pseudo-random peer selection algorithms and obfuscating the network topology. The massive data replication and the protection of the network topology properties stand out when compared to traditional P2P networks, since they will not even be considered useful.

In Chapter 4 we have analysed one of the core components of Bitcoin, the UTXO set. A proper use of the UTXO set is fundamental for the health of the system, since not doing so will harm every single Bitcoin full node. Our analysis has first shown the structure of the set, and later on we have analysed its content. The analysis has focused especially in dust and unprofitability, two metrics that define how usable a coin is. Our results show how a huge part of the UTXO set is filled with dust and unprofitable outputs for low fee rates, pointing out that the use of Bitcoin has been far from optimal so far. Such findings point out the need of a better management of UTXOs, as well as a consolidation of dust.

The rest of the thesis has been focused on developing solutions for different issues within Bitcoin.

First, Chapter 5 introduced private key locked transactions, a proposal for a new type of transactions within Bitcoin. Private key locked transactions create a new type of lock in which a private key needs to be discoloured in order to redeem an output. Such new type of locks are used as a building block in the last two chapters of the thesis.

In Chapter 6 we proposed a prevention mechanism for double-spending on zero-confirmation transactions based on private key locked transactions. The proposal aims to discourage potential attackers to perform double-spending in zero-confirmation transactions, allowing any user who detects such a behaviour to punish the attacker by claiming the output to himself. The main goal of the proposal is to provide a way to use zero-confirmation transaction in scenarios in which time is crucial, and where waiting for a confirmation is not possible. In such a way, a merchant can be sure that if the customer cheats, he risks losing way more than what he was willing to pay for the product.

Finally, Chapter 7 presents a fair protocol for data trading based on private key locked transactions and Bitcoin scripting. Our proposal aims to remove the trust a customer has to have in a merchant when buying some goods online using Bitcoin. While one of the main properties of Bitcoin is that it is a trustless system, in the sense that everything can be verified, when using Bitcoin to buy anything online a customer has to perform a non-refundable transaction to a merchant, who upon receiving the payment will deliver some goods. However, such last statement cannot be enforced. Therefore, customers are in a disadvantageous position with respect to merchants. With our fair data trading protocol we help to even up such a relationship by forcing the exchange to be atomic. Hence, customers cannot received any (useful) data without performing a payment and merchants will not receive any payment until providing the data as agreed.

8.2 Future work

The work done in the thesis open several future lines of research. In this section we will present some of our ideas.

Regarding the Bitcoin network study, it is clear that a better understanding of the requirements that a cryptocurrency network has are necessary in order to build stronger and more secure systems in the future, as well as to avoid attacks that may pose severe threats to the existing ones. Furthermore, the knowledge of the topology of the network will be desirable. Even though we have stated that the obfuscation of the topology is important in terms of avoiding network based attacks, the absence of such knowledge does not let us see if some undesirable behaviour is going on in the network, such as network centralization, censoring, etc.

With respect to the UTXO set, the analysis should be extended filling the current gaps in Bitcoin, like a more accurate estimation for the unprofitable output metric (one that could estimate more precisely the minimum input of some types of transactions, such as P2SH). Moreover, the analysis could also be extended to other cryptocurrencies using the UTXO approach, such as Bitcoin

Cash or Litecoin.

As private key locked transactions are concerned, a deeper study should be performed to see how it could be applicable in layer two protocols, such as the Lightning Network, or even ported to other currencies, like Ethereum. Moreover, there are other applications of private key locked transactions that we have not fully analysed, like using in open access systems, or even use them as an incentive to move funds bound to old non-compressed keys in order to reduce the size of the UTXO set.

Finally, other research lines will be also interesting to follow, such as layer two routing protocol design.

BIBLIOGRAPHY

[1] E. ANDROULAKI, G. KARAME, M. ROESCHLIN, T. SCHERER, AND S. CAPKUN, *Evaluating user privacy in bitcoin*, in Financial Cryptography and Data Security, A.-R. Sadeghi, ed., vol. 7859 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 34–51.

[2] M. ANDRYCHOWICZ, S. DZIEMBOWSKI, D. MALINOWSKI, AND L. MAZUREK, *Secure multiparty computations on bitcoin*, Commun. ACM, 59 (2016), pp. 76–84.

[3] A. M. ANTONOPOULOS, *Transaction scripts and script language*, in Mastering Bitcoin: unlocking digital cryptocurrencies, O'Reilly Media, Inc., 2014, ch. 5.

[4] M. APOSTOLAKI, A. ZOHAR, AND L. VANBEVER, *Hijacking bitcoin: Routing attacks on cryptocurrencies*, in 2017 IEEE Symposium on Security and Privacy (SP), May 2017, pp. 375–392.

[5] N. ASOKAN, V. SHOUP, AND M. WAIDNER, *Optimistic fair exchange of digital signatures*, IEEE Journal on Selected Areas in Communications, 18 (2000), pp. 593–610.

[6] A. BACK ET AL., *Hashcash-a denial of service counter-measure*, (2002).

[7] T. BAMERT, C. DECKER, L. ELSEN, R. WATTENHOFER, AND S. WELTEN, *Have a snack, pay with bitcoins*, in Proceedings of the IEEE Internation Conference on Peer-to-Peer Computing (P2P), Trento, Italy, 2013., 2013.

[8] F. BAO, R. H. DENG, AND W. MAO, *Efficient and practical fair exchange protocols with off-line ttp*, in Proceedings. 1998 IEEE Symposium on Security and Privacy, 1998, pp. 77–85.

[9] S. BASU, I. EYAL, AND E. G. SIRER, *Falcon*. `https://www.falcon-net.org/`, 2016.

[10] D. BAYER, S. HABER, AND W. S. STORNETTA, *Improving the efficiency and reliability of digital time-stamping*, in Sequences II, Springer, 1993, pp. 329–334.

[11] M. BELLARE, S. GOLDWASSERY, AND D. MICCIANCIOZ, *Pseudo-random number generation within cryptographic algorithms: the DSS case*, 1997.

[12] S. BELLOVIN, *Security aspects of napster and gnutella*, in 2001 Usenix Annual Technical Conference, 2001.

[13] M. BEN-OR, O. GOLDREICH, S. MICALI, AND R. L. RIVEST, *A fair protocol for signing contracts*, IEEE Transactions on Information Theory, 36 (1990), pp. 40–46.

[14] I. BENTOV AND R. KUMARESAN, *How to use bitcoin to design fair protocols*, in Advances in Cryptology – CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II, J. A. Garay and R. Gennaro, eds., Springer Berlin Heidelberg, 2014, pp. 421–439.

[15] A. BIRYUKOV, D. KHOVRATOVICH, AND I. PUSTOGAROV, *Deanonymisation of clients in bitcoin p2p network*, in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014, pp. 15–29.

[16] A. BIRYUKOV AND I. PUSTOGAROV, *Bitcoin over tor isn't a good idea*, in Security and Privacy (SP), 2015 IEEE Symposium on, IEEE, 2015, pp. 122–134.

[17] BITCOIN WIKI, *Zero knowledge contingent payment*. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment, feb 2016.

[18] BITCOIN.ORG, *Android security vulnerability*, 2013.

[19] BITICOIN WIKI, *Hashlock*. https://en.bitcoin.it/wiki/Hashlock, 2016.

[20] BLOCKCHAIN.INFO, *Blockchain*. https://blockchain.info/, 2018.

[21] M. BLUM, *How to exchange (secret) keys*, ACM Trans. Comput. Syst., 1 (1983), pp. 175–193.

[22] N. BORISOV AND J. WADDLE, *Anonymity in structured peer-to-peer networks*, Tech. Rep. UCB/CSD-05-1390, Computer Science Division (EECS) University of California, may 2005.

[23] M. BRINKMEIER, G. SCHÄFER, AND T. STRUFE, *Optimally DoS resistant P2P topologies for live multimedia streaming*, IEEE Transactions on Parallel and Distributed Systems, 20 (2009), pp. 831–844.

[24] M. CARLSTEN, H. KALODNER, S. M. WEINBERG, AND A. NARAYANAN, *On the instability of bitcoin without the block reward*, in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, ACM, 2016, pp. 154–167.

[25] M. CASTRO, P. DRUSCHEL, A. GANESH, A. ROWSTRON, AND D. S. WALLACH, *Secure routing for structured peer-to-peer overlay networks*, SIGOPS Oper. Syst. Rev., 36 (2002), pp. 299–314.

[26] D. CERRI, A. GHIONI, S. PARABOSCHI, AND S. TIRABOSCHI, *Id mapping attacks in p2p networks*, in GLOBECOM '05. IEEE Global Telecommunications Conference, 2005., vol. 3, nov 2005, pp. 1785–1790.

[27] D. CHAUM, *Security without identification: Transaction systems to make big brother obsolete*, Communications of the ACM, 28 (1985), pp. 1030–1044.

[28] D. CHAUM, A. FIAT, AND M. NAOR, *Untraceable electronic cash*, in Conference on the Theory and Application of Cryptography, Springer, 1988, pp. 319–327.

[29] J. CLARK AND A. ESSEX, *Commitcoin: Carbon dating commitments with bitcoin*, in Financial Cryptography and Data Security, vol. 7397 of LNCS, Springer, 2012, pp. 390–398.

[30] COPAY, *Copay*.
`https://copay.io/`, 2018.

[31] C. CRAMER, K. KUTZNER, AND T. FUHRMANN, *Bootstrapping locality-aware p2p networks*, in Networks, 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on, vol. 1, nov 2004, pp. 357–361.

[32] L. DASHJR, *getblocktemplate - Fundamentals*.
`https://github.com/bitcoin/bips/blob/master/bip-0022.mediawiki`, 2012.

[33] ——, *getblocktemplate - Pooled Mining*.
`https://github.com/bitcoin/bips/blob/master/bip-0023.mediawiki`, 2012.

[34] C. DECKER, *Data propagation: How fast does information move in the network?*
`http://bitcoinstats.com/network/propagation/`, 2017.

[35] C. DECKER AND R. WATTENHOFER, *A fast and scalable payment network with Bitcoin duplex micropayment channels*, in Symposium on Self-Stabilizing Systems, Springer, 2015, pp. 3–18.

[36] S. DELGADO-SEGURA, C. PÉREZ-SOLÀ, J. HERRERA-JOANCOMARTÍ, AND G. NAVARRO-ARRIBAS, *Bitcoin private key locked transactions.*, IACR Cryptology ePrint Archive, 2016 (2016), p. 1184.

[37] S. DELGADO-SEGURA, C. PÉREZ-SOLÀ, J. HERRERA-JOANCOMARTÍ, G. NAVARRO-ARRIBAS, AND J. BORRELL, *Cryptocurrency networks: A new p2p paradigm*, Mobile Information Systems, 2018 (2018).

[38] S. DELGADO-SEGURA, C. PÉREZ-SOLA, G. NAVARRO-ARRIBAS, AND J. HERRERA-JOANCOMARTÍ, *A fair protocol for data trading based on bitcoin transactions*, Future Generation Computer Systems, (2017).

[39] S. DELGADO-SEGURA, C. PÉREZ-SOLA, G. NAVARRO-ARRIBAS, AND J. HERRERA-JOANCOMARTI, *Analysis of the bitcoin utxo set*, in Proceedings of the 5th Workshop on Bitcoin and Blockchain Research Research (in Assocation with Financial Crypto 18), Lecture Notes in Computer Science, 2018.

[40] J. DINGER AND O. P. WALDHORST, *Decentralized bootstrapping of p2p systems: A practical view*, in NETWORKING 2009: 8th International IFIP-TC 6 Networking Conference, Aachen, Germany, May 11-15, 2009. Proceedings, Springer Berlin Heidelberg, 2009, pp. 703–715.

[41] J. A. D. DONET AND J. HERRERA-JOANCOMARTÍ, *Cryptocurrency P2P networks: a comparison analysis*, in Actas de la XIV Reunión Española de Criptología y Seguridad de la Información (RECSI 2016), J. L. Ferrer and M. Payeras, eds., Menorca, Illes Balears, October 2016, pp. 423–428.

[42] J. J. DOUCEUR, *The sybil attack*, in Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

[43] C. DWORK AND M. NAOR, *Pricing via processing or combatting junk mail*, in Annual International Cryptology Conference, Springer, 1992, pp. 139–147.

[44] S. EVEN, O. GOLDREICH, AND A. LEMPEL, *A randomized protocol for signing contracts*, Commun. ACM, 28 (1985), pp. 637–647.

[45] I. EYAL AND E. G. SIRER, *Majority is not enough: Bitcoin mining is vulnerable*, in Financial Cryptography and Data Security, N. Christin and R. Safavi-Naini, eds., Berlin, Heidelberg, 2014, Springer Berlin Heidelberg, pp. 436–454.

[46] M. FELDMAN AND J. CHUANG, *Overcoming free-riding behavior in peer-to-peer systems*, SIGecom Exch., 5 (2005), pp. 41–50.

[47] FIBRE, *Fibre*.
`http://bitcoinfibre.org/`, 2018.

[48] M. K. FRANKLIN AND M. K. REITER, *Fair exchange with a semi-trusted third party (extended abstract)*, in Proceedings of the 4th ACM Conference on Computer and Communications Security, CCS '97, 1997, pp. 1–5.

[49] A. GERVAIS, S. CAPKUN, G. O. KARAME, AND D. GRUBER, *On the privacy provisions of bloom filters in lightweight bitcoin clients*, in Proceedings of the 30th Annual Computer Security Applications Conference, ACM, 2014, pp. 326–335.

[50] S. GHEMAWAT AND J. DEAN, *Leveldb*.
`https://github.com/google/leveldb`, 2014.

[51] A. GHODSI, L. O. ALIMA, AND S. HARIDI, *Symmetric replication for structured peer-to-peer systems*, in Databases, Information Systems, and Peer-to-Peer Computing: International Workshops, DBISP2P 2005/2006, Trondheim, Norway, August 28-29, 2005, Seoul, Korea, September 11, 2006, Revised Selected Papers, Springer Berlin Heidelberg, 2007, pp. 74–85.

[52] D. GOLDSCHLAG, M. REED, AND P. SYVERSON, *Onion routing for anonymous and private internet connections*, Communications of the ACM, 42 (1999), pp. 39–41.

[53] M. GRUNDMANN, T. NEUDECKER, AND H. HARTENSTEIN, *Exploiting transaction accumulation and double spends for topology inference in bitcoin*, (2018).

[54] S. HABER AND W. S. STORNETTA, *How to time-stamp a digital document*, in Conference on the Theory and Application of Cryptography, Springer, 1990, pp. 437–455.

[55] ——, *Secure names for bit-strings*, in Proceedings of the 4th ACM Conference on Computer and Communications Security, ACM, 1997, pp. 28–35.

[56] D. A. HARDING AND P. TODD, *Opt-in Full Replace-by-Fee Signaling*. `https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki`, 2015.

[57] E. HEILMAN, A. KENDLER, A. ZOHAR, AND S. GOLDBERG, *Eclipse attacks on bitcoin's peer-to-peer network*, in Proceedings of the 24th USENIX Conference on Security Symposium, SEC'15, Berkeley, CA, USA, 2015, USENIX Association, pp. 129–144.

[58] J. HERRERA-JOANCOMARTÍ AND C. PÉREZ-SOLÀ, *Privacy in bitcoin transactions: New challenges from blockchain scalability solutions*, in Modeling Decisions for Artificial Intelligence, Springer, 2016, pp. 26–44.

[59] D. HUGHES, G. COULSON, AND J. WALKERDINE, *Free riding on gnutella revisited: the bell tolls?*, IEEE Distributed Systems Online, 6 (2005).

[60] M. JAKOBSSON AND A. JUELS, *Proofs of work and bread pudding protocols*, in Secure Information Networks, Springer, 1999, pp. 258–272.

[61] D. JAYASINGHE, K. MARKANTONAKIS, AND K. MAYES, *Optimistic fair-exchange with anonymity for bitcoin users*, in 2014 IEEE 11th International Conference on e-Business Engineering, 2014, pp. 44–51.

[62] C. JENNINGS, B. LOWEKAMP, E. RESCORLA, S. BASET, AND H. SCHULZRINNE, *REsource LOcation And Discovery (reload) Base Protocol*.
IETF, RFC 6940, jan 2014.

[63] D. JOHNSON, A. MENEZES, AND S. VANSTONE, *The elliptic curve digital signature algorithm (ecdsa)*, International Journal of Information Security, 1 (2001), pp. 36–63.

[64] G. O. KARAME, E. ANDROULAKI, AND S. CAPKUN, *Double-spending fast payments in bitcoin*, in Proceedings of the 2012 ACM conference on Computer and communications security, ACM, 2012, pp. 906–917.

[65] G. O. KARAME, E. ANDROULAKI, M. ROESCHLIN, A. GERVAIS, AND S. ČAPKUN, *Misbehavior in bitcoin: A study of double-spending and accountability*, ACM Trans. Inf. Syst. Secur., 18 (2015), pp. 2:1–2:32.

[66] P. KIRK, *Gnutella protocol development*, Retrieved June, 27 (2003), p. 2011.

[67] P. KOSHY, D. KOSHY, AND P. MCDANIEL, *An analysis of anonymity in bitcoin using p2p network traffic*, in Financial Cryptography and Data Security, N. Christin and R. Safavi-Naini, eds., vol. 8437 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 469–485.

[68] J. A. KROLL, I. C. DAVEY, AND E. W. FELTEN, *The economics of bitcoin mining, or bitcoin in the presenceof adversaries*, in The Twelfth Workshop on the Economics of Information Security (WEIS 2013), jun 2013.

[69] B. N. LEVINE, C. SHIELDS, AND N. B. MARGOLIN, *A survey of solutions to the sybil attack*, Tech. Rep. 2006-052, University of Massachusetts Amherst, oct 2006.

[70] E. LOMBROZO, J. LAU, AND P. WUILLE, *Segregated Witness*. `https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki`, 2015.

[71] E. K. LUA, J. CROWCROFT, M. PIAS, R. SHARMA, AND S. LIM, *A survey and comparison of peer-to-peer overlay network schemes*, IEEE Communications Surveys Tutorials, 7 (2005), pp. 72–93.

[72] S. MARTI, P. GANESAN, AND H. GARCIA-MOLINA, *Sprout: P2p routing with social networks*, in Current Trends in Database Technology - EDBT 2004 Workshops: EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14-18, 2004. Revised Selected Papers, Springer Berlin Heidelberg, 2005, pp. 425–435.

[73] G. MAXWELL, *The first successful zero-knowledge contingent payment*. Bitcoin Core Blog, `https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/`, feb 2016.

[74] G. MAXWELL, *A deep dive into bitcoin core 0.15*. SF Bitcoin Developers Meetup, `http://diyhpl.us/wiki/transcripts/gmaxwell-2017-08-28-deep-dive-bitcoin-core-v0.15/`, Sept. 2017.

[75] A. J. MENEZES, P. C. VAN OORSCHOT, AND S. A. VANSTONE, *Handbook of applied cryptography*, CRC press, 1996.

[76] R. C. MERKLE, *Protocols for public key cryptosystems*, in Security and Privacy, 1980 IEEE Symposium on, IEEE, 1980, pp. 122–122.

[77] A. MILLER, J. LITTON, A. PACHULSKI, N. S. GUPTA, D. LEVIN, N. SPRING, AND B. BHATTACHARJEE, *Discovering bitcoin's public topology and influential nodes*, 2015.

[78] MYCELIUM, *Mycellium wallet*. `https://wallet.mycelium.com/`, 2018.

[79] S. NAKAMOTO, *Bitcoin: A peer-to-peer electronic cash system*. `https://bitcoin.org/bitcoin.pdf`, oc 2008.

[80] A. NARAYANAN AND J. CLARK, *Bitcoin's academic pedigree*, Communications of the ACM, 60 (2017), pp. 36–45.

[81] K. NAYAK, S. KUMAR, A. MILLER, AND E. SHI, *Stubborn mining: Generalizing selfish mining and combining with an eclipse attack*, in 2016 IEEE European Symposium on Security and Privacy (EuroS P), March 2016, pp. 305–320.

[82] J. NEWSOME, E. SHI, D. SONG, AND A. PERRIG, *The sybil attack in sensor networks: Analysis & defenses*, in Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks, IPSN '04, 2004, pp. 259–268.

[83] P. Q. NGUYEN AND I. E. SHPARLINSKI, *The insecurity of the digital signature algorithm with partially known nonces*, Journal of Cryptology, 15 (2002).

[84] S. J. NIELSON, S. A. CROSBY, AND D. S. WALLACH, *A taxonomy of rational attacks*, in Peer-to-Peer Systems IV: 4th International Workshop, IPTPS 2005, Ithaca, NY, USA, February 24-25, 2005. Revised Selected Papers, Springer Berlin Heidelberg, 2005, pp. 36–46.

[85] M. OBER, S. KATZENBEISSER, AND K. HAMACHER, *Structure and anonymity of the bitcoin transaction graph*, Future Internet, 5 (2013), pp. 237–250.

[86] O. OSUNTOKUN, A. AKSELROD, AND J. POSEN, *Client Side Block Filtering*. https://github.com/bitcoin/bips/blob/master/bip-0157.mediawiki, 2017.

[87] C. PÉREZ-SOLÀ, S. DELGADO-SEGURA, G. NAVARRO-ARRIBAS, AND J. HERRERA-JOANCOMARTÍ, *Double-spending prevention for bitcoin zero-confirmation transactions.*, IACR Cryptology ePrint Archive, 2017 (2017), p. 394.

[88] J. POON AND T. DRYJA, *The Bitcoin lightning network: Scalable off-chain instant payments*, tech. rep., 2015.

[89] P. PRIHODKO, S. ZHIGULIN, M. SAHNO, A. OSTROVSKIY, AND O. OSUNTOKUN, *Flare: An approach to routing in lightning network*, White Paper (bitfury. com/content/5-white-papers-research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016. pdf), (2016).

[90] I. RAY AND I. RAY, *Fair Exchange in E-commerce*, SIGecom Exch., 3 (2002), pp. 9–17.

[91] C. RESEARCH, *Sec 2: Recommended elliptic curve domain parameters*, tech. rep., Certicom Corp., Jan 2010.

[92] D. RON AND A. SHAMIR, *Quantitative analysis of the full bitcoin transaction graph*, in Financial Cryptography and Data Security, A.-R. Sadeghi, ed., vol. 7859 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 6–24.

[93] T. RUFFING, A. KATE, AND D. SCHRÖDER, *Liar, liar, coins on fire!: Penalizing equivocation by loss of bitcoins*, in Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15, ACM, 2015, pp. 219–230.

[94] N. SCHNEIDER, *Recovering Bitcoin private keys using weak signatures from the blockchain*, 2013.

[95] A. SINGH, M. CASTRO, P. DRUSCHEL, AND A. ROWSTRON, *Defending against eclipse attacks on overlay networks*, in Proceedings of the 11th Workshop on ACM SIGOPS European Workshop, EW 11, ACM, 2004.

[96] E. SIT AND R. MORRIS, *Security considerations for peer-to-peer distributed hash tables*, in Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers, vol. 2429 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2002, pp. 261–269.

[97] K. SUTO, H. NISHIYAMA, N. KATO, T. NAKACHI, T. FUJII, AND A. TAKAHARA, *Thup: A p2p network robust to churn and dos attack based on bimodal degree distribution*, IEEE Journal on Selected Areas in Communications, 31 (2013), pp. 247–256.

[98] THE BITCOIN CORE DEVELOPERS, *Bitcoin core 0.10.0rc3 source code: transaction.h, line 137*.
Github, `https://github.com/bitcoin/bitcoin/blob/v0.10.0rc3/src/primitives/transaction.h#L137`, dec 2014.

[99] ——, *Bitcoin core 0.11.0 release notes*.
`https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md`, jul 2015.

[100] ——, *Bitcoin core 0.12.0 release notes*.
`https://bitcoin.org/en/release/v0.12.0`, feb 2016.

[101] ——, *Bitcoin core 0.14 source code: `coins.h`*.
Github, `https://github.com/bitcoin/bitcoin/blob/0.14/src/coins.h`, 2017.

[102] ——, *Bitcoin core 0.15.0 release notes*.
`https://bitcoin.org/en/release/v0.15.0`, sep 2017.

[103] P. TODD, *OP_CHECKLOCKTIMEVERIFY*.

https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, 2014.

[104] P. TODD AND A. TAAKI, *Paypub: Trustless payments for information publishing on bitcoin*.
Github Project, https://github.com/unsystem/paypub, oct 2004.

[105] D. S. TOUCEDA, J. M. SIERRA, A. IZQUIERDO, AND H. SCHULZRINNE, *Survey of attacks and defenses on p2psip communications*, IEEE Communications Surveys Tutorials, 14 (2012), pp. 750–783.

[106] H. TRAN, M. HITCHENS, V. VARADHARAJAN, AND P. WATTERS, *A trust based access control framework for p2p file-sharing systems*, in Proceedings of the 38th Annual Hawaii International Conference on System Sciences, jan 2005, pp. 302c–302c.

[107] T.W. SANDHOLM AND V.R. LESSER, *Advantages of a leveled commitment contracting*, in Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference,AAAI 96,, vol. 1, 1996, pp. 126–133.

[108] F. VALSORDA, *Exploiting ECDSA failures in the Bitcoin blockchain*, 2014.

[109] W. J. VAN DER LAAN, *Obfuscate database files*.
Bitcoin Core Github Issue 6613, https://github.com/bitcoin/bitcoin/issues/6613, jul 2015.

[110] D. S. WALLACH, *A survey of peer-to-peer security issues*, in Software Security—Theories and Systems, Springer, 2003, pp. 42–57.

[111] W. WANG, Y. XIONG, Q. ZHANG, AND S. JAMIN, *Ripple-stream: Safeguarding P2P streaming against DoS attacks*, in 2006 IEEE International Conference on Multimedia and Expo, jul 2006, pp. 1417–1420.

[112] H. WEATHERSPOON AND J. D. KUBIATOWICZ, *Erasure coding vs. replication: A quantitative comparison*, in Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers, Springer Berlin Heidelberg, 2002, pp. 328–337.

[113] P. WUILLE, *Answer to: What are the limits of m and n in m-of-n multisig addresses?*
Bitcoin StackExchange, `https://bitcoin.stackexchange.com/a/28092/30668`, 2014.

[114] A. YEOW, *Bitnodes*.
`https://bitnodes.earn.com/`, 2018.

[115] Y. ZHANG, X. LI, J. HUAI, AND Y. LIU, *Access control in peer-to-peer collaborative systems*, in 25th IEEE International Conference on Distributed Computing Systems Workshops, jun 2005, pp. 835–840.

[116] J. ZHOU AND D. GOLLMAN, *A fair non-repudiation protocol*, in Proceedings 1996 IEEE Symposium on Security and Privacy, may 1996, pp. 55–61.

[117] J. ZHOU AND D. GOLLMANN, *An efficient non-repudiation protocol*, in Proceedings 10th Computer Security Foundations Workshop, 1997, pp. 126–132.

Sergi Delgado Segura
Cerdanyola del Vallès, July 2018