IMT School for Advanced Studies, Lucca
Lucca, Italy

Models and analysis for the Bitcoin ecosystem

PhD Program in Systems Science

Track in Computer Science and Systems Engineering

XIV Cycle

By

Ivan Mercanti

2022

The dissertation of Ivan Mercanti is approved.

PhD Program Coordinator: Prof. Rocco De Nicola, IMT School for Advanced Studies Lucca

Advisor: Prof. Rocco De Nicola, IMT School for Advanced Studies Lucca

Co-Advisor: Prof. Stefano Bistarelli, University of Perugia

The dissertation of Ivan Mercanti has been reviewed by:

Prof. Massimo Bartoletti, Universty of Cagliari

Prof. Damiano di Francesco Maesa, University of Cambridge

IMT School for Advanced Studies Lucca
2022

To my family

Contents

List of Figures **xi**

List of Tables **xvi**

Acknowledgements **xviii**

Vita and Publications **xx**

Abstract **xxii**

1 Introduction **1**

2 Background **7**

2.1 Bitcoin 7

2.1.1 Transactions 8

2.1.2 Addresses 10

2.1.3 Blockchain 12

2.1.4 Scripting Language 14

2.1.5 Standard Transactions in Blockchain 15

2.2 Distributed Consensus (The Byzantine Generals Problem) . . . 22

2.2.1 Proof of work (PoW) 24

2.3 The PRISM Language 30

3 BlockchainVis Suite **33**

3.1 Architecture 34

3.2 Visualizer 37

3.3 Case Study 45

3.3.1 Studying a pattern . . . 45
3.3.2 CryptoTorLocker2015 . . . 47
3.3.3 WannaCry . . . 49
3.4 Bitcoin Addresses Scraper . . . 54
3.5 Conclusions . . . 57

4 Non-standard Transactions **58**
4.1 An Analysis of Bitcoin transactions . . . 59
4.1.1 Standard Transactions . . . 59
4.1.2 Non-standard Transactions in blockchain . . . 61
4.2 An analysis of Pay to Script Hash . . . 67
4.2.1 Standard Transactions in P2SH . . . 67
4.2.2 Non-standard Transactions in P2SH . . . 68
4.3 An analysis of OP_RETURN . . . 75
4.3.1 Data Dimension . . . 75
4.4 Transaction Information Tool . . . 77
4.5 Related work . . . 79
4.6 Conclusions . . . 80

5 Study on reused Addresses **82**
5.1 Analysis of reused Bitcoin addresses . . . 83
5.2 Hidden addresses . . . 85
5.2.1 Analysis of Bitcoin hidden reused addresses . . . 87
5.2.2 Visualisation of Hidden Addresses . . . 89
5.3 De-anonymisation of reused addresses . . . 89
5.3.1 Evaluation of de-anonymisation . . . 92
5.3.2 Mixing services addresses . . . 93
5.3.3 Mining pool addresses . . . 96
5.4 Conclusions . . . 97

6 Arbitrage and Bubbles in the Bitcoin market **99**
6.1 Modeling the Bitcoin price dynamics and arbitrage opportunities 100
6.1.1 Data description . . . 100
6.1.2 Arbitrage opportunities . . . 102
6.1.3 Experiments . . . 104

6.2 Arbitrage opportunities in practice . . . 105
6.2.1 How it works . . . 106
6.2.2 Results . . . 107
6.3 The speculative bubble in BTC/USD rates . . . 108
6.4 Mixed causal-noncausal autoregressive models . . . 111
6.4.1 Introduction to noncausality . . . 111
6.4.2 Mixed Causal-Noncausal Autoregressive Model . . . 112
6.5 The Data . . . 115
6.5.1 Price decomposition . . . 115
6.6 Estimated models . . . 117
6.6.1 Noncausal analysis of the bubble component . . . 119
6.6.2 Noncausal analysis of the observed price BTC/USD . . . 122
6.6.3 Residual analysis . . . 125
6.7 Conclusions . . . 128

7 Stochastic Modelling and Analysis of the Bitcoin Protocol 129
7.1 The Extended PRISM Language . . . 132
7.1.1 Blocks . . . 132
7.1.2 Ledgers . . . 133
7.1.3 Sets . . . 135
7.2 The Bitcoin Model . . . 135
7.2.1 General Model . . . 136
7.2.2 Churn Nodes . . . 141
7.2.3 Network Topologies . . . 143
7.3 Stochastic analysis . . . 145
7.3.1 Coherence of the abstract model . . . 146
7.3.2 Variation of Cryptopuzzle Difficulty . . . 148
7.3.3 Churn Nodes . . . 151
7.3.4 Different Topologies . . . 153
7.4 Conclusions . . . 155

8 Related Works 156

9 Conclusion and Future Works 163

A Appendix Title **170**
A.1 Technologies 170
A.2 The Database of Transactions 171
A.3 Web Interface 175

List of Figures

1 Transaction fields. 9
2 Bitcoin's address generations. 11
3 Block fields. 13
4 Bitcoin script validation. 15
5 P2PKH script validation. 16
6 P2PK script validation. 18
7 Multi-signature script validation. 19
8 P2SH script validation. 20
9 The position of new signatures in witness transactions. 21
10 Scenario of the Byzantine Generals Problem where the lieutenant 2 is a traitor. 22
11 Scenario of the Byzantine Generals Problem where the commander is a traitor. 23
12 Proof of work's protocol. 25
13 A Fork in Blockchain. 26
14 An example of a PRISM module 30
15 Transition system of module `C` 31

16 BlockchainVis Suite architecture. 35
17 Visualizer architecture flow. 36
18 A summary of the architecture and technologies. 37
19 (a) The whole Bitcoin archipelago, and (b) by keeping only islands with 2-10 miners (min-max). 38
20 Duration of Bitcoin transaction island. 39

21 Time based visualisation. 40
22 Summary of island statistics panel. 41
23 An example (a) of an island as visualised by BlockChainVis. Larger nodes are addresses, and smaller ones are transactions. The darker larger node has a higher (incoming) budget. In (b) we filter (a) by keeping only transactions in the first half of the block interval considered in (a). 41
24 The three slide-bars of the filter panel. 42
25 An example of filter application on (a) the initial graph: (b) by highlighting miners, (c) hiding coinbase transactions (rewarding the miners), (d) highlighting leaves, (e) applying a transaction-value interval, (f) showing only roots, (g) visualising paths (most of the two-colour nodes are the same roots in (f), the others are the leafs), (h) combines b and f together (darker nodes highlight the same miners in (b)), (i) focuses on a given transaction. . . . 43
26 Visualizer drop-down menu. 45
27 From the full visualisation of an island (top-left), to the only two addresses involved in all the transactions (bottom-right). First, all changes and binary transactions are collasped to discover a single leaf. Then, all the root-to-leaf paths are extracted. 46
28 CryptoTorLocker2015 address (in purple) income. 48
29 CryptoTorLocker2015 address after (in purple) path filter. 48
30 Wannacry address 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94 income. 50
31 Wannacry address 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw income. 51
32 Wannacry address 115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn income. 51
33 Highlighting the three addresses in Figure 30. 52
34 Addresses Scraper architecture flow. 55
35 Addresses Scraper DB structure. 55
36 Visualizer info generated by addresses scraper. 57

37 Distribution (number of occurrences in blockchain) of standard transactions (left) and distribution of Multi-signature transactions (right). 60
38 Evolution over time of the output type used in coinbase transactions. 60
39 Distribution of non-standard transactions (left) and distribution of their miners (right). 66
40 Distribution of non-standard transactions over time (left) and percentage distribution of non-standard transactions type over time (right). 67
41 Distribution of standard (left) and CLVT transactions inside P2SH (right). 67
42 Distribution of OP_DROP (left) and OP_HASH160 OP_EQUALVERIFY transactions inside P2SH (right). 70
43 Distribution of OP_IF(left) and non-standard transactions inside P2SH (right). 74
44 Distribution of OP_RETURN transactions divided by size. . . . 76
45 Distribution of OP_RETURN transactions over time (left) and Distribution of miners in "non-standard" OP_RETURN transactions (right). 77
46 Chart visualization on transaction information module. 78
47 Bitcoin script compiler on Transaction information module. . . . 79

48 The fist 100 most reused addresses. 84
49 Distribution of transactions in the blockchain. 85
50 Distribution of transactions inside P2SH ones. 86
51 The first 100 most reused addresses considering also hidden ones. 87
52 The distribution of address repetitions in time. 87
53 Type address distribution on July 2015. 88
54 Type address distribution on May 2017. 88
55 Type address distribution on December 2017. 89
56 Standard representation of P2SHs in BlochainVis. 89
57 Control panel for a nested address (circled in red the button to show the inside transaction). 90

58 A new representation in BlochainVis of a 2-3 multi-signature transaction inside a P2SH transaction. 90
59 Distribution of the top 100 nested reused addresses. 92
60 Distribution of the top 100 legacy reused addresses. 94
61 Time distribution of legacy addresses reused more than 500 times, each class differently coloured. 94

62 The Bitcoin price in USD according to 5 different Exchanges and the Index value (top) and two sub-samples with only two exchange rates (bottom). 101
63 An example with two simulated paths for two months of daily prices: parameters are set to $\mu_1 = \mu_2 = 1.5$ and $\sigma_1 = 0.75$ (solid), $\sigma_2 = 0.9$ (dashed) respectively. 102
64 Total profit dynamics in USD from January, 1, 2018 to March 30, 2018 (90 days), investing on Gdax and CEX.IO Exchanges with an initial investment $C = 1\$$. 105
65 TradeBitcoin DB schema 106
66 Exchange prices table. 107
67 Number of times that BID has been greater and ASK lower. . . . 108
68 Gain (in USD) over time. 109
69 Bitcoin/USD observed time series 110
70 TradeBitcoin data price download panel 114
71 Bitcoin/USD price decomposition 118
72 Bitcoin/USD price vs. production cost 118
73 Noncausal AR(1) model residuals 120
74 Mixed causal-noncausal MAR(1,1) residuals 121

75 An example of Ledger. 134
76 Two ledgers in a state of fork of length 1. 134
77 The Bitcoin model architecture. 136
78 Daisy chain topology. 144
79 Ring topology. 144
80 Tree topology. 145
81 Hashrate distribution of Bitcoin mining pools on May 2020. . . 147
82 Probability of mining a block. 147

83 Probability of reaching a fork of length 1 by varying the broadcast delay; the bound time T is set to 600 seconds. 148
84 Probability of a fork of length k; the bound time T is set to k*600 seconds. 149
85 Probability of mining a block within 600 seconds. 149
86 Probability of forks; the bound time T is set to 600 seconds. . . 150
87 Probability of mining a block within 3000 seconds. 150
88 Probability of a fork of length 1 with different difficulty level of the cryptopuzzle; the bound time T is set to 600 seconds. 151
89 Probability of mining a block within 3000 seconds. 152
90 Probability of a fork of length k, the bound time T is set to $k*600$ seconds. 153
91 Probability that the node can synchronize in a minute with mean sleep time ranging from 2 to 10 hours. 153
92 Probability of mining a block within 3000 seconds. 154
93 Probability of a fork of length k, the bound time T is set to $k*600$ seconds. 154

94 Docker network. 171
95 Bitcoin DB schema 173
96 Bitcoin connected components DB schema 175
97 Blockchainvis web interface. 176
98 Bitcore control panel. 176
99 DB population control panel. 177

List of Tables

1 Opcodes description. 17

2 Statistics about address reuse. 85
3 Distribution of the 100 most-frequently reused legacy-addresses in blockchain transactions. 93
4 Characteristics of some mixing services. 95
5 Dataset characteristics. 95
6 Similarity of address sets (first seven days). 96
7 Similarity of address sets (second seven days). 96
8 Statistics about reused addresses in mining pools that use addresses to identify users. 97
9 Statistics about reused addresses in the top 5 mining-pools (top considering the amount of mined blocks). 97

10 Parameters fit of Black and Scholes model. 104
11 AR(1) model's estimated parameters 119
12 BDS test results, purely noncausal model AR(1). 119
13 Information Criteria 120
14 MAR(1,1) estimated parameters. 121
15 BDS test results for the MAR(1,1) model residuals on y_t. 121
16 Information Criteria, MAR model on $rate_t$ 122
17 MAR(0,1) estimated autoregressive parameters on $rate_t$ 123
18 Estimated parameters of the Student's t error distribution with MAR(0,1) model on $rate_t$ 123

19 BDS test results for the MAR(0,1) model residuals 123
20 MAR(1,1) estimated autoregressive parameters on $rate_t$ 123
21 Estimated parameters of the Student's t error distribution with MAR(0,1) model on $rate_t$ 124
22 BDS test results for the MAR(1,0) model residuals on $rate_t$. . . 124
23 BDS test results for the MAR(1,1) model residuals on $rate_t$. . . 125
24 MAR(r,s) estimated parameters on $rate_t$, y_t & yc_t 126
25 BDS test results for the MAR(1,0) model residuals on yc_t. . . . 127
26 BDS test results for the MAR(1,1) model residuals on yc_t. . . . 127

Acknowledgements

Working on this thesis, I met many amazing people, and now I'm happy to have the opportunity to thank them properly. This work would not have been possible without the constant support, guidance, and assistance of my advisors Prof.Stefano Bistarelli and Prof Rocco de Nicola. Their levels of patience, knowledge, and creativity are something I will always keep aspiring to. I would also like to thank Dr. Letterio Galletta for his guidance, constructive feedback, support, and advice during my PhD. I also thank Prof. Francesco Santini for his great feedback and excellent encouragement. I would be amiss if I did not mention Dr. Fabio Rossi for his incredible help from the DMI laboratory. Tanks also to Dr. Andrea Bracciali, who made my time in Stirling unique and productive.

My life as a PhD student was surrounded by pleasant moments spent with the people I met and the friends who shared this long path with me. Life in Lucca would not have been so lovely and stimulating without all the IMT mates. Togher, we shared incredible moments that I'll never forget. Among my mates, I want to thank Nicole and Fra Micocci for putting up with me and for the little help on the thesis. Special thanks also go to "casa SAI" (Sedric, Arturo and Ilaria) for hosting me many times in the last months. Continuing with fellow PhD students, thanks to Carlo (Dr. Carlo actually), Francesco (Galt) and Adele. They shared the joys and sorrows of the doctorate with me from different places. I am indebted also to other several friends. The group "Sezione (L)aurea(ti)" group: Luca, Matteo, Emanuele and Francesco (Fra). You always make me laugh. Andrea, Antonietta, Filippo, Francesca and Giulia for their friendship and "crazy travel ideas". Last but not least, "Caccia & Focaccia" Group (too many to list). You made every summer in Panicale unforgivable. Thanks to all my other friends. You always made the

difference.

Finally, my warm and heartfelt thanks go to my family for the tremendous support and hope they have given to me. Without that hope, this thesis would not have been possible. Thank you all for the strength you gave me.

Vita

Mach 18, 1993	Born, Castiglione del Lago (Perugia), Italy
2012-2015	Bachelor Degree in Computer Science Final mark: 98/110 University of Perugia, Perugia, Italy
2015-2017	Master Degree in Computer Science Final mark: 110/110 University of Perugia, Perugia, Italy
January-September 2018	Research Fellow University of Perugia, Perugia, Italy
November 2018-Present	Phd Student IMT Lucca, Lucca, Italy
February-May 2022	Visiting Period University of Stirling, Stirling, UK

Publications

1. Bistarelli, S., Mercanti, I., and Santini, F. An analysis of non-standard bitcoin transactions. In *Crypto Valley Conference on Blockchain Technology*. IEEE Computer Society, 2018.
2. Bistarelli, S., Cretarola, A., Figà-Talamanca, G., Mercanti, I., and Patacca, M. Is arbitrage possible in the bitcoin market? In *GECON - Economics of Grids, Clouds, Systems, and Services* . Springer LNCS, 2018.
3. Bistarelli, S., Mercanti, I., and Santini, F. A suite of tools for the forensic analysis of bitcoin transactions: Preliminary report. In *Proceedings of Euro-Par 2018: Workshop on Future Perspective of Decentralised Applications*. Springer LNCS, 2018.
4. Bistarelli, S., Mercanti, I., and Santini, F. An analysis of non-standard transactions. *Frontiers in Blockchain*, 2, 7, 2019.
5. Bistarelli, S., Mercanti, I., Santancini, P., and Santini, F. End-to-end voting with non- permissioned and permissioned ledgers. *Journal of Grid Computing*, 17(1), 97-118, 2019.
6. Bistarelli, S., Figà-Talamanca, G., Lucarini, F., and Mercanti, I. Studying forward looking bubbles in Bitcoin/USD exchange rates. In *Proceedings of the 23rd International Database Applications & Engineering Symposium* pp. 1-9, 2019.
7. Bistarelli, S., Mercanti, I., Faloci, F., and Santini, F. Highlighting poor anonymity and security practice in the blockchain of Bitcoin. In *Proceedings of the ACM Symposium on Applied Computing*, pp. 265–272, 2021.
8. Bistarelli S., De Nicola R., Galletta L., Laneve C., Mercanti I., and Veschetti A. Stochastic modeling and analysis of the bitcoin protocol in the presence of block communication delays. In *Concurrency and Computation: Practice and Experience*, 2021.

Abstract

Cryptocurrencies are widely known and used principally as a means of investment and payment by more and more users outside the restricted circle of technologists and computer scientists. However, like fiat money, they can also be used as a means for illegal activities, exploiting their pseudo-anonymity and easiness/speed in moving capitals. This thesis aims to provide a suite of tools and models to better analyze and understand several aspect of the Bitcoin blockchain.

In particular, we developed a visual tool that highlights transaction islands, i.e., the sub-graphs disconnected from the super-graph, which represents the whole blockchain. We also show the distributions of Bitcoin transactions types and define new classes of non-standard transactions. We analyze the addresses reuse in Bitcoin, showing that it corresponds to malicious activities in the Bitcoin ecosystem. Then we investigate whether solids or weak forms of arbitrage strategies are possible by trading across different Bitcoin Exchanges. We found that Bitcoin price/exchange rate is influenced by future and past events. Finally, we present a Stochastic Model to quantitative analyze different consensus protocols. In particular, the probabilistic analysis of the Bitcoin model highlights how forks happen and how they depend on specific parameters of the protocol.

Chapter 1

Introduction

The white-paper on Bitcoin appeared in November 2008 [Nak08a], written by a computer programmer using the pseudonym "Satoshi Nakamoto". His invention is an open-source, peer-to-peer digital currency. Money transactions do not require a third-party intermediary, so without traditional financial institutions involved: the Bitcoin network is entirely decentralised. A complete transaction record of every bitcoin and every Bitcoin user's encrypted identity is maintained on a public ledger, called the *blockchain*. For this reason, Bitcoin transactions are thought to be *pseudonymous*, not wholly anonymous.

The actors in the Bitcoin network are the *users* who own a *wallet* associated with a couple (or more) of private/public cryptographic keys. Users employ these keys to sign the transactions they generate to transfer their value to other users; transactions are broadcast to the Bitcoin peer-to-peer network. Notice that this example is just the simplest to understand the form of ownership proof available in Bitcoin. The *miners* update the blockchain, a public distributed data structure that implements the database of every transaction ever executed.

Transactions represent the mechanism that allows a user to pass value to another user. A user can prepare a new transaction referring to the ones through which he/she received value, called the (multiple) *inputs* of this new transaction. The *output* of a transaction describes the destination of bitcoins instead. There can be multiple *outputs*, allowing an owner to make multiple payments at once; one output often represents the change w.r.t. a previous transaction. Notice that

bitcoin in reality are not transferred, but is transferred the right to spend such value.

Miners keep the blockchain consistent, complete, and unalterable: they repeatedly verify and collect newly broadcast transactions into a new group of transactions, called a *block*. To validate a block, a miner needs to compute a random nonce that becomes part of a block and makes it have a hash that starts with a given amount of zeroes (i.e., the *proof-of-work*). This proof is easy to verify but extremely time-consuming to generate.

Criminals often find ways to exploit legitimate technologies for nefarious uses. Bitcoin is not different: due to its pseudo-anonymity and untraceability, it is currently used for criminal activities such as extortion attack ransom, money laundering, and selling illegal goods or services. Bitcoin has been the de-facto currency of the Dark Web, the "hidden" Internet accessible only by *Tor*[1], since the pioneering marketplace *Silk Road*, also known as the "eBay of drugs", came up in 2011. After its shutdown, new *darknet markets* proliferated: these digital markets primarily are black markets, selling or brokering transactions involving drugs, cyber-arms, weapons, counterfeit currency, stolen credit card details, forged documents, unlicensed pharmaceuticals, steroids, other illicit goods, as well as fully-legal products.

Laundering money using Bitcoin is attractive[2] because of its low fees, instantaneous transactions, and virtual anonymity. For instance, bitcoins collected from illegal activities, e.g., proceeds of previously-mentioned markets, can be mixed with "clean" value by using *mixing services* (also called *tumblers*). These services[3] are third parties used to break the connection between a Bitcoin address sending coins, and the address(es) they are sent to. Basically, the destination address of every mixed transaction receives the same amount of value from possibly many different addresses.

Another illicit use of the blockchain is Ransomware [Kha+15a]. It is a software that performs a cryptoviral extortion attack that encrypts data until a ransom is paid to a given Bitcoin address. Thus, ransomware leads to a denial-of-access attack that prevents users from accessing files on the infected computer.

[1] Tor project: `https://www.torproject.org`.
[2] The Guardian: `http://tinyurl.com/hmp2ll7`.
[3] E.g., *Bitmixer*: `https://bitmixer.io`.

Some sadly-famous names of such software are *Cryptolocker*, *Cryptowall*, *TeslaCrypt*, and *Locky*. For what concerns Cryptolocker, it affected 500,000 users until 2014, and an analysis indicates that only 1.3% of all the users hit by the malware paid the ransom of 400$.[4] A more recent and very effective piece of ransomware, which started to spread on May 12th 2017, is WannaCry.

Our answer at all these problems is *BlockChainVis*[5] [BMS18a; BPS18; BS17] Suite, a suite of tools dedicated to the visual analysis of flows of Bitcoin transactions. *BlockChainVis* aims to help analyze such scenarios in depth. In this thesis we present some of its tools:

- The *visualizer* tool highlights transaction islands, i.e., the sub-graphs disconnected from the super-graph, which represents the whole blockchain. It is also possible to apply different filters on, e.g., the interval of dates, blocks, transaction values, or the number of addresses used in transactions, using visual analytics techniques. Such views highlight specific nodes (e.g., roots and leaves of flows or miners) and exclude useless and confusing information: e.g., transactions representing changes can be hidden.

- *Transaction Information* is focused on providing additional information about Bitcoin transactions. The most common kind of transaction is in the form "Bob pays Alice", and it is based on the *Pay to-Public Key Hash*(*P2PKH*) [AW18] script, which is resolved by sending the public key and a digital signature created by the corresponding private key. P2PKH transactions are just one among many standard classes: a transaction is standard if it passes Bitcoin Core's *IsStandard()* and *IsStandardTx()* tests. However, the creation of ad-hoc scripts to lock (and unlock) transactions allows for also generating non-standard transactions, which can be nevertheless broadcast and mined as well. This module explores the Bitcoin blockchain to analyze and classify standard and non-standard transactions, understanding how much the standard behaviour is respected. We also highlight how Bitcoin addresses are reused in transactions (implicitly or explicitly), despite recommendations and best practices strongly advise to

[4] `http://www.bbc.com/news/technology-28661463`.
[5] `http://blockchainvis.dmi.unipg.it/`.

avoid this. We analyze the blockchain to find and count highly reused Bitcoin addresses, starting from the birth of the protocol in 2009. We also include hidden addresses: these are not the explicit addresses of a transaction, but they are somehow contained in it (this also helps to find the same owner addresses, called *wallet*). The other way to find wallets is using some heuristics. For example, the *Multi-Input Heuristic* [RH11a] considers only transactions with more than one input. All the inputs of this transaction are deemed to belong to the same owner.

- *TradeBitcoin* checks the possibility of real-time gain from arbitrage in the Bitcoin Market. In this thesis, we use it to: investigate whether a strong or weak form of arbitrage strategies are indeed possible by trading across different Bitcoin Exchanges; and give validation to a bubble behaviour in exchange rates between Bitcoin and traditional currencies, proving also that the bubble effect is due to confidence in Bitcoin future values.

- *Consensus Analyzer* will be a module to analyze different consensus protocols. Until now, we analyze the Bitcoin one by: extending PRISM, a probabilistic model checker, with a ledger datatype, modelling the behaviour of blockchain's key participants (the miners) and describing the whole protocol as a parallel composition of processes. The probabilistic analysis of the model highlights how forks happen and how they depend on some parameters of the protocol, such as the difficulty of the cryptopuzzle and the network communication delays.

The thesis is organized as follow. In Chapter 2 we introduce some background: some basic information related to Bitcoin, Distributed Consensus protocols and the PRISM Language.

In Chapter 3 we describe a suite of different software tools whose aim is to facilitate the analysis of bitcoin flows and let the forensic scientist extract and visualize useful insights on target (pools of) addresses. We name the whole suite *BlockChainVis*, inheriting from the visualization module [BS17].

Chapter 4 presents the *Transaction Information* module, which shows the distributions of Bitcoin transactions types and defines new classes of non-standard transactions. Moreover, go through P2SH transactions to classify their scripts.

Finally, we show that most miners and users behave respecting the standard way, i.e. using standard transactions.

In Chapter 5 we analyze the addresses reuse in Bitcoin and, using the module of Chapter 4, we define a new type of address (*hidden address*). This type links nested addresses to legacy ones; this is helpful also for find addresses wallets. We also show that such misuse in reusing the address often corresponds to malicious activities in the Bitcoin ecosystem. These results push towards a single mandatory usage, which mitigates dangerous consequences.

In Chapter 6 we investigate whether a solid or weak form of arbitrage strategies are indeed possible by trading across different Bitcoin Exchanges. Our theoretical and practical investigation gives that arbitrage is indeed possible. We also show that Bitcoin price/exchange rate is influenced by future and past events, but that the bubble behaviour is strictly connected to trust in the future of the Bitcoin system. These results were possible thanks to the module *TradeBitcoin* of *BlockChainVis* Suite.

Chapter 7 presents a Stochastic Model to quantitative analyze different consensus protocols. In particular, the probabilistic analysis of the Bitcoin model highlights how forks happen and how they depend on specific parameters of the protocol, such as the difficulty of the cryptopuzzle and the network communication delays. Our results confirm that considering transactions in blocks at a depth larger than five as permanent is reasonable because most miners have consistent blockchains up to that depth with a probability of almost 1.

Chapter 8 discuss our results with the ones in the literature.

Chapter 9 wraps up the thesis with final conclusions and adds possible destinations for future research.

The Origin of the Chapters

Many chapters of this thesis are based on already published papers, jointly with my supervisors Stefano Bistarelli and Rocco De Nicola, and Alessandra Cretarola, Gianna Figà-Talamanca, Francesco Faloci, Letterio Galletta, Cosimo Laneve, Francesco Lucarini, Marco Patacca, Francesco Santini e Adele Veschetti. In particular:

- The architecture and the technologies of BlockchainVis Suite described in

Chapter 3 have been already presented in [BMS18a; BPS18; BS17].

- Chapter 4, that describes non standard transaction and P2SH, is presented in [BMS18b; BMS19].
- The study on reused addresses in blockchain and the definition of the hidden addresses in Chapter 5 is based on the ideas developed in [Bis+21]
- Chapter 6 describes possibilities of arbitrage in Bitcoin market and develops the ideas presented in [Bis+18; Bis+19c]
- Chapter 7 is based on the ideas of a Stochastic Model to analyze consensus protocols developed in [Bis+]

Chapter 2

Background

In this chapter, we are going to describe the Bitcoin features needed for our research, its consensus protocol and provide an introduction to the PRISM framework that will be used for simulations and the analyses.

2.1 Bitcoin

The white-paper on Bitcoin (₿, is the commonly used currency-symbol) appeared in November 2008 [Nak08a], under the pseudonym authorship of "Satoshi Nakamoto". His invention is an open-source, peer-to-peer digital currency. Money transactions do not require a third-party intermediary, so no traditional financial-institution is involved. The identity of Satoshi Nakamoto is still a mystery.

The payer and payee directly interact (peer-to-peer), but without using their real identities, and no personal information is transferred from one to the other. However, unlike a fully anonymous transaction, there is a transaction record-maintained on a public ledger, called the *blockchain*. For this reason, Bitcoin transactions are thought to be *pseudonymous*, not anonymous: Bitcoin addresses are pseudonyms of real individuals (one can have several pseudonyms).

The only way to create new bitcoins is through the *mining* process: *miners* are the nodes that verify the transactions and add them to the blockchain. The bitcoins created each time a miner discovers a new block represents a reward for its job.

The actors in the Bitcoin network are the *users* who own a *wallet* associated with a (or more) couple of private/public cryptographic keys. In Bitcoin, a private key is usually a 256 bit random number, and by using the *Elliptic Curve Digital Signature Algorithm* (*ECDSA*) [JMV01], a 512 bit public key can be obtained from it. Afterwards, from the public key it is possible to obtain a Bitcoin *address*, e.g., applying an hashing function on it. Users use these keys to sign the transactions they generate in order to transfer their value to other users; transactions are then broadcast to the Bitcoin peer-to-peer network. The miners update the blockchain, containing every transaction ever executed.

2.1.1 Transactions

Transactions are the basic bricks of the Bitcoin network: they represent the mechanism that allows a user to send value to another user, e.g., from a buyer to a seller. A bitcoin owner can prepare a new transaction referring to the ones he/she received value in the past. These links are called the (multiple) *inputs* of this new transaction. A transaction *input* must store the proof that the value belongs to who received them in a previous transaction. In fact, a Bitcoin *wallet* stores a collection of public/private key-pairs of a user (and not directly bitcoins). The *output* of a transaction describes the destination of bitcoins instead. There can be multiple *outputs*, allowing an owner to make multiple payments at once; one output often represents the change w.r.t. a previous transaction. So the ownership of the coins is expressed and verified through links to previous transactions. For example, to send 3 ₿ to Bob, Alice needs to refer to other transactions she has previously received, whose amount is at least 3 ₿. Moreover, to lock the coin (output), a script called *scriptPubKey* is used, while to prove the ownership of a coin (input), a script called *scriptSig* is used instead. In the following, we will refer to them as "locking script" and "unlocking script". Finally, a user broadcasts her transaction to the Bitcoin network.

The structure of a transaction is presented in Figure1: txid contains the transaction hash; hex contains all the transaction data hexadecimal encoded; size and vsize are the transaction dimension in byte; version contain the transaction version; blockhash has the hash of the block that contains the transaction; locktime and time have the transaction time information; coinbase is the info of the coin-

```
{
  "hex" : "data",       (string) The serialized, hex-encoded data for 'txid'
  "txid" : "id",        (string) The transaction id (same as provided)
  "hash" : "id",        (string) The transaction hash (differs from txid for witness transactions)
  "size" : n,             (numeric) The serialized transaction size
  "vsize" : n,            (numeric) The virtual transaction size (differs from size for witness transaction
  "version" : n,          (numeric) The version
  "locktime" : ttt,       (numeric) The lock time
  "vin" : [               (array of json objects)
     {
       "txid": "id",    (string) The transaction id
       "vout": n,         (numeric)
       "scriptSig": {     (json object) The script
         "asm": "asm",  (string) asm
         "hex": "hex"   (string) hex
       },
       "sequence": n      (numeric) The script sequence number
       "txinwitness": ["hex", ...] (array of string) hex-encoded witness data (if any)
     }
     ,...
  ],
  "vout" : [              (array of json objects)
     {
       "value" : x.xxx,            (numeric) The value in BTC
       "n" : n,                    (numeric) index
       "scriptPubKey" : {          (json object)
         "asm" : "asm",          (string) the asm
         "hex" : "hex",          (string) the hex
         "reqSigs" : n,            (numeric) The required sigs
         "type" : "pubkeyhash",  (string) The type, eg 'pubkeyhash'
         "addresses" : [           (json array of string)
           "address"        (string) bitcoin address
           ,...
         ]
       }
     }
     ,...
  ],
  "blockhash" : "hash",   (string) the block hash
  "confirmations" : n,      (numeric) The confirmations
  "time" : ttt,             (numeric) The transaction time in seconds since epoch (Jan 1 1970 GMT)
  "blocktime" : ttt         (numeric) The block time in seconds since epoch (Jan 1 1970 GMT)
}
```

Figure 1: Transaction fields.

base transaction (it is empty if the transaction is not a coinbase). Vin contains all the transaction inputs: txid prev and vout are the link to the hash of the spent transaction and his output position; txid is the hash of the transaction; asm and hex contain the script to spent the transaction (unlocking script). Vout has all the transaction outputs: asm and hex contain the script to lock the transaction (locking script); n is the output number; value is the bitcoin value in the transaction; reqsigs is the number of signature required to unlock the output; type is the output type.

UTXO and Memory Pool A UTXO is an *Unspent Transaction Output* that can be spent as an input in a new transaction. To assemble the candidate block,

a Bitcoin miner selects transactions from the memory pool (*mempool* for short): a pool of memorized transactions collected by a miner. The data that is stored in the mempool consists of unconfirmed transactions (that are not inserted in blockchain) which still needs to be processed and inserted in a block by the Bitcoin Network, following a priority scale (depending from the "age" of the transaction, from the value and from other parameters). Today's miners choose which transactions to mine mostly considering the fee-rate, thus prioritizing the transactions with highest fees per kilobyte of transaction size. Any transaction left in the mempool, after the block is filled, will remain in the pool for inclusion in the next block. As transactions remain in the mempool, they increase the "age" parameter. Eventually, also a transaction without fees might reach a high enough priority to be included in the block for free [Ant17].

2.1.2 Addresses

A Bitcoin address (that is an identifier of 26-35 alphanumeric characters) represents a payment destination in Bitcoin. It strictly derives from the hash of a generated public key (*pubkey* in the following) [Ant17]. Addresses can be generated without cost by any user of the network. For example, using Bitcoin Core[2], every user can just click "New Address" and will receive an address. It is also possible to get a Bitcoin address using an account at an exchanger Website, or using an online wallet service. Like e-mail addresses or phone number, you can send them to a person by sending bitcoins to her address. However, people should have different Bitcoin addresses and, to preserve privacy and security, a unique address should be used for each transaction. They can begin with value *1* (classified as *Legacy*), *3* (classified as *Nested SegWit*, or simply *Nested*) or *bc1* (classified as *Native SegWit*)[3] In Figure 2 we can see how to generate an address:

- *Legacy*: It strictly derives from the base58[4] symbols hash of a generated (by the private key) *pubkey*.

[1] Source: `https://en.bitcoin.it/wiki/Address`.
[2] `https://bitcoin.org/it/`.
[3] `https://tinyurl.com/legacy-segwit`.
[4] It is a group of binary-to-text encoding schemes that represent binary data without non-alphanumeric characters (as) and ambiguous letters (0 – zero and O – capital o)

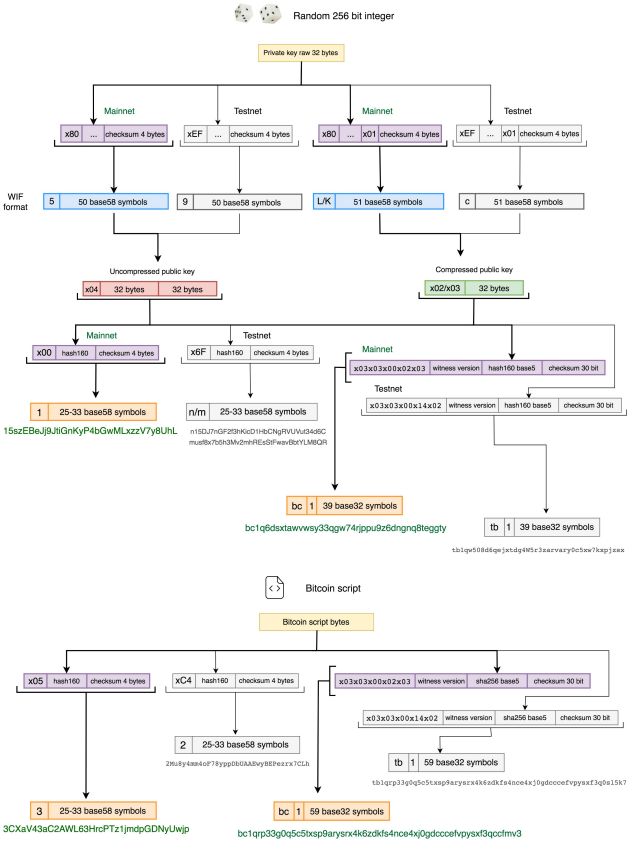

Figure 2: Bitcoin's address generations[1].

- *Nested*: Generated from the base58 symbols hash of a generic Bitcoin script (see 2.1.4).

- *Native SegWit*: Similarly at the Legacy, but using bench32[5] symbols instead of base58 symbols.

Since they strictly derive from a Bitcoin script, Nested addresses can be thought of as the equivalent of writing a check to two parties - "pay to the order of somebody AND somebody else" - where both parties must endorse the check to receive funds. The actual requirement (number of private keys needed, their corresponding public keys, etc.) that must be satisfied to spend the funds is decided in advance by the user who creates the script. Once an address is created, such a requirement cannot be changed without generating a new address (so a new script). We can consider these addresses as a pool of legacy addresses. In Chapter 5 we will provide more details.

2.1.3 Blockchain

The modern notion of the Blockchain was introduced by Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the Bitcoin. In particular, Nakamoto improved the method used by Hash-cash [Bac+02] by proposing to timestamp and signing blocks without resorting to a third trusted party [Nar+16]. The invention of the Blockchain for Bitcoin made it the first digital currency to solve the double-spending problem without a trusted authority or central server. The words block and chain were used separately in Satoshi Nakamoto's white paper [Nak08a], but then became famous as a single word: Blockchain.

In the details, miners keep the Blockchain consistent, complete, and unalterable: they repeatedly verify and collect newly broadcast transactions into a new group of transactions, called a *block*. Mining is used to introduce bitcoins into the system, due to a reward (now 6.25฿[6]) created and assigned to the winning miner. This serves the purpose of disseminating new coins in a decentralised manner, as well as motivating people to provide security for the system.

[5] A checksummed base32 format. For more information, see `https://en.bitcoin.it/wiki/Bech32`.

[6] This reward is halving every four years. Originally in 2009 was 50฿

The first step accomplished by a miner, after collecting transactions, is to perform a verification on them. This implies to check a set of rules, e.g., if transactions format is syntactically correct w.r.t the protocol, or to reject them if the sum of input values is less than sum of output values. Valid transactions (all checks are passed) are added to a block. A block consists of a header and a list of transactions (the block body). Each block contains information that chains it to the previous block in the blockchain, that is the hash of the previous block. Thanks to this field, a block (and consequently the blockchain) is computationally impractical to be modified, since every block after it would also have to be regenerated. The remaining field of the header, i.e., the *nonce*, is obtained from the computation of the proof-of-work by miners.

```
{
  "hash" : "hash",     (string) the block hash (same as provided)
  "confirmations" : n,   (numeric) The number of confirmations, or -1 if the block is not on the main chain
  "size" : n,            (numeric) The block size
  "strippedsize" : n,    (numeric) The block size excluding witness data
  "weight" : n           (numeric) The block weight as defined in BIP 141
  "height" : n,          (numeric) The block height or index
  "version" : n,         (numeric) The block version
  "versionHex" : "00000000", (string) The block version formatted in hexadecimal
  "merkleroot" : "xxxx", (string) The merkle root
  "tx" : [               (array of string) The transaction ids
     "transactionid"     (string) The transaction id
     ,...
  ],
  "time" : ttt,          (numeric) The block time in seconds since epoch (Jan 1 1970 GMT)
  "mediantime" : ttt,    (numeric) The median block time in seconds since epoch (Jan 1 1970 GMT)
  "nonce" : n,           (numeric) The nonce
  "bits" : "1d00ffff", (string) The bits
  "difficulty" : x.xxx,  (numeric) The difficulty
  "chainwork" : "xxxx",  (string) Expected number of hashes required to produce the chain up to this block
(in hex)
  "nTx" : n,             (numeric) The number of transactions in the block.
  "previousblockhash" : "hash",  (string) The hash of the previous block
  "nextblockhash" : "hash"       (string) The hash of the next block
}
```

Figure 3: Block fields.

Figure3 shows the structure of a block: hash contains the block hash; confirmations is the number of block confirmations; size and strippedsize are the block dimension in byte, the second one excludes the transaction txinwitness field; weight contains block weight; version and versionhex contain respectively the block version and the hexadecimal of the block version; merkleroot store the block root node; time and mediantime have the block time information; nonce

contains the block nonce; difficulty is the difficulty to create the block; previousblockhash and previousblockhash have that hash of the previous and the next block. Tx contains all the transaction in the block and nTx is their number.

2.1.4 Scripting Language

The Bitcoin transactions language *Script* is a Forth-like [RCM93] stack-based execution language. Script requires minimal processing and it is intentionally not Turing-complete (no loops) to lighten and secure the verification process of transactions. An interpreter executes a script by processing each item from left to right in the script. Script is a stack-based language: data is pushed onto the stack, instead the operations can push or pop one or more parameters onto/from the execution stack, operate on them, and possibly push their result back in the stack to be used by successive operations.

For example, the operator OP_ADD pops two items from the stack, add them, and finally push the resulting sum onto the stack [Ant14]. There are also conditional operators as OP_EQUAL: it pops two items from the stack and pushes TRUE (represented by number 1) if operands are equal, or FALSE (represented by 0) if they are not equal. In Bitcoin, transaction scripts usually contain a final conditional operator, so that they can produce the result TRUE, which points to a valid transaction.

Most locking scripts refer to a public key address: they require the proof of ownership of the address in order to spend value. However, this is not mandatory [And+14]: any combination of locking and unlocking scripts that result in a final TRUE value is valid. Figure 4 shows the step-by-step validation procedure of the locking and unlocking script. We have this locking script: "2 OP_ADD 8 OP_EQUAL", which can be satisfied by the unlocking script: "6"'. The validation software combines the locking and unlocking scripts and produces the following script: "6 2 OP_ADD 8 OP_EQUAL". This script is interpreted left-to-right as: first 6 is pushed onto an empty stack, then 2, and then the operation OP_ADD is performed between the two last operands in the stack, which are also popped from it. The result, i.e. 8, is pushed onto the stack, and then the 8 in the script is pushed as well. Finally, OP_EQUAL is performed, thus removing the two 8 and pushing TRUE as result.

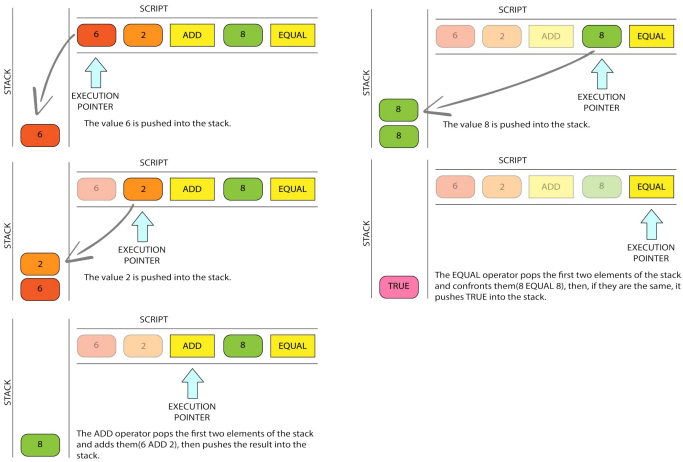

Figure 4: Bitcoin script validation.

Opcodes Opcodes are the operators of the scripting language. In Table 1 we describe some operators that we will refer to in the next chapters.

2.1.5 Standard Transactions in Blockchain

In the first few years of Bitcoin history, developers introduced some limitations in the scripts that could be processed by the reference client. In fact, transactions can be accepted by the network if their locking and unlocking scripts match a small set of believed-to-be-safe templates. This is the *isStandard()* and *isStandardTx()* test, and transactions passing it are called standard transactions.[7] More accurately, the *isStandard()* function gives TRUE if all the outputs (locking script) use only standard transaction forms. On the other hand, the *isStandardTx()* function gives TRUE if all the inputs (unlocking script) use only standard transaction forms according to the output that they are spending. The main reason behind defining and checking standard transactions is to prevent someone from attacking Bitcoin by broadcasting harmful transactions. There are seven standard types of transactions [BMS19].

[7] Valid transactions are reported in the reference source code of *Bitcoin Core*: `https://github.com/bitcoin/bitcoin`.

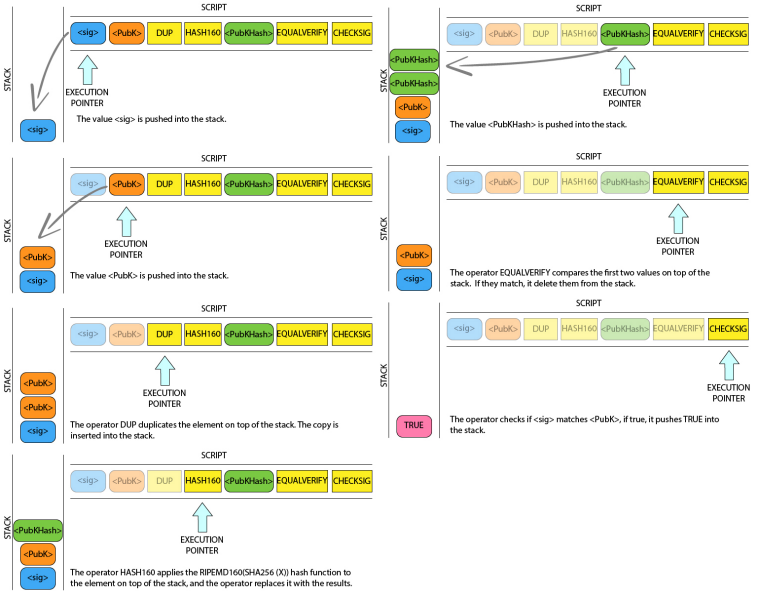

Figure 5: P2PKH script validation.

Table 1: Opcodes description.

Opcodes	Description
OP_HASH160	It hashes twice the top stack element: first with SHA-256 and then with RIPEMD-160.
OP_CHECKSIG	The entire transaction outputs, inputs, and script are hashed. The signature used by OP_CHECKSIG must be a valid signature for this hash and public key. If it is, 1 is returned, 0 otherwise.
OP_MIN	It returns the smaller of the two top elements into the stack.
OP_DROP	It removes the top stack element.
OP_DEPTH	It puts the number of stack elements onto the stack.
OP_2DUP	It duplicates the two elements on top of the stack.
OP_IF	It executes the statements only if the top stack value is not False. The top stack value is removed.
OP_ELSE	It executes its statements if the preceding OP_IF or OP_ELSE was not executed.
OP_ENDIF	It ends an if/else block. All blocks must end, or the transaction is invalid. An OP_ENDIF without an OP_IF before is also invalid.

Pay to Public Key Hash (P2PKH): The transaction *Pay to public key hash* is the most used in the network. This is because it is the default transaction in a Bitcoin client. These transactions contain a locking script that encumbers the output with a public key hash: "OP_DUP OP_HASH160 <PUBLIC KEY A HASH> OP_EQUAL OP_CHECKSIG". Figure 5 shows an example of P2PKH script validation. A P2PKH output can be unlocked (spent) by a public key and a digital signature created with the corresponding private key: "<SIGNATURE A> <PUBLIC KEY A>".

Data output (OP_RETURN): The *Data output* transactions are used to store data not related to Bitcoin payments. Their form is "OP_RETURN <DATA>". Since any output with OP_RETURN is provably un-spendable. Thus, the output can be immediately pruned from the UTXO[8] set even if it has not been spent. These transactions can be used to save different kinds of information on the blockchain,

[8]Unspent Transaction Output, i.e. the transactions that can be spent as an input in a new transaction.

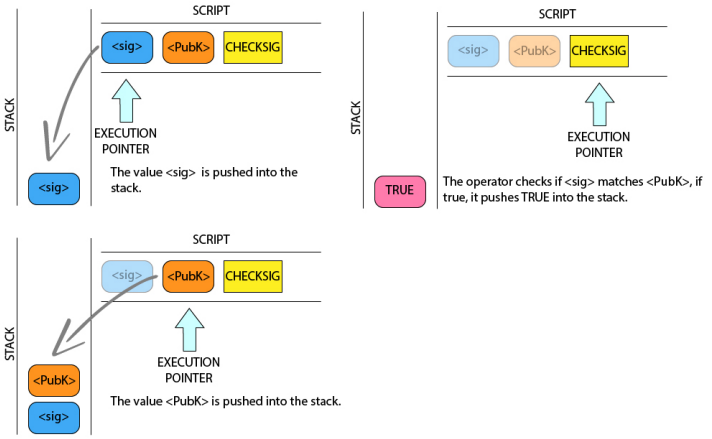

Figure 6: P2PK script validation.

which is in this way used as an immutable distributed ledger by applications, as e-voting ones [Bis+17; Bis+19b]. Such transactions are unlockable. Many members of the Bitcoin community believe that use of OP_RETURN is irresponsible in part because Bitcoin was intended to provide a record for financial transactions, not a record for arbitrary data[9]. Additionally, it is trivially obvious that the demand for external, massively-replicated data store is essentially infinite. Despite this, OP_RETURN has the advantage of not creating bogus UTXO entries, compared to some other ways of storing data in the blockchain. This helps miners to be faster in calculating the priority transactions function.

Pay to Public Key (P2PK): The *Pay to Public Key* scheme is simpler than P2PKH; it was used in coinbase transactions, i.e. the one with the miners are paid for their job. P2PK, as the name suggests, has in its locking script directly the pubkey, instead of its hash: "<PUBLIC KEY A> OP_CHECKSIG". Figure 6 shows the P2PK script validation process. To unlock this transaction, only the corresponding signature of pubkey in the locking script is needed: "<SIGNATURE A>".

Multi-signature: *Multi-signature* scripts set a condition where N public keys are recorded in the script, and at least M of those signatures must be used to

[9] https://en.bitcoin.it/wiki/OP_RETURN.

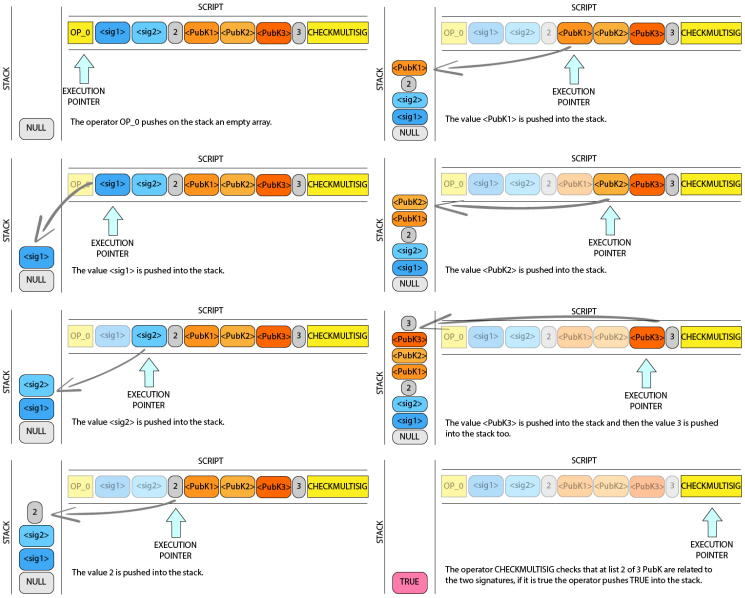

Figure 7: Multi-signature script validation.

unlock a transaction. This is also known as an *M-of-N* scheme, where *N* is the total number of keys and *M* is the lower threshold of signatures required for a validation. The maximum *M* for the current Bitcoin Core implementation[10] is 15. The general form of a locking script setting an *M-of-N* multi-signature condition is: "M <PUBLIC KEY 1> <PUBLIC KEY 2> ... <PUBLIC KEY N> N OP_CHECKMULTISIG". In Figure 7 the validation steps of this script is visually represented. The locking script can be satisfied with an unlocking script containing: "OP_0 <SIGNATURE 1> <SIGNATURE 2> ... <SIGNATURE M>"). Notice that the prefix OP_0 is required because of a bug in the original implementation of CHECKMULTISIG: due to this bug, one more argument on the stack is required. CHECKMULTISIG simply considers it as a placeholder.

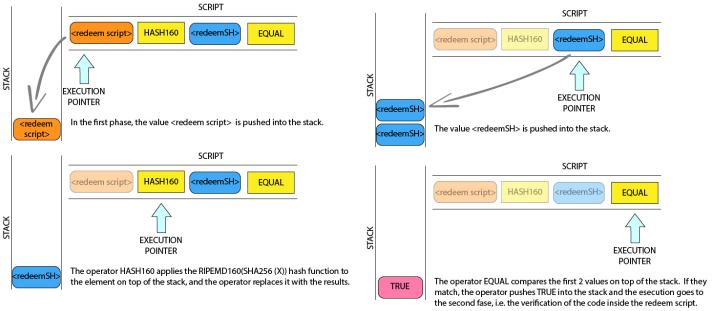

Figure 8: P2SH script validation.

Pay to Script Hash (P2SH): A *Pay to Script Hash* transaction contains the hash of a script of a different transaction (called *redeem script*) in its locking script. For example, we can hash a *2-of-5* multi-signature transaction. Instead of "pay to this 5-key multi-signature script", the P2SH equivalent transaction is "pay to a script with this hash". Hence, the script only stores a 20-byte hash instead of five pubkeys (around 180 byte using the compressed form). Figure 8 shows an example of locking script: "OP_HASH160 <2-OF-5 MULTI-SIGNATURE SCRIPT HASH> OP_EQUAL", with the unlocking script as: "<SIG1> <SIG2> <2-OF-5 MULTI-SIGNATURE SCRIPT>". This particular transaction is recognisable by the nested address, which is generated from the redeem script. This means that

[10] 0.21.1

by knowing the redeem script, a user could extract from it a set of different legacy addresses, which is linked to the real address. For example, if inside the redeem script we have a *2-of-5* multi-signature, we can associate all the 5 addresses of this multi-signature transaction to the address generated by the redeem script. See Figure 8 for an example of P2SH script validation.

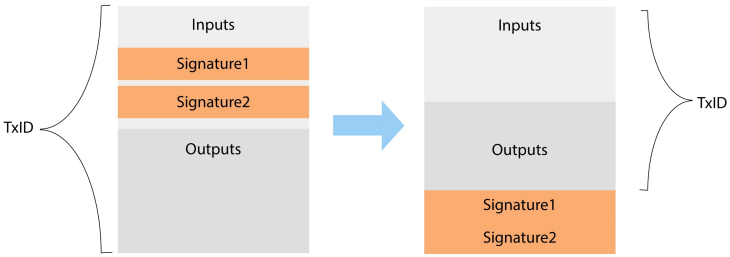

Figure 9: The position of new signatures in witness transactions.

Pay to Witness Public Key Hash (P2WPKH) and Pay to Witness Script Hash (P2WSH): With the introduction of the Segregated Witness[11] (*SegWit*) in Bitcoin, the default transaction P2PKH can be also obtained in a different way. The main differences of the Segregated Witness are the locking script shorter and the signature that are moved outside the unlocking script, see Figure 9. In fact, in place of the longer script in P2PKH, the script in P2WPKH is shortened to: "OP_0 <PUBLIC KEY A HASH>" A P2WPKH output can be unlocked (spent), as the P2PKH, by a public key and a digital signature created by the corresponding private key: "<SIGNATURE S> <PUBLIC KEY A>". The difference is that these components are no longer in the unlocking script, but in the witness field instead.

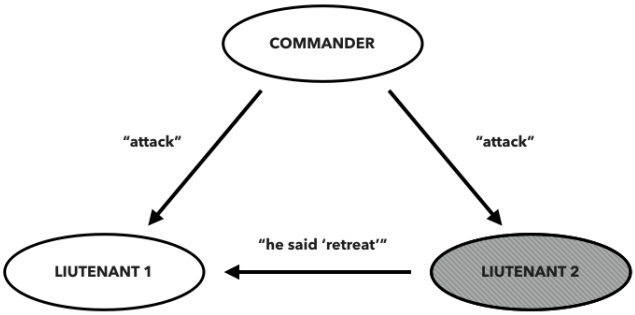

Figure 10: Scenario of the Byzantine Generals Problem where the lieutenant 2 is a traitor.

2.2 Distribuited Consensus (The Byzantine Generals Problem)

The distributed consensus can be described as an instance of The Byzantine Generals Problem [LSP82]. This problem involves a commander and a group of lieutenants which are distributed around a small town. They need to cooperate in order to attack the town at the exact same time. The commander sends an oral message between all the lieutenants, however there might exist some traitors to them, and if there is at least a traitor, the attack will fail. The attack will be successful if more than 2/3 of generals are loyal. By this fact, it comes that with three generals, and with only one traitor, the problem can not be solved. In fact, in Figure 10 the Commander gives orders to both the Lieutenants to attack but Lieutenant 2 is a traitor because he tells lieutenant 1 that the commander ordered him to retreat. Now Lieutenant 1 receives two different orders. Let's assume that Lieutenant 1 follows the commander's order because of strict hierarchy. 1/3 of the army is weaker by force as Lieutenant 2 is a traitor and this creates a lot of problems and confusions and eventually the army will fail. However not only the lieutenants can be traitors, even the commander can be a traitor, as showed by Figure 11. In this second scenario, the attack fails because the first lieutenant obeys to the "retreat order" proposed by the second lieutenant, who in this case

[11] `https://bitcoincore.org/en/2016/01/26/segwit-benefits/`.

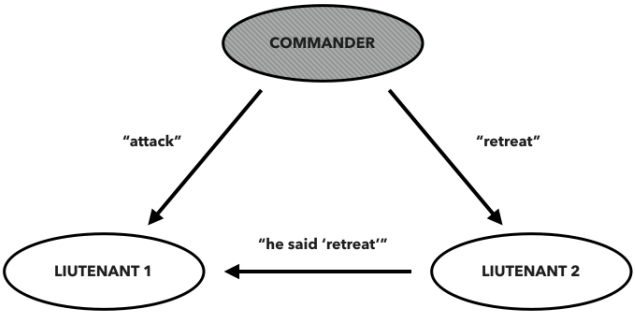

Figure 11: Scenario of the Byzantine Generals Problem where the commander is a traitor.

is not a traitor. It is proved that in this situation, with the use of oral messages, no solution with less than 2/3 of loyal generals can overcome against the rest of traitors.

To sum up, the problem consists of trying to agree on some actions by exchanging information over an unreliable and potentially compromised network. All participating nodes have to agree upon every message that is transmitted between the nodes. This agreement is called as distributed consensus on a distributed system.

Therefore, distributed systems can be divided in two classes:

1. Those systems which are Byzantine Fault Tolerant.

2. Those systems which are Not Byzantine Fault Tolerant.

Systems belonging to the first class have the ability to ensure that consesus is always reached by all the participants of the network, whereas the others belonging to the second class can not always guarantee the achievement of a distributed consensus. This is an important problem for distributed systems such as the blockchain, because, in this particular system a failure to reach a consensus implies the non-publication of a new block, that means no valid transactions between users. Therefore it is crucial for the blockchain to be tolerant to failures regarding the distributed consensus. In fact, a blockchain is resistant to modification of the data, by design. Currently, the main application of the

blockchain is the implementation of a distributed ledger that can securely store transactions without any trusted third party. In this case, a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol. This protocol regulates how the peers of the network communicates and how they update the blockchain by adding new blocks. Once the data are recorded in a block, they cannot be altered retroactively without alteration of all subsequent blocks, which requires consensus of the network majority. Although blockchain records are not unalterable, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance. Decentralized consensus has therefore been claimed using blockchain[12].

2.2.1 Proof of work (PoW)

Proof of work is the consensus algorithm used and proposed for the Bitcoin blockchain [Nak08a]. It is a permissionless consensus process meaning that everyone can join the network and start mining, i.e. to validate transaction. The strategy of this algorithm consists of doing some work in order to validate a new block and add it to the blockchain system. This work is done by all nodes in the network that want to ”mine” a block. In the Bitcoin system, the work that has to be done consists of solving a mathematical problem using the header of the block of transactions and a special number which is called ”nonce” whose size is 32-bit (4-byte) long. The header, combined with the ”nonce”, is then hashed with the sha256 algorithm in order to produce a digital fingerprint uniquely associated with that specific header and so to the block. Hence, the main goal of each miner is to find this nonce in a way that the digital fingerprint produced at the end is composed at the beginning by some zeros. If a nonce does not produce the desired result, then it is replaced with another nonce, and the hash is recalculated until the desired result is reached. The number of zeros required at the beginning of the fingerprint is determined by a special target. A higher target means a lower difficulty while lower target means that it is more difficult to find a hash below that target. The average work required is exponential in the number of zeros required and can be verified by executing a single hash. When the target is reached by a node, this ”winning” node needs to prove that

[12]https://en.wikipedia.org/wiki/Blockchain

```
   /* Joining network                                    */
1  Join the network by connecting to known peers;
2  Start BlockGen();
   /* Main loop                                          */
3  while running do
4      if BlockGen() returns block then
5          Write block into blockchain;
6          Reset BlockGen() to the current blockchain;
           /* Gossiping rule                             */
7          Broadcast block to peers;
8      end
       /* Longest-chain/validation rule                  */
9      if block received & is valid & extends the longest
        chain then
10         Write block into blockchain;
11         Reset BlockGen() to the current blockchain;
12         Relay block to peers;
13     end
14 end
   /* PoW-based block generation                         */
15 Function BlockGen():
16     Pack up transactions (including coinbase);
17     Prepare a block header context 𝒞 containing the
        transaction Merkle tree root, hash of the last block
        in the longest chain, timestamp, and other essential
        information reflecting blockchain status;
       /* PoW hashing puzzle                             */
18     Find a nonce that satisfies the following condition:

                        Hash(𝒞|nonce) < target

        wherein more preceding zero bits in target indicates
        a higher mining difficulty;
19     return new block;
20 end
```

Figure 12: Proof of work's protocol.

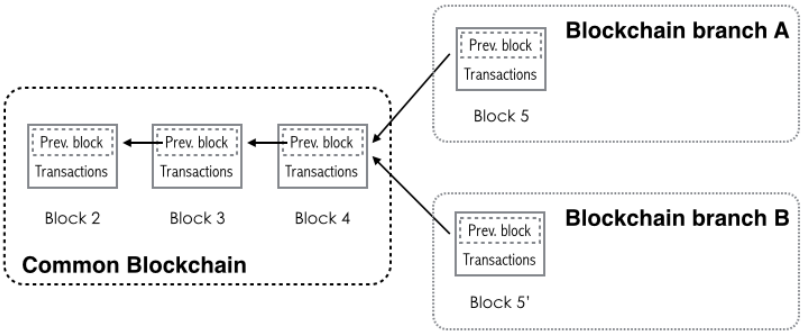

Figure 13: A Fork in Blockchain.

the work is correct, and to do so, he transmits his block and his nonce to all the other nodes on the network. When this happens, all these other nodes check the correctness of the work and reach a consensus. After reaching a distributed consensus on the proposed solution, the block is published on the blockchain. After all these steps, they start working again on a new block of transactions. In Figure 12 [Xia+19] the pseudo-code of PoW protocol. The target presented here, and so the difficulty of the work can be changed by the Bitcoin protocol in order to keep the block generation time at 10 minutes. If the network is finding blocks faster than every 10 minutes, the difficulty increases (the target decreases). If block discovery is slower than expected, the difficulty decreases (the target increases). This kind of process is also required to prove that the node that wants to publish a new block is not likely to attack the network, because, it has to perform all the steps described before. In this process, there are also incentives reserved for those who can solve the problem, including a reward in bitcoin[13] plus the fees, an amount of cryptovalue that user can add to the transactions in order to speed up the validations. By convention, any block with more than six confirmations is considered irrevocable, because it would require an immense amount of computation to invalidate and recalculate six blocks [AW18]. Nodes always consider the longest chain to be the authentic one and will keep working on extending it, however, because of the way this whole process was thought, it

[13] 12.5 in October 2019

can happen that two nodes reach the consensus at the same time. This situation results in a fork of the blockchain, meaning that the main branch is split into two branches, as can be seen in Figure 13. This happens because the blocks are found almost simultaneously by miners on opposite sides of a previous fork, but, the chance of that happening is very low. Even if this will happen, a node that has received different versions of a new block will continue working on the first one received, but saving also the other branch in case it becomes longer. Nodes that were working on a shorter branch will switch to the longer one and the tie connecting the shorter branch will be broken. Such condition ensures two important facts:

- A consensus will always be found by the majority of the nodes.
- Proof of work is byzantine fault tolerant.

There are four main drawbacks with this consensus algorithm:

1. It is not considered to be ”energy saving”.
2. Miners have to do a lot of work in order to publish the block, and all this work is done only to find a special number compatible with the target. For this reason, many consider it a waste of resources because of the unuseful performed work. To address this issues there have been created some cryptocurrencies which use POW but in a more efficient way. One of these cryptovalues is Primecoin [Kin13], where instead of searching simple numbers, miners have to search for long chains of prime-numbers such as Cunnigham chains[14]. The efficient part is that the prime-numbers that are found by the miners, are then used for mathematical research, and they are easily verifiable by all nodes because the protocol specifies that they do not have to be too large. Primecoin adjusts its difficulty slightly every block and blocks come at a rate of one per minute. This was done to increase the speed of the proof-of-work because six confirmations may take fifty minutes in Bitcoin, but they take only six minutes in Primecoin[15].
3. The proof-of-work is subjected to a vulnerability called ”51% attack” and it involves the presence of a group of miners owning the 51% of the total

[14] `https://primes.utm.edu/glossary/page.php?sort=CunninghamChain`.
[15] `https://tinyurl.com/Primecoin-pow`.

hashing power [Wat+16]. By owning this amount of resources, the group would be able to prevent the confirmation of new blocks from the other miners and they also would be able to double-spend coins by storing them in a private blockchain. This is made possible because by having the 51% of the total hash power, the private blockchain would be certainly longer than the original one, and so, after a future publication of it, the protocol would certainly choose it. It is very difficult to realize this type of attack because of the large amount of power required, however it would be possible doing that if all the mining-pool groups would form a unified group.

4. It is not ASIC-resistant. ASICs are integrated circuits that are created to serve a specific use case, performing a particular computing task. In the world of cryptocurrencies, ASIC devices are designed to participate in the process of mining Bitcoin (or other cryptocurrencies). An ASIC-resistant cryptocurrency has its protocol and mining algorithm configured in such a way that using ASIC machines to mine the coin is either impossible or brings no significant benefit when compared to traditional GPU mining. In some cases, using ASICs on ASIC-resistant cryptocurrencies may be even worse than using the more conventional hardware. Since mining involves multiple attempts of finding a solution for a sort of mathematical problem, the job of an ASIC is to perform as many attempts as possible (i.e., as many hashing functions per second as possible). This means that using ASICs to mine Bitcoin or other Proof of Work cryptocurrencies is much better than using a general-purpose piece of hardware, such as a GPU card.

The Bitcoin Protocol

Bitcoin is a peer-to-peer asynchronous network whose nodes host a ledger recording economic transactions that are grouped into *blocks*. The ledgers are trees of blocks with a pointer (*handle*) to a *leaf block at maximal depth*; the *blockchain* is the sequence of blocks from the handle to the root block, (*genesis block*). Blocks are created by special nodes of the network – the *miners* – and contain a number of information, including transactions and a pointer to the current handle of

miner's ledger.

Once a block has been mined, the miner (*i*) adds the block to its own ledger (therefore the depth of the ledger increases and the handle is updated); and (*ii*) broadcasts it to all the connected nodes of the network. Every node receiving the new block updates its local copy of the ledger by inserting the block in the right position and, if necessary, it also updates its own handle. If the block cannot be connected to the ledger (because, due to network delays, a previous block has not been delivered) then it is added to the local set of the miner and will be inserted afterwards (orphan blocks).

Because of asynchrony, it may happen that two nodes mine and broadcast a block almost concurrently, yielding different ledgers with different handles (and, therefore, with different blockchains). This phenomenon, called *fork*, is at the core of the inconsistencies of Bitcoin and, to overcome this problem, the protocol uses a probabilistic algorithm. In particular, Bitcoin has a technique to regulate the mining of blocks, called *Proof of Work* (PoW). According to PoW, miners can add a block only if they solve a computational problem. Technically, the problem consists of finding a number (a nonce) that, when concatenated with the block header and hashed together with the (header of the) block, produces a solution that starts with a certain number of binary zeros and is lower than a predefined condition, which is called a target. The only way to find such a nonce is through an exhaustive search. So the difficulty of the problem is proportional to the size of the nonce search space and to the number of zeros required by the target. The time needed to mine a new block depends on the difficulty of the PoW and the hashing power of the miners. Clearly, the faster miners are, that is the more of computational power they own, the higher is the probability of conflicts and, thus, the more likely is the inconsistency between miners. For this reason, Bitcoin PoW difficulty is determined by a moving average targeting a certain number of blocks per hour. If they are generated too fast, the difficulty increases, as shown by Nakamoto [Nak08b]. The current protocol modulates PoW in order to have 6 blocks per hour on average.

To further reduce the probability of inconsistencies, Bitcoin also uses the so-called *eventual consistency* (also known as *n*-consistency [PSS17]). This is a weak version of consistency, according to which the protocol considers consistent those ledgers with the corresponding blockchains equal up to the last few

blocks. In particular, Bitcoin considers both transactions and miner's rewards in blocks at depth greater than 5 as permanent [AW18].

```
module C
    x : [0..2] init 0;

    [] x=0 -> 2 : (x′ = 1);
    [] x=1 -> 3 : (x′ = 2);
    [] x=1 -> 5 : (x′ = 0);
    [] x=2 -> 2 : (x′ = 1);

endmodule

label "full" = x = 2;
label "empty" = x = 0;
```

Figure 14: An example of a PRISM module

2.3 The PRISM Language

PRISM [KNP02] is a probabilistic model checker that inputs a formal description of a system and computes the likelihood of the occurrence of certain events. The formal description of systems in PRISM is given by means of a process algebra that allows one to specify interacting *modules*. A module contains a number of local variables whose values at any given time constitute its state. Module's behaviour is described by a set of commands whose basic form is described in equation 2.1.

$$[a]g- > rho : update; \tag{2.1}$$

In this command, the guard `g` is a predicate over all the variables in the model (including those belonging to other modules). The update `update` describes a transition that the module can make if the guard is true, and it is specified by assigning the new values of the variables in the module, possibly as an expression formed from other variables or constants. PRISM uses the prime notation to specify updates, *e.g.* `x′ = x+1` means that, in the next state, `x` takes the

value stored in it plus one. The expressions `rho` are used to assign probabilistic information to the transitions – in the following they are called *rates* because we will consider a stochastic semantics. The name `a` is an *action*: when it is present in command of different modules then the corresponding commands must be executed *at the same time*, *i.e.* they synchronize, and the overall rate is the product of the corresponding rates. Finally, a module may also contain labels that are a way of identifying sets of states that are of particular interest.

A PRISM system consists of modules and *global variables*; its *global state* is determined by the local state of all modules, together with the values of the global variables. We refer to Kwiatkowska et al. [KNP11] for a full account of the formalism. Figure 14 reports a PRISM module defining a two-items queue. The variable `x` encodes the state, which has three possible values: `0` (the queue is empty), `1` (the queue has one element) and `2` (the queue is full, *i.e.* it has two elements). For example, from the state `x=1` the system can evolve either in `x=2` with rate 3 and in `x=1` with rate 5.

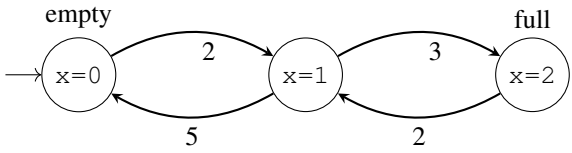

Figure 15: Transition system of module `C`

We observe that the above module `C` may be also described by a transition system where states are those of the system and transitions are labelled by rates (see Figure 15). PRISM supports different kinds of probabilistic formalism. In Chapter 7 we focus on Continuous Time Markov Chains (CTMC) models, which are transition systems (as the one above) whose semantics is defined by rates. In particular, if the rate of a transition from a state s to s' is `r` then the probability of moving from s to s' within $t \geq 0$ time units is $1 - e^{-\mathtt{r} \cdot t}$, that is rates are used as parameters of an exponential distribution. Note that, higher is the rate, higher is the probability to leave s in a given time. This is the exact abstraction used by Nakamoto [Nak08b] to analyze the Bitcoin protocol, where the parameter `r` depends on the miner hashing power and the difficulty level

of the crytopuzzle, and the abstraction discussed by Decker and Wattenhofer in [DW13] where an exponential distribution approximates the probability of delivering blocks across the Bitcoin network within t time units. When a CTMC state has several exiting transitions, *e.g.* the above state `x=1`, then the probability of choosing one transition depends on the rates of the corresponding transitions – this is known as *race condition*. For example, if $\mathtt{r}_1, \cdots, \mathtt{r}_n$ are the rates of transitions exiting from a state `s` (and entering on pairwise different states), then the probability of taking a transition with rate $\mathtt{r}_i$ is $\mathtt{r}_i/\mathtt{R}$, where $\mathtt{R} = \mathtt{r}_1 + \cdots + \mathtt{r}_n$. In this setting, the probability of moving from s in t time units is $1 - e^{-\mathtt{R} \cdot t}$. Since, in Markov chains, the events are independent from the previous events in the history (the Markov property), the probability of reaching a state in a given time t is a function of the product of the probabilities in the intermediate states (this function is not simple because one has to consider all the possible partitions of t and all the possible paths to reach the state from the initial step).

In the PRISM framework, properties of CTMC models are expressed in Continuous Stochastic Logic (CSL) [Azi+96; Bai+00; Bai+03], which is an extension of temporal logic with a probabilistic operator. In particular, in chapter 7 we will analyze formulas of the form `P=?[F<=t property]` that return the probability that the `property` is true in a state of the model within t time units (starting from the initial state). For example, if we want to express the probability that the two-items queue is "full" within t time units, we will write `P=?[F<=5 "full"]`. To compute this probability, PRISM performs Statistical Model Checking. That is, since models may bear infinite sets of paths (in general and in this case, in particular), PRISM does not perform an exhaustive exploration of the state-space. Rather, it imposes a maximum path length to avoid the need to generate excessively long paths and returns the probability of the formula in the finite model. The core idea of the approach is to conduct some simulations of the system, monitor them, and then decide whether the system satisfies the property or not with some degree of confidence. All the simulations in this thesis have been driven with a Confidence Interval (CI) method for Statistical Model Checking that gives an approximate value for a `P=?` property based on a confidence level and the number of samples generated. For instance, if we compute the probability that the two-items queue is "full" within 5 time units in PRISM with the CI method, we obtain that `P=0.957`.

Chapter 3

BlockchainVis Suite

In this chapter we describe the *BlockChainVis*[1] Suite: a suite of tools mainly dedicated to the visual analysis of flows of Bitcoin transactions, but also to other connected activities.

Following the popularity of Bitcoin [Ant14; Nak08a], also other cryptocurrencies have experienced a massive increase in acceptance/use (e.g., Ethereum [But+13], Litecoin[2], Ripple[3], Monero[4]). Hundreds of new cryptocurrencies (coins and tokens) have been offered to the market, currently reaching slightly less than two thousand proposals. Cryptocurrencies are no longer relegated (only) to *darknet markets*[5] or technology enthusiasts. Still, they are nowadays a matter of discussion and investment products known by a large part of the population who has access to ICT. However, due to the pseudo-anonymity offered to users, Bitcoin payments have also become an attractive and frequently used means for collecting money from illegal activities perpetrated by criminals. For instance, Bitcoin payments are requested by most of the last *ransomware*, as *WannaCry* [BPS18] and *Petya*[6]. Other activities are represented by demanding payments for illegal services/goods, as software *exploits* or *Ransomware-as-a-Service* (*RaaS*) target-

[1] `http://blockchainvis.dmi.unipg.it/`.

[2] `https://litecoin.org/it/`.

[3] `https://ripple.com/`.

[4] `https://www.getmonero.org/`.

[5] Commercial Websites that are only reachable through overlay networks implemented by communication anonymisation projects as *Tor* or *I2P*.

[6] `https://tinyurl.com/petyaRamson`.

ing a desired victim. A new frontier could be the use of cryptocurrencies as tax heavens.

BlockChainVis aim to help analyse all such scenarios in deep. At a first step, the tool highlights transaction islands, i.e., the sub-graphs disconnected from the super-graph, which represents the whole blockchain. Then it is possible to apply further filters on, e.g., the interval of dates, blocks, transaction values, or number of addresses used in transactions, using *Visual Analytics* (*VA*) [WT04] techniques. Such views highlight specific nodes (e.g., roots and leaves of flows, or miners) and exclude useless and confusing information: e.g., transactions representing changes can be hidden.

BlockChainVis is dedicated to the visual analysis of flows of bitcoin transactions. Since the blockchain is an example of Big Data, a straightforward visualisation in its entirety is not very significant. Hence, we have exploited some techniques from VA to filter out undesired information, with the purpose to obtain a forensic-tool to efficiently and visually analyse the blockchain and help investigations.

In this chapter, we also show how *BlockChainVis* [BS17] (Section 3.3) visualises all the Bitcoin transactions that have as output one of the addresses ascribed to the ransomware WannaCry. Thanks to their visualisation, we can immediately understand the structure of WannaCry transactions and draw interesting properties, besides retrieving the whole amount of money transferred to them. The chapter is organised as follows: Section 3.1 shows the whole architecture of *BlockChainVis*, a suite of different software tools whose aim is to facilitate the analysis of bitcoin flows and let the forensic scientist extract and visualise useful insights on target (pools of) addresses. In Section 3.2 provides details on the VA techniques used to visualise only needed information. Section 3.3 shows three case-study from the real-world. Section 3.4 presents the addresses scraper. Finally, conclusions and future work (Section 3.5).

3.1 Architecture

BlockchainVis Suite [BS17; BPS18; BMS18a] is a suite of different software tools, whose aim is to facilitate the analysis of bitcoin flows, and let the forensic scientist extract and visualise useful insights on target (pools of) addresses

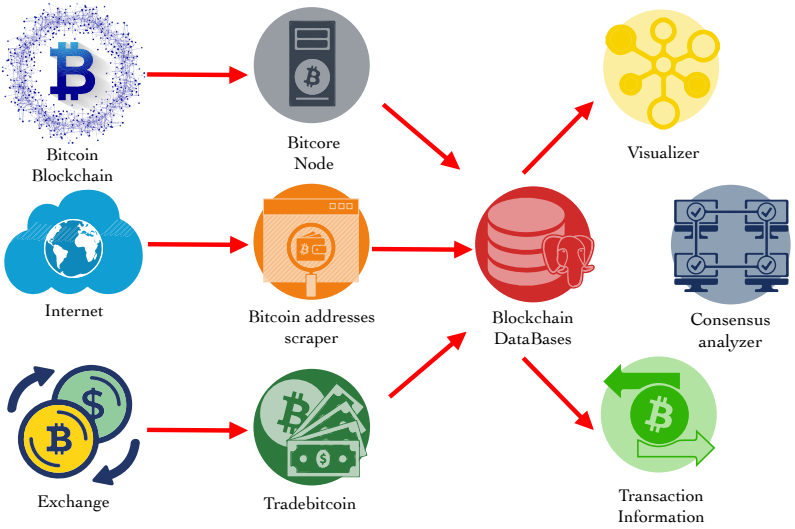

Figure 16: BlockchainVis Suite architecture.

and transactions. The BlockChainVis architecture is designed to accommodate a modular and expandable framework with the purpose to build complex applications for the forensic analysis of the Bitcoin blockchain. Figure 16 summarises the suite. We have three modules (Bitcore node, addresses scraper and TradeBitcoin) that retrieve different data from the web and store them in some DB. Then the last two modules use this data to analyse and visualize the information. The module TradeBitcoin, which analyse arbitrage possibility in the Bitcoin market, will be presented in chapter 6. Instead, the module Transaction information that studies bitcoin transaction type and scripts is described in chapter 4. Finally, consensus analyser is a module to evaluate the properties of consensus protocols. It is described in chapter 7. More details on the used technologies are in Appendix A.1.

In Figure 17 we focus on the Visualizer work flow. The Bitcore node gets the blockchain data from the network and inserts them into the database parsed for the visualization. In particular, BlockChainVis is a client-server Web application (see Figure 18). It consists of a back-end (server-side) and a front-end (client-

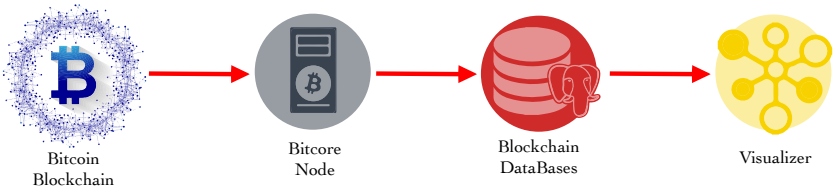

Figure 17: Visualizer architecture flow.

side). After some initial attempts, we discarded the idea to use a *block-explorer*[7] because of their current limitations (traffic volumes and omission of some information). Therefore, we opted for *Bitcore* as "full node"[8] Bitcoin client. The raw blockchain can be queried by using Insight API [Red11], and the result is presented to the user as a JavaScript Object Notation (JSON[9]) file, which is a simple text-document where the basic structure is a set of name-value pairs and an ordered list of values. For example, the Bitcore API call *getrawtransaction* having a hash as parameter, allows for receiving all the information about a transaction; for instance, its block number, all the inputs and the outputs, the number blocks following in the blockchain (i.e., *confirmations*).

The tool downloads the entire blockchain in the back-end as a first step. The second step consists in extracting the desired information from the blockchain to populate the database. The DB makes the retrieves of information from blockchain easier and faster. A DB also allows us to create our queries to search for specific information. As database we use *PostgreSQL* (see Appendix A.2 for more information on the DB structure). The PHP script that populates the DB is *getBlock.php*, which takes as input a range of blocks. Starting from the first block, by calling the API offered by BitCore, the script extracts all the information related to it (the result of the invocation is a JSON file) and encodes it in a PHP object to better handle it. Then, the *Coinbase* (the first transaction in a block, which generates new bitcoins as a reward) and all the other transactions are added to the database. At the end of the scan, the script generates a report

[7] Websites that allows for reviewing information about the blockchain by using dedicated Web services.

[8] Full nodes download every block in the blockchain, currently more than 200Gb of raw data.

[9] `http://www.json.org`.

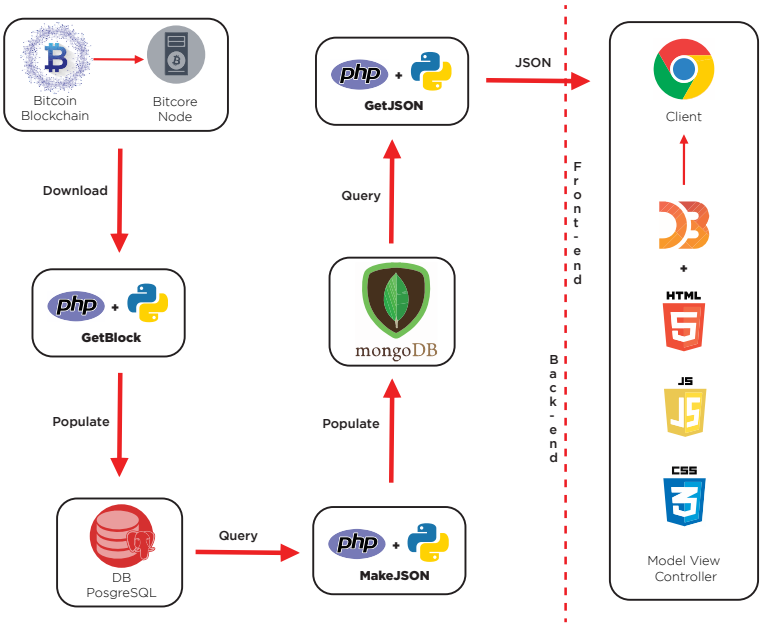

Figure 18: A summary of the architecture and technologies.

message and switches to the next block.

PHP and Python scripts are also used to operate on the PostgreSQL database to serve requests from the client application: the extracted information is then translated to JSON. In addition, we store these JSONs in the MongoDB database to use them as cache and do not recreate every time the same one and speed up the process. More information and details on the control panel to create and manage this JSONs are in Appendix A.3.

3.2 Visualizer

Big Data analytics examines large amounts of data to uncover hidden patterns, correlations and other insights. The blockchain can be considered as Big Data: by the end of February 2022, the blockchain contains over than 1 billion transactions, for more than 200Gb.

Being VA task-oriented [WT04], we have identified nine main tasks: *i)* find miners; *ii)* find transaction sources and understand how they are connected; *iii)* find the main addressees of transactions; *iv)* find the "richest" and "poorest" addresses; *v)* find the addresses with a break-even budget; *vi)* find bitcoin flows from an arbitrary address; *vii)* find bitcoin flows from a set of different addresses; *viii)* filter the blockchain on intervals of time or block identifiers; *ix)* filter the blockchain on specific transaction amounts of bitcoins, or on their number of involved addresses.

To reach such tasks, in the initial window it is possible to select among three different kinds of visualisation: *Single Transaction*, *Address Transactions*, and *Archipelago*. The first view allows for manually inserting the hash of one desired transaction, and then the tool shows the input addresses (in the following mentioned simply as "inputs") and the output addresses (in the following, "outputs") as a graph as in Figure 23(a). The second option is the dual of the former: it is possible to type in an address and the tool shows all the transactions that have such address as output, and all the inputs of these transactions. Hence, the first two options offer a more targeted view: the user already has some initial information.

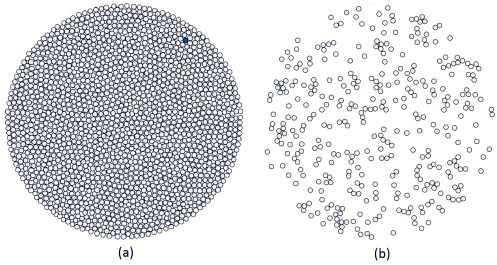

Figure 19: (a) The whole Bitcoin archipelago, and (b) by keeping only islands with 2-10 miners (min-max).

The Archipelago view is the third and most challenging one: it displays all the islands of the archipelago of Bitcoin transactions. An island is a connected component of a graph, where each couple of nodes is connected through a path, and each of the nodes is not connected to any other vertex of the super-graph. Using the DB described in Appendix A.2, a bipartite graph of addresses and

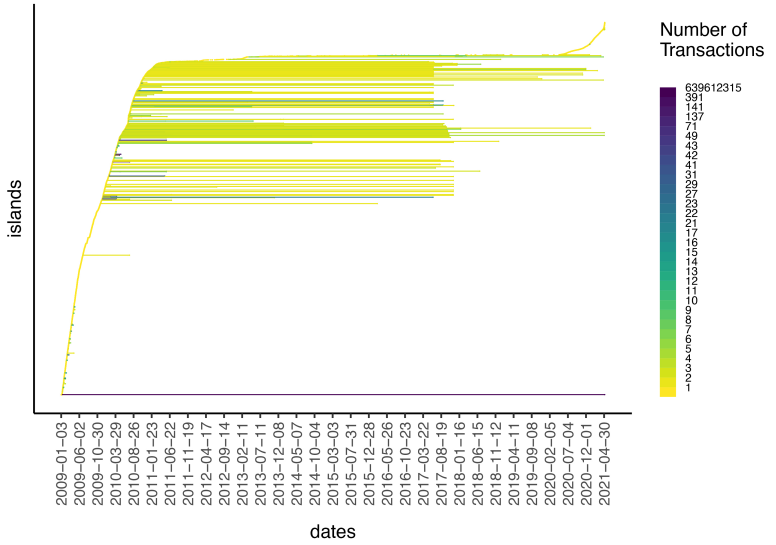

Figure 20: Duration of Bitcoin transaction island.

transactions is created using the transaction's inputs as edges. In this graph (archipelago), we found 38,838 distinct connected components (island). Figure 20 shows islands' distribution over times and their number of transactions. Every line represents a transaction connected component. Its length makes us understand the duration along time. The darkest colour represents a greater number of transactions. We can see that almost the totality of transactions is contained in only one island. It is also distributed along with all the blockchain history. The second island has just 391 transactions, but in a short period (from March 2010 to July 2010). Instead, more than 38,800 islands contain just one transaction. Notice that the yellow points between January and October 2009 are due to the coinbase created by "Satoshi Nakamoto" and never spent.

In Figure 19(a) we show such archipelago for the block interval 1-111,111, which is made by 1,700 islands.[10] As it can be seen from Figure 19(a), the number of islands is too large to be useful. For this reason we have created four

[10]For the sake of testing, at the moment only the first two years of transactions are available to the public, until 2011-02-28.

slide-bars, each one operating on a different data-filter:

- A block-interval filter, by date or by height position (from-to) in the blockchain. Only the transactions in such blocks are visualised in the archipelago.
- A filter on the number of transactions: only the islands with the specified minimum and maximum number of transactions are shown.
- A value-based filter: it specifies the minimum and maximum amount of bitcoins considering all the transactions of a single island (i.e., their sum).
- A filter on the number of miners: it specifies the minimum and maximum number of miners in each visualised island.

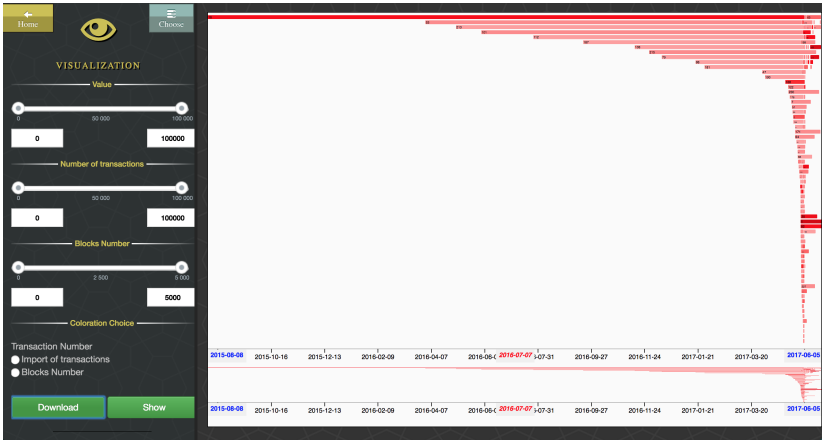

Figure 21: Time based visualisation.

In Figure 19(b) we lighten the visualisation w.r.t. Figure 19(a) by only showing islands with 2-10 miners. We obtain 354 islands (20% of the total): most of the islands only have one miner. We can visualise the Archipelago along time, as shown in Figure 21. In particular, instead of seeing the islands as nodes, we see them as time bars. The left position is the data of the oldest transaction on the island. On the contrary, the right one is the data of the most recent transaction of the island. This visualization is similar to the one in Figure 20.

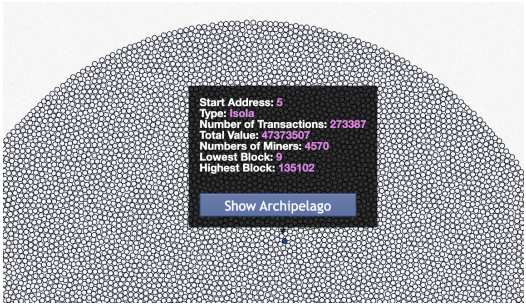

Figure 22: Summary of island statistics panel.

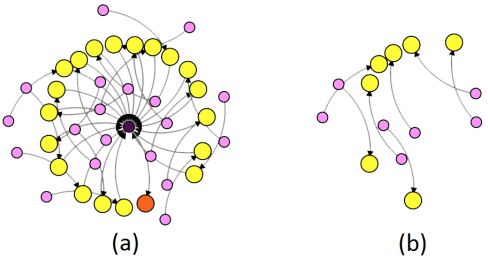

Figure 23: An example (a) of an island as visualised by BlockChainVis. Larger nodes are addresses, and smaller ones are transactions. The darker larger node has a higher (incoming) budget. In (b) we filter (a) by keeping only transactions in the first half of the block interval considered in (a).

By clicking on any island of the archipelago, a summary of its statistics pops-up (Figure 22). It is possible to enter into an island and visualise all its transactions. However, it is also possible to pre-filter the transactions before visualising them. The visualisation employs an oriented graph: a node can represent either a transaction or an address, and each transaction may have 1-n outgoing edges and 1-n incoming edges (see Section 2.1). The graph is bipartite: transactions can only be connected to addresses, and vice-versa. An example directly taken from BlockChainVis, is provided in Figure 23(a). Larger nodes are addresses: they are lighter if their budget is balanced (bitcoin inputs equal to outputs). The colour of address nodes gets darker as their budget moves from balance: the sink of all the transactions in Figure 23(a) receives 750Ƀ as input, without any output.

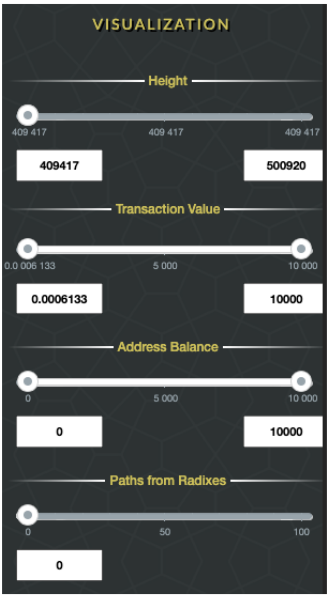

Figure 24: The three slide-bars of the filter panel.

Smaller nodes represent transactions, and their colour is darker if the amount of transferred bitcoins is larger.

As for the archipelago view, even the visualisation of a single island can contain too much information to be useful: three slide-bars can filter out undesired information (Figure 24):

- A block-interval filter by height position in the blockchain. Only transactions in such a block interval are visualised.
- A value-based filter: expressed as a min/max interval, it hides all the transaction whose value is outside this range.
- A balance-based filter: expressed as a min/max interval, it hides all the addresses whose balance (sum of the inputs minus sum of the outputs) is outside this range.

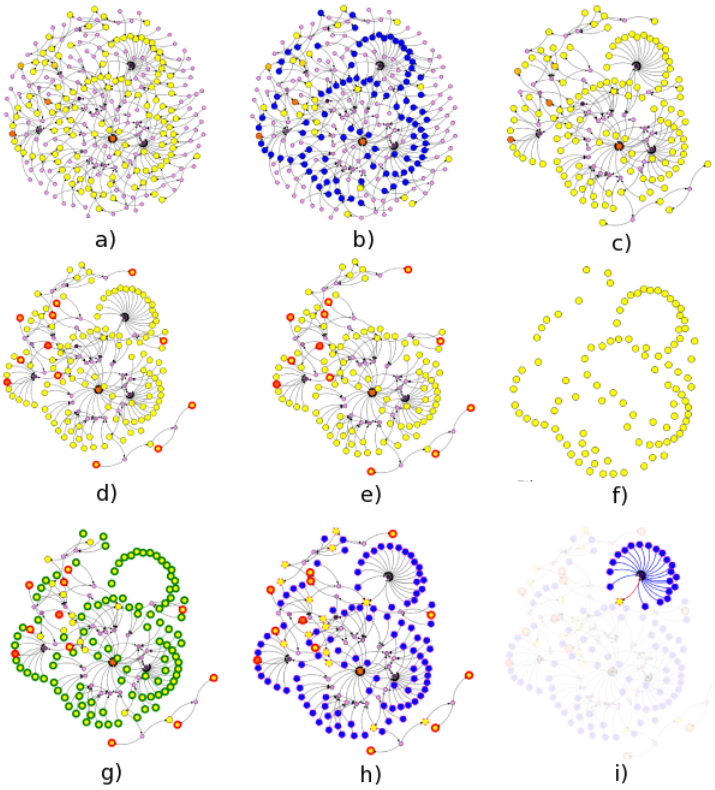

Figure 25: An example of filter application on (a) the initial graph: (b) by highlighting miners, (c) hiding coinbase transactions (rewarding the miners), (d) highlighting leaves, (e) applying a transaction-value interval, (f) showing only roots, (g) visualising paths (most of the two-colour nodes are the same roots in (f), the others are the leafs), (h) combines *b* and *f* together (darker nodes highlight the same miners in (b)), (i) focuses on a given transaction.

As an example, in Figure 23 we show (a) an island and (b) the same island with only the transactions in the first half of the block interval considered for (a). Besides the previous main filters for islands, BlockChainVis has some secondary filters to *i)* show only the roots of an island, *ii)* only the leaves, *iii)* hide the transactions with a fee, *iv)* collapse binary transactions, and *v)* collapse changes: with *iv)* we hide the nodes that represent the transactions with only one input and only one output, while with *v)* we hide the transactions that return a change to the same address. The effect of some of the implemented filters is shown in Figure 25.

Moreover, by clicking on an island in the archipelago view, a *tooltip* (or *infotip*) appears to show numerical information related to the selected island. The tooltip reports the number of transactions, the total amount of bitcoins transferred in all the considered transactions, the number of miners, the time interval between the first and the last transaction. A tooltip is also available for addresses and transactions. For a transaction the tooltip reports the hash, the timestamp, the transferred value, the fee for the miner, the number of inputs and outputs. For an address node, the information consists of its hash, its name (if any), the amount received and sent from this address, its balance, and the number of inputs and outputs. Moreover, by clicking on a transaction or an address, all the incoming and outgoing edges are automatically coloured in respectively blue and red, in order to understand the extent of their in/out-degree.

Finally, there is the possibility to select an address (or a group of them), and then to show all the paths departing from it and reaching the leafs of an island, immediately highlighting the bitcoin flows from these addresses. It is also possible to restrict only to paths with a desired number of hops (e.g., 2); this is accomplished by typing in the number of hops and then pushing the button *Show Paths*. The graph is visited through a *Breadth-first* search (*BFS*).

We want to stress that all the views presented in this subsection are in real-time, i.e. there is no delay in using them and their filter. This is possible since we pre-calculate all the information necessary to draw the graphs. Following the idea in [BBT18] we create a connected components DB (see Appendix A.2 for more details). This DB allows us to calculate the blockchain archipelago in a brief amount of time (around two days). This efficiency in the archipelago computation is due to a small data storage, i.e. we store only a compressed

version of the transactions ID and his output and addresses, according [BBT18]. We use the lower 63 bits of the transaction ID as a vertex ID for a transaction instead of 254 bits of the traditional transaction ID. Having this graph is very helpful in creating the views for the module. In fact, using this DB is very simple to get the flow (addresses transaction path) of a specific address we want to analyze in the visualizer module. We need around six hours to create a view with 100 counting both transactions and addresses, more than one day if we have 1000 among transactions and addresses.

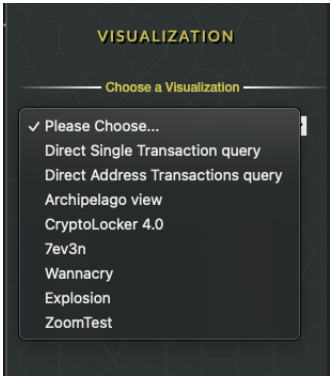

Figure 26: Visualizer drop-down menu.

3.3 Case Study

This section shows two real use from the blockchain. It is possible to find this view using the menu of the visualizer in Figure26. The first one shows how to filter transactions that connect two addresses, the second how to track Wannacry's flow.

3.3.1 Studying a pattern

In the first case, we select a pattern, which is interesting to be studied due to its huge amount of transactions. In particular, we show how easily we have managed to spot a set of $15,964$ transactions that correspond to only two flows

exchanging bitcoins between two addresses.

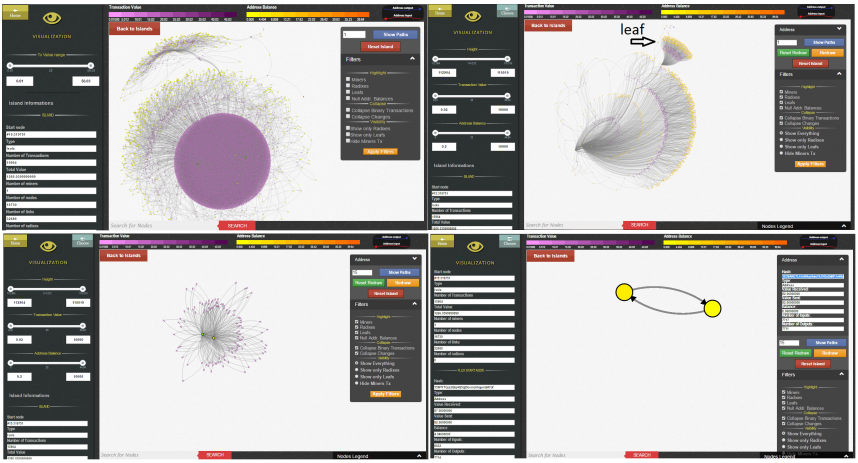

Figure 27: From the full visualisation of an island (top-left), to the only two addresses involved in all the transactions (bottom-right). First, all changes and binary transactions are collasped to discover a single leaf. Then, all the root-to-leaf paths are extracted.

In Figure 27 we start by considering all the information contained in an island (top-left); this image directly reports a screenshot of BlockChainVis. Indeed this screenshot is not very informative since it shows $15,964$ transactions, with a total value of $1,266$฿. However, by applying different filters, the situation becomes much clearer: the top-right screenshot is obtained from the first one by both collapsing binary transactions and changes (see Section 3.2), and by highlighting the miners (only one in this island), the roots, the leaves, and the addresses with a null balance. By doing this, we immediately see that the number of visualised transactions drops, and that the island contains only one leaf node. The next step consists in visualising only the paths that end to this leaf address: we select this node and we click on *Show Paths*. The result is represented in the bottom-left screenshot in Figure 27; since we have outputs and changes still collapsed, each path is aggregated as a single transaction. If we filter out also such transactions we obtain the last screenshot in Figure 27 (bottom-right). It is now very easy to see that all the involved transactions exchange money between

only two nodes.

3.3.2 CryptoTorLocker2015

CryptoTorLocker2015 is a ransomware that, once installed, will scan the system and lock all data files and shortcuts. What is specific about CryptoTorLocker2015 is not intelligibly by design. In fact, it generates a messy message that said: "Your important files strong encryption RSA-2048 produces on this computer:Photos,Videos,documents,usb disks etc.Here is a complete list of encrypted files,and you can personally verify this.CryptoTorLocker2015! which is allow to decrypt and return control to all your encrypted files.To get the key to decrypt files you have to pay 0.5 Bitcoin 100$ USD/EUR. Just after payment specify the Bitcoin Address.Our robot will check the Bitcoin ID and when the transaction will be completed, you'll receive activation,Purchasing Bitcoins,Here our Recommendations 1. Localbitcoins.com This is fantastic service,Coinbase.com Exchange,CoinJar =Based in Australia,We Wait In Our Wallet Your Transaction WE GIVE YOU DETAILS! Contact ME if you need help My Email = information@jupimail.com AFTER YOU MAKE PAYMENT BITCOIN YOUR COMPUTER AUTOMATIC DECRYPT PROCEDURE START! YOU MUST PAY Send 0.5 BTC To Bitcoin Address: 1KpP1YGGxPHKTLgET82JBngcsBuifp3noW".

It also displays a wallpaper unskillfully put together along with payment address and XOR encryption. Nathan Scott, a malware analyst from Florida, has discovered that it is possible to bypass the password to decrypt the files[11]. Our research visualises the money flows associated with the CryptoTorLocker2015 Bitcoin address, i.e. where to send the ransom in Figure 28). CryptoTorLocker2015 money flow. The first thing we notice is that the address received just one transaction of 0.5฿, the price of the ransom. In particular, using a path filter (Figure 29), we can see that the other five input transactions are less than 0.3฿. Two of them are also output transactions, where the CryptoTorLocker2015 address is used as change. Digging into these two transactions, we can see that one sent value to another address; the other, instead, received value from another address. The two addresses involved in these two transactions belong

[11] https://sensorstechforum.com/remove-cryptotorlocker2015-and-restore-the-encrypted-files/

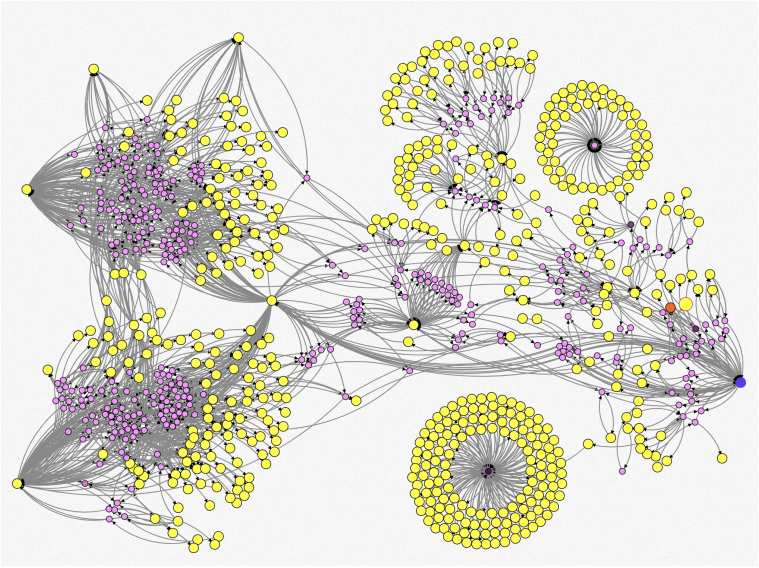

Figure 28: CryptoTorLocker2015 address (in purple) income.

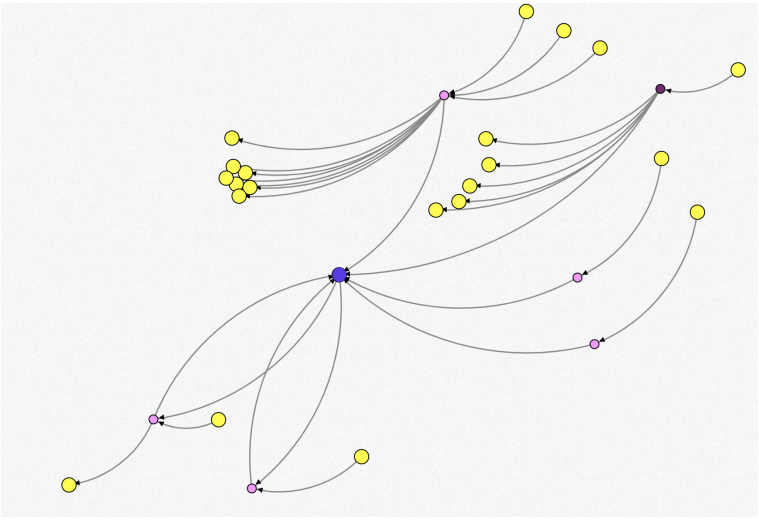

Figure 29: CryptoTorLocker2015 address after (in purple) path filter.

to SatoshiDice[12].SatoshiDICE is a "blockchain-based betting game" operating since 2012. In 2014, off chain session-based bets were also made available. Unlike traditional online gaming software, wagers with SatoshiDice can be sent without access to the website or running any client software. To play, a Bitcoin transaction is made to one of the static addresses operated by the service, each having differing payouts. The service determines if the wager wins or loses, sends a transaction in response with the payout to a winning bet, or returns a tiny fraction of the house's gain to a losing bet. As a result, the game spams the p2p network and blockchain with useless data. SatoshiDice forces players to pay a transaction fee on each result so the spam will flood the p2p relay network and the blockchain successfully[13]. On the left side of Figure 28 we can see all the traffic of SatoshiDice connected with CryptoTorLocker2015. In particular, we saw that the owner of CryptoTorLocker2015 spent all the ransom (more than 5฿) gambling on SatoshiDice. So he/she did not keep the money.

3.3.3 WannaCry

WannaCry (also known as *WannaCrypt*, *WanaCrypt0r 2.0*, *Wanna Decryptor*) is a ransomware computer worm that targets the *Microsoft Windows* operating systems family. It was initially identified on Friday, May 12th 2017, and it has infected over 200,000 machines in over 150 countries[14]. On the same day, a security researcher observed traffic from a copy of WannaCry he was testing led to an unregistered domain, halting the spread of the virus. He ended the virus by registering emphiuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com, a "kill switch" to control it[15].

WannaCry wants a $300 bitcoin ransom at the time of infection, which rises to 600 dollars after three days. Data will be permanently unrecoverable after seven days without payment. This ransomware encrypts practically every essential file type that a user might have on her computer, including png,.zip,.jpg,.docx, and.rtf. WannaCry takes advantage of a flaw in Microsoft's implementation of the *Server Message Block* (*SMB*) protocol to get control of the target and infect

[12] `https://satoshidice.com`.
[13] `https://en.bitcoinwiki.org/wiki/Satoshi_Dice`.
[14] `https://blog.kaspersky.com/wannacry-ransomware/16518/`.
[15] `http://tinyurl.com/k7ea9y2`.

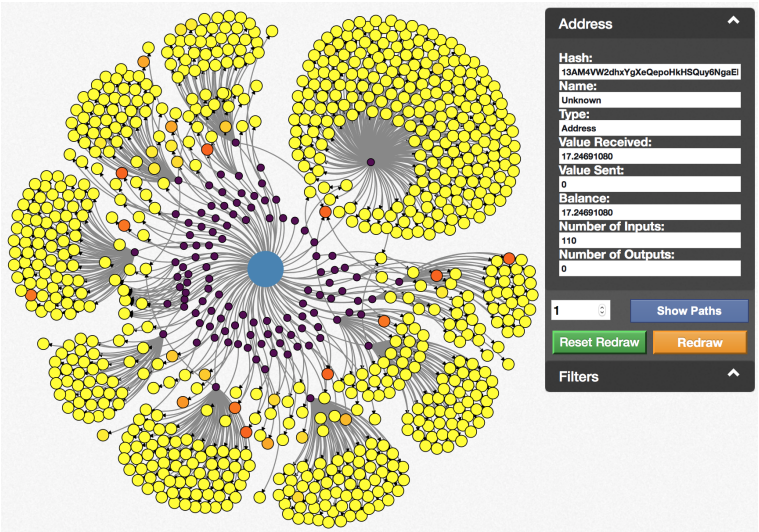

Figure 30: Wannacry address 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94 income.

it (as well as encrypt the data of the victim).

Our research visualises the money flows associated with the top three known WannCry Bitcoin addresses by incoming budget, where a victim is needed to send the ransom.[16] We show these flows in Figure 33, where we filter out all the blocks outside of the range 465960-466949, or all transactions mined between 2017-05-12 00:21:05 and 2017-05-18 09:42:34. Furthermore, we filter out all transaction inputs to focus solely on ultimate value destinations. The transactions that concern such three addresses (at the centre of each image) are highlighted in Figure 30, Figure 31, and Figure 32, i.e., all transactions for which one of the outputs is one of these three addresses. All three incriminated addresses are highlighted in the same image in Figure 33(bigger nodes). Smaller dark nodes represent transactions in such images, which were collected using BlockChainVis (see Section 3.1). All larger and lighter nodes represent the involved addresses of such transactions.

During the first week, $17.247 + 16.037 + 11.518 = 44.802$Ƀwere sent to

[16]Addresses: `https://github.com/GregorSpagnolo/WannaCrypt`.

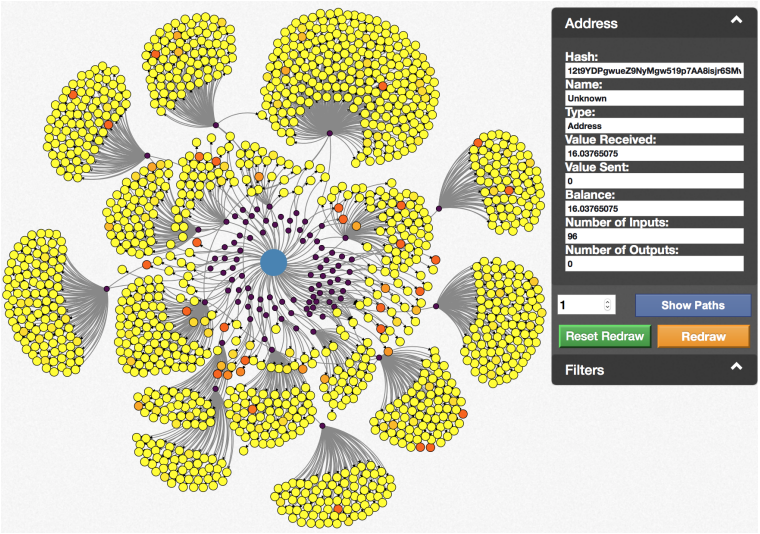

Figure 31: Wannacry address 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw income.

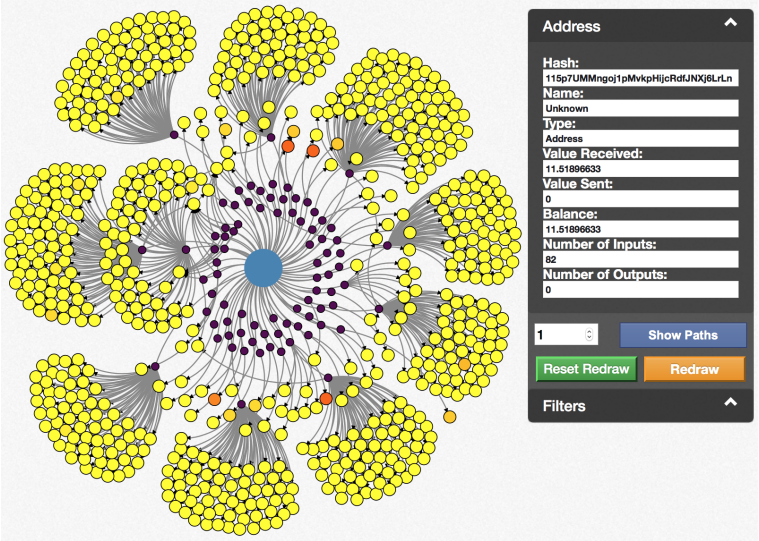

Figure 32: Wannacry address 115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn income.

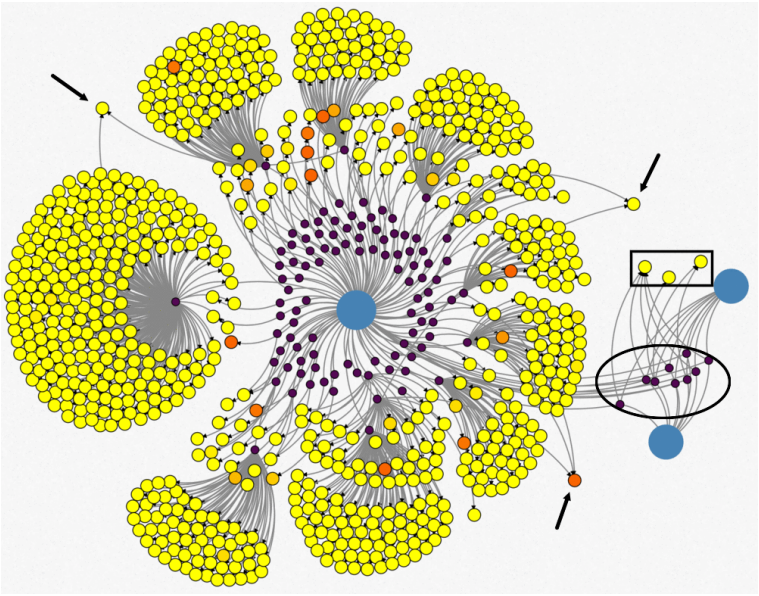

Figure 33: Highlighting the three addresses in Figure 30.

these three addresses. Because of the extreme volatility of Bitcoin in 2017, we do not provide amounts in dollars or euros. The total number of inputs that all three are concerned with is $110+96+82=288$.

We find some significant features in the flows given in Figure 33. First, the majority of the transactions have only one output to one of the three incriminated addresses (no payer splits the ransom among them), where they transfer a number of bitcoins that is very close to 300$ or 600$ at the time (i.e., precisely what WannaCry requested): between 0.15Ƀand 0.18Ƀ(respectively, 0.3Ƀ-0.36Ƀ). Only one of the 288 transactions (for the address in Figure 31) paid a more considerable ransom: 1.99Ƀ, which corresponds to 11 infected machines, according to the analysis.

Instead, looking at the entire history up until two weeks after the diffusion (May 26th), we can see that *i* no transaction is dated before May 12th, and *ii* there is still no outbound transaction at the moment (also visible in Figure 33 for the first week): the ransom funds are still unspent. Furthermore, the number of input transactions grows from 288 to 333 in the second week, with only new 45 transactions, and the total balance of the three nodes increases from 44.802Ƀto 50.14Ƀ: 89% of the ransoms has been collected during the first week. Considering these 333 transactions, 177 transfer $[0.1,0.2)$Ƀ, while 41 move $[0.3,0.4)$Ƀ as ransom. The second group is concentrated in the last three days of the study period, implying that victims who paid after three days paid twice the ransom, exactly as WannaCry instructed (unless those few cases where they paid for two infected machines). Furthermore, 85 of these 333 transactions are less than 0.01Ƀ($\sim$ 2$ at the time). As a result, we estimate that no more than $333-85=248$ victims paid the required ransom in the first two weeks, focusing on the three addresses identified. We hypothesise that a large number of low-value transactions will accumulate payment mistakes or first tries, hidden messages (see below), or simply the desire to be included in such a list.

Some transactions in Figure 33 have a significant number of outputs, visually comparable to "flowers with many petals"; for example, in Figure 33, there are 12 of such many-output transactions, the largest of which has 246 outputs. Most of these outputs receive fewer bitcoins for all of the 12 "flowers," while few addresses receive more than the ransom. If we investigate the largest of them, it moves a total amount of 147.83Ƀ, but 144 addresses (out of 246) re-

ceive less than 0.1฿. Six of these receivers are owned by *Poloniex.com*[17], a cryptocurrency exchange and lending service provider situated in the United States. Some other addresses refer to different betting, investing, or wallet services, e.g. *Cubits.com*[18].

A second characteristic is the presence of three addresses (pointed by arrows in Figure 33) that are a common output of two/three different transactions used to pay a ransom. One of them belongs to Poloniex.com. The second address is linked only to the two transactions used to pay two ransoms, and has a low unspent budget (0.45฿). The third address has been involved in $1,073$ transactions and received more than 169฿ (with a current null balance): the last transaction is towards an online gambling platform.

A third feature observable in Figure 33 is a set of 9 transactions (grouped by an ellipse) that moved some bitcoin to all the three WannaCry addresses. However, such sums are less than 1$ and do not correlate to the amount owed for a ransom. Seven of these transactions (out of 9) include a fourth output (one additionally has a fifth); these addresses are inside a rectangle in Figure 33. These are messages sent to WannaCry addresses in order to gain high awareness. Although only five have a single input address, all 5 addresses begin with the string "1DoDiK". Another transaction has four input addresses, each containing a portion of an insult, such as "You are a ****." A transaction has two inputs whose sub-strings advertise a different crypto-currency: "Use ****".

Finally, the number of victims still paying the ransom drastically reduced after May 16th: e.g., only two transactions have an incriminated address as output on May 25th.

3.4 Bitcoin Addresses Scraper

The *Bitcoin addresses scraper*[19] crawls the Web for Bitcoin addressees to be associated with real users, or to Web URL. The aim is to fully de-anonymise addresses where possible. Figure 34 shows the Scraper workflow. The tool gets addresses info from the network and inserts them into the Bitcoin database. The

[17] `https://poloniex.com`.
[18] `https://cubits.com`.
[19] `http://scraping.dmi.unipg.it/ci`.

Figure 34: Addresses Scraper architecture flow.

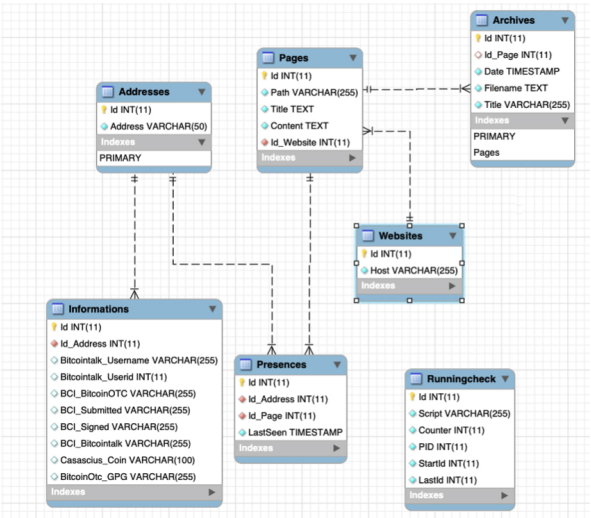

Figure 35: Addresses Scraper DB structure.

DB structure is based on seven tables (Figure35):

- Table *Addresses* has two columns: *id* is the incremental number of the rows (it is in all tables), *address*, instead contains the address name.
- Table *Information* has all the address info related to different forums. Here the column name represents the address information connected to the specific websites. *id_Address* is the link to the id in table addresses.
- Table *Pages* contains the info of scraped web pages: *Path* has the path of the saved web pages; *Title* and *Content* are the website title and his content; *id_Website* is the link to the table website.

- Table *Websites* contains *Host* that is the URL of the sites.
- Table *Presences* has a row for all addresses found on websites: *id_Address* is the link to the id in table addresses, *id_Page* is the link to the table Pages, *LastSeen* has the timestamp of the scraping date.
- Table *Archives* has the info of the saved pages: *id_Page* is the link to the table Pages; *Date* has the timestamp of the save date; *Filename* is the saved file name.

We use a set of scrapers [SMZ14a] that crawl specific data form Web sites connected to the Bitcoin world:

- user-names on Bitcoin Talk[20] forum and Bitcoin-OTC[21] marketplace;
- physical coins created by Casascius (https://www.casascius.com) along with their Bitcoin value and status (opened, untouched);
- known scammers, by automatically identifying users that have significant negative feedback on the Bitcoin-OTC and Bitcoin Talk trust system.
- name tags on blockchain.info[22], e.g., *"Wannacry ransomware 1"*.

The tool helps users build lists of gambling addresses, online wallet addresses, mining pool addresses, and addresses which were subject to seizure by law enforcement authorities. All these addresses are entered in the database and they are used to de-anonymise further addresses.

This module allows us to search Bitcoin addresses from the web, using the power of the Bitcoin scraper module. It is possible inserting a URL in the web panel. Then the module will search for all addresses in the chosen URL and all the links present on the web page.

Figure 36 shows the integration between the Scraper and the Visualazier. Clicking the button *info* in the visualizer is possible see the information of the scraped addresses.

[20] `https://bitcointalk.org/`.
[21] `https://bitcoin-otc.com/`.
[22] `https://blockchain.info/tags`.

Figure 36: Visualizer info generated by addresses scraper.

3.5 Conclusions

In this chapter, we have presented BlockChainVis, a suite of tools for flows of bitcoins. The main goal is to, given a specific task, filter out not interesting information, in order to better analyse: the Bitcoin blockchain, cluster addresses, identify mixing services, visualise information about transactions, and allow for using scripting languages.

We want to stress that all the views presented in this subsection are in real-time, i.e. there is no delay in using them and their filter. This is possible since we pre-calculate all the information necessary to draw the graphs. Following the idea in [BBT18] we create a connected components DB (see Appendix A.2 for more details). This DB allows us to calculate the blockchain archipelago in a brief amount of time (around two days). This efficiency in the archipelago computation is due to a small data storage, i.e. we store only a compressed version of the transactions ID and his output and addresses, according [BBT18]. We use the lower 63 bits of the transaction ID as a vertex ID for a transaction instead of 254 bits of the traditional transaction ID. Having this graph is very helpful in creating the views for the module. In fact, using this DB is very simple to get the flow (addresses transaction path) of a specific address we want to analyze in the visualizer module. We need around six hours to create a view with 100 counting both transactions and addresses, more than one day if we have 1000 among transactions and addresses.

Chapter 4

Non-standard Transactions

In this chapter we investigate *standard* and *non-standard* transactions in the Bitcoin blockchain. Transactions are standard if they pass the controls implemented in the reference Bitcoin-node software, i.e., *Bitcoin Core*[1]. Our interest is mainly focused on non-standard ones, of which we provide a classification in nine different types, extending some previous analysis for Bitcoin[2] [BMS18b] and in a manner similar to what done for Ethereum [Bis+19a].

The main motivation behind this chapter is to provide an updated and comprehensive snapshot of standard and non-standard transactions from Bitcoin origins until today. In particular, the goal is to understand what and how partial compliance to the Bitcoin protocol or flaws has been exploited so far, accidentally or not. Hence, we can evaluate the errors and misuses accepted by some of the miners in the network. The main result is that only the 0,02% of transactions is non-standard (2009 - November 2018). In addition, the study presented in this chapter addresses further general questions. For example, there is no particular miner pool that validates only some specific classes of non-standard transactions: all pools that deviate from the standard behavior accept these classes. We also saw that only 2,615 bitcoins (out of more then 17 million in circulation) were definitely "burned" (i.e. made no longer spendable) due to non-standard transactions.

[1] `https://bitcoin.org/`.
[2] `https://tinyurl.com/bytestxsurvey`.

The chapter is organized as follows: Section 4.1 shows the non-standard transactions that we found in the blockchain, and related statistics; Section 4.2 shows the statistics of standard and non-standard Bitcoin transactions nested in *P2SH* transactions, focusing on non-standard ones reported in the literature; Section 4.3 classifies OP_RETURN transactions by their byte size, and it also presents related statistics. Finally, Section 4.6 draws the final conclusions and proposes ideas about future work.

4.1 An Analysis of Bitcoin transactions

In this section we describe standard and non-standard transactions in the Bitcoin blockchain, by reporting statistics on their number and frequency with the purpose to have a clear view on their "popularity" and acceptance. To accomplish such an analysis we take advantage of a Bitcoin Core node, which we used to fill a PostgreSQL Database[3] in which we have stored all the blockchain blocks up to number 550,000: until November the 14th 2018. Such a tool is part of the BlockchainVis Suite [BMS18a].

4.1.1 Standard Transactions

Considering the first 550 000 blocks in the blockchain, there are 356 588 805 transactions that generate a total of 968 098 854 outputs, of which 910 274 680 have been spent (94,03%). These include 967 874 499 standard transaction outputs (99,98% of the total number of outputs). Hence, non-standard transaction outputs are only the 0,02% of the total.

In Figure 37 we show the distribution of standard transactions. As introduced before, the most common class is represented by P2PKH transactions, since they are the default ones in the Bitcoin client. The P2SH scheme is the second mostly frequently used class of transactions, with almost 150 million outputs. Interestingly, the number of P2SH and OP_RETURN transactions has considerably increased if compared to the data from early 2018 [BMS18b]. In Figure 37 we see that the most used *M-of-N* multi-signature class corresponds to the scheme *1-3*, with 58% of all the multi-signature transactions. The second

[3]PostGreSQL: `https://www.postgresql.org`.

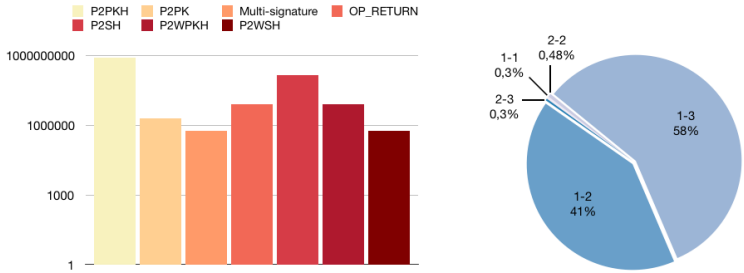

Figure 37: Distribution (number of occurrences in blockchain) of standard transactions (left) and distribution of Multi-signature transactions (right).

most used scheme is *1-2*, with 41%. We found also: 38 repetitions of *3-3*, 3 repetitions of *0-1*, only one 1 transaction in the form *3-5*, 1 in *1-9*, and, finally, 1 in the *9-9* form.

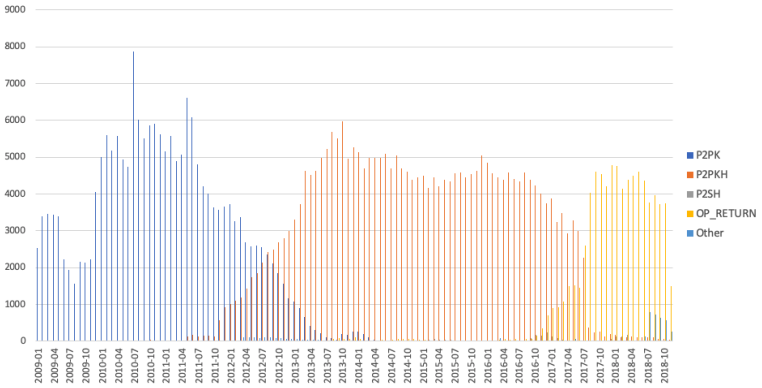

Figure 38: Evolution over time of the output type used in coinbase transactions.

Figure 37 shows the distribution over time of the output type used in coinbase transactions. P2PK was the most used until July 2012, then they started use the P2PKH (see Figure 38). Since 2017 the most used output in a coinbase transaction is the OP_RETURN, which is a provably unspendable (see Section 2.1.5). This coinbase transactions have some P2PKH outputs in order to pay the miners and some OP_RETURN outputs, which do not carry bitcoin (the value of

the outputs is 0 ฿). These OP_RETURN outputs are related to Segregated Witness[4] (SegWit) implemented soft fork: They are linked to the Merkle root of the witness tree. The SegWit needs an extended blockheader, but the blockheader cannot be extended without a hardfork, so segwit blocks commit to this witness tree by including the root in an OP_RETURN in the coinbase transaction[5].

4.1.2 Non-standard Transactions in blockchain

Transactions are validated through *isStandard()* and *isStandardTx()* functions in the Bitcoin Core reference implementation. In case they do not pass such tests, they are simply discarded. However, some transactions that deviate from the standard enforced by Bitcoin Core can be mined as well: these transactions can be issued in the blockchain thanks to miners that relax these checks enforced by such control functions, as for example Eligius[6]. Non-standard transactions use more complex script forms, represent challenges, or just result from bugs. Their singularity comes from non-standard inputs or outputs.

Correctly validating non-standard transactions can make the creation of future transactions harder for two different reasons:

1. Some scripts might cause harm to the network.

2. Some scripts might make future upgrades harder.

Concerning the first reason, the non-standard transaction check was first implemented by Nakamoto before P2SH existed, so it could not be so easily circumvented. This gave developers time to better analyze the script language and fix problems with the remaining opcodes: e.g., in some cases[7] it was possible to create a transaction that took five hours to be verified. However, even if the scripting language is perfectly safe, each script has to be stored by every full node until it is spent as part of the UTXO database. Since a locking script is limited to 10,000 bytes, this means that an attacker can add up to 10 KBytes to the UTXO set for every output he creates, potentially quickly adding enough data to degrade performance enough that the rate of stale blocks (orphan blocks) mined

[4] https://en.bitcoin.it/wiki/Segregated_Witness.
[5] https://tinyurl.com/RETURNcoinbase
[6] https://btc.com/stats/pool/Eligius.
[7] https://tinyurl.com/5hours2verify.

increases, which would reduce miner profits and encourage them to centralize further to recover that lost revenue.

Concerning the second reason, there are some opcodes the network does not want people to use. These are opcodes that might be redefined in the future, e.g. the OP_NOPX opcodes that have been used for soft forks in the past[8]. Lately, Bitcoin Core 0.16.1 quit the use of OP_CODESEPARATOR in non-segwit in preparation for another potential soft fork that will reduce some lingering problems with expensive verification. In those cases, standard transactions forbid both the scriptPubKey and the redeemScript (P2SH) versions (and, when applicable, the segwit P2WSH version), thus the easy circumvention is not possible in that case.[9]

One of the reasons to include non-standard transaction in the blockchain could be that miners have a long-term investment in the health of the Bitcoin Network. If Bitcoin collapses, then their expensive ASICs are worthless. Miners particularly need bitcoins to remain valuable over the long term because their hardware produces bitcoins over time. If nobody includes transactions in blocks, then bitcoins would be useless and therefore worthless. That would impact on miners long-term investment. If this ever became a problem, transactions would just wind up with higher fees to encourage miners to include them. Right now, enough miners include a transaction with a very small fee and there is no reason in paying more[10].

We searched in the blockchain for these particular transactions and we obtained nine patterns of non-standard transactions. We now describe them by also highlighting the interval of years in which they were confirmed in the blockchain.

Pay to Public Key Hash 0 [2011]: It corresponds to a distortion of P2PKH, with the difference that instead of the hash of a pubkey, there is a 0 value. The locking script is in the form "OP_DUP OP_HASH160 0 OP_EQUALVERIFY OP_CHECKSIG". These transactions are unspendable because, as P2PKH ones, in order to verify them a miner needs *i)* the pubkey corresponding to the hash in the locking script, and *ii)* the private key to generate the corresponding signature.

[8] OP_NOP1 became OP_CHECKLOCKTIMEVERIFY and OP_NOP2 became OP_CHECKSEQUENCEVERIFY

[9] https://tinyurl.com/non-standard.

[10] https://tinyurl.com/minerNS.

However, we know that HASH160 returns a 20 byte long hash: therefore, no key passed to the hash function can return 0.[11] A Bitcointalk thread[12] indicates that this deviation was mainly performed by MtGox. The outputs represent a value of 2609.36304319 BTC, around 8,000$ at that time (around 20 million dollars nowadays).

P2PKH NOP [2011, 2014]: This transaction is identical to P2PKH with the only difference that a NOP (an operation that does nothing) is in the locking script: "OP_DUP OP_HASH160 <HASHPUBKEY> OP_EQUALVERIFY OP_CHECKSIG OP_NOP". This transaction was probably used to test the OP_NOP operator. It can be unlocked by a script identical to that of P2PKH.[13]

OnlyHash [2011-2014]: The *OnlyHash* transactions are the most numerous non-standard class in the blockchain. These transactions contain a hash in the locking script, which is usually the hash of a file for using the blockchain as a ledger to register documents. Therefore, the security and resilience of the blockchain system can be used by applications such as digital-note services, stock exchange certificates, and smart contracts. The blocking script corresponds to "<HASH-OF-SOMETHING>". These transactions were used before the introduction of the OP_RETURN, which can be used for the same purpose.[14]

P2Pool Bug [2012]: These transactions are due to a bug[15] in the P2Pool[16] mining tool between February 2nd, 2012 and April 1st, 2012. Instead of a plain P2PKH locking script, the following script was added: "OP_IFDUP OP_IF OP_2SWAP OP_VERIFY OP_2OVER OP_DEPTH". This script does not make any sense and it is not even valid as the OP_IF is not closed by a corresponding OP_ENDIF, hence it is consequently unlockable.[17]

OP_CHECKLOCKTIMEVEIRFY OP_DROP (CLTV) [2012]: In this case the locking script is in the form: "<DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP".

[11] An example is in the first output of the transaction in `https://blockchain.info/it/tx-index/1793329/1`.

[12] `https://bitcointalk.org/index.php?topic=50206.0`.

[13] An example is in output 2 in `https://blockchain.info/it/tx-index/629343/2`. The unlocking script is in the input 2 in `https://blockchain.info/it/tx-index/781227`.

[14] For instance output 0 in `https://blockchain.info/it/tx-index/2557393/0`.

[15] `https://bitcointalk.org/index.php?topic=140097.5;imode`.

[16] `http://p2pool.in/`.

[17] See output 1 of `https://blockchain.info/it/tx$-$index/2915138/1`.

The OP_CHECKLOCKTIMEVEIRFY operator makes the transaction invalid if the element at the top of the stack is greater than the *nLockTime*[18] field of a transaction. In practice, using OP_CHECKLOCKTIMEVEIRFY it is possible to make funds provably un-spendable until a certain point in the future, i.e. it is a way to freeze funds up to certain future day.[19] Therefore, by checking the <DATA> element against the rules above, we can verify the transaction by using an unlocking script that inserts TRUE into the stack.[20]

OP_MIN OP_EQUAL [2012]: These transactions are related to a script as "OP_MIN 3 OP_EQUAL", which simply needs two numbers corresponding to the equation $x \leq y \wedge x = 3$ to be verified. Hence, to unlock it is possible to prepare an unlocking script as "3 4". We can see this transaction as a proof of how an equation can be used to generate a Bitcoin transaction. This means that anyone can easily unlock such a transaction without any private key.[21]

Pay to Hash (P2H) [2012-2015]: These transactions are a simplification of plain P2SH ones; we have a blocking script identical to P2SH, with the difference that the internal hash does not refer to a redeem script, but it is the hash of an hexadecimal string: "OP_HASH160 <HASH160OFSOMETHING> OP_EQUALVERIFY". There are two variants, where only the type of hashing operator changes: one corresponds to HASH256, while the other one to SHA256. The two blocking scripts are "OP_HASH256 <HASH256OFSOMETHING> OP_EQUAL" and "OP_SHA256<SHA256OFSOMETHING> OP_EQUAL". We can consider these transactions as "contest" in the network to find the correct value of the hash in the transactions.

A P2H cannot be considered a *Hash Time-locked Contract* (HTLC). A HTLC is essentially a type of payment in which two people agree to a financial arrangement where one party will pay the other party a certain amount of cryptocurrency. However, the receiving party only has a certain amount of time to accept the payment, otherwise value is returned to the sender. Instead a P2H transaction is different from a HTLC because it generates a payment that can be accepted

[18] https://en.bitcoin.it/wiki/NLockTime.

[19] https://tinyurl.com/freezefunds.

[20] See output 1 in https://blockchain.info/it/tx-index/3000496/1, and input 1 in https://blockchain.info/it/tx-index/3000536.

[21] See output 1 https://blockchain.info/it/tx-index/3118220/1, and the verification in input 1 in https://blockchain.info/it/tx-index/3126754/0 .

by the receiver without any time-constraint.

These outputs can only be spent by providing data that once hashed (called *hashlock*) by a cryptographic function is equal to a given hash.[22]

The verification step is simple: the hexadecimal string of the hash in the blocking script is enough to spend the transaction.[23]

UnLocked (UL) [2015]: This transaction has an empty locking script: it can be unlocked by simply having TRUE as unlocking script. Almost all of these transactions carry an amount of 0 BTC, i.e., they are valueless transactions. Such transactions can be used as a way to donate funds to miners in addition to transaction fees: any miner who mines such a transaction can also include[24] an additional one after sending funds to an address they control.[25]

OP_RETURN ERROR [2016-2017]: These transactions are identical to OP_RETURN, with the difference that there is an error in the script code. The code asks to push onto the stack a number of opcode higher than what is really in the code itself. For example an OP_RETURN script could ask to insert in the stack the next 40 bytes in the code, but if there are only 28 bytes, the execution fails. This transaction is apparently due to a programming error. The locking script is: "OP_RETURN ERROR" (an error is returned).[26]

OP_2 OP_3 ERROR [2017-2018]: These transactions are similar to OP_RETURN ERROR, but they have no OP_RETURN in the script code. As the OP_RETURN ERROR, their code asks to push onto the stack a number of bytes higher than what is really in the code itself. Probably these transactions, like the OP_RETURN ERROR, are related to an implementation error. This is the locking script: "OP_2 OP_3 ERROR" (an error is returned).[27]

[22] `https://tinyurl.com/NSoutput`.

[23] An example is in the output 0 in `https://blockchain.info/it/tx-index/12864193/0`, and how it is verified in the input 0 of the transaction in `https://blockchain.info/it/tx-index/12874719`.

[24] `https://en.bitcoin.it/wiki/Script#Anyone-Can-Spend_Outputs`.

[25] An example is in output 0 in `https://blockchain.info/it/tx-index/56730867/0`.

[26] An example of it is in output 1 in `https://blockchain.info/it/tx-index/134023577/1`.

[27] An example is represented in output 2 in `https://tinyurl.com/txopreturn`.

Statistics for non-standard transactions

Non-standard transactions are 224 355 (0,02%), even if the majority of them, i.e., 219 174, are unlocked transactions with a 0 BTC value (all of them in year 2015). This means that they are transactions without blocking scripts, and they do not carry any value: in practice they are "fake" transactions. Thus, if we do not consider such transactions, "real" non-standard ones are only 5 181: 0,000 5% of the total number in the blockchain.

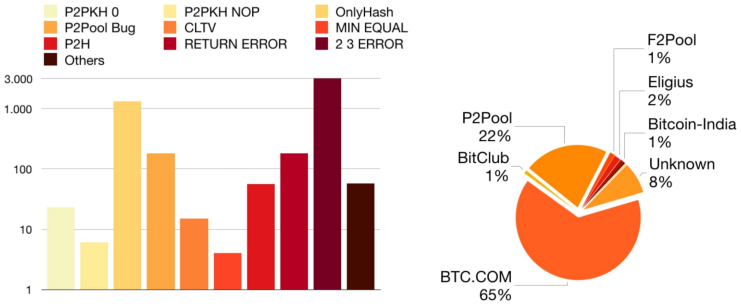

Figure 39: Distribution of non-standard transactions (left) and distribution of their miners (right).

In Figure 39 we show the distribution of non-standard transactions. Without considering unlocked ones, OP_2 OP_3 ERROR is the most common class, with more than three thousand transactions. The second class is the OnlyHash, with almost one thousand outputs, while the remaining ones have few outputs. In Figure 39 we show the percentage of non-standard transactions associated with each miner. In order to identify the miner of a transaction we looked for the block of this transaction. Then we took the coinbase transaction of the block, in this particular transaction there is a field called just coinbase where the miners put their id. We classified the miners according to these tags.[28]

In Figure 40 we associate the percentage of non-standard transactions: we found that year 2018 has more than 3,200 occurrences of this kind of transactions, all of type OP_2 OP_3 ERROR. Overall, these non-standard transactions contain almost 2,615 bitcoins that are no longer spendable.

[28] `https://tinyurl.com/labelpools`.

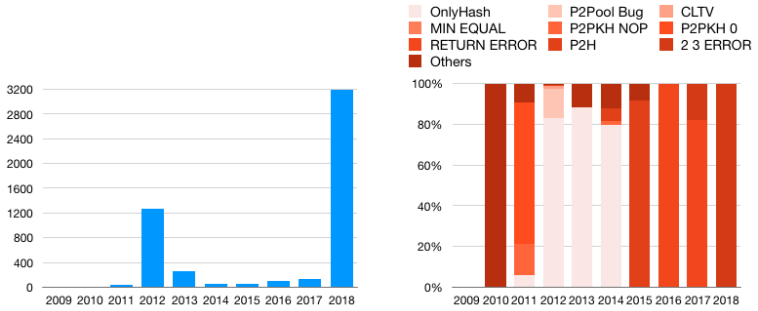

Figure 40: Distribution of non-standard transactions over time (left) and percentage distribution of non-standard transactions type over time (right).

4.2 An analysis of Pay to Script Hash

As we introduced before, the P2SH transactions contain the hash of a script (called *redeem script*) in their locking script.
In the blockchain there are 149 410 668 P2SH transactions of which 140 620 401 are spent (94,12%). Hence, we decided to analyse the content of the redeem script inside the unlocking script of the spent P2SH transactions. In the following of this section we show the results we obtained.

4.2.1 Standard Transactions in P2SH

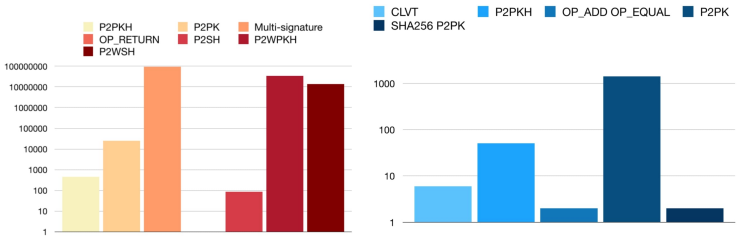

Figure 41: Distribution of standard (left) and CLVT transactions inside P2SH (right).

Like in the blockchain analysis, also inside P2SH transactions the majority of

the transactions is standard. In fact there are 140 509 279 standard transactions, which are 99,92% of the total. In Figure 41 we show the distribution of standard transactions inside P2SH ones. The most frequent class of transactions is not the P2PKH (only 447), as in Section 4.1, but it is the multi-signature one, with more than 90 million occurrences (65,7%). The second one is the P2WKH with almost 35 million transactions (24,6%).

4.2.2 Non-standard Transactions in P2SH

In the *redeem scripts* of P2SH, we found 111 122 non standard transactions (0,08% of the total); this amount is four times more than what we obtained when we "simply" analyzed the blockchain (0,02%). We found several new classes of non-standard transactions, which are different from previous ones.

OP_CHECKLOCKTIMEVERIFY OP_DROP (CLVT)

We found five different types of transactions that take advantage of the CLVT operator, that, as we have already seen in Section 2.1.5, makes transaction provably unspendable until a certain date. Essentially, it allows users to create a Bitcoin transaction of which the outputs are spendable only at some point in the future. CLTV is necessary for properly functional payment channels (e.g. lightning network). These channels are effectively a series of "off-chain" transactions, that benefit from all the security of typical on-chain transactions, and with some added benefits.[29]

One of them is equal to a group of transactions already present in the blockchain: "<DATA> OP_CHECKLOCKTIMEVERIFY OP_DROP".

The second is a P2PK locked with the CLVT operator, and this is the locking script: "<DATA> OP_CHECKLOCKTIMEVERIFY OP_DROP <PUBLIC KEY A> OP_CHECKSIG". This script means it is not possible to use the signature (related to the public key) to spend this transaction before the date in the script.

The third one is a P2PKH locked with a CLVT operator, this is the script: "<DATA> OP_CHECKLOCKTIMEVERIFY OP_DROP OP_DUP OP_HASH160 < PUBLIC KEY A HASH> OP_ EQUAL OP_CHECKSIG". Then we have the class "<DATA>

[29] https://tinyurl.com/CLTVboost.

OP_CHECKLOCKTIMEVERIFY OP_DROP 1 OP_ADD 2 OP_EQUAL", which simply needs the number corresponding to the equation $x+1=2 \wedge x=1$: it consequently waits until the date has expired.

The last one is a P2PKH variant that also needs a further hash to be unlocked; the locking script is "<DATA> OP_CHECKLOCKTIMEVERIFY OP_DROP OP_SHA256 <SHA256OFSOMETHING> OP_DUP OP_HASH160 <PUBLIC KEY A HASH> OP_EQUAL OP_CHECKSIG". As we can see in Figure 41, the most frequently type adopted in transaction is the P2PK one, with almost 1 500 outputs.

OP_DROP

This transaction allows for storing some data in blockchain without making the transaction unspendable. In fact, all these transaction start with <DATA> OP_DROP where <DATA> is what we want to put in blockchain and OP_DROP is the operator that removes the data from the stack in order to make the execution same as of a standard transaction.

We found four different types of transactions that use OP_DROP. The first type does not need anything to be unlocked, that is the script is: "<DATA> OP_DROP 1". Then there is the $2-2$ multi-signature type "<DATA> OP_DROP 2 <PUBLIC KEY A> <PUBLIC KEY B> 2 OP_CHECKMULTISIG", which needs the two signatures to be unlocked, like the normal $2-2$ multi-signature. The third one is a P2PKH with OP_DROP, identified by the "<DATA> OP_DROP OP_DUP OP_HASH160 <PUBLIC KEY A HASH> OP_EQUAL OP_CHECKSIG". Also this one needs only the signature, as for P2PKH transactions. The last one is an OP_DROP with a P2PK: "<DATA>OP_DROP <PUBLIC KEY A> OP_CHECKSIG". It can be unlocked as a classical P2PK transaction.

In Figure 42 we can see that the most used type is the $2-2$ multi-signature one, with almost 25 000 occurrences. We can say that these transactions are just standard outputs in disguise, using the OP_DROP operator to add data that is discarded during verification.[30]

[30] `https://tinyurl.com/NStxoutput`.

Figure 42: Distribution of OP_DROP (left) and OP_HASH160 OP_EQUALVERIFY transactions inside P2SH (right).

OP_HASH160 OP_EQUALVERIFY

We found four different types of transactions that start with OP_HASH160 OP_EQUALVERIFY. The first one has this locking script: "OP_HASH160 <HASH160OFSOMETHING>OP_EQUAL 1" or "OP_HASH160 <HASH160OFSOMETHING> OP_EQUAL 0 1". They could be transactions made to have a P2SH that can be unlocked by revealing only the redeem script, in fact these scripts do not need anything to be unlocked because at the end they push 1 in the stack; hence, the transaction is always verified.

The second is a P2PK that starts with a series of OP_HASH160 OP_EQUALVERIFY, with the locking script:
"(OP_HASH160 <HASH160OFSOMETHING>OP_EQUALVERIFY)*N <PUBLIC KEY A> OP_CHECKSIG"[31]. To unlock the script, it is needed to know all the strings of the hashes and the private key corresponding to the public key in the script. We can consider this transaction like a challenge to a specific person (the owner of the private key corresponding to the public key in the script): when he found all the strings of the hashes in the script, he can take the value.

There is also a variant with this script: "(OP_HASH160 <HASH160OFSOMETHING>OP_EQUALVERIFY)*N <PUBLIC KEY A> OP_CHECKSIGVERIFY <DATA> OP_DROP OP_DEPTH 0 OP_EQUAL", that can be unlocked like the previous one because the new part checks that there is no element in the stack and it is possible only if all the given strings and the signature are correct. Similar

[31] In this script, N is a integer number greater than 0 and it represents the number of times that a particular part of code is repeated.

to the previous one, this transaction is a challenge to a specific person, which can take the value only if he knows all the the strings of the hashes in the script, but in addition there is the <DATA> OP_DROP sequence, as we saw in Section 4.2.2, allows for storing some data in blockchain without making the transaction unspendable.

The last one is only with OP_HASH160 OP_EQUALVERIFY, whose indentifying script is "(OP_HASH160 <HASH160OFSOMETHING> OP_EQUALVERIFY)*N OP_HASH160 <HASH160OFSOMETHING>OP_EQUAL". To unlock it we need to know all the hashing strings. Also this transaction could be a challenge: the first user who finds all the strings of the hashes, can take the value.

In Figure 42 we show the distribution of OP_HASH160 OP_EQUALVERIFY transactions. The one with OP_DEPTH is the most common class, with more than 6 500 transactions extracted from the blockchain.

OP_IF

These transactions are characterised by the presence of the OP_IF. In fact this operator, as in C or JAVA, generates different execution branches, and the receiver can chose the one he prefers.

We found seven different classes of transactions that start with OP_IF, each of them characterized by the following scripts:

1. "OP_IF
   OP_SIZE 32 OP_EQUALVERIFY OP_SHA256 <SHA256OFSOMETHING>
   OP_EQUALVERIFY OP_DUP
   OP_HASH160 <PUBLIC KEY A HASH>
   OP_ELSE
   <DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP OP_DUP OP_HASH160
   <PUBLIC KEY B HASH>
   OP_ENDIF
   OP_EQUAL OP_CHECKSIG".
   This transaction can be unlocked in two ways: the first with a 32-byte string obtained from the SHA256 hash and the signature of A; the second one, after the date in the script, only with the signature of B. We can see this transaction like a challenge between A and B (the owner of the private

keys corresponding to the public keys A and B in the script): if A finds the 32-byte string before the date in the script can take the value, otherwise B will take the bitcoins.

2. There is also another type equal to the last one but without the size check:
   "OP_IF
   OP_SHA256 <SHA256OFSOMETHING>OP_EQUALVERIFY OP_DUP
   OP_HASH160 <PUBLIC KEY A HASH>
   OP_ELSE
   <DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP OP_DUP OP_HASH160
   <PUBLIC KEY B HASH>
   OP_ENDIF
   OP_EQUAL OP_CHECKSIG".
   This transaction presents the same challenge as in the previous one, but the string's length is not fixed, so A has to find the string before the date in the script or B will take the value.

3. We found also a version with the branch reversed and using Hash160 instead of SHA256:
   "OP_IF
   <DATA> OP_CHECKLOCKTIMEVEIRFY OP_DROP <PUBLIC KEY A> OP_CHECKSIG
   OP_ELSE
   OP_HASH160 <HASH160OFSOMETHING>OP_EQUALVERIFY <PUBLIC KEY B> OP_CHECKSIG
   OP_ENDIF".
   This is identical to the second one but with the branch inverted, i.e. now B has to find the strings or A will take the value.

4. One more variant, which needs 15 original strings, is this:
   "OP_IF
   (OP_RIPEMD160 <RIPEMD160OFSOMETHING>OP_EQUALVERIFY)*15
   <PUBLIC KEY A> OP_CHECKSIG
   OP_ELSE
   <DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP <PUBLIC KEY B> OP_CHECKSIG

OP_ENDIF."

Also this transaction can be considered like a challenge very similar to the second one, but A has to find 15 strings of the hashes before the date in the script, otherwise B will take the value.

5. A class that can be spent immediately with two signatures or, after the date inside the script, with one signature only:
   "OP_IF
   <PUBLIC KEY A> OP_CHECKSIGVERIFY
   OP_ELSE
   <DATA>OP_CHECKLOCKTIMEVERIFY OP_DROP
   OP_ENDIF
   <PUBLIC KEY B> OP_CHECKSIG".
   This could be consider a transaction between two people (A and B) who do not trust each other, in fact if everything is well and good and A does not disappear, they can take the bitcoins, otherwise after the date in the script B can take the bitcoins.

6. A class with 2-2 multi-signatures and CLVT:
   "OP_IF
   2 <PUBLIC KEY A> <PUBLIC KEY B> 2 OP_CHECKMULTISIG
   OP_ELSE
   <DATA>OP_CHECKLOCKTIMEVEIRFY OP_DROP <PUBLIC KEY A> OP_CHECKSIG
   OP_ENDIF".
   This transaction can be considered exactly like the previous one, in fact the sequence 2 <PUBLIC KEY A> <PUBLIC KEY B> 2 OP_CHECKMULTISIG is identical to <PUBLIC KEY A> OP_CHECKSIGVERIFY <PUBLIC KEY B> OP_CHECKSIG.

7. The last class is represented by transactions that do not need anything to be unlocked: at the end the script pushes 1 in the stack. For example, the script is
   "OP_IF
   <DATA> 15 <PUBLIC KEY A> OP_CHECKMULTISIG
   OP_ENDIF

1.”

They could be transactions prepared to have a P2SH that can be unlocked by revealing only the redeem script, in order to make easier the unlock.

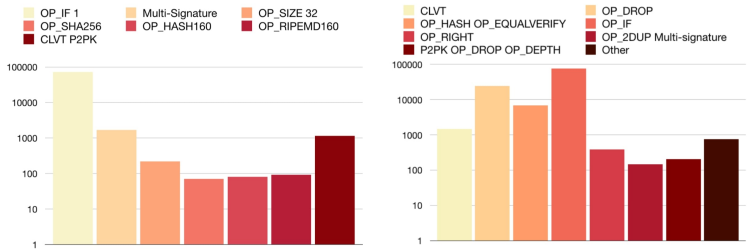

Figure 43: Distribution of OP_IF(left) and non-standard transactions inside P2SH (right).

As we can see in Figure 43, the most used transaction is the OP_IF 1, with more than 70 000 results.

OP_RIGHT

This type of transaction has a script that contains only “OP_RIGHT”. This operator[32] takes a string and a position and it pushes only the characters on the right w.r.t. that position in the string. To unlock this script, it is is enough to assemble an unlocking script with a number and a string in a way that the result of the OP_RIGHT is different from 0. Like the OP_HASH 1 or the OP_IF 1 in the previous sections this transaction could be used to make a P2SH that can be unlocked by revealing only the redeem script.

OP_2DUP Multi-signature

These transactions are similar to the multi-signature transactions, but with some noticeable differences: “OP_2DUP OP_EQUAL OP_NOT OP_VERIFY 2 <PUBLIC KEY A> OP_DUP 2 OP_CHECKMULTISIG”. To unlock this script, two signatures from the same private key are needed. The reason is that the first operator OP_2DUP duplicates the two signatures, and then OP_EQUAL checks if they are

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

the same; however, they are different, and then it pushes 0 in the stack. Now, OP_NOT changes 0 in 1 and OP_VERIFY removes 1 from the stack. At the end, the presented scheme corresponds to a plain multi-signature.

This transaction is very dangerous for the receiver, in fact it requires two signature from the same private key. This exposes the private key to the risk of being recovered from the public key, i.e. the risk that anyone can have this private key[33].

P2PK OP_DROP OP_DEPTH

This transaction looks like a P2PK but at the end of the script it checks whether the stack is empty, this is the script: "<PUBLIC KEY A> OP_CHECKSIGVERIFY <DATA> OP_DROP OP OP_DEPTH 0 OP_EQUAL". So to unlock this script, only the signature generated by the private key of A is needed. Like the OP_DROP (see Section 4.2.2), this transaction allows for storing some data in blockchain without making the transaction unspendable.

In Figure 43 we show the distribution of non-standard transactions inside P2SH transactions. The most used is the OP_IF class, with almost 80 000 outputs. The second one, with almost 25 000 occurrences, is the OP_DROP transaction.

4.3 An analysis of OP_RETURN

We analyze the nulldata transaction outputs, also called OP_RETURN, and in this way we update with new data the work by [BP17]. As introduced before, this transaction cannot be spent, and it is used only to store data in the blockchain, exploiting its immutability. In this case, our goal is to study the use of OP_RETURN over the year.

4.3.1 Data Dimension

The data dimension inside the OP_RETURN transaction is different also because the standard dimension changed during years. In fact, in the Bitcoin Core

[33] `https://allprivatekeys.com/random-vulnerability.php`.

0.9.0[34] the limit was only 40 bytes. Then from the 0.10.0[35] release on February 16th 2015, Bitcoin nodes could choose whether to accept or not OP_RETURN transactions, and to set a maximum dimension. The 0.11.0[36] release on July 12th 2015 extended the data limit to 80 bytes. Finally, the 0.12.0[37] release on February 23th 2015 set the maximum to 83 bytes (80 byte default, plus three bytes overhead).

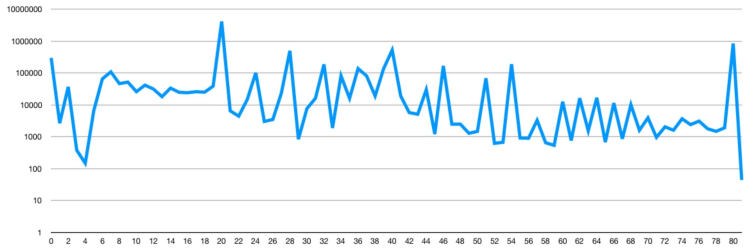

Figure 44: Distribution of OP_RETURN transactions divided by size.

In Figure 44 we show the distribution of OP_RETURN data size. The most used size is 20 bytes with more than 4 million occurrences, the second one is 80 bytes with almost 1 million outputs, and the third is the 40 bytes with more than 500 thousands occurrences. There are also other transactions with only one occurrence, which we do not show in Figure 44, exactly with size 95, 114, 134, 190, 449 and 983 bytes. There are also 296 518 output with data size 0, i.e. empty, of which 222 348 were made in September 2015 by compromised addresses[38]. These transactions were definitely used as stress tests for the Bitcoin Network [[Baq+16]].

In Figure 45 we show the distribution of OP_RETURN transactions over time. It seems clear that every year the OP_RETURN occurrences double, and this proves that their use is increasing at a high rate.

[34] https://bitcoin.org/en/release/v0.9.0.
[35] https://bitcoin.org/en/release/v0.10.0.
[36] https://bitcoin.org/en/release/v0.11.0.
[37] https://bitcoin.org/en/release/v0.12.0.
[38] https://allprivatekeys.com/random-vulnerability.php.

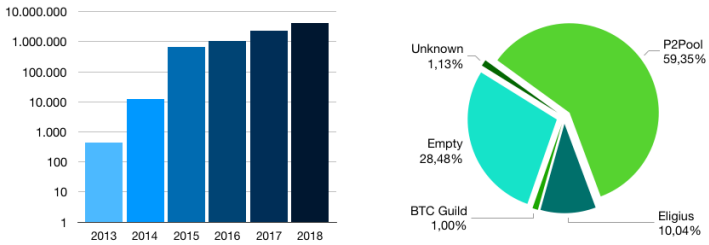

Figure 45: Distribution of OP_RETURN transactions over time (left) and Distribution of miners in "non-standard" OP_RETURN transactions (right).

"Non-standard" OP_RETURN

We analyzed transactions according to the aforementioned Bitcoin Core update, with the purpose to find those that did not respect size rules, thus we can call that "non-standard". We found 797 results, of which 784 were registered before the release of Bitcoin Core 0.9.0 (on average they have a size of 38 bytes) and 13 registered during the 0.9.0 release, but with more than 40 bytes (on average they have a size of 61 bytes). In Figure 45 we show the distribution of miners that accepted transactions with "non-standard" OP_RETURN transactions. P2Pool is the miner pool that mined the largest number of them.

The last analysis that we did is the amount of Bitcoin "lost" in these transactions. We see that only 57 038 (0.68% of all OP_RETURN transactions) spent bitcoins with a total amount of 3.71572552 BTC. The maximum spent for a transaction is 0.018454 BTC, the minimum is 1 Satoshi (10^{-8} BTC).

4.4 Transaction Information Tool

Transaction information module[39] is focused on providing additional information about transactions. First, it classifies transactions into standard and non-standard types, according to the Bitcore function *isStandard()*[40]. Then, it shows the distributions of standard and non-standard transactions in the blockchain (Figure 46), i.e. all the information that we studied in this chapter. This mod-

[39] `http://btctransaction.dmi.unipg.it/`.
[40] `https://github.com/bitcoin/bitcoin`.

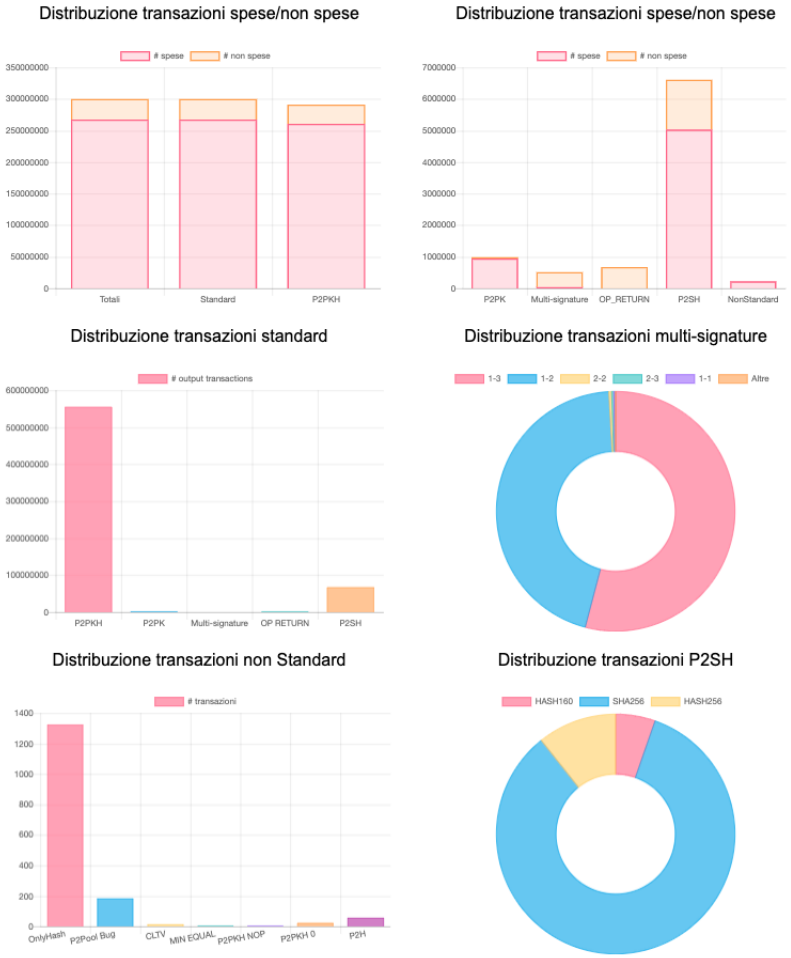

Figure 46: Chart visualization on transaction information module.

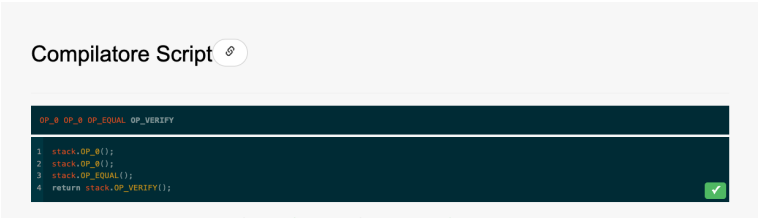

Figure 47: Bitcoin script compiler on Transaction information module.

ule also allows for interacting with a Bitcoin scripting compiler, see Figure 47. The Bitcoin transaction language *Script* is a Forth-like [RCM93] stack-based execution language. Script requires minimal processing and it is intentionally not Turing-complete (no loops) to lighten and secure the verification process of transactions. An interpreter executes a script by processing each item from left to right in the script. Data is pushed onto the stack, as well as operations, which can push or pop one or more parameters on/from the execution stack, operate on them and possibly push their result onto the stack.

4.5 Related work

Several analysis can be found in the literature.

In 2014 Ken Shirriff's blog[41] studied some methods for inserting arbitrary data into Bitcoin blockchain and also what kind of data can be (or is already) stored.

A few months later QuantaBytes[42] surveyed Bitcoin transactions in blockchain and found three classes of non-standard transaction.

In 2018 [SVS18] improved the study on inserting arbitrary data into Bitcoin's blockchain.

Then [Mat+18b] describes the problem of inserting harmful content into a blockchain; in particular they propose conceptual countermeasures to heuristically reject transactions holding unintended content with high probability. They find that mandatory minimum fees and mitigation of transaction manipulability

[41] `https://tinyurl.com/asciibw`.
[42] `https://tinyurl.com/txType`.

significantly raise the bar for inserting malicious content into a blockchain.

In he same period, [Mat+18a] the authors provide a systematic analysis of the benefits and threats of arbitrary blockchain content. They show that certain illegal content can render the mere possession of a blockchain illegal. Their analysis reveals more than 1600 files on the blockchain, e.g. links to child pornography. This analysis highlights the importance for future blockchain to be designed to address the possibility of unintended data insertion and protect users.

In 2019 [BP17] empirical study the usage of OP_RETURN over the years and they identify several protocols based on OP_RETURN, which they classify by the application domain and their space consumption.

4.6 Conclusions

We reported statistics concerning standard (seven classes) and non-standard (nine classes) transactions in the Bitcoin blockchain, by considering up to block number 550 000, i.e. until November 14th 2018. The most populated class of transactions is P2PKH. The second most used class is P2SH. This study showed the adherence of the Bitcoin protocol to the intended purposes, by quantifying past and present deviations. As a result we have obtained that only 0,02% of transactions outputs corresponds to non-standard ones: this implies that most of miners and users behave in a standard way. We noticed that only the 0,015% of all the circulating bitcoins was burned by non-standard transactions. We saw that the most used transaction inside P2SH transactions is the multi-signature one. We also show that the most used size in bytes of an OP_RETURN transaction is 20 bytes.

We remark that, since the massive amount of data, these results took a significant amount of time to be realized, also with a powerful machine like the one described in Appendix A.1. First of all, our first implementation needed three months to include the first 550 000 blocks in the DB. Then we inserted the records as a TSV[43] file. In this way, we can insert thousands of records in one query. The process speeds up to less than one week. Then querying the DB to get the class distribution took one day. After we got all the non-standard transactions, we classified them for around two weeks. We could not use a fully

[43] tab separated values

automatic tool because we did not know them structure. Now it will be easier since we know some classes. Finally, we spent around one week getting all the redeem scripts of the P2SH and translating them from hexadecimal to Script. Also in this case, we spent almost a month classifying them since we did not know their pattern.

Chapter 5

Study on reused Addresses

In this chapter we study the reuse of the same Bitcoin address in different transactions. We investigate the reuse of *legacy* addresses (legacy ones are the original Bitcoin addresses), whose addresses start with 1, and the repetition of addresses in *Nested SegWit* (P2SH) transactions, whose addresses start with a 3 instead. In addition, we "open" P2SH transactions and we count the repetitions of legacy addresses included in their script. In this way we have a comprehensive view on address reusing.

The motivations behind this chapter are multi-faced. The reuse of Bitcoin addresses is a practice that refers to the use of the same address for multiple transactions. It is often an unintended practice adopted by users, abusing the privacy and security of transaction participants as well as future holders of its value. The most private and secure procedure is instead represented by using a new address each time a user needs to receive value. After the received coins have been spent, the address should never be used again. Reusing the same address multiple times increases the amount of information that relates an address to the identify of its wallet owner. Using a new address each time is feasible, since, as a remainder, the total number of possible Bitcoin addresses is 2^{160}. The practice of reusing the same addresses is also susceptible to well-known attacks undermining the security of coins and exposing them to theft. For instance, *timing side-channel* attacks [YB14; FWC16], or possible weak randomness in ECDSA [Wan+20; Bro05; BH19].

For this reason, first we investigate the reuse of Bitcoin addresses, also including hidden addresses, i.e. addresses that are not an explicit output of a transaction, but they are still contained in it (e.g., in the script of a Pay to Script Hash transaction). As a second and final step, we de-anonymise as more addresses as possible in the first 100 positions of the most-frequently reused legacy-address ranking: as a result we obtain addresses linked to scams or ransomware. We consequently derive that such a metric can be considered in the automatic computation of a trust-score to be associated with Bitcoin addresses. This value can be taken into account before interacting with these users, e.g., sending to or receiving coins from them.

The chapter is structured as follows: Section 5.1 presents the results obtained by querying and then looking into the transactions that contain reused Bitcoin addresses. Section 5.2 presents the distribution of the different classes of transactions in the blockchain; we curb to transactions that may contain hidden addresses. Then, we present the results of reused hidden Bitcoin addresses. Moreover, we also show how we can visualize such hidden information in *Blockchain-Vis*. In Sec 5.3 we identify the top 100 reused addresses and we relate them to well-known malware and exchange services: this section proves that the motivations we highlighted find application in the real world of Bitcoin and Cybersecurity. Finally, in Section 5.4 we wrap up the chapter with conclusive thoughts and ideas about possible future work.

5.1 Analysis of reused Bitcoin addresses

People should have different Bitcoin addresses and, to preserve privacy and security, a unique address should be used for each transaction. The reason is that several vulnerability issues may affect the protocol: for example, ECDSA weak randomness [Wan+20] leads to private key leakage and even fund theft. In fact, due to this weakness, knowing more than one digital signature could be enough to calculate the private key needed to spend bitcoins. Besides security, also motivations behind privacy are important: the more an address is used, the more information about is owner could be discover. A further problem of reusing addresses is confusion. For example, a merchant tells his clients to send value to a single address. Customer *A* sends him bitcoins. Since Bitcoin transactions are

public on the blockchain, another malicious customer B could see the transaction made by A and send to the merchant an email saying that he/she paid. At this point the merchant is not capable of knowing who really sent coins to his address. This means that each customer need to have a different address where to send her payment. To solve these problems, most Bitcoin software and websites generate a new address each time a payment request is generated. However, in the Bitcoin blockchain, if we consider all the blocks up to the 550 000th one (end of 2018), there are 442 742 034 distinct addresses. They are used in total 951 320 268 times, so on the average they are used 2,15 times each. Since the Bitcoin protocol advises against using more than once the same key (so the same address), we decided to analyse how many addresses are used more than once and for what reason. First of all, we found out that in the blockchain there are 42 172 956 reused addresses, of which only 6 751 674 are legacy ones.

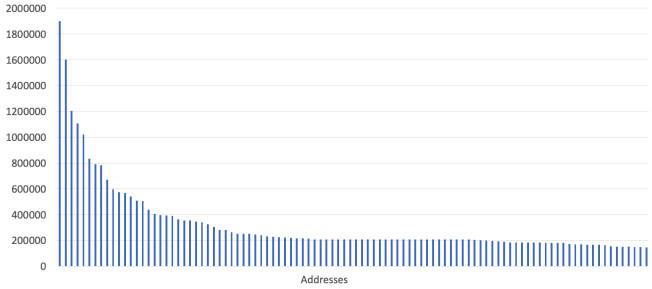

Figure 48: The fist 100 most reused addresses.

In Figure 48 we present the distribution of the first 100 most reused addresses. The most reused is a legacy one and it's used almost 1 990 000 times. The second one is reused more than 1 600 000 The first reused nested address has more than 1 million repetitions.

In Table 2 we present a comparison between [Bar14] and our work. The table shows that the number of addresses and uses noticeably increases, while the mean value is lower. This could mean that users are more adherent to the protocol than in 2014.

5.2 Hidden addresses

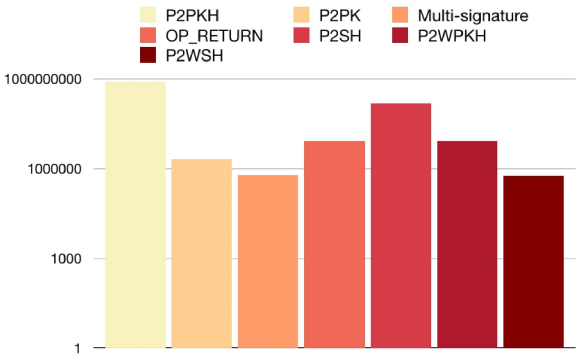

Figure 49: Distribution of transactions in the blockchain.

In the Bitcoin blockchain we now have around 1 billion transaction outputs. In Figure 49 we see how the most used transaction is the default one in the Bitcoin client: the *Pay to PublicKey Hash* type (*P2PK*)[1]. The P2SH is the second mostly frequently used class of transactions, with almost 150 million outputs [BMS19].

As we introduced before, the P2SH transactions contain the hash of a script (called *redeem script*) in their locking script, which in turn may contain different classes of Bitcoin transactions. In the blockchain there are 149 410 668 P2SH

[1] About transaction classes in the legend of Figure 49 and Figure 50, please refer to chapter 4

	Barcelo [Bar14] (January 2014)	Our result (December 2018)
Number of addresses	12,963,199	442,742,034
Number of uses	41,244,997	951,320,268
Mean	3.18	2.15
Number of max uses	1,238,931	1,899,538
Addresses used once	10,476,899	400,569,078

Table 2: Statistics about address reuse.

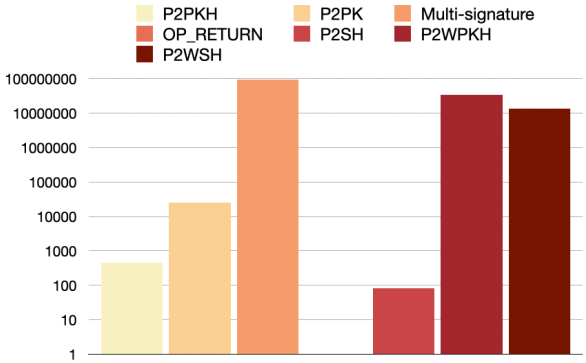

Figure 50: Distribution of transactions inside P2SH ones.

transactions, of which 140 620 401 are spent (94,12%).

Hence, we decided to analyse the content of the redeem script inside the unlocking script of the spent P2SH transactions. As we can see in Figure 50, inside P2SH transactions the majority of the transactions is standard. In fact there are 140 509 279 standard transactions, which are 99,92% of the total. The most frequent class of transactions is the multi-signature one, with more than 90 million occurrences (65,7%). Like plain transactions in the blockchain, also the ones inside the P2SHs have addresses related to them. In particular we can "translate" a nested address in one or more (in case of multi-signature transactions) legacy ones.

If we look inside this class of transactions, we obtain more than 90 million nested addresses, out of which only 32 176 523 are distinct. By extracting the content of the redeem script of these transactions, we obtain 89 143 036 legacy addresses, that we can consider as hidden and related to nested addresses. Then, we check if some of these hidden addresses were used also in clear and we found 56 702 matches. Thus, these addresses are used both inside and outside P2SH transactions.

5.2.1 Analysis of Bitcoin hidden reused addresses

To better analyse the reuse distribution, we decide to convert all the nested addresses to the related legacy ones. Hence, from now on all these nested addresses are considered as legacy ones in the analysis; we will refer to them as *hidden*.

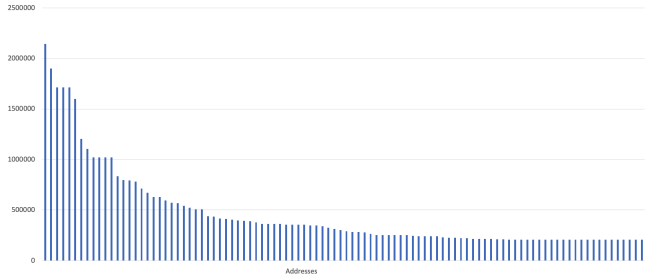

Figure 51: The first 100 most reused addresses considering also hidden ones.

In Figure 51 we present the distribution of the first 100 most reused addresses. The most reused one is a hidden address, and it is used more than 2 100 000 times; the second one in the list is a legacy one, and it is reused almost 1 990 000 times. The first reused nested addresses, which we cannot convert to hidden, has 540 090 repetitions.

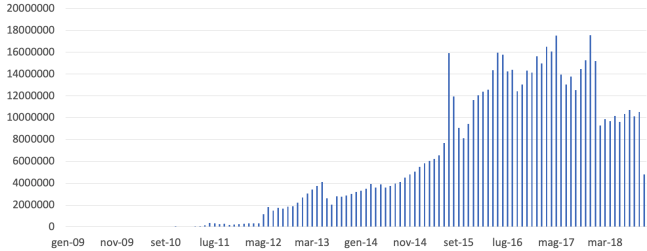

Figure 52: The distribution of address repetitions in time.

In Figure 52 we show the distribution of address repetitions in time.

There are three peaks: the first is July 2015, the second one is in May 2017, and the third one, which is the highest, occurs December 2017, the period when

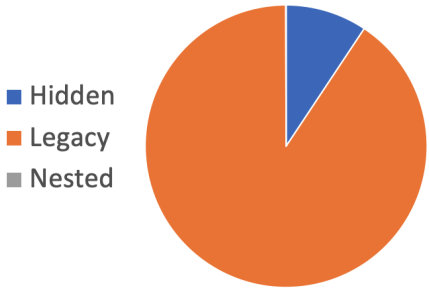

Figure 53: Type address distribution on July 2015.

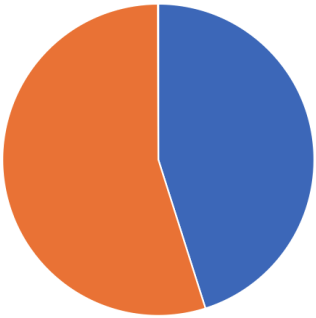

Figure 54: Type address distribution on May 2017.

the bitcoin value reached the highest price ever[2]. Considering the peak of July 2015 in Figure 53, we can see that more than 90% is generated from legacy addresses. There are only 7 815 nested ones. During May 2017 instead, as represented in Figure 54, we can see a large number of hidden addresses. Still the nested ones are less the 9 000. Finally, in December 2017 we still have a majority of legacy addresses, but we have a huge increase of nested ones: they are more than 4 000 000. In general we can conclude that the majority of repetitions are due to legacy addresses.

[2] `https://tinyurl.com/bitcoinrecord2000`

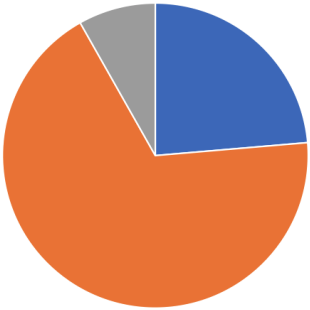

Figure 55: Type address distribution on December 2017.

Figure 56: Standard representation of P2SHs in BlochainVis.

5.2.2 Visualisation of Hidden Addresses

Our Visualizer can visualize also the transactions inside the P2SH. In Figure 56 we can see the standard visualization of a P2SH: pink nodes represent transactions, while orange nodes represent addresses. As shown in Figure 57, the tool now allows to click on nested addresses and thus check the actual inside transactions and their hidden addresses.

For example in Figure 58 we show a multi-signature transaction inside a P2SH and his hidden addresses. The edges obtained in the node explosion allow us to see which are the spent addresses.

5.3 De-anonymisation of reused addresses

In this section we try to analyse the set of reused Bitcoin wallet addresses based on their classes: respectively legacy and nested addresses. The set of reused addresses was ordered and divided into two distinct subsets where the same analysis was executed on each.

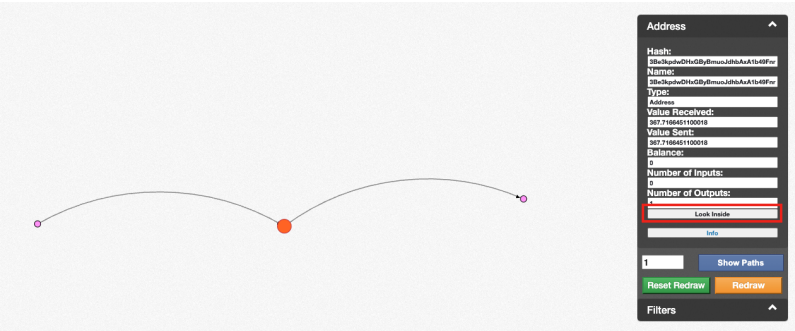

Figure 57: Control panel for a nested address (circled in red the button to show the inside transaction).

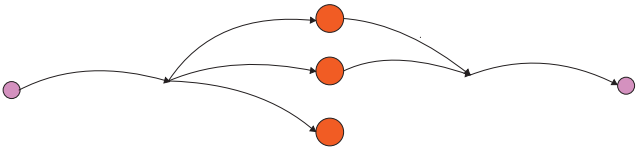

Figure 58: A new representation in BlochainVis of a 2-3 multi-signature transaction inside a P2SH transaction.

A first analysis is based on a public data sets [PHD18][Hua+18] of correlation between Bitcoin wallet addresses and plausible owner. The data set was built for, that is a study on ransomware and their related economic income. In this data set the main addresses are registered from the most recent fraud, malware and ransomware until 2017: based on this data set, most of the recurring addresses have been identified with respect to the ranking of most frequent repetitions. Secondly, the set of ordered addresses was subjected to a further analysis through a data set previously built for some of our studies [BMS18a]. In this data set, some of the wallets are identified as malicious or harmful activities for the blockchain network, but there are also wallets of activities related to malware. As last step, a long work was executed to analyse and identify publicly declared addresses trying to discover the remaining correlations catalogued as “no Reference”. Most of the addresses are extrapolated through other stud-

ies [HCS17][Kha+15b], forums or platforms of expert users[3,4], also by analysts of the Bitcoin Network[5]. In addition, the main exchange sites were scanned[6], trying to identify the wallet of one of them.

After the first phase of association between addresses and plausible references, we select the one hundred higher reused addresses. In these sets the total amounts are computed, then the relative distributions are analysed with respect to the global amount of addresses, as it is shown in Table 3. In this table, the distribution of the first 100 reused legacy addresses is shown (addresses are grouped by the owner). The entry marked as "No reference" means that it was not possible to attribute either a holder or a source to the address.

In Figure 60 we provide a graph of the time distribution of the most relevant 100 legacy reused addresses in the blockchain; the reported values refer to Table 3, with values grouped by type of address holder. Ransomware, exchanges and "No Reference" type. Around 75% of the reused addresses have been deanonymized, for a total of about 21 668 886 repetitions out of 29 286 669 total repetitions of the list of 100 most reused. In Figure 61 we show the time distribution of legacy addresses reused more than 500 times, colored by class. The result show that the most receptions are due to CryptoLocker activity.

A similar procedure was used on a nested addresses list. The study in [PHD18] presents a recurring address in abnormal transactions study [7], which is specified as a "scam wallet" address. However, it was not possible to deanonymize most of the nested addresses list.

In Figure 59 we show a graph on the analysis of the list of the top 100 reused nested addresses. Even according to the very few results found during the address association phase, the presence of repetitions (about 1 021 574) of the wallet containing the malicious address is relevant.

[3] `https://tinyurl.com/Stoopwasting` an analysis of the exploit bug of Gemini

[4] `https://tinyurl.com/extortscam` an analysis of types of legal transactions related to sites prohibited to minors

[5] `https://whitesunset.github.io/wannacrypt_balance/` in-depth study on the distribution of three wallets of the WannaCry ransomware

[6] `https://tinyurl.com/whoControlBC` a research on wallets related to a known exchange code bug

[7] `https://bit.ly/2Fj5vbY` appeared on a forum that investigates the malicious address, type-3, which is referred to as "scam wallet"

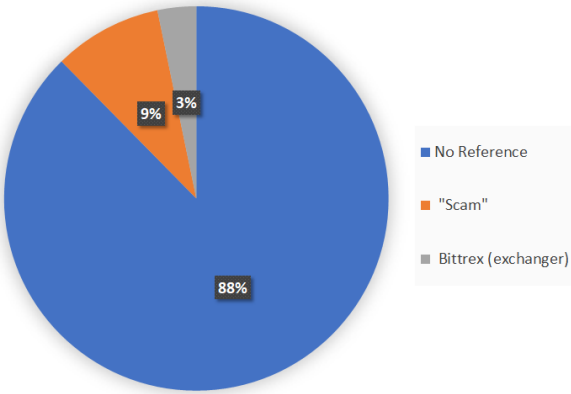

Figure 59: Distribution of the top 100 nested reused addresses.

5.3.1 Evaluation of de-anonymisation

Based on the data collected during the de-anonymisation and analysis phases, two considerations arise: a substantial percentage of the reused addresses belongs to exchanges, both for reasons of errors and bugs known on the transaction scripts associated with them, and for the obvious capability of the exchanges to be the hub for many transactions; a much larger and unexpected percentage refers to malicious addresses. In this first analysis we studied and tried to de-anonymise the first 100 addresses with higher repetitions. Yet from this first sample the amount of transactions related to malicious intents is evident: the first 4 reused addresses refer to the family of CryptoLocker ransomware. Furthermore, these alone characterise a large share of the address repetitions - around 5 810 459. By assuming a complete de-anonymisation, we suppose the percentage of 59% of reused addresses can only increase. A different and in-depth study should be carried out on nested type guidelines; study that could be improved by explicating the addresses and working on the list of addresses thus identified.

With the discovery of these addresses, it is possible to start mapping the critical points highlighted in Section 5.2.1 and Figure 52, when most of repetitions occur. In these periods we identify that: in July 2015 there were numerous transactions related to “cryptolockers”, as it was discovered and declared in the

following months; in December 2017, the period related to the peak value of the exchange of bitcoins, the majority of reused addresses is related to some exchanges identified in our study.

Wallet Reference	#Reuse
CryptoTorLocker2015	12398905
Others	7617783
SatoshiDICE Hot Wallet	4967566
Gemini (exchange)	792458
POLONIEX (exchange)	752445
Omni Layer (exchange)	670397
Huobi (exchange)	572297
Bittrex (exchange)	567945
Binance (exchange)	303486
FoxBit Hot Wallet (exchange)	280103
Coinimal.com (exchange)	196266
Bitcointoyou Hot Wallet (exchange)	167018

Table 3: Distribution of the 100 most-frequently reused legacy-addresses in blockchain transactions.

5.3.2 Mixing services addresses

When a Bitcoin users pay for some service or good, he/she will of course need to provide her name and address to the seller for billing or delivery purposes. It means that a third party[8] can trace her transactions and associate her address with her name. To avoid this, *mixing services* (also called *tumblers*) [SP06] provide the ability to interrupt a direct money-flow from one user to another by using addresses that do not belong to the original owner. Mixing services are used to mix one's funds with other people's money, intending to confuse the trail back to the original source. In traditional financial systems, the equivalent would be moving funds through banks located in countries with strict bank-secrecy laws, such as the Cayman Islands.[9]

[8] e.g. police authority.

[9] `https://en.bitcoin.it/wiki/Mixing_service`.

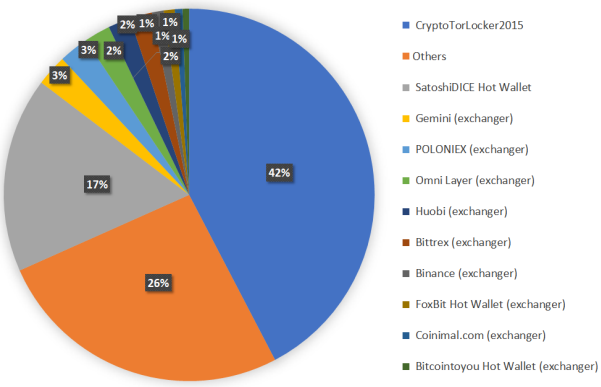

Figure 60: Distribution of the top 100 legacy reused addresses.

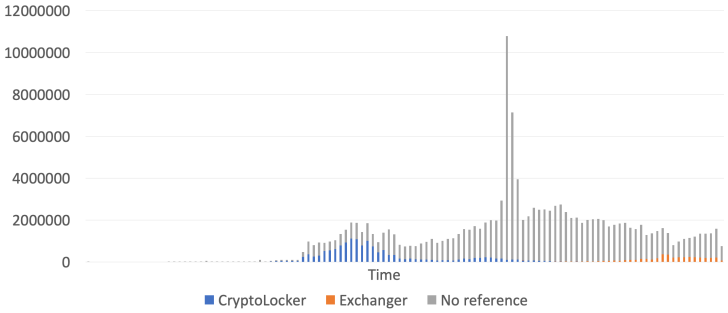

Figure 61: Time distribution of legacy addresses reused more than 500 times, each class differently coloured.

Service	Fees	Return Time	Min import	Max import
Helix Light	2.5%	max 24 h	0.01 BTC	43 BTC
BTC Blender	1-3%	max 99 h	0.01 BTC	None
Coin Cloud	1.25%	max 1 h	0.01 BTC	None
CoinMixer	1-3% + 0.0006 BTC	max 5 h	0.01 BTC	None
BitClock	2% + 0.0008 BTC	max 5 h	0.02 BTC	10 BTC

Table 4: Characteristics of some mixing services.

Mixing services transactions	All transactions
Made with mixing services	Obtained from the blockchain
Time range:	Time range:
25 September 2017, 22 October 2017	25 September 2017, 22 October 2017
Label with the name of the service	No label
973	7 852 074

Table 5: Dataset characteristics.

The goal of this subsection is to find mixing services in the Bitcoin Network. In particular, to extract related behavioural patterns in terms of payments and understand how a mixing service works. In practice, this allows for tracking a desired bitcoin flow also through a mixing service.

To experimentally find such patterns, we executed some real bitcoin-payments by using different mixing services: the final goals is to identify those addresses that belong to tumblers. In Table 4 we can see some features of the mixing services we used. We extracted two datasets to proceed with the investigation: one with all the transactions sending and receiving value from tumblers addresses, while the other one with all the other transactions performed in the same time interval. The characteristics of these two datasets are shown in Table 5.

Finally, we studied the behaviour of these addresses with Machine Learning, and in particular by using hierarchical clustering techniques considering the following nine features: input addresses, output addresses, balance, average balance, transaction ID, time of creation, number of inputs, number of outputs.

Unfortunately, in this way we were not able to spot a different behaviour between the two datasets. Then, in a second experiment we focused on a Data-mining analysis instead; in the tumblers dataset we noticed that 4.9% of CoinMixer transactions generates 89.7% of the edges, and 14 transactions generated more than 1 000 output addresses. These transactions show the following fea-

	TX 1	TX 2	TX 3	TX 4	TX 5	TX 6	TX 7
TX 1	100%	98.16%	97.31%	96.05%	94.54%	93.28%	92.58%

Table 6: Similarity of address sets (first seven days).

	TX 8	TX 9	TX 10	TX 11	TX 12	TX 13	TX 14
TX 1	91.78%	90.91%	90.2%	89.55%	88.99%	88.28%	87.55%

Table 7: Similarity of address sets (second seven days).

tures: *i)* number of input addresses equal to 2, *ii)* number of output addresses in the range $[2530, 2534]$, *iii)* they were collected one a day, for 14 consecutive days.

We decided to compare the sets of output addresses and we noticed that the similarity between the two datasets decreased day by day (see Table 6 and Table 7). This feature allowed us to conclude that the output addresses are gradually renewed over time with new addresses that work in the same way as those deleted. These results identify a behavioural pattern of the CoinMixer service, generated by a specific internal algorithm. Hence, through an analysis of transactions, and in particular of their output addresses, it is possible to also recover all similar past and future transactions.

To conclude, a further source of addresses reuse is clearly due to tumblers.

5.3.3 Mining pool addresses

A different source of reuse consists in mining pools. Some of them, such as *Eligius*,[10] have no registration process, and Bitcoin addresses are just used as usernames. This means that every node of the pool is always paid to the same address. For this reason, we decided to investigate in coinbase transactions how many addresses are reused (hence, in transactions strictly related to miners).

Table 8 shows some stats about addresses reused by these particular mining pools. We see that the pool that reuses addresses more frequently is Eligius. On the opposite side we have *BitMinter* (now definitely closed), which always pays to the same single address; because of this it is also the pool with the highest average per transaction. The most privacy-oriented pool is *SlushPool*, which

[10] `https://en.bitcoin.it/wiki/Eligius`.

	Eligius	Huobi	SlushPool	BitMinter	HHTT	BTC Guild	Bitparking
#addresses	39366	3	4818	1	139	22	14
#reused addresses	34441	3	16	1	108	21	14
% of reuse	87.49	100	0.33	100	77.70	95.45	100
max #reuse	10334	268	14560	6451	592	26204	425
mean	21.19	98.00	5.61	6451.00	21.52	1497.05	84.14

Table 8: Statistics about reused addresses in mining pools that use addresses to identify users.

	F2Pool	BTC.com	Poolin	58COIN&1THash	AntPool
#addresses	153	11	1	2	70
#reused addresses	3	11	1	2	70
% of reuse	1.96	100	100	100	100
max #reuse	34892	8065	194	352	6881
mean	229.14	1251.55	194.00	321.00	563.17

Table 9: Statistics about reused addresses in the top 5 mining-pools (top considering the amount of mined blocks).

shows only 0,33% of reuse. BTC Guild instead reused a single address more than the others, over 26 000 times.

Finally, we also checked repeated addresses in the 5 most successful mining pools, i.e. those that mined a larger number of blocks in the considered period. In Table 9 we can see the results. The pool that seems more coherent w.r.t. the Bitcoin protocol is *F2Pool*, which has a low percentage of reuse. *AntPool* has the highest percentage of reuse instead: it only uses 70 addresses for all its payments. In general, we can say that all the considered mining pools do not behave very well in terms of address reuse: their average is almost 100 greater than the Bitcoin Network's mean.

To sum up, we can say that also the behaviour of mining pools is a cause behind reused addresses in blockchain, even if in a small percentage of compared to cryptolocker and exchanges.

5.4 Conclusions

In this chapter we studied Bitcoin addresses that are most frequently reused in transactions. Our analysis is important to evaluate how much the privacy of addresses is at risk, and also how much the security of coins is exposed to attacks.

We investigated the repetitions of legacy and nested addresses; we have also considered and define hidden addresses, i.e. legacy addresses hidden in P2SH transactions, in order to have a more comprehensive view. We saw that most of top 100 reused legacy-addresses can be linked to ransomware payments and scams. Such a metric can be also adopted as a addresses trust indicator.

Notice that we need a lot of computational power and time to realise our analysis because of the massive quantity of data. First, querying our DB to get the occurrences of all the addresses took around two days. Then starting from the redeem scripts used in the previous chapter, we extracted all the public keys and translated them into hidden addresses. This took around one day. At this point, we spent twelve hours of computation to see if some of the hidden addresses were already in the blockchain. Finally, we took two days to recreate the new addresses distribution considering the hidden addresses.

Chapter 6

Arbitrage and Bubbles in the Bitcoin market

The key innovation in Bitcoin is its decentralized nature [Böh+15]. The system enables an independent currency, not subject to the control of central authority and without inflation. Recent literature claims Bitcoin as a volatile stock rather than a currency, [Yer15]. An important issue about Bitcoin prices is that it is traded on different web-exchanges for different prices. Hence it does not obey the usual law of unique price. On the web there are also some sites that compare in real time the price of Bitcoin in different exchange as well as the price history in order to decide what are the best exchanges for buying and for selling Bitcoins at a fixed point in time. The most important sites are Bitcoin Analytics, CoinDesk, Cryptohopper and AvaTrade[1].

Some studies try to understand the reasons for special activities in the market. In particular, several papers evidenced a bubble behaviour in exchange rates between Bitcoin (BTC henceforth) and traditional currencies (Euro or Dollars usually)[Gar+14; Kri15].

Traditional econometrics models within the class of AutoRegressive Integrated Moving Average (ARIMA) are *backward looking* since the only time-dependence admitted regards the past [BJ90] and are usually referred to as causal

[1]Available at: `http://bitcoin-analytics.com/`, `https://www.coindesk.com/price/`, `https://www.cryptohopper.com/`, `https://www.avatrade.com`.

models. Recently, models known as Mixed causal-noncausal AutoRegressive (MAR) have been introduced in order to extend time dependence to the future [Bre+91; GJ16; LS08] thus reflecting a *backward-forward looking* behaviour.

In this chapter, we investigate whether strong or weak form of arbitrage strategies are indeed possible by trading across different Bitcoin Exchanges. The aim of this chapter is to explore the conjecture that the bubble effect is due to confidence in Bitcoin future values so that its price/exchange rate is influenced both by past events and by views about future ones. In Section 6.1 we describe the dynamics of Bitcoin price by applying the well-known Black and Scholes model (see [BS73]) in a multi-variate fashion; under this assumption a strong arbitrage opportunity exists in the market even if the strategy is performed with discrete time revision of the portfolio. This theoretical arbitrage might be much outperformed in practice; in Section 6.2 we present an example of arbitrage strategy which take also advantage of the bid-ask spread mismatch in market Exchanges. The rest of the chapter is structured as follows: Section 6.3 is devoted to the economic explanation of our conjecture about the relation of the speculative bubble in BTC/USD exchange rates with the monetary policy of the Bitcoin system; then, in Section 6.4 the theory behind the *Mixed Causal–Noncausal autoregressive models* is briefly described. Section 6.5 describes the dataset and Section 6.6 summarizes the results of the estimation of the MAR model on the observed data. Section 6.7 gives conclusions and final remarks.

6.1 Modeling the Bitcoin price dynamics and arbitrage opportunities

6.1.1 Data description

Bitcoin is traded in multiple online trading platforms (Exchanges), where different exchange rates are applied against the same fiat currency. Even ignoring market frictions such as the bid/ask spread, by considering the mid-price as the unique price, still there are multiple prices for Bitcoin available from different Exchanges. We consider daily prices from 01/01/2015 to 31/12/2017 available

on the Exchanges website for Bitstamp, Gdax, Kraken, CEX.IO and BitKonan[2] and the corresponding value of the BlockChainInfo Index[3]. In Figure 62 we plot the above Bitcoin prices and Index in US Dollar (USD) over the whole time series and two selected sub-periods to better appreciate the different exchange rates. As we can notice from Figure 62, the behavior of Bitcoin price in the dif-

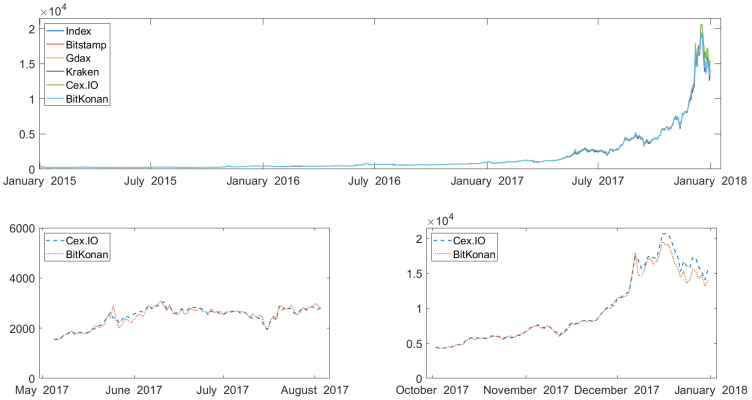

Figure 62: The Bitcoin price in USD according to 5 different Exchanges and the Index value (top) and two sub-samples with only two exchange rates (bottom).

ferent Exchanges is pretty the same. Indeed, the correlation between time series of Bitcoin prices obtained in the five considered Exchanges is approximately perfect, since it is close to 1.

In this chapter, we assume that there are I Exchanges trading Bitcoin in the same fiat currency (i.e. USD) and denote by $S_t^{(i)}$ the price of one Bitcoin quoted in Exchange i at time t. In order to take into account the almost perfect correlation between the different Exchanges, we consider a a common risk-source for all platforms. More precisely, we assume that the price dynamics of Bitcoin is described, in every Exchange, by the well-known Black and Scholes model [BS73], where

$$\mathrm{d}S_t^{(i)} = \mu_i S_t^{(i)}\mathrm{d}t + \sigma_i S_t^{(i)}\mathrm{d}W_t, \quad S_0^{(i)} = s_0^{(i)} > 0. \tag{6.1}$$

[2]Available at: `https://www.bitstamp.net/`, `https://www.gdax.com/`, `https://www.kraken.com/`, `https://cex.io/`, `https://bitkonan.com/`.

[3]`http://www.blockchain.info/`.

Here, for every $i = 1, 2, \ldots, I$, μ_i, $\sigma_i > 0$, represent model parameters, $W = \{W_t,\ t \in [0,T]\}$, is a standard Brownian motion modeling the fluctuation of the Bitcoin market and $T > 0$ denotes a fixed finite time horizon. Note that the process $\mu_i S_t^{(i)}$ is the so-called drift coefficient, which represents the "average appreciation" of the process $S^{(i)}$ at time t, while $\sigma_i S_t^{(i)}$ is the diffusion coefficient, measuring the intensity of the source of randomness given by W.

Note that, the Bitcoin price processes $S^{(1)}, \ldots, S^{(I)}$ are perfectly correlated since they have the same "driving" Brownian motion W; in addition, different Exchanges are characterized by (possibly) different parameters values in the dynamics.

In Figure 63 we plot two possible paths for two months of daily prices simulated according to model in (6.1) with parameters $\mu_1 = \mu_2 = 1.5$ and $\sigma_1 = 0.75, \sigma_2 = 0.9$. The picture exhibits a similar pattern to the one observed in Figure 62.

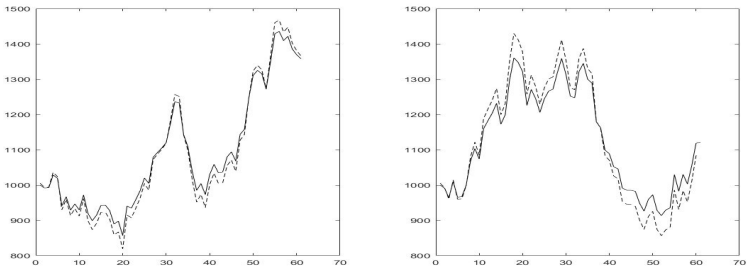

Figure 63: An example with two simulated paths for two months of daily prices: parameters are set to $\mu_1 = \mu_2 = 1.5$ and $\sigma_1 = 0.75$ (solid), $\sigma_2 = 0.9$ (dashed) respectively.

6.1.2 Arbitrage opportunities

To show that arbitrage is theoretically possible, we focus on two different Exchanges and denote the corresponding Bitcoin prices as $S^{(1)}$ and $S^{(2)}$. In our framework, we have $S_t^{(i)}$ with $i = 1, 2$.

Let us denote by r the constant risk-free rate. The risk premia for the the Bitcoins quoted in Exchanges $i = 1, 2$ is defined as $\text{RiskPremium}_i = \mu_i - r$.

The risk premium can be seen as the return in excess of the risk-free rate of return an investment in Bitcoin is expected to yield. Then, the corresponding *Sharpe ratios*, that is, the average returns earned in excess of the risk-free rate per unit of volatility or total risk, are given by

$$\text{SharpeRatio}_i = \frac{\text{RiskPremium}_i}{\sigma_i} = \frac{\mu_i - r}{\sigma_i}, \quad i = 1,2. \tag{6.2}$$

Note that two assets that are perfectly correlated as $S^{(1)}$ and $S^{(2)}$, must have the same Sharpe ratio. Otherwise, it is possible to exploit an arbitrage opportunity, i.e. a trading strategy with zero initial outlay and a non-negative future payoff (so it does not expose to any risk) that is positive with positive probability.
This means that a sufficient condition to realize this investment strategy is

$$\text{SharpeRatio}_1 > \text{SharpeRatio}_2. \tag{6.3}$$

Indeed, let us consider the self-financing portfolio $(\alpha^1, \alpha^2, \beta)$, where, for any $t \in [0,T]$, we buy the amount $\alpha_t^1 = C\left(S_t^{(1)}\sigma_1\right)^{-1}$ of Bitcoin with price $S^{(1)}$ on Exchange 1, we short-sell the quantity $\alpha_t^2 = C\left(S_t^{(2)}\sigma_2\right)^{-1}$ of Bitcoin with price $S^{(2)}$ on Exchange 2 and we invest/borrow the risk-free bond in the amount of the cost difference $C\left(\frac{1}{\sigma_1} - \frac{1}{\sigma_2}\right)$, where C is an arbitrary positive constant.
If V_t denotes the corresponding portfolio value at time t, then $(\alpha^1, \alpha^2, \beta)$ is a strategy with null initial value, since

$$V_0 = -\alpha_0^1 S_0^{(1)} + \alpha_0^2 S_0^{(2)} + \beta_0 = C\left(-\frac{1}{\sigma_1} + \frac{1}{\sigma_2} + \frac{1}{\sigma_1} - \frac{1}{\sigma_2}\right) = 0. \tag{6.4}$$

Moreover, the return of the above strategy is

$$\begin{aligned} \mathrm{d}V_t &= \alpha_t^1 \mathrm{d}S_t^{(1)} + \alpha_t^2 \mathrm{d}S_t^{(2)} - r\beta_t e^{rt}\mathrm{d}t && (6.5)\\ &= \frac{C}{S_t^{(1)}\sigma_1}\mathrm{d}S_t^{(1)} - \frac{C}{S_t^{(2)}\sigma_2}\mathrm{d}S_t^{(2)} - rC\left(\frac{1}{\sigma_1} - \frac{1}{\sigma_2}\right)\mathrm{d}t && (6.6)\\ &= C\left(\frac{\mu_1 - r}{\sigma_1} - \frac{\mu_2 - r}{\sigma_2}\right)\mathrm{d}t = C\left(\text{SharpeRatio}_1 - \text{SharpeRatio}_2\right)\mathrm{d}t > 0, && (6.7) \end{aligned}$$

Table 10: Parameters fit of Black and Scholes model.

	Bitstamp	Gdax	Kraken	Cex.IO	BitKonan
μ_i	1.5313	1.6363	1.5317	1.5404	1.6770
σ_i	0.7323	0.8947	0.7451	0.7065	0.9139
SharpeRatio_i	2.0911	1.8289	2.0557	2.1803	1.8350

and therefore it gives rise to an arbitrage opportunity since it produces a certain profit that is strictly greater than 0. The total gain of the strategy in the time interval $[0,s]$, for $s > 0$, is given by $C\left(\frac{\mu_1 - r}{\sigma_1} - \frac{\mu_2 - r}{\sigma_2}\right)s$. Here, C represents a scale factor which may leverage the total gain. Note that the above arbitrage opportunity is due to perfect correlation of the two dynamics. This is not the case in traditional financial markets; though some common risk factors may be identified to describe the systematic fraction of the variance of each asset, the idiosyncratic part of the variance is non negligible.

Example. Assume that parameters are set to $\mu_1 = \mu_2 = 1.5$, $\sigma_1 = 0.75$, $\sigma_2 = 0.9$ and $r = 5\%$ and that at time $t = 0$ we have $S_0^{(1)} = 1000, S_0^{(2)} = 1005$ and $C = 1000$ USD. Then, the arbitrage strategy consists in buying $\alpha_0^1 = 1.3086$ Bitcoins in Exchange 1, selling $\alpha_0^2 = 1.1056$ in Exchange 2 and an investment of 222.22 USD in the money market account. The initial cost of the investment is exactly V_0 by construction, while at time $t = \frac{1}{365}$ (one day) the profit is $\frac{1000}{365}(\text{SharpeRatio}_1 - \text{SharpeRatio}_2) = 0.88$ USD. This is an arbitrage. Investments quotes should be revised in continuous time to keep the profit riskless.

6.1.3 Experiments

In what follows we consider the time interval from 01/01/2015 to 31/12/2017 as a training time for the model and we estimate the above model on daily Bitcoin prices available on Gdax, Bitstamp, Kraken, CEX.IO and BitKonan.
In Table 10 we report parameter values estimated on the above Exchanges. For the sake of simplicity, we have set $r = 0$ in the Sharpe ratios row.

First, we observe that parameter estimates are different across Exchanges, meaning that arbitrage opportunities arise; Gdax and CEX.IO Exchanges exhibit

the most diverging Sharpe ratios.

The theoretical trading strategy considered in Section 6.1.2 is tested over the next 90 days by computing the overall daily profit of the investment applied to Gdax and CEX.IO; it is worth noticing that the strategy is performed by applying daily revisions of the investment rather than in continuous time as suggested in the theoretical model. Nevertheless it is still very profitable.

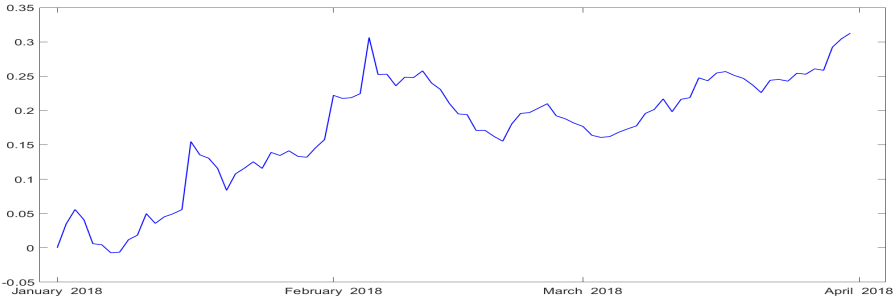

Figure 64: Total profit dynamics in USD from January, 1, 2018 to March 30, 2018 (90 days), investing on Gdax and CEX.IO Exchanges with an initial investment $C = 1\$$.

As an example, we plot in Figure 64 the total theoretical profit obtained by investing according to the above strategy on Gdax and CEX.IO Exchanges which exhibit the largest difference in the Sharpe Ratio values. The Overall Gain is computed from January, 1 to march, 30, 2018 for $C = 100$. Clearly, by choosing a higher scale C the total gain increases proportionally.

6.2 Arbitrage opportunities in practice

To understand if there are arbitrage opportunities in practice we decided to make an application. Our application (TradeBitcoin), part of the suite BlockChainVis used for Bitcoin analysis and visualization. This module[4] is based on finding the price options on the Bitcoin exchange and writing possible arbitrage operations on a database to see if it is possible to correctly perform an arbitrage on the Bitcoin market. It provides various calls: to check the possibility of real-time gain

[4] `http://tradebitcoin.dmi.unipg.it/`.

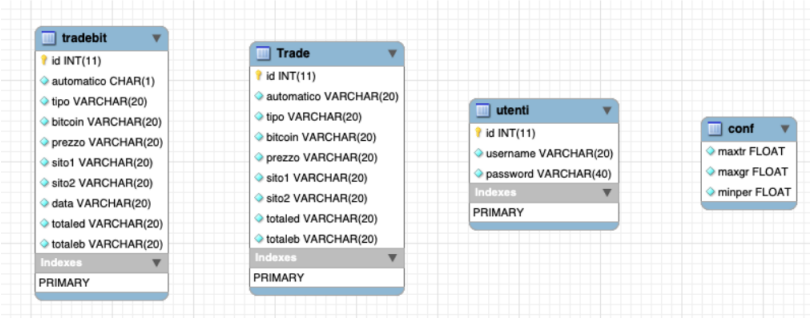

Figure 65: TradeBitcoin DB schema

from arbitrage between exchanges, automatic execution function every x hours (results are stored in a SQL DB); to download historical price data from bitcoincharts.com and insert them in SQL DB; statistical calculation of the places where have been most often the lowest buy price (BID) and the highest sell price (ASK) (saved in SQL DB). In Figure 65 the db structure: the table *utenti* contains the informations for login; table *conf* store the information to manage the arbitrage algorithm: *maxtr* is the max amount to spend for a single transaction, *maxtr* is the max amount to spend in a day and *minper* is the time to wait between two check; *tradebit* and *Trade* contain all the info of the arbitrate opportunities find by TradeBitcoin. To perform this, we choose the principal Bitcoin Exchanges that allow the use of web API to get Bitcoin prices in real time. In our application we used: Bitstamp, Kraken, CoinBase, ANX, Bitbay, Bitfinex, BitKonan, HitBTC and TheRock. In Figure 66 we can see a table, from our application, that shows in real time the price of all considered exchanges. Instead in Figure 67 a chart, from our application, that shows the number of times the BID has been greater and the ASK lower on a exchange in the last year.

6.2.1 How it works

Starting with an initial endowment of 1000 USD, we build an investment strategy that:

- Obtains with a fixed time frequency, bid and ask prices by the aforesaid exchanges.

Site	Bid	Ask	Fee(ask/bid)
Bitstamp	7640.12 $	7643.99 $	0.25 %
Kraken	7647.2 $	7652.9 $	0.34 %
CoinBase	7645.8 $	7667.45 $	1 %
ANX	7576.52387 $	7588.85163 $	0 %
Bitbay	7104.28 $	7500 $	0.43 %
Bitfinex	7556.2 $	7557 $	0.2 %
BitKonan	7511.31 $	7603.05 $	0.29 %
HitBTC	7581.82 $	7584.27 $	0.1 %
TheRock	7201.53 $	7543.67	0.2 %

Figure 66: Exchange prices table.

- Checks and compares BID and ASK values in different exchanges, also controlling for the **amounts offered** and **exchange's fees**.
- has a frequency of 6 hours.
- spends maximum of 100 USD per transaction.
- spends maximum of 100 USD per day.
- makes transactions only with a minimum gain of 2%
- simulates the arbitrage position and the corresponding gain added to our portfolio value.

6.2.2 Results

The evolution of the portfolio value (Figure 68) leads to a total outcome of about 3500 USD in October 2017 that is a 250% return in less than two years with no risk! By relaxing some of our constraints we could obtain a still better performance. Of course, there are practical issues in order to make this arbitrage a real strategy; instead for many exchanges, it is not possible to do real-time payments from a selected bank account so we need to store a certain amount of USD in all the exchanges we are wishing to invest on. Further the Bitcoin trading is not

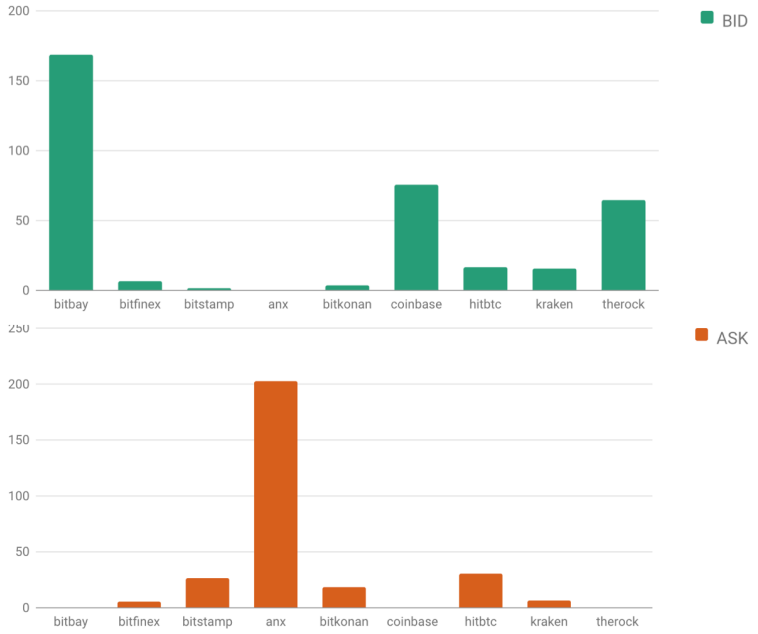

Figure 67: Number of times that BID has been greater and ASK lower.

instantaneous, but need many minutes to be completed. Now, it is also possible to make immediate money transfers[5] with an average cost of to 2 USD, so if we subtract these costs to our profit we still have 2000 USD.

6.3 The speculative bubble in BTC/USD rates

By simply watching the trajectory of the BTC/USD exchange rate time series it is easy to notice how often its pattern surges and bursts rapidly mimicking the one of speculative bubbles. The definition of speculative bubble, considered in this chapter, is the one proposed by Blanchard [Bla79] in the framework of rational expectations models where it is assumed that the economic variable of

[5] `https://www.europeanpaymentscouncil.eu/what-we-do/sepa-instant-credit-transfer`.

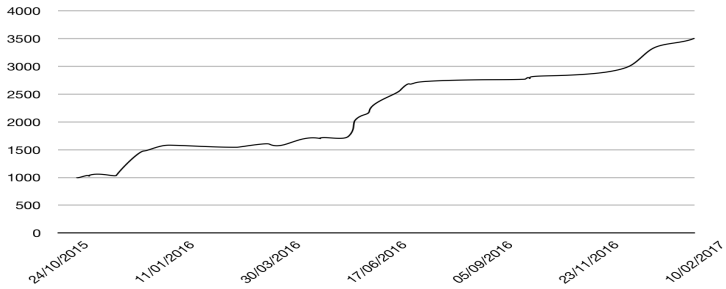

Figure 68: Gain (in USD) over time.

interest, say x_t, has two components: the first one depicts the fundamental path of x_t, while the second represents the bubble effect. In this context a bubble results from the departure of x_t from it's *fundamental path*. In Figure 69 one of the major bubbles occurred in 2013 for the BTC/USD rate is recorded.

Bitcoins are produced through a "mining" process which involves computers (nodes from now on) solving complex mathematical problems (cryptography) to keep the system secure; when the node find a solution to the problem it is rewarded with an amount of Bitcoins which is referred to as "Block reward". The protocol running Bitcoin is programmed to halve every 4 years the "Block reward" by suitably increasing the difficulty of the mathematical problems to be solved. Hence, the volume of new coins will decay to zero with time and the long-term monetary supply will be fixed.

This chapter aims at investigating whether the peculiar "deflationary" mechanism running the system's monetary issuance is the main responsible of the formation, and the subsequent crash, of speculative bubbles (against the USD and other currencies). Indeed economic agents, before undertaking any action within the system, already include the system's monetary issuance in their preferences/expectations, *i.e.* they already know that monetary issuance will never diverge from their expectations inasmuch the system has a unalterable monetary policy programmed to ever decrease the monetary issue over time. Therefore as the system grows (think of it as the Gross Domestic Product of a national economy) the demand of Bitcoins will increase, boosting upwards its price against other traditional currencies, given the *ex-ante* fixed monetary supply.

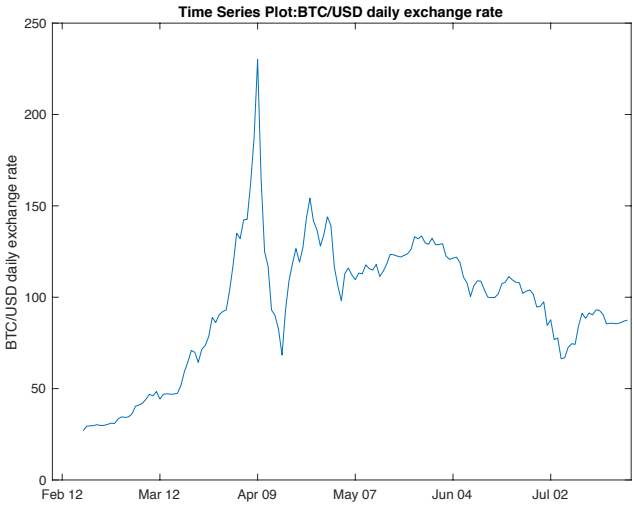

Figure 69: Bitcoin/USD observed time series

The reason why this issuance mechanism is hereby defined "deflationary" is that as long as the general belief that the system as a whole will keep growing stands, the price against other currencies will inevitably increase, increasing the inter-temporal opportunity cost of spending any given amount of BTC. As a matter of fact since the agents know that the price will increase then they are encouraged to withhold any transaction in BTC and increase their *savings* in BTC. A very interesting effect of such mechanism is that any steep fall in the price may boost the awareness of BTC as a system, potentially increasing it's diffusion among the general public, thus incrementing the aforementioned self–sustaining dynamic [Kri15].

It must be noticed that if the system had a flexible monetary policy, where changes are not known *ex-ante*, then the economic agents within and without the system wouldn't be able to include it in their preferences, thus neutralizing the aforementioned self–sustaining mechanism, even if the Bitcoin system is flourishing. After the explanation given above it must be clear now the reason why it is expected and tested below in this study that the speculative bubble in

BTC/USD rates is a *forward–looking* phenomena.

6.4 Mixed causal-noncausal autoregressive models

For a long time, as mentioned by Gouriéroux & Hencic [HG15], speculative bubbles were considered as nonstationary phenomena and treated similarly to the explosive, stochastic trends due to unit roots. Gouriéroux & Zakoian [GZ13] propose a different approach and assume that the bubbles are rather short-lived explosive patterns caused by extreme valued shocks in a noncausal, *stationary* process. In particular they assume a noncausal AR(1) (Auto Regressive) model, strictly *forward looking*, with Cauchy distributed errors.

A useful feature of such models is that shocks are non-fundamental, combining this trait with the extended time dependance (to the future) allows these models to perfectly fit the peculiar pattern of the aforementioned (definition of) speculative bubble.

6.4.1 Introduction to noncausality

Let y_t be the observed time series onto which estimate the traditional autoregressive model:

$$\begin{aligned} a(L)\,y_t &= \varepsilon_t \\ (1 - a_1 L - \cdots - a_p L^{-p})\,y_t &= \varepsilon_t \end{aligned} \tag{6.8}$$

with L being the backshift operator, *i.e.*, $L y_t = y_{t-1}$ gives lags and $L^{-1} y_t = y_{t+1}$ produces leads and a the autoregressive parameters. It is known [LS08] that if s out of p of the polynomial's ($a(L)$) *zeros* are inside the unit circle, then the model is *non stationary* causing the impossibility to estimate the traditional autoregressive model. ε_t represent the usual error term of the model.

In Lanne & Saikkonen [LS08] it is shown that when $p = r + s$, with r being the zeros outside the unit circle, one can factor the polynomial $a(z)$ as[6]:

$$a(z) = \varphi^*(z)\,\phi(z) \tag{6.9}$$

where $\phi(z)$ is the usual causal polynomial of the autoregressive parameters and $\varphi^*(z)$ has its zeros inside the unit circle.

[6] In order to maintain the same notation as in Lanne & Saikkonen [LS08] the polynomial $a(L)$ will be referred to as $a(z)$ for the following proof.

The polynomial $\varphi^*(z)$ can be expressed as:

$$\begin{aligned}\varphi^*(z) &= 1-\varphi_1^* z-\cdots-\varphi_s^* z^s \\ &= -\varphi_s^* z^s\left(1+\tfrac{\varphi_{s-1}^*}{\varphi_s^*} z^{-1}+\cdots+\tfrac{\varphi_1^*}{\varphi_s^*} z^{1-s}-\tfrac{1}{\varphi_s^*} z^{-s}\right) \\ &= -\varphi_s^* z^s\, \varphi(z^{-1})\end{aligned} \tag{6.10}$$

where $\varphi(z^{-1}) = 1-\varphi_1 z-\cdots-\varphi_s z^s$ in view of the fact that $\varphi_{s-j}^*/\varphi_s^* = -\varphi_j$ for $j = 1,\ldots,s$ and $1/\varphi_s^* = \varphi_s$.

Because the zeros of $\varphi^*(z)$ lie inside the unit circle those of $\varphi(z)$ lie outside of the unit circle. Thus, (6.8) can be written as:

$$\phi(L)\,[-\varphi_s^* L^s\, \varphi(L^{-1})] = \varepsilon_t$$

given the decomposition shown in (6.10). Also, the latter expression can be rearranged as:

$$\phi(L)\,\varphi(L^{-1})\, y_t = \varepsilon_t \tag{6.11}$$

where $\varepsilon_t = -(1/\varphi_s^*)\, L^{-s}\varepsilon_t = -(1/\varphi_s^*)\,\varepsilon_{t+s}$. It is important to notice that $E_t[\varepsilon_t] \neq \varepsilon_t$ since this variable is not determined by any informations available at time point t (see above).

6.4.2 Mixed Causal-Noncausal Autoregressive Model

The univariate mixed causal-noncausal autoregressive model, denoted MAR(r,s), shown with equation 6.11 is usually written as:

$$(1-\phi_1 L-\cdots-\phi_r L^r)(1-\varphi_1 L^{-1}-\cdots-\varphi_s L^{-s})\, y_t = \varepsilon_t \tag{6.12}$$

When $\varphi_1 = \cdots = \varphi_s = 0$, the process y_t represent a purely causal autoregressive process denoted AR$(r,0)$:

$$(1-\phi_1 L-\cdots-\phi_r L^r)\, y_t = \varepsilon_t \tag{6.13}$$

where y_t is regressed on past values, giving the process y_t a *backward looking* autoregressive dynamic.

The process y_t is *purely noncausal* when $\phi_1 = \cdots = \phi_r = 0$, hence defined as:

$$(1-\varphi_1 L^{-1}-\cdots-\varphi_s L^{-s})\, y_t = \varepsilon_t. \tag{6.14}$$

usually referred to as *forward looking* AR$(0,s)$ process, being the exact counterpart of the model specification given in (6.13), since it's regressed on future values rather than past ones.

Models containing both lags and leads of the dependent variable are called *mixed causal–noncausal models*.

Assuming that the roots of the causal and noncausal polynomial are outside the unit circle, that is:

$$\phi(z) = 0 \qquad \text{per} \quad |z| > 1 \qquad \text{e} \qquad \varphi(z) = 0 \qquad \text{per} \quad |z| > 1 \tag{6.15}$$

than these conditions imply that the series y_t admits a two-sided Moving Average (MA) representation:

$$y_t = \sum_{j=-\infty}^{\infty} \psi_j \, \varepsilon_{t-j} \tag{6.16}$$

such that $\varphi_j = 0$ for all $j < 0$ implies a purely causal process t and a purely noncausal model when $\varphi_j = 0$ for all $j > 0$ [LS11]. More in detail, the ψ_j's are the coefficients of an infinite order polynomial in positive and negative powers of the Lag operator and such that $\Xi(z) = \sum_{j=-\infty}^{\infty} \psi_j z^j = [\Psi(z^{-1})]^{-1} [\Phi(z)]^{-1}$.

Error terms ε_t are assumed *iid* non-Gaussian with $E(|\varepsilon_t|^{\delta}) < \infty \; \forall \; \delta \in (0,1)$ [GZ13]. Following Gouriéroux & Jasiak [GJ16] the unobserved causal and noncausal components of the process y_t are defined as follows:

$$u_t \equiv \phi(L) y_t \leftrightarrow \varphi(L^{-1}) u_t = \varepsilon_t, \tag{6.17}$$

$$v_t \equiv \varphi(L^{-1}) y_t \leftrightarrow \phi(L) v_t = \varepsilon_t \tag{6.18}$$

The specification of these values will prove useful for the following part regarding the estimation of mixed causal-noncausal processes.

The non-Gaussianity assumption for the error term ensures the identifiability of the causal and the noncausal part. Most papers by Lanne & Saikkonen et al. use Student's t_ν distributions, with $\nu \geq 2$ while Gouriéroux et al. rely on the Cauchy or a mixture of Cauchy and Normal distributions. As shown by Hecq et al. [HLT16] it emerges that the Cauchy has too strong fat tails features and many series would have a degree of freedom between 1.5 and 2.5[7].

[7]Notice that when $\nu < 2$ then the Student's t expected value is undefined.

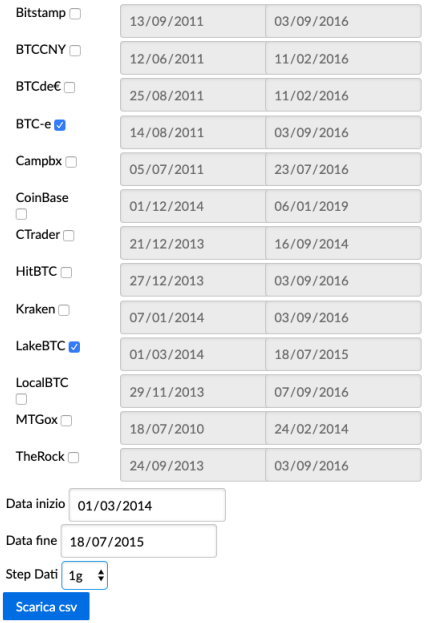

Figure 70: TradeBitcoin data price download panel

6.5 The Data

The sample consists in 151 observation of the BTC/USD price spanning from February 20 to July 20 2013. The dynamic of the data is shown in Figure69, where it is possible to notice the speculative bubble behaviour of the BTC/USD path, boosting and bursting rapidly around the month of April. In fact, in the April 2013 there was a famous bubble, commonly called simply the April bubble, that was a rally, all-time high and subsequent crash of the bitcoin exchange rate. The bubble resulted in a momentary all-time high of $266 USD per bitcoin on Mt. Gox[8] on 10th April 2013. Then Mt. Gox suspended trading on 11th April 2013 until 12th April 2013 2 am UTC for a "market cooldown". The value of a single bitcoin fell to a low of $55.59 after the resumption of trading before stabilizing above $100[9] (a price decline of 61%).

The data is obtained from TradeBitcoin. It also collects all this data price from 17 different exchanges and it allows to download that data with a detection time of 1 day or 1 hour or 15 minutes (Figure 70).

6.5.1 Price decomposition

As a first issue it is important to disentangle the fundamental component from the bubble component of the BTC/USD prices. The fundamental value of the Bitcoin is still under debate. While in [Dwy15] it is argued that this fundamental value is zero, in [Gar+14] it is linked to the reputation of the Bitcoin system measured by internet queries, moreover it is suggested (still in [Gar+14]) that the production cost of Bitcoin, due to the mining process, should be considered as the lower limit of the fundamental value of Bitcoin. Since this study assumes that Bitcoin has a fundamental value indeed, the price will be firstly decomposed following the approach in [HG15], where the fundamental path of the BTC/USD rate is assumed to be a nonlinear deterministic trend modelled as a 3rd degree polynomial in time and the bubble part is obtained by subtraction from the observed prices, as it is done still in [HG15]. The other decomposition that will be undertaken builds upon the suggestion made in [Gar+14], by setting

[8] https://en.bitcoin.it/wiki/Mt._Gox.

[9] https://bitcoincharts.com/charts/mtgoxUSD#rg5zczsg2013-04-10zeg2013-04-12ztgSzm1g10zm2g25zl.

apart the production cost of Bitcoin and the bubble component using the cost of production model shown in [Hay17].

Nonlinear deterministic trend

As mentioned above the BTC/USD rate is defined as follows:

$$rate_t = trend_t + y_t, \tag{6.19}$$

with $rate_t$ being the observed prices, $trend_t$ the fundamental component and y_t the bubble component and the estimated trend is given by:

$$trend_t = 0.000073t^3 - 0.0316t^2 + 3.6590t - 3.2951.$$

The corresponding time-series are plotted in Figure 71.

Production cost as the lower limit of the fundamental value

The Bitcoin production cost model shown in [Hay17] assumes the perspective of a generic *miner* that is deciding whether to mine or not for Bitcoin. The *miner* will decide to join the mining process in case of positive profit expectations and to abandon it on the contrary case. The variables considered to be influencing the mining process and hence the production cost in [Hay17] are: the *block reward* β, the *hashing power* (computational power) of the mining hardware equipment ρ, the *difficulty* set by the network δ, the *cost* per kilowatt-hour[10] $\$kW/h$ and the *average energy efficiency* WGH/s of the mining hardware deployed.

As shown in [Hay17] the expected number of cryptocurrency coins to be mined per day on average given the difficulty and block reward (number of coins issued per successful mining attempt) per unit of hashing power is given by:

$$BTC/day = \frac{\beta\, \rho\, sec_{hr}}{\delta\, 2^{32}}\, hr_{day}$$

sec_{hr}= 3600 being the seconds in 1 hour and hr_{day}= 24 being the hours in a day.

The cost of mining can be expressed as:

$$E_{day} = (\rho/1000)\,(\$kW/h\ WGH/s\ hr_{day})$$

[10]In this study we consider the same average cost for electricity that is considered in [Hay17], although it must be noticed that it changes depending on the geographical location of the miner and therefore on the national electricity supplier.

with $\$\,kW/h$ being the electricity cost and $W\,GH/s$ the average energy efficiency. Bitcoin production cost estimates over the considered time span (Feb.-Jul. 2013) are shown in Figure 72, where it is assumed an average energy efficiency of $W\,GH/s = 500$ as suggested by Garcia et al. [Gar+14], a computational power[11] of $GH/s = 1000$, an average global electricity cost of $\$\,kW/h = 0.115$. In 2013 the block reward set by the network was $\beta = 25\,BTC$, the values of the ever changing difficulty over the considered time-period can be found in the public database `https://blockchain.info`.

Assuming the lower limit of the fundamental value, given by the aforementioned definition of production cost, as the actual fundamental value, therefore the BTC/USD rate is defined similarly as in 6.19:

$$rate_t = cost_t + yc_t, \tag{6.20}$$

with $rate_t$ being the observed prices, yc_t the bubble component and where the fundamental component in this case is given by the aforementioned production cost $cost_t$. Assuming that the production cost is correctly estimated, it must be noticed that the bubble component could be considered as the *added market value*.

The corresponding time series are plotted in Figure 72

6.6 Estimated models

The following part of the study undertakes a mixed causal/non-causal analysis by estimating MAR models on the BTC/USD price and on the bubble component according to the two different definitions of the fundamental part. As already discussed in the introduction it is expected that the *forward looking* dependence is stronger in the isolated bubble component than in the observed price. The model specifications in what follows are chosen by applying *information criteria* which are useful tools to select the number of lags (and leads) to be included in the model. The information criteria hereby considered are the *Akaike*

[11] It must be noticed that in this example varying the computational power does not change the cost, *i.e.*, the cost of 1 BTC in USD is only affected by the difficulty, the electricity cost and the average energy efficiency of the mining hardware, increasing the scale of production in this case doesn't lead to economies of scale.

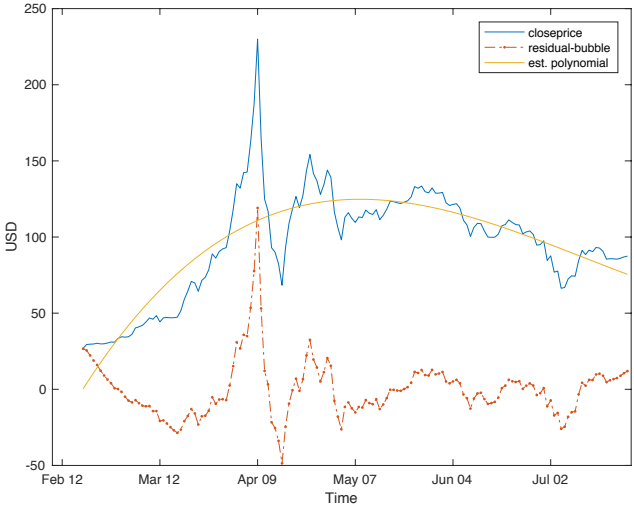

Figure 71: Bitcoin/USD price decomposition

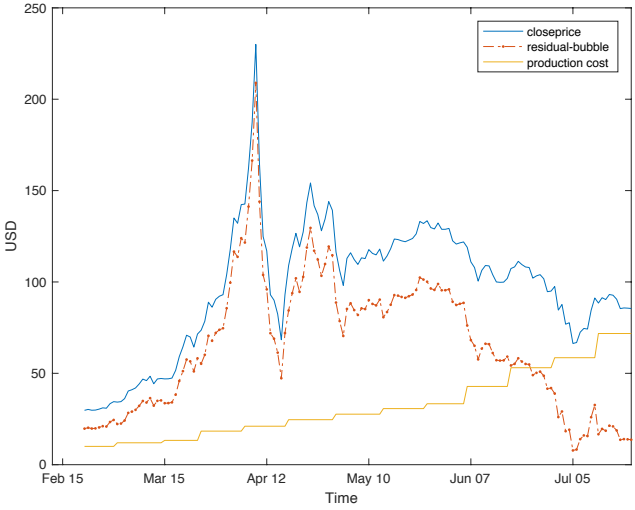

Figure 72: Bitcoin/USD price vs. production cost

information criterion (AIC), the *Bayesian information criterion* (BIC) and the *Hannan-Quinn information criterion* (HQ) (for a general review see the book by Hamilton [Ham94]). Once the number of lags/leads have been detected, models are estimated by maximizing the approximated log-likelihood function based on the Student's t density function for the error term; a detailed description of the procedure may be found in Hecq et al. [HLT16]. The related Matlab routines used in this work are kindly provided by the authors of the above quoted paper.

Table 11: AR(1) model's estimated parameters

AR(1) Model		t **distribution**	
φ_1	Std. Dev.	λ	ν
0.8066	0.0234	4.3928	2.5013

Table 12: BDS test results, purely noncausal model AR(1).

m	w	$p-value$	m	w	$p-value$
2	5,978547545	1,13E-09	9	13,3666165	0
3	6,525463574	3,39E-11	10	14,65204326	0
4	7,420797806	5,82E-14	11	15,91260972	0
5	8,615265114	0	12	17,42915916	0
6	9,743131337	0	13	19,3469674	0
7	10,83386529	0	14	21,49415105	0
8	12,07560832	0	15	24,05813227	0

6.6.1 Noncausal analysis of the bubble component

Firstly is considered a *strictly noncausal* AR(1) (forward looking):

$$y_t = \varphi_1 y_{t+1} + \varepsilon_t .$$

where ε_t are *iid* Student's t distributed errors, with location 0 and scale parame-

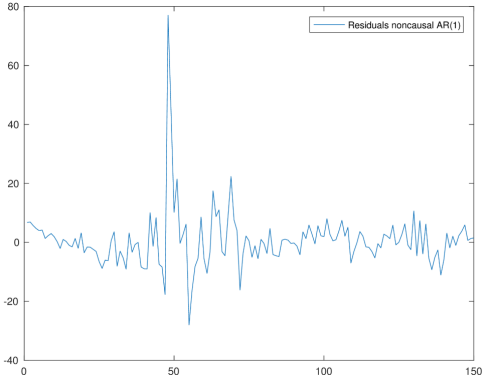

Figure 73: Noncausal AR(1) model residuals

ter λ, $\varepsilon_t \sim (0, \lambda)$. Estimated parameters are reported in Table 11. The residuals of the models are shown in Fig 73. In order to test the model's goodness of fit, the results of the BDS test (Brock, William, Davis Dechert & Scheinkman, 1987) [Kan99], used to test whether the residual are truly a sequence of *iid* Student's t random variables, are reported in Table 12. The test fails to accept the null hypothesis of *iid* distributed residuals, this implies that the present model must be discarded.

Table 13: Information Criteria

p	BIC	AIC	HQ	p	BIC	AIC	HQ
0	6,0119	5,9565	5,9649	5	4,8862	4,5537	4,6042
1	4,7057	4,5949	4,6117	6	4,914	4,526	4,585
2	4,6823	4,5161	4,5413	7	4,9494	4,506	4,5734
3	4,7513	4,5296	4,5633	8	4,987	4,4882	4,5639
4	4,818	4,5409	4,583				

Table 14: MAR(1,1) estimated parameters.

Parameter	Estimate	Confidence bounds	
ϕ_1	0.5255	0.4585	0.5925
φ_1	0.6503	0.5897	0.7110

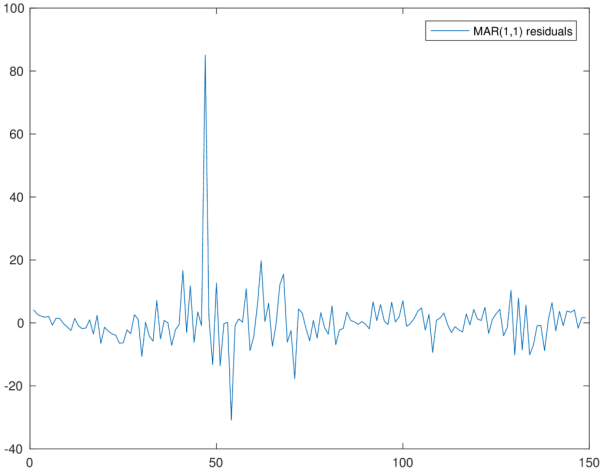

Figure 74: Mixed causal-noncausal MAR(1,1) residuals

Table 15: BDS test results for the MAR(1,1) model residuals on y_t.

m	w	$p-value$	m	w	$p-value$
2	1,703389269	0,0442	9	6,924875969	2,18E-12
3	2,249277819	0,0122	10	7,294838916	1,50E-13
4	3,342684301	4,15E-04	11	7,639695322	1,09E-14
5	4,384330303	5,82E-06	12	8,143710723	2,22E-16
6	5,191782337	1,04E-07	13	8,892820344	0
7	5,887119414	1,96E-09	14	9,500629834	0
8	6,412692984	7,15E-11	15	10,18321732	0

Mixed causal-noncausal AR model

The following specification of the model is derived by the suggestions of the *information criteria*, these are very useful tools to determine the time dependencies to be included in the model, *i.e.* they are used to determine the order of the autoregressive polynomial p (see equation 6.8). The information criteria hereby considered are the *Akaike information criteria* AIC, the *Bayesian information criteria* and the *Hannan-Quinn* information criterion HQ [HLT16], Hecq et al. [HLT16] show that simulation results would favour the use of BIC. As reported in Table 13 the information criteria suggest setting $p = 2$.

When $p = 2$ the estimated *Mixed causal-noncausal* model is a MAR(1,1):

$$(1 - \phi_1 L)(1 - \varphi_1 L^{-1}) y_t = \varepsilon_t .$$

Table 14 shows the estimated parameters of the model. Figure 74 shows the sequence of the MAR(1,1) model's estimated residuals $\hat{\varepsilon}_t$.

As shown in Table 15, the BDS test for independence fails to accept the null hypothesis of *iid* distributed residuals for most of the tested *embedded dimensions*, thus suggesting to discard the model just now estimated.

Table 16: Information Criteria, MAR model on $rate_t$

p	BIC	AIC	HQ	p	BIC	AIC	HQ
0	7,0358	6,9803	6,9888	5	4,9587	4,6262	4,6767
1	4,7483	4,6374	4,6543	6	5,0137	4,6257	4,6847
2	4,7546	4,5883	4,6136	7	5,0781	4,6348	4,7021
3	4,8215	4,5998	4,6335	8	5,0705	4,5716	4,6474
4	4,8906	4,6135	4,6556				

6.6.2 Noncausal analysis of the observed price BTC/USD

Since the interpretation of the aforementioned estimated parameters can be rather misleading and therefore hard to be extended to the market reality, given the arbitrary choice for the fundamental component, it is of great interest to estimate the MAR model directly on the BTC/USD time series ($rate_t$).

Table 17: MAR(0,1) estimated autoregressive parameters on $rate_t$

Parameter	Estimate	Confidence bounds	
ϕ_1	0.9809	0.9740	0.9878

Table 18: Estimated parameters of the Student's t error distribution with MAR(0,1) model on $rate_t$

λ	ν
3.3073	1.4863

Table 19: BDS test results for the MAR(0,1) model residuals

m	w	H_0	m	w	H_0
2	6,45069	1	9	16,22165	1
3	8,26635	1	10	18,05194	1
4	9,30074	1	11	20,14559	1
5	10,38450	1	12	22,50443	1
6	11,66036	1	13	25,59976	1
7	13,03694	1	14	29,31705	1
8	14,59404	1	15	33,57504	1

Table 20: MAR(1,1) estimated autoregressive parameters on $rate_t$

Parameter	Estimate	Confidence bounds	
ϕ_1	0.9747	0.9650	0.9844
φ_1	0.2781	0.2077	0.3485

Table 21: Estimated parameters of the Student's t error distribution with MAR(0,1) model on $rate_t$

λ	ν
3.3901	1.6043

Table 22: BDS test results for the MAR(1,0) model residuals on $rate_t$.

m	w	$p-value$	m	w	$p-value$
2	6,450686379	5,57E-11	9	16,22165279	0
3	8,266350334	1,11E-16	10	18,05194201	0
4	9,300742867	0	11	20,14558908	0
5	10,38450476	0	12	22,50443038	0
6	11,66035838	0	13	25,5997597	0
7	13,03694212	0	14	29,31704918	0
8	14,59403959	0	15	33,57504129	0

The aforementioned information criteria, in application to the BTC/USD time series, suggest to set the order of the autoregressive polynomial to $p = 1$ (see Eq. 6.8) or $p = 2$ depending on the selected criterion (see Table 16).

Estimated MAR model, case $p = 1$

When $p = 1$ the estimated model that best fits the observed time series $rate_t$ is a *purely causal* AR(1,0). Table 17 and Table 18 display the estimated parameters of the autoregressive polynomial and of the error distribution, respectively.

Since the distribution's degrees of freedom $\nu = 1.4863 < 2$, then the estimated sequence of error terms $\hat{\varepsilon}_t$ cannot be likened to the case $\varepsilon_t \sim iid\ t_\nu(0,\lambda)$, given the fact that when $\nu < 2$ the expected value of the distribution is not defined. Anyhow the BDS test (Table 22) for independence does not accept the null hypothesis of *iid* distributed residuals, thus suggesting once again to discard the estimated model.

Table 23: BDS test results for the MAR(1,1) model residuals on $rate_t$.

m	w	$p-value$	m	w	$p-value$
2	5,121150647	1,52E-07	9	15,28926444	0
3	7,537452115	2,40E-14	10	16,97636905	0
4	8,912767967	0	11	18,98947641	0
5	10,07953357	0	12	21,39229563	0
6	11,15351875	0	13	24,08437913	0
7	12,36324983	0	14	27,23839452	0
8	13,78408522	0	15	30,78448895	0

MAR model, p=2

When $p = 2$ the estimated model is a *Mixed causal-noncausal* MAR; estimates of the model are displayed in Table 20 and 21. Once again the estimated t distribution's degrees of freedom is $\nu = 1.6043 < 2$, therefore the estimated sequence of error terms cannot be likened to the case $\varepsilon_t \sim iid\ t_\nu(0,\lambda)$, suggesting to discard the model once again. In any case, the BDS test(Table 22) for independence does not accept the null hypothesis of *iid* distributed residuals, thus suggesting once again to discard the estimated model.

6.6.3 Residual analysis

To sum up, MAR models are estimated directly on the BTC/USD time series ($rate_t$) and then on the bubble part (y_t, yc_t); the aforementioned information criteria suggest to set the order of the autoregressive polynomial to $p = 1$ or $p = 2$ depending on the selected criterion, both for the BTC/USD rate and for the bubble terms. In the former case, $p = 1$, a *strictly causal backward looking* AR(1) is the preliminary reference specification for both the full rate $rate_t$ and the bubble component yc_t whereas a *strictly non-causal forward looking* AR(1) is the preliminary reference for the bubble component y_t. For the latter case, $p = 2$, a MAR(1,1) model is found to be fitting all the time series $rate_t$, y_t and yc_t.

The estimation results are reported in Table 24.

Table 24: MAR(r,s) estimated parameters on $rate_t$, y_t & yc_t

Time Series	MAR(r,s)	Parameter	Est.	Conf. bounds		Parameter	Est.	Conf. bounds	
$rate_t$	MAR(1,0)	ϕ_1	0.9809	0.9740	0.9878	φ_1	-	-	-
	MAR(1,1)	ϕ_1	0.9747	0.9650	0.9844	φ_1	0.2781	0.2077	0.3485
y_t	MAR(0,1)	-	-	-	-	φ_1	0.8066	0.8028	0.8103
	MAR(1,1)	ϕ_1	0.5255	0.4585	0.5925	φ_1	0.6503	0.5897	0.7110
yc_t	MAR(1,0)	ϕ_1	0.9803	0.9702	0.9904	φ_1	-	-	-
	MAR(1,1)	ϕ_1	0.3424	0.2604	0.4245	φ_1	0.9396	0.9216	0.9576

It is evident from the results in Table 24 that there is a very strong *backward looking* dependence in one lagged value, for the BTC/USD rate and for the bubble component yc_t; conversely, for the isolated bubble term y_t, there is a very strong *forward looking* dependence in one led value.

The estimation of a *Mixed causal/non-causal* MAR(1,1) gives further insights on the *backward* and *forward* dependence; outcomes are summed up in Table 24 respectively for the full rate $rate_t$, the bubble term y_t and the bubble term yc_t.

Particularly interesting is the difference in the parameter ϕ_1 and φ_1 when estimating the MAR(1,1) model separately on the bubble component y_t and on the original time series $rate_t$. As shown in Table 24 the non-causal parameters (*forward looking*) φ are stronger in the bubble component y_t than in the observed price $rate_t$, whereas the causal parameter ϕ is much stronger in the observed price $rate_t$ than in the bubble component y_t. This is consistent with the conjecture made in the introduction, that the speculative bubble is rather a forward looking phenomena than a past one, since the *forward looking* estimated parameters on the bubble part are stronger than the ones on the observed BTC/USD price $rate_t$. This evidence is strengthen by the MAR(1,1) estimated parameters on the bubble part yc_t, indeed it can be noticed that the value of the forward looking and backward looking components almost trade places when estimating the model on the full price time series $rate_t$ and the bubble component yc_t respectively.

As mentioned in Section 6.4.2, if the model is correctly specified then the model residuals ε_t should be a sequence of *Independent Identically Distributed*

Table 25: BDS test results for the MAR(1,0) model residuals on yc_t.

m	w	$p-value$	m	w	$p-value$
2	5,528256527	1,62E-08	9	12,29309902	0
3	7,009012288	1,20E-12	10	13,36389728	0
4	7,773551059	3,77E-15	11	15,15913039	0
5	8,506760111	0	12	17,1200305	0
6	9,287013257	0	13	19,26464437	0
7	10,17111148	0	14	21,78528929	0
8	11,24555355	0	15	24,56128819	0

Table 26: BDS test results for the MAR(1,1) model residuals on yc_t.

m	w	$p-value$	m	w	$p-value$
2	4,236007075	1,14E-05	9	10,32346963	0
3	5,315716933	5,31E-08	10	11,13612446	0
4	6,272423899	1,78E-10	11	12,51271091	0
5	7,320422719	1,24E-13	12	14,01078341	0
6	8,132958153	2,22E-16	13	15,68841368	0
7	8,838578863	0	14	17,59084596	0
8	9,612705164	0	15	19,49717776	0

Student's t observations. In this study the *IID* hypothesis is tested through the BDS test for independence. This test is based on the correlation dimension, with *m embedded dimension*, since it can be shown [Kan99] that the test statistic w is asymptotically normally distributed $\sim \mathcal{N}(0,1)$, it is quite feasible to obtain $p-values$. The Tables 25 and 26 reporting the outcome of performing such test on the residuals ε_t of the yc_t estimated models. As shown in Table 15 it can be noticed that the only model for which the null hypothesis of *IID* residuals cannot be rejected is the MAR(1,1) model on y_t, and only for $m = 1$ or $m = 2$, depending on the selected confidence bound width.

6.7 Conclusions

In this chapter, we prove both via a theoretical model and via an empirical strategy that arbitrage opportunities are indeed possible by trading on different Exchanges. This study also undertook a Mixed causal-noncausal analysis of the BTC/USD exchange rates time series, over the period February-July 2013, to test whether the bubble effect disentangled on observed data may be explained by a forward looking behaviour of the economic agents. The conjecture underlying this study is that the forward looking parameters should be stronger in the bubble part than in the observed price. Indeed this turns out to be the case, when estimating the model on the observed data, however the residuals analysis, conducted by performing the BDS test for independence, suggests not to consider this models valid but for one case.

Chapter 7

Stochastic Modelling and Analysis of the Bitcoin Protocol

The implementation of a distributed ledger has to solve a well-known critical issue: the inconsistency of updates that are performed by different nodes. This problem, which has been demonstrated unsolvable in 1985 [FLP85], is overcome in the Bitcoin protocol by using a probabilistic approach where the probabilities depend on the rate of the addition of new blocks – called *mining* – and on the delays of broadcasts. Additionally, to further reduce the probabilities of inconsistencies in the copies of the ledger, Bitcoin guarantees the so-called *eventual consistency* whereby the various replicas may be temporarily inconsistent in at most the last m blocks. In particular, Bitcoin users generally consider as "*confirmed*" (and therefore "*can be paid*") every block that is at a depth greater than m ($m = 5$ in the current protocol [Gra20]).

It turns out that the Bitcoin protocol is complex and many researchers are actively involved in studying its properties and its criticalities. Clearly, understanding the details of the protocol is of paramount importance because overlooking some of them might introduce vulnerabilities and pave the way to attacks. For example, inconsistencies of the ledger replicas, called *forks*, occurring when there are two or more blocks at the same height of the ledger, may be used

for rewriting the transaction histories and for letting the blockchain evolve to a wrong state. A typical example of wrong state is a state where a transaction is paid twice – called *double spending attack*.

In this Chapter, we analyze the probabilistic behaviour of the Bitcoin protocol by using formal methods. In particular, following the approach of Pass et al. [PSS17] and of Gramoli [Gra20], we use an abstract model that defines the network of nodes as the *parallel composition* of processes – the *miners* – and the time needed to mine a block and to broadcast a message as an *exponential distribution* with a *rate parameter* associated to process actions. As done by Biais et al. [Bia+19], by Eyal and Sirer [ES14] and by Zamyatin et al. [Zam+18], we use Continuous Time Markov Chains (CTMCs) for providing a probabilistic model of our processes. To examine the probability of reaching a state of fork in different settings, we analyse our model with a probabilistic model checker. In our model, we consider every detail of the real implementation of the Bitcoin protocol that influences the probability of reaching a state of fork. The details of the protocol that do not impact on our analysis are ignored. The overall aim of our analysis is to study the resilience of the protocol by varying some parameters such as network topology, the presence of nodes that leave and join the network dynamically.

The formal definition of the abstract model of Bitcoin is described using PRISM [KNP02], a process calculus with a probabilistic and stochastic semantics. This framework has been chosen because it includes an automatic model checker that enable us to perform our analyses. Actually, in order to deal with the complex data types used by Bitcoin, namely `ledger`, `block` and `set`, and with the operations upon them, we extend PRISM and introduce the richer variant PRISM+ that enables us to naturally model these concepts. Implementing PRISM+ has required a significant programming effort in order to make the foregoing data types native. With PRISM+, we can analyze several models of the Bitcoin protocol and measure the impact on probabilities obtained by tuning, up or down, the rates of mining and that of the delays of broadcasts.

As an initial step, we assess the coherence of our model by verifying that the probabilities of mining a new block within a given amount of time and the probability of having a fork are both in full agreement with the values available from the literature [DW13]. In particular, we confirm that the probability

of a fork strictly depends on the broadcast delay and on the difficulty of the cryptopuzzle, i.e. the difficulty of the computational problem that miners must solve to be enabled to add a new block to the ledger. We also corroborate the statement [Gra20] that waiting for at least 6 confirmations before considering a transaction as permanent in the ledger is reasonable, because at that time the majority of miners has a consistent blockchain with probability 10^{-12}.

After validating our model, we study the trade-off between the security guarantees and the difficulty of the cryptopuzzle. More precisely, we analyze the variability of the probability of reaching fork states when the speed of the mining process increases, that is when the difficulty of the cryptopuzzle decreases. We observe that the speed of mining can be boosted by decreasing slightly the difficulty level at the cost of an almost irrelevant increase of the probability of forking. Obviously, the easier is the cryptopuzzle, the faster the entire system mines a block. We also show that a good balance between speed and safety can be obtained with an average mining rate equals to 1/500. In fact, with this rate, the process of mining a block is faster than in Bitcoin (1/600 [DW13; Bia+19]), but the probability of reaching a state of fork is not much higher.

We also study a network with churning nodes, i.e. nodes that can leave and rejoin the network. In particular, we analyze how the presence of this kind of nodes affects the probability of mining new blocks and, consequently, the probability of forks. Our analyses show that this probability is lower when there are churning nodes in the network. The same considerations stays valid for forks. This is due to the fact that, in our network model, each node is connected to every other node, thus the presence of churning nodes does not lead to message losses.

Finally, we consider several network topologies (daisy topology, round topology, tree topology and fully connected topology) and analyze them in order to estimate how the connections between nodes can affect the likelihood of a fork. Our simulations show that the probability of inconsistency increases when new blocks, instead of being forwarded to all the nodes of the network directly, are transmitted only to a subset of them. Thus, the smaller is the subset of the receiving nodes per miner, the higher is the probability of having a fork. The different topologies have instead no impact on the mining process because the rate of communications (of blocks) is much higher than the mining rate.

We conclude by remarking that our PRISM primitives (`ledger`, `block` and `set`) are generic, namely they are not tied to a particular protocol and can be used for modelling and analysing other blockchain protocols, such as proof of stake and its variants.

The rest of the Chapter is structured as follows. Section 7.1 presents PRISM+, our extension of PRISM with the new data types, `ledger`, `block` and `set`. Our model of Bitcoin is presented in Section 7.2. Section 7.3 contains our simulations and compares the results we obtain when considering different type of networks. Section 7.4 draws some conclusions and discusses possible future work.

7.1 The Extended PRISM Language

To have a faithful abstraction of the Bitcoin protocol we extend the PRISM language with three dynamic data types: `block`, `ledger` and `list`. We call the extended language PRISM+ and, in this section, we overview its main features. Our extension has been implemented in a prototype that the reader can find online at this url[1] together with all the data of our simulations, and the instructions for the installation and the use of tool.[2]

7.1.1 Blocks

As discussed in Section 2, a Bitcoin block records information items such as the transactions, the nonce, a timestamp, the Merkle root and a pointer to its parent. In our model, we ignore the hash value, because we consider all blocks and all transactions valid. In particular, our blocks are either `genesis` or terms of the form $(\mathtt{m}^{\mathtt{n}};\mathtt{p})$, where $\mathtt{m}^{\mathtt{n}}$, called *name*, is such that `m` is a miner's name and `n` is a unique numeric label. The term `p`, called *father*, is the name of another block to which $(\mathtt{m}^{\mathtt{n}};\mathtt{p})$ points. For instance, $(\mathtt{m3}^{0};\mathtt{m4}^{7})$ denotes a block named $\mathtt{m3}^{0}$, which is the first block created by the miner `m3`, and whose father is the block named $\mathtt{m4}^{7}$, which is the seventh block created by the miner `m4`. The block `genesis`, called *genesis block* is the root of every ledger.

[1] `https://github.com/adeleveschetti/bitcoin-analysis`.

[2] A docker image with our prototype is also available at `https://hub.docker.com/repository/docker/meivan/bitprism`

We introduce the following operation for creating blocks:

- `createBlock(m, n, L)`, which returns a block $(\mathtt{m}^{\mathtt{n}},\mathtt{p})$, where `p` is the handle of the ledger `L` (see below).

7.1.2 Ledgers

The `ledger` data type, noted `L`, $\mathtt{L}'$, $\cdots$, is a pair $\langle \mathtt{T};\mathtt{p}\rangle$ where `T` is a tree of blocks and `p` is the name of a leaf block at maximal height, called the *handle* of `L`. If there are several leaf blocks at maximal depth, the pointer is set to the *first received* leaf at maximal depth, precisely following the description of protocol presented by Nakamoto [Nak08b]. The height of a block `b` is the length of path the connects `b` to the genesis block. The root of `T` is the *genesis block*. Given a ledger `L`, the corresponding *blockchain* is the sequence of blocks which starts from the handle and reaches the genesis. When a miner with a ledger $\langle \mathtt{T};\mathtt{p}\rangle$ receives a block `b`, it adds `b` to `T`. Note that this is not always possible because the father of `b` may be not in `T`; in this last case `b` is added to the local set of the miner and is inserted into the blockchain afterwords. In case the depth of `b` in `T` is higher than the one of the block pointed by `p`, the miner updates `p` to point to `b`. When a miner mines a new block, the corresponding father is set to `p` and `p` is updated accordingly. We define the following operations for the new data types:

- `addBlockLedger(L,b)`: the first argument is a ledger $\mathtt{L} = \langle \mathtt{T};\mathtt{p}\rangle$ and the second is a block `b`. The operation adds `b` to `T` if the father of `b` is in `T`. When `b` is at higher depth than the handle `p`, then the handle is updated with the name of `b`;
- `canbeInserted(L,b)`: checks if the block `b` can be inserted in the ledger `L`. Thus, if the father of `b` is already in tree of blocks of `L`, the function returns true, false otherwise.

In Figure 75, we depict the ledger

$$\mathtt{L}{=}\langle\{\mathtt{genesis},(\mathtt{Miner}_1^0;\mathtt{genesis}),(\mathtt{Miner}_1^1;\mathtt{Miner}_1^0),(\mathtt{Miner}_2^0;\mathtt{genesis})\};\mathtt{Miner}_1^1\rangle$$

that has two blocks created by $\mathtt{Miner}_1$ and one created by $\mathtt{Miner}_2$. The handle, which is represented by a green arrow, is the block whose name is $\mathtt{Miner}_1^1$. The

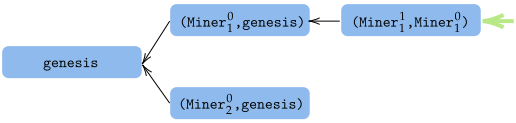

Figure 75: An example of Ledger.

blockchain of $\mathtt{L}$ is the sequence of blocks

$$(\mathtt{Miner}_1^1;\mathtt{Miner}_1^0)\cdot(\mathtt{Miner}_1^0;\mathtt{genesis}^0)\cdot(\mathtt{genesis}^0;\mathtt{genesis}^0).$$

We notice that the block created by $\mathtt{Miner}_2$ is a valid stale block.

A basic notion of ledgers is that of fork: Let $\mathtt{L}_1 = \langle \mathtt{T}_1;\mathtt{p}_1\rangle, \ldots, \mathtt{L}_n = \langle \mathtt{T}_n;\mathtt{p}_n\rangle$ be a set of ledgers and let $\mathtt{m}$ be the maximal height of the handles $\mathtt{p}_1, \cdots, \mathtt{p}_n$. Let also $\mathtt{L}_{i_1}, \ldots, \mathtt{L}_{i_k}$ be the ledgers in the above set with the handle at height m. The set $\mathtt{L}_1, \ldots, \mathtt{L}_n$ has a fork of length $\mathtt{m}-\mathtt{h}$, where h is the length of the maximal common suffix of the blockchains of $\mathtt{L}_{i_1}, \ldots, \mathtt{L}_{i_k}$.

For example, the two ledgers in Figure 76 have a fork of length 1. Our

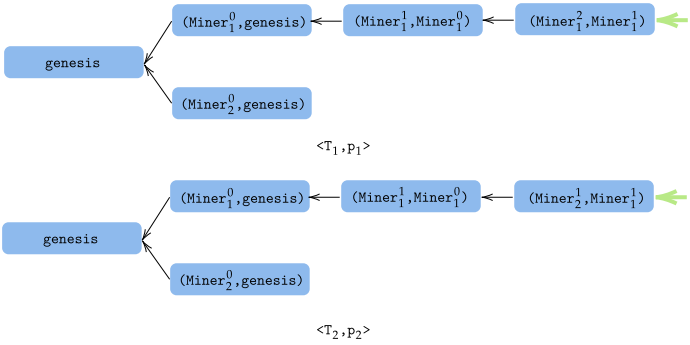

Figure 76: Two ledgers in a state of fork of length 1.

simulator will compute the probability that a fork of a certain length happens. To this aim, we have implemented the operation

- `calculateFork`($\langle \mathtt{T}_1;\mathtt{p}_1\rangle,\ldots,\langle \mathtt{T}_n;\mathtt{p}_n\rangle$): it takes in input a set of ledgers and return the length of the fork; it return 0 if there is no fork.

The function `calculateFork` compares the handles of the ledgers given as input and keeps the blockchains with different handles at maximal height. Then `calculateFork` uses an auxiliary function that recursively iterates over the resulting blockchains *without the leading block*: the length of the resulting fork is obtained by summing 1 to the length obtained from the previous recursive invocation.

7.1.3 Sets

The `set` data type is implemented as a list of blocks without duplication. In our Bitcoin model, this data type is used to maintain, for each miner, the collection of *blocks* to be added to the blockchain. The data type has the following operations:

- the extraction operation `extractBlockSet(S)` which returns a block randomly extracted from the set `S`;
- `addBlock(S,b)` which takes in input a set `S` and a block `b`; it returns $\mathtt{S} \cup \mathtt{b}$. When $\mathtt{b} \in \mathtt{S}$, `addBlockSet(S,b)` = `S`;
- `removeBlock(S,b)` which returns the set $\mathtt{S} \setminus \mathtt{b}$;
- `isEmpty(S)` which returns `true` if the set `S` is empty, `false` otherwise.

7.2 The Bitcoin Model

In our model, a Bitcoin system is the result of the parallel composition of n Miner processes, n *Hasher* processes and a process called *Network*. *Hasher* processes model miners' attempts to solve the cryptopuzzle, while the *Network* process model the broadcast communication among miners. At the beginning, we shall assume that miners are connected through a network guaranteeing broadcast. Later on, we shall consider other topologies (see Section 7.2.3); the actual architecture is illustrated in Figure 77. The abstraction also uses an auxiliary process, called *Global*, that computes the length of forks, see Section 7.3.

As already said, in order to abstract out the solution of the cryptopuzzle and the emission of new blocks, we use rates. It turns out that every process can be modelled as a CTMC because

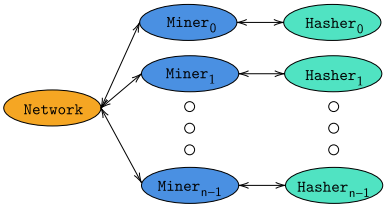

Figure 77: The Bitcoin model architecture.

1. the time spent by a miner m_i to mine a block can be described by an exponential distribution $1 - e^{-\lambda_{m_i} t}$, where the parameter λ_{m_i} depends on the miner hashing power and the difficulty level of the crytopuzzle (see Nakamoto [Nak08b]);

2. the communication delay across the Bitcoin Network can be also approximated by an exponential distribution [DW13].

7.2.1 General Model

For the sake of clarity, we present a simplified version of the PRISM+ code implementing our processes. The actual abstraction, the analyzed properties with tests and the instructions for the use of the library are available on the online repository.

```
1  module Hasher_i
2     Hasher_i_STATE : 0;
3
4     [win_i] (Hasher_i_STATE=0) -> mR : Hasher_i_STATE'=0 ;
5     [lose_i] (Hasher_i_STATE=0) -> lR : Hasher_i_STATE'=0 ;
6  endmodule
7
8  module Miner_i
9     Miner_i_STATE : [Mine,Winner,Lost,Add,Move] init Mine;
10    b_i : block (Miner_i^0;genesis^0);
11    L_i : ledger ⟨{ genesis };genesis ⟩;
12    c_i : [0..100] init 0;
13    setMiner_i : set [];
14
15    [win_i] (Miner_i_STATE=Mine) ->
16                  hR_i : (b_i'=createBlock(Miner_i,c_i,L_i))&(c_i'=
                        c_i+1)&(Miner_i_STATE'=Winner);
17    [lose_i] (Miner_i_STATE=Mine) -> hR_i : (Miner_i_STATE'=Lost);
18
19    [addBlock_i] (Miner_i_STATE=Winner) ->
20                  1 :(L_i'=addBlockLedger(L_i,b_i))&(Miner_i_STATE
                        '=Mine);
21
22    [] (Miner_i_STATE=Lost) & !isEmpty(set_i) ->
23                  1 : (b_i'=extractBlock(set_i))&(Miner_i_STATE'=
                        Move);
24    [] (Miner_i_STATE=Lost) & isEmpty(set_i) ->  1 : (Miner_i_STATE
          '=Mine);
25    [] (Miner_i_STATE=Lost) & !isEmpty(setMiner_i) ->
26                  1 : (b_i'=extractBlock(setMiner_i))&(
                        Miner_i_STATE'=Add);
27    [] (Miner_i_STATE=Lost)& isEmpty(setMiner_i) -> 1 : (
          Miner_i_STATE'=Mine);
28
29    [removeBlock_i] (Miner_i_STATE=Move) ->
30                  1 : (setMiner_i' = addBlock(setMiner_i, b_i))&(
                        Miner_i_STATE'=Mine);
31
32    [] (Miner_i_STATE=Add)&(canbeInserted(L_i,b_i)) ->
33                  1 : (L_i'=addBlockLedger(L_i,b_i)&(setMiner_i'=
                        removeBlock(setMiner_i,b_i))
34                      &(Miner_i_STATE'=Mine);
35    [] (Miner_i_STATE=Add)&(!canbeInserted(L_i,b_i)) ->  1 : (
          Miner_i_STATE'=Mine);
36 endmodule
```

Listing 7.1: Simplified model of the Hasher and a miner.

A Hasher process is defined in Listing 7.1 (lines 1 to 6). It represents the

PoW algorithm performed by miners: those miners who want to solve the cryptopuzzle synchronize with the Hasher which "answers" telling them if they succeeded or not. The Hasher consists of two transitions: the first one with action `[win_i]` and rate `mR` is triggered when the synchronizing miner finds a solution for the PoW (`mR` is taken such that $0 < \texttt{mR} < 1$); the second one with action `[lose_i]` and rate `lR` ($\texttt{lR} = 1 - \texttt{mR}$) is triggered when the synchronizing miner does not find a solution to PoW. In both cases the Hasher process makes a silent action.

A Miner, described in Listing 7.1 lines 8 to 36, has a ledger, called `L_i`, a set containing the blocks to be added to the ledger, called `setMiner_i`, a block `b_i` and a integer `c_i`. The variable `b_i` is used to store the block the Miner creates and to store the newly extracted block from the set. The integer `c_i` is a counter whose value ranges between 0 and 100 (initially is zero) and which keeps track of the number of blocks created by the Miner, so that we can assign unique names to blocks.

It has five states and behaves as follows:

- the initial state is `Mine`. From this state, the miner may win the criptopuzzle and transit into the `Winner` state or lose and transit into the `Lost` state.
- in the `Winner` state, the miner may create (e.g. mine) a new block: in our setting, mining a block amounts to synchronizing with the Hasher on the action `win` (which means "winning the PoW"). This synchronization has a rate `hR_i`, with $0 < \texttt{hR\_i} < 1$, that indicates nodes' computational power. When a block is created by a miner, it is added to the local ledger and it is forwarded to all the other miners of the network, by synchronizing with the Network process.
- in `Lost` state, the miner may receive a block from the network or may try to add blocks stored in his local set to `L_i`.
- the states `Add` and `Move` are used to add a block into `setMiner_i` and into `L_i`, respectively.

The Network process is defined in Listing 7.2. It contains a set of blocks `set_i` for each `Miner_i` that represents the messages to be delivered to the miner.

The set N_i, in the case of the broadcast topology, is equal to $\{0,\cdots,i-1,i+1,\cdots,n-1\}$, e.g. the indexes of the miners to whom a block must be sent (line 9). Also, the Network process contains a transition for each Miner to model sending and receiving messages.

```
1  module Network
2      n : numberOfMiners
3
4      for i from 0 to n-1:
5              set_i : set [];
6              N_i : set [];
7
8      for i from 0 to n-1:
9          [addBlock_i] ->  r_b : foreach k in N_i { set_k'=addBlock(
               set_k,b_i); };
10
11     for i from 0 to n-1:
12         [removeBlock_i] -> 1 : set_i' = removeBlock(set_i,b_i);

13 endmodule
```

Listing 7.2: Simplified model of the Network.

More precisely, the Network synchronizes with the Miner who won the PoW (in the winner state) using the addBlock_i action. As an effect of this synchronization, Network updates the sets of blocks of the miners contained in the set N_i. [3]

Below we describe in detail how our processes abstract the Bitcoin protocol. As the reader can observe from the code in Listing 7.1, Miner's state is initially set to Mine. In this state it can synchronize with Hasher_i using either the win_i or lose_i actions. As already said, this synchronization abstracts the cryptopuzzle solution. Note that the time needed for the creation of a new block at miner i is a random number sampled from an exponential distribution with rate P proportional to the ratio of the difficulty of the PoW problem and the hashing power of the miner. Therefore, the chosen action, win_i or lose_i, depends on the difficulty of the problem (represented by the hasher values mR and lR) and on the hashing power of the miner (represented by hR_i).

Since the rate of a synchronization is equal to the product of the rates of the two actions, the rate of mining a new block is mR×hR_i, which corresponds

[3] Actually, the set N_i is not present in the PRISM+ model. The actions are replicated for all the Miners without the for loop.

to the parameter λ_{m_i} introduced in the previous section. Whereas the rate of loosing the competition is `lR×hR_i`. If the miner wins, it changes its status in `Winner` (lines 15 and 16), updates its ledger and sends the new block to the Network (action `addBlock_i` at lines 19 and 20) in order to forward it to the other miners (i.e. updating other miners' sets with the new block). If the miner loses, its status becomes `Lost` (line 17) and it checks for new blocks in the Network process with a certain rate r_b, which simulates the latency of the network and corresponds to the product between the rate 1 (for the action of the Miner) and the rate r_b of the Network action (lines 22 and 23 and line 9 of Listing 7.2). If there are new blocks, the miner chooses randomly one of them (with the operation `extractBlock()`). This random choice simulates the delay due to the topology of the network. In our model, the rate r_b and the random selection of blocks from the sets simulates the communication delay of messages in the Bitcoin Network. Then, the state of the Miner becomes `Move` and the Miner adds to its local set `setMiner_i` the block `b_i` (lines 29 and 30). Moreover, the Miner synchronizes with the process Network with action `removeBlock_i`. The Network removes the block `b_i` from the set of the Miner `set_i`. Then the state of the Miner becomes `Mine`. Otherwise, the Miner can try to take a block from its local set (lines 25-26). A block is randomly extracted from the local set `setMiner_i`. If the block taken from the local set can actually be added in the ledger (which means that the function `canbeInserted(L_i,b_i)` returns `True`), the Miner adds the block to its ledger and removes it from the local set (lines 32-34). Finally, its status is set to `Mine` and the process starts again. Otherwise, the block is not removed from the local set and the process starts again (line 35). If both the local set and the set stored in the Network process are empty, the Miner does nothing and its status returns `Mine` (lines 24 and 27). The time spent in performing these actions is simulated by the rate 1. This rate is much higher than the other rates (which are numbers in the $[0,1]$ interval) because it corresponds to local management operations of the Miner. Therefore, the probability that a Miner tries to add a received block in its ledger is higher than the probability of receiving or mining a new block. It is worth noticing that a block is added in the correct position of the ledger, even if it is a *stale block*. In our model stale blocks are represented as valid blocks which are not part of the blockchain. In contrast, an orphan block is modeled as a block received by a

miner, but that does not have its entire ancestry (yet) in the local ledger and thus cannot be added. So an *orphan block* is not added to the ledger and is left in the local set `setMiner_i`.

7.2.2 Churn Nodes

Nodes that may leave the Bitcoin Network and rejoin after some time are called *churn nodes*. As described by Motlagh et al.[MMM; MMM20b], while a node is away from the network, other active nodes continue processing transactions, mining and adding blocks to their respective blockchains. When a node rejoins the network, its ledger is out of date and needs to be updated before the node can take part in network activities. Therefore, the first action to be taken after rejoining is to download all blocks that were added to the set of the Network during its sleep. When the blocks have been downloaded, the miner can start adding them to the ledger. In the model of Listing 7.1, this is performed by transiting to the state `Mine`. In order to model churn miners, we define a controller process that awakes and shuts down miners following a given policy explained below. The uncertainty is modelled by rates and the controller consists of a sequence of states that alternate `awake` and `sleep` synchronizations with the corresponding miner. Listing 7.3 shows a controller `Controller_i` for a miner `Miner_i` with with 11 states.

```
1  module Controller_i
2    Controller_i_STATE : [s0,s1,. . . , s10] init s0;
3
4    [sleep_i] (Controller_i_STATE = s0) -> r_i0 : Controller_i_STATE
         ' = s1 ;
5    [awake_i] (Controller_i_STATE = s1) -> r_i1 : Controller_i_STATE
         ' = s2 ;
6    ...12
7    [sleep_i] (Controller_i_STATE = s8) -> r_i8 : Controller_i_STATE
         ' = s9 ;
8    [awake_i] (Controller_i_STATE = s9) -> r_i9 : Controller_i_STATE
         ' = s10 ;
9  endmodule
```

Listing 7.3: Model of a controller with 11 states.

Note that the controller is a finite state system that, when the number of states are even, will leave the corresponding miner active forever; when the number

is odd, it will leave the miner inactive forever. It is easy to define alternative controller with cyclic behaviours.

```
1  module Miner_i
2     Miner_i_STATE : [Mine,Winner,Lost,Add,Move,Update,
         MoveUpdate,Sleep] init Mine;
3     b_i : block (Miner_i^0;genesis);
4     L_i : ledger ⟨{ genesis };genesis ⟩;
5     c_i : [0..100] init 0;
6     setMiner_i : set [];
7
8     [sleep_i] (Miner_i_STATE = Mine) -> 1 : (
         Miner_i_STATE' = Sleep);
9     ...
10    [addBlock_i] (Miner_i_STATE=Update) & !isEmpty(
         set_i) ->
11                   1 : (b_i'=extractBlock(set_i))&(
                        Miner_i_STATE'=MoveUpdate);
12
13    [removeBlock_i] (Miner_i_STATE=MoveUpdate) ->
14                   1 : (setMiner_i'=addBlockSet(
                        setMiner_i, b_i))&(Miner_i_STATE
                        '=Update);
15
16    [] (Miner_i_STATE=Update) & isEmpty(set_i) -> 1 :
         Miner_i_STATE'= Mine ;
17
18    [awake_i] (Miner_i_STATE = Sleep) -> 1 : (
         Miner_i_STATE' = Update);
19 endmodule
```

Listing 7.4: Simplified model of a *dynamic* miner.

The churn miner of Listing 7.4 extends the one of Listing 7.1 with three additional states: `Sleep`, `MoveUpdate`, and `Update`.

As before, the initial state of this Miner is `Mine` where, in addition, it may synchronize with the controller (action `sleep_i`) and, after a certain amount of time, modeled by the exponential parameter `r_i0`, the Miner state becomes `Sleep` (line 8). In this state, the miner may synchronize with the controller again (action `awake_i`) and its state becomes `Update` (line 10). In the `Update`

state, the Miner synchronizes with the `Network` process and extracts all the blocks from the corresponding set in `Network` by moving them into its local set `setMiner_i` (lines 13-14). When `set_i` becomes empty (line 16) the Miner state is set to `Mine` and the Miner can start the standard behaviour, which is the one defined in Section 7.2.1 in Listing 7.1 (lines 15-35).

In our simulations we consider three controllers: one with four states (so sleep-awake-sleep synchronizations, the first sleep has a very high rate, therefore the corresponding miner goes asleep immediately), the second with two states (one sleep synchronization only: when the miner shuts down, it will be down forever) and the third one with five states (sleep-awake-sleep-awake synchronizations). Additional experiments with larger number of churn miners and with cyclic behaviour are left to future work.

```
1  module Network
2      n : numberOfMiners
3
4      for i from 0 to n-1:
5              set_i : set [];
6              N_i : set [];
7
8      for i from 0 to n-1:
9          [addBlock_i] ->  r_b : foreach k in N_i { set_k'=addBlock(
               set_k,b_i); };
10
11     for i from 0 to n-1:
12         [removeBlock_i] -> 1 : set_i' = removeBlock(set_i,b_i)
13                              & foreach k in N_i { set_k'=
                                    addBlock(set_k,b_i); };

14 endmodule
```

Listing 7.5: Simplified model of the Network.

7.2.3 Network Topologies

We modeled several kinds of network topologies and analyzed how they affect the likelihood of a fork. Network topology refers to how nodes are connected with each other and transmit new blocks. We study three network topologies: the ring topology, the tree topology and the daisy chain. The three topologies have been modeled by changing only the Network process; Hasher and Miner are those defined in Section 7.2.1.

Listing 7.5 shows the modified code for the Network process. As the reader can observe, the code is the same as of the one presented in the Section 7.2.1, except for the line 13. In particular, we modified the set `N_i` containing, for each node, the set of nodes to whom the new blocks have to be forwarded. When a Miner extracts a block received by the Network, the block is forwarded to the nodes contained in the set `N_i`. This set is defined according to the topology as follows:

- *Daisy chain* (Figure 78): `N_i` contains the previous node and the next one (except for the terminal nodes where `N_i` are singletons). For instance, for miner `Miner_i`, assuming `i` $\neq$ `0,n`, `N_i = { Miner_i-1, Miner_i+1 }`, while `N_0 = { Miner_1) }` and `N_n = { Miner_n-1 }`.
- *Ring* (Figure 79): the set `N_i` contains the previous and the next node for each miner. Thus, for every miner `N_i = { Miner_((i-1)%n), Miner_((i+1)%n)}` .
- *Tree topology* (Figure 80): every `N_i` contains the parent node and the children node, except for the root of the tree and the leaves. Roots have only children nodes; leaves have only the parent.

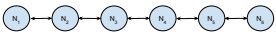

Figure 78: Daisy chain topology.

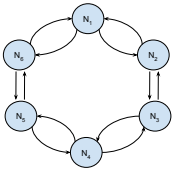

Figure 79: Ring topology.

The above topologies have been selected because they are the simplest to realize in practice. The goal is to compare the resilience to forks of these topologies

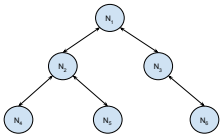

Figure 80: Tree topology.

with respect to the broadcast topology, (where every miner forwards the block to all the other miners). Henceforth, one can choose the best topology according to the preferred trade-off between risks of forks and connection costs.

7.3 Stochastic analysis

In this section, we report the results of the simulations of our model, the confidence level is set to 99% and the samples to be generated are 100000. In the caption of the figures we always show the `t` time units (bound time) considered for each property. We used PRISM+, presented in Section 7.1, to analyze the behavior of Bitcoin in different settings. The first analysis validates our model with respect to the real Bitcoin Network; the second one studies the trade-off between the security and the efficiency/scalability of the network, i.e. we study the interdependence between the probability of reaching a fork of length k and the difficulty of a cryptopuzzle. The last two analyses study the time needed to mine a new block and the probability of reaching a fork of length k for a network with churn miners and in networks using different kind of topologies, respectively.

We always assume that all miners work honestly, for example, they never try to mine new blocks and attach them in internal nodes of the ledger, as would occur in a double spending attack or in a block withholding attack [ES14]. All the simulations have been carried out on a Virtual Machine with 8 VCPU and 64 GB RAM.

Following Section 2.3, we define the properties of interest in the stochastic temporal logic CTL. For example, the formula

```
P=?[F<=T "winner"]
```

defines the probability that some miner mines a new block within the first `T` time

units. Thus, when checking the above property, PRISM must check whether `F<=T "winner"` is true for each path. This is the formula we use to assess the coherence of our model with respect to Bitcoin – see Section 7.3.1. Another example is the formula that checks the occurrence of forks in ledgers. To formalize this formula we introduced in our model a suitable module to compute forks, called `Global`, whose code is reported in Listing 7.6.

```
1 module Global
2     difference : [0..100] init 0;
3
4     for i from 0 to n:
5         [] (Miner_i_STATE = Add) -> 1 : (difference' =
              calculateFork(L_1, ..., L_n));
6 endmodule
```

Listing 7.6: The Global process.

The process `Global` computes the difference between the ledgers of the system every time a ledger is modified, e.g. when the `Miner_i` changes its state to `Add`, using the PRISM+ operation `calculateFork` (see Section 7.1). The value returned by `calculateFork` is stored in the state variable `difference`. Therefore, the probability of reaching a state of fork of length `k` within the first `T` time units is defined as:

```
P=?[F<=T difference = k]
```

The complete definition of the considered properties can be accessed in the on-line repository.

7.3.1 Coherence of the abstract model

To assess the coherence of our model with respect to Bitcoin, we take some well-known values of the protocol and we compare them with the result of our simulations.

We begin by studying mining rates. According to the hashrate distribution of Bitcoin mining pools on May 2020, the probability that a block is mined in Bitcoin within 600 seconds is about 63% (or $1-e^{-1}$). In 30 minutes (1800 seconds) a block has about 95% chance of being found and in 3000 seconds

[4]Source: `https://www.blockchain.com`

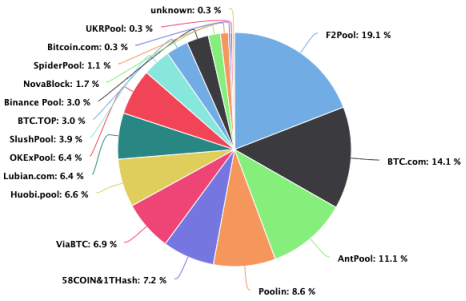

Figure 81: Hashrate distribution of Bitcoin mining pools on May 2020[4].

the probability that someone has found the block is close to 1 [5]. If we model a system with 16 miners whose hashing power (the rate) corresponds to the hashing power distribution of (the main) Bitcoin pools as illustrated in Figure 81, we obtain a probability of mining a new block which varies over time as shown in Figure 82.

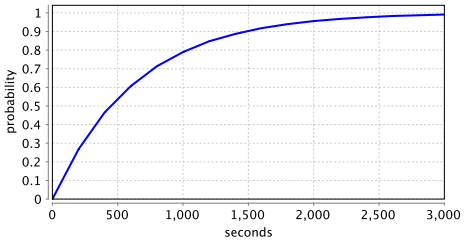

Figure 82: Probability of mining a block.

The figure displays the probability that someone in the system has mined a new block. From out plot it is easy to see that the probability of mining a new block in our system has an exponential behaviour as expected from the literature. In particular, the reader may verify that the probability that a block is mined in 3000 seconds is almost 1. Similarly, the results at 600 and 1800 seconds are in line with those of the real execution of the protocol.

[5] `https://en.bitcoin.it/wiki/Confirmation`

In another analysis, we study the probability of reaching a state with a fork of length 1 by varying the communication delay. The expected output is that the higher is the rate for the communications, the smaller is the time for the transitions to occur.

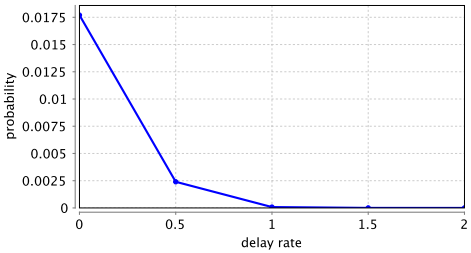

Figure 83: Probability of reaching a fork of length 1 by varying the broadcast delay; the bound time `T` is set to 600 seconds.

In fact, this is what Figure 83 highlights. In particular, when the communication rate is $r_b = 0.08$, we obtain results in line with Bitcoin, as presented by Decker and Wattenhofer [DW13].

As a last analysis for validating our modelling, we study the probability of having forks of increasing length when the broadcast rate is fixed to $r_b = 0.08$. Our choice derives from the observation that the average communication delay in the Bitcoin Network is 12.6 seconds [DW13] and that can be approximated by an exponential distribution with mean λ. Thus, taking λ equal to 12.6, we have that $r_b = 1/12.6 = 0.08$. The results of our simulations are in Figure 84. The reader can observe that the probability to have a fork of length 5 is of the order of 10^{-8}, whereas it is approximately zero when the length of the fork reaches 6. This is a key result because blocks at depth 6 are considered as permanent in Bitcoin and therefore payed.

7.3.2 Variation of Cryptopuzzle Difficulty

We start our study of the resilience of the Bitcoin protocol to relevant changes of the rates. We begin by analyzing the probability of having a fork while varying the difficulty of the cryptopuzzle (in Bitcoin this difficulty is adjusted with respect to the computational power of the miners, in order to have a new block

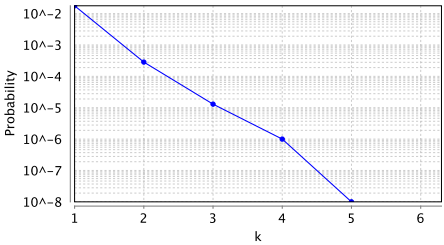

Figure 84: Probability of a fork of length `k`; the bound time `T` is set to `k*600` seconds.

on average every 10 minutes). Figure 85 highlights the relationship between the probabilities of mining a block within a specific amount of time with two two different average mining rates, denoted with *D* in the figure.

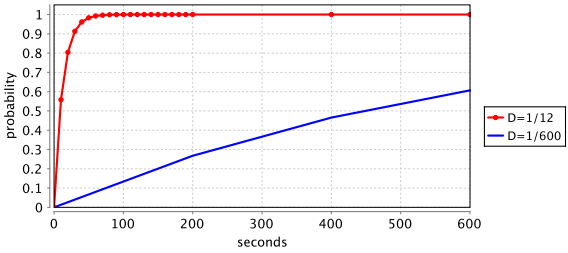

Figure 85: Probability of mining a block within 600 seconds.

The comparison is between a system with Bitcoin average mining rate (1/600) and a system where a new block is produced every 12 seconds (1/12). Of course, the probability that a miner finds a new block in the second system is much higher than Bitcoin. In particular, after 100 seconds the probability that a miner mines a new block is 1 when the average mining rate is 1/12; on the contrary, with the Bitcoin rate, the probability is less than 0.2.

Figure 86 shows the relationship between the length of the forks and theaverage mining rates. In this case, the probability of reaching a fork of length 6 with average mining rate 1/12 is greater than 0, whereas it becomes zero with the average mining rates of Bitcoin.

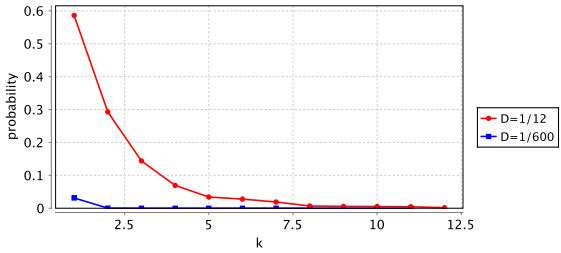

Figure 86: Probability of forks; the bound time T is set to 600 seconds.

Finally, we study how the time required to mine a block varies when we consider different cryptopuzzle difficulties. Since the difficulty of the cryptopuzzle is inversely proportional to the rate of the mining process, one might be interested in studying the trade-off between speed and security

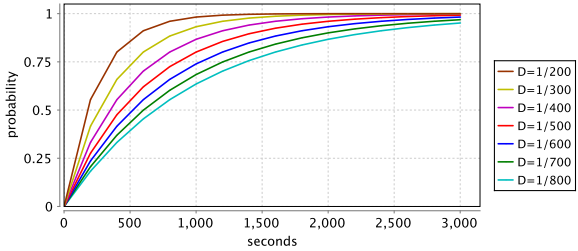

Figure 87: Probability of mining a block within 3000 seconds.

The results of the simulations in Figure 87 confirm that the easier the cryptopuzzle is, the faster the entire system mines a new block. This also follows by the fact that by design the time required for mining a new block is $\frac{2^{32}D}{H}$ where D is the cryptopuzzle difficulty and H is the hash rate. Figure 88 displays how the probability of reaching a fork of length 1 varies depending on different average mining rates. Our results show that a good balance between speed and safety can be obtained with a mining rate equal to 1/500 per second. Indeed, with this rate, the process of mining a block is faster than in Bitcoin (1/600), but the probability of reaching a state of fork is not much higher. Even if this is a theoretical result, it shows that a better trade-off between the rate of the mining process and the

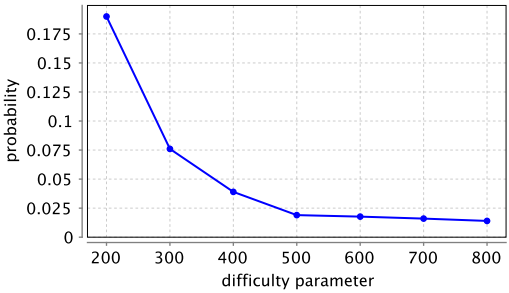

Figure 88: Probability of a fork of length 1 with different difficulty level of the cryptopuzzle; the bound time `T` is set to 600 seconds.

security of the network can be obtained and can be measured. Of course, changing this trade-off may impact the behaviour of Bitcoin in different aspects. Since mining is a energy consuming task, one may expect that speeding up the process may lead to some energy savings in practice. Although this seems reasonable in theory, actually, it also depends on the behaviour and strategies of miners, e.g., they may decided to invest more in mining since it is now "easier" to mine new Bitcoins. It is not easy to predict with certainty how this change impacts miners' strategies. We leave studying this problem as a future work. A related aspect concerns understanding how the value of Bitcoins in the market varies, when the cryptopuzzle difficulty changes. Actually, a recent paper by Fantazzini and Kolodin [FK20] seems to suggest that the hashrate is not useful in predicting the Bitcoin price on its own. However, we believe that the change could affect the fees miners receives for their work, so impacting their strategies. Also, studying this problem is left as a future work.

7.3.3 Churn Nodes

In this subsection, we focus on the simulations of a system using the broadcast topology (see the `Network` process in Listing 7.2) but with three churn nodes. As anticipated in Section 7.2.2, our model consists of 13 "static" miners and three churn miners. The first miner goes asleep as soon as the process starts and then awakes after a while. The other two miners, at the beginning, participate at the mining process, but shut down after a given amount of time. The difference

between the two is the fact that one of them, when it shuts down, does it forever, whereas the other awakes again after a while. A churn miner impacts on the Bitcoin protocol because, when a node leaves or joins the network, the overall hashing power changes [MMM20a; MMM20b; MMM].

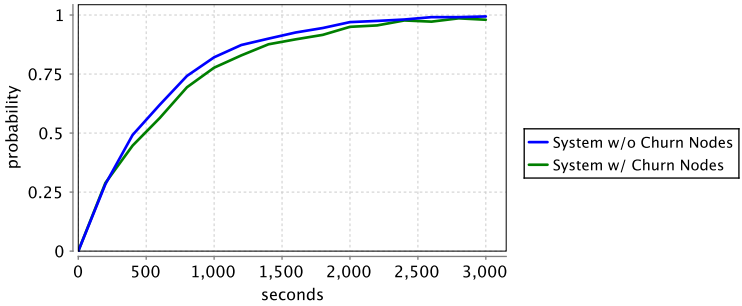

Figure 89: Probability of mining a block within 3000 seconds.

This remark is confirmed by Figure 89, which compare the time needed to mine a new block with the presence of churn nodes to the time needed in the Bitcoin system with only static nodes. Since in the dynamic system, there are less nodes trying to solve the cryptopuzzle, the probability that someone win is lower. Also the probability of reaching a state of fork is lower as reported in Figure 90. This is due to the fact that there are fewer miners and, in this setting, each miner is connected to every other, thus the presence of churning nodes does not lead to message losses.

Finally, we study the latency of a churn miner when it rejoins the network because it has to download all the blocks that were mined during its absence. In particular, we assume that the mean node sleep rate ranges from 2 to 10 hours, as suggested by Motlagh at el. [MMM; MMM20b]. Our experiments highlight that the synchronization process requires little time, as the reader can observe from Figure 91 which shows the probability that a node (with different sleep rates) synchronizes the missing blocks within a minute after rejoining the network.

Clearly, the longer the nodes is down the lower the probability that it quickly synchronizes because the number of blocks mined during its sleeping period increases.

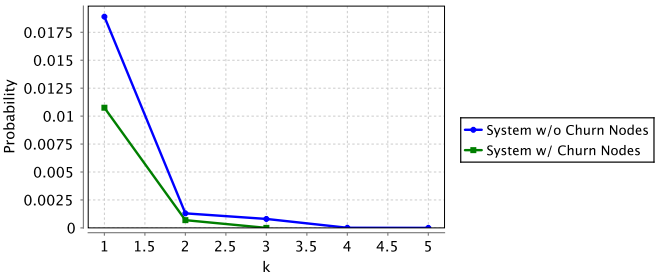

Figure 90: Probability of a fork of length k, the bound time T is set to $k * 600$ seconds.

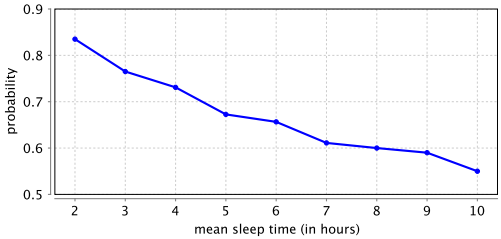

Figure 91: Probability that the node can synchronize in a minute with mean sleep time ranging from 2 to 10 hours.

7.3.4 Different Topologies

After having described the models of different kinds of topologies in Section 7.2.3, we show the results of our simulations. The rates $\mathtt{r}_b$ presented in Section 7.2.3 are set to 1 in the simulations of daisy chain, tree and ring topologies, In fact, in this settings, we do not model the latency of the network by means of a rate, but we use the set of connected nodes. In particular, we choose to set $\mathtt{r}_b$ exactly to 1 so that it can be considered as an instantaneous action (the other actions have lower rates).

In Figure 92 we show that the probability of mining a block is not affected by changing the network topology. This outcome is trivial because we are changing how the nodes receives and sends the new blocks. This has very high rate with respect to the mining process, which remains unchanged. It turns out that

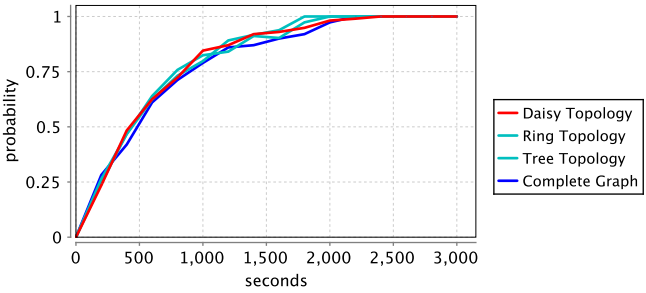

Figure 92: Probability of mining a block within 3000 seconds.

the main difference between topologies is the probability of reaching a fork of length `k`. As the reader would expect, when a new block is mined, if it is not for-

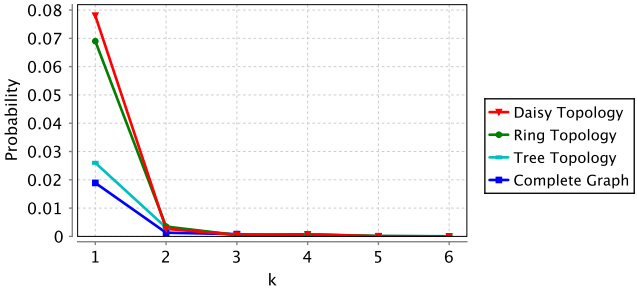

Figure 93: Probability of a fork of length k, the bound time T is set to $k * 600$ seconds.

warded to all the other nodes but only to a subset of the total, the probability of inconsistency increases. In Figure 93 we compare the four kinds of topologies: broadcast, daisy chain, ring and tree topologies. Our result is that the smaller is the subset of the receiving nodes per miner, the higher is the probability of a fork.

7.4 Conclusions

In Chapter 7 we analyzed the consensus protocol of Bitcoin by using an extension of the probabilistic model checker PRISM. In particular, we extended PRISM with a library implementing the notion of block of transactions and ledger natively adding the new data types `ledger`, `block` and `set`and we named it PRISM+ Then, We have modeled the Bitcoin system as a parallel composition of processes and analyzed its behavior. Indeed, in our model each blockchain is selected as the sequence of blocks starting from the pointer to a leaf at maximal depth with root the genesis block. We performed some probabilistic analyses covering different features of the protocol. The first analysis showed that our model simulates accurately the probability of mining a new block within a given amount of time and the probability of reaching a state of fork respect with the real Bitcoin system. The second analysis was concerned with the trade-off between security and the difficulty of the cryptopuzzle. It has been observed that a slight decrease of the difficulty level of the cryptopuzzle leads to a significant increase of the speed of mining at the cost of an almost irrelevant increase of the probability of a fork. We also pointed out that when a node leaves the network frequently, there is an immediate consequence on how the hashing power is distributed in the network. Finally, we simulated the Bitcoin protocol taking into account different kinds of network topologies. The results of our simulations clearly pointed out that the less the nodes are connected, the higher is the probability of reaching a state of fork.

Chapter 8

Related Works

The results presented in the thesis can be roughly partitioned into two classes: 1) the empirical analysis of data in the Bitcoin blockchain and the development of visualisation tools to enable the forensics analysis of suspect value flows; 2) the theoretical analysis of arbitrage, a crucial economic mechanism of the Bitcoin ecosystem, and the probability of forks in the Bitcoin protocol. In this chapter, we discuss our results by contextualising them with the ones in the literature.

Analysis of the Bitcoin blockchain and visualisation tools

First, we discuss the literature on the empirical analysis of the Bitcoin blockchain and the development of visualisation tools for forensics analysis. In particular, we face three Bitcoin arguments: visualisation tools, transaction surveys and addresses reuse. Since the blockchain is open, in combination with the networked structure of transactions, in the literature we can find several tools for visualising the Bitcoin Network.

In [SMZ14b] the authors present *BitIodine*; BitIodine labels users (semi) automatically with information on their identity and actions, which is scraped from openly-available information sources; for instance, individuals that a have significant negative feedback on the *BitcoinTalk*[1] trust system. An interesting deployment of small-scale visualisation to directly analyse transactions is presented in [Bat+15]. The chapter presents a tool, i.e., *BitCoinView*, which per-

[1] BitcoinTalk: `https://bitcointalk.org`.

forms a bottom-up visual analysis of the influence of selected source-transactions on subsequent flows in the transaction graph. Some other approaches that permit the visualisation of part of the graph are described in [Mei+13; RH11b; RS13]). Blocksci [Kal+17] is an applications of blockchain analysis, that allow to get information from transactions graph, but can't visualize it. In [SAA18] the authors present visualisation mechanisms for taint propagation in Bitcoin that display how cyber criminals launder money. Differently from us their visualisation is static and does not have filter. However, the work that is more related to our approach is [McG+16]. In [McG+16] the authors present a systemic top-down visualisation of Bitcoin transaction activity to explore dynamically generated patterns of algorithmic behaviour. Differently, in our work we explicitly make use of Visual Analytics with the purpose to adopt different filter and views that allow a user to visually capture only the information of interest.

In addition to research works, on the Internet it is possible to find a plethora of Bitcoin data analysis tools. *Bitnodes*[2] shows the distribution of Bitcoin nodes across the globe, using a Bitcoin crawler implemented in Python. The live-map of *Wizbit*[3] displays both the transactions and the latest discovered Bitcoin blocks. *Interaqt* shows the live transaction-graph: the size of the nodes represents the transaction volume; every node also carries a link to its information, taken from *Bitcoin.info*[4]. *Skry*[5] is conceived for performing in-depth investigations across the blockchain. The interface can be used to analyse the relationships and to explore the transaction graph, which is enriched with addresses (possibly clustered) and other entities. *Coin Viz*[6] is a financial visualizer for Bitcoin transactions. It contains a tool, based on a force-directed approach, that shows in real-time the transactions entering the blockchain. For each of them, the sender and recipient addresses are shown. However, it is not possible to visualise all the information as we instead do in our tool. A tool with similar features is *Bitcoin-tx-graph-visualizer*[7]. *Blockseer*[8] also allows for labelling and clustering addresses by using publicly available or self-created labels. Chainaly-

[2] `https://bitnodes.21.co`.
[3] `https://blocks.wizb.it`.
[4] Bitcoin.info: `http://bitcoin.info`.
[5] `https://skry.tech`.
[6] `http://tinyurl.com/jyb3dx4`.
[7] `http://tinyurl.com/hpj36wd`.
[8] `https://www.blockseer.com`.

sis[9] is a commercial Bitcoin forensic suites, that allows to detect and investigate cryptocurrency laundering and frauds. Other tools, like *Blockchain.info*[10] and *blockr.io*[11] allow a user to navigate the graph of transactions and present several charts on financial data (some of the information about Bitcoin markets used in this chapter has been taken from Blockchain.info). All such tools are more oriented to showing statistical, geographical, or aggregated information, rather than offering means to focus on specific islands and to show only information filtered according to the user's needs.

Analyzing and understanding the Bitcoin blockchain is as complicated (due to the amount of data) as interesting. Several empirical analysis of Bitcoin data can be found in the literature. In 2014 Ken Shirriff's blog[12] studied some methods for inserting arbitrary data into Bitcoin blockchain and also what kind of data can be (or is already) stored. A few months later QuantaBytes[13] surveyed Bitcoin transactions in blockchain and found three classes of non-standard transaction. Our study update their survey, discovering also other seven classes of non-standard transaction. In 2018 [SVS18] improved the study on inserting arbitrary data into Bitcoin's blockchain. Then [Mat+18b] describes the problem of inserting harmful content into a blockchain; in particular they propose conceptual countermeasures to heuristically reject transactions holding unintended content with high probability. They find that mandatory minimum fees and mitigation of transaction manipulability significantly raise the bar for inserting malicious content into a blockchain. In the same period, [Mat+18a] the authors provide a systematic analysis of the benefits and threats of arbitrary blockchain content. They show that certain illegal content can render the mere possession of a blockchain illegal. Their analysis reveals more than 1600 files on the blockchain, e.g. links to child pornography. This analysis highlights the importance for future blockchain to be designed to address the possibility of unintended data insertion and protect users. Our study shows that Bitcoin is going in this direction. In fact, we saw that only 0,02% (224 355 out of 968 098 854) of transaction outputs corresponds to non-standard ones: this shows that most

[9] `https://www.chainalysis.com/`
[10] `https://www.blockchain.com/explorer`
[11] `https://www.crunchbase.com/organization/blockr-io`.
[12] `https://tinyurl.com/asciibw`.
[13] `https://tinyurl.com/txType`.

miners and users behave in a standard way. In 2019 [BP17] empirical study the usage of OP_RETURN over the years and they identify several protocols based on OP_RETURN, which they classify by the application domain and their space consumption. We extend their work providing an updated OP_RETURN classification by their space consumption. We also found that in blockchain there are just 797 "non-standard" OP_RETURN. Finally, We saw that only 57 038 (0.68% of all OP_RETURN transactions) lock bitcoins with a total amount of 3.71572552 BTC.

The first to study Bitcoin addresses reuse was the work in [Bar14], by focusing on privacy issues. That work also shows a graph of the first 100 most reused addresses, but differently from us, it does not consider hidden addresses and the source of such repetitions. A discussion about security and privacy issues of Bitcoin is already presented in [Con+17]. The authors describe the major attacks on the Bitcoin system and issues, as the address reuse, but without further investigation. Then in 2018 Gaihre, Luo and Liu [GLL18] present a study to understand if Bitcoin users care about anonymity. To do that, they consider three anonymity metrics of an address: reuse frequencies, zero balance, attempts to hide its intention. However, also in this case, the authors did not consider hidden addresses and did not try to de-anonymise reused addresses.

Also related to this work, but from a different perspective, in [NJ20] the authors how to keep safe an identity through a digital wallet by using *Self-Sovereign Identity* (*SSI*). Also some blockchain frameworks, as Alastra [IM20], were developed to implement SSI and could suffer from anonymity problems.

Finally, [WOD18] examines how cyber-criminal can launder money by using services that are offered on the Dark Web (by using tumblers, for example). Like our work, the authors use these services and saw that: some are excellent, professional and well-reviewed at competitive cost; others, instead, turned out to be scams. They discuss what these findings mean to law enforcement, and how bitcoin laundering chains could be disrupted. This work do not focuses on understanding how the mixing services work and how easy identify them in the network.

Theoretical analysis of arbitrage and Bitcoin protocol

According to our work and results, we can divide the discussion into two arguments: theoretical arbitrage analysis in Bitcoin and Bitcoin protocol models. Considering the theoretical analysis on Bitcoin arbitrage, our approach is complementary to other studies such as [PV15; Bar17] where the authors study triangular arbitrage with Bitcoin, i.e. buying bitcoin in US dollar and selling them in Renminbi. The paper by Gouriéroux & Hencic [HG15] represents a valid anchor to refer to, at least in this area of study, as it undertakes a non-causal analysis of the BTC/USD rates in order to predict its future evolution. The present study shares with [HG15] both the same decomposition of the BTC/USD price in a bubble and in a fundamental part, and the observed time series; though, here the main objective is to investigate whether confidence in future values of the BTC/USD rate (i.e. the *forward looking* part) is the one responsible of the bubble effect, while in [HG15] the focus was on forecasting future rates. If this is the case, a significant change in the estimated parameters should be detected when the MAR model is estimated separately in the observed time series and in the bubble component. In particular the *forward looking* parameters should be stronger in the bubble part than in the observed price. The conjecture underlying this study is that the forward looking parameters should be stronger in the bubble part than in the observed price. Indeed this turns out to be the case, when estimating the model on the observed data, however the residuals analysis, conducted by performing the BDS test for independence, suggests not to consider this models valid but for one case (partially). Since the results of this test are asymptotical (for $n \to \infty$) and given the low entity of the residuals a more extensive residual analysis could be performed in order to assess the capability of the chosen model to describe the dynamics of BTC/USD rate and/or the isolated bubble term (y_t, yc_t). Several techniques are available such as the classical *Ljung-Box-Q* test on residuals autocorrelation (see [Ham94]). Although the focus of this study is not to come across the true Data Generating Process for the Bitcoin, a deeper investigation of this issue is beyond the scope of the present study and will be tackled in future research.

The blockchain protocol was introduced by Haber and Stornetta [HS90] and only in the last few years, because of Bitcoin, the problem of analyzing the

consistency of the ledgers has attracted the interest of several researchers. The discussion of the mainstream blockchain consensus algorithms and the way the classic Byzantine consensus can be revisited for the blockchain context is presented in a paper by Gramoli [Gra20], where the consensus algorithm at the heart of Bitcoin and Ethereum is described and the behaviour of each process involved in the system is illustrated through pseudo-code. Garay et al. [GKL15] prove the correctness of the protocol when the network communications are synchronous and focusing on *persistence* and *liveness*. The extension of this analysis to dynamic asynchronous networks with bounded delays can be found in a paper by Pass et al. [PSS17]. There, the authors also provide an abstract model of the Bitcoin protocol that ignores all irrelevant implementation details. The abstract model enables them to formally study the behaviour of the protocol and to detect where there is room for improvement. Pîrlea and Sergey [PS18], instead, propose a formalization of blockchain consensus with a proof of its consistency mechanised in an proof assistant. They present an operational model that provides an executable semantics of the system and prove a form of eventual consistency focusing on the notion of global system safety. Similarly to these papers, we propose an abstract model of Bitcoin where we ignore all the implementation details which do not affect the properties of interest. The main difference between these contributions and our work is that we formalize the blockchain protocol as a stochastic system (with exponential distribution of duration) and prove its properties by simulating the model through the PRISM model checker.

There are few works in the literature that have followed a research line similar to ours, i.e. studying the properties of the Bitcoin protocol using a probabilistic model checker. DiGiacomo-Castillo et al. [DiG+20] and Chaudhary et al. [Cha+15] use UPPAAL [Ben+96; Dav+15]. The formers study the security of the proof of work consensus when the network has an adversary miner that leverages the selfish mining strategy introduced by Garay et al. [GKL15]. In particular, their experiments show the effectiveness of selfish mining against various deployment parameters. Chaudhary et al. [Cha+15] analyze the probability of success of a double spending attack in the Bitcoin protocol. They show that double spending can be achieved if the parties in the Bitcoin protocol behave maliciously. In these two works, the main goal is to verify the resilience of Bitcoin by analyzing the probability of a successful attack. Differently, in our work,

we do not consider malicious miners in the network, but study the properties of the protocol under different circumstances. Another difference is that the foregoing works do not model churning nodes and only consider the case in which a block is broadcasted to all the other nodes in the system. Bastiaan [Bas15] uses PRISM to analyze the so-called 51%-attacks (a pool can attack the network as soon as it reaches a substantial percentage of hash power) in an extension of the Bitcoin protocol (the two phase proof of work). The author proves that the extension of Bitcoin is good at preventing the 51%-attacks. As in our work, each miner is modeled as a module in the PRISM language; however the work focuses on the actual cryptographic problem and does not implement blocks and the blockchain data structures. In particular, in our case, every cryptographic detail that do not affect our analyses is overlooked.

Some recent papers propose stochastic models to analyze specific parts of blockchain systems. Biais et al. [Bia+19] focus on miners and propose a game-theoretic approach to analyze the strategies miners can adopt and the kind of equilibrium these strategies can lead to in blockchain dynamics. A similar approach is adopted by Zhang et al. [ZZP20] which propose a formal mathematical framework, to model the core concepts in blockchain-enabled economies. They illustrate the dynamics of the blockchain economies simulating and testing two different block reward strategies. The main difference with respect to our work is that their analyses focus on economic aspects. In fact, their main goal is to understand what the economic forces at the root of forks are; our analysis instead focuses on the security and the integrity aspects of the system. Moreover, Piriou and Dumas [PD18] propose a basic stochastic model for the blockchain protocol to capture the block creation and broadcasting process. They model blocks as abstract objects with just the necessary information to analyze the ledger evolution. They also propose a framework to ease the tuning of the model and exploit Monte Carlo simulations to obtain probabilistic results on consistency of the ledgers. In contrast with our purpose, their main goal is to check the ability to detect and prevent double-spending attacks of blockchain protocols.

Chapter 9

Conclusion and Future Works

In this thesis we presented a suite of tools to analyze and visualize the Bitcoin blockchain. We have been involved in several research topics; in the following points we summarize the main results.

The suite of tool (Chapter 3)

We presented BlockChainVis, a suite of tools for tracking flows of bitcoins. The main goal is to, given a specific task, filter out not interesting information, in order to better analyse: the Bitcoin blockchain, cluster addresses, identify mixing services, visualise information about transactions, and studying addresses. In simple terms, we showed several integrated tools that simplify the life of the forensic scientist, by automating some of the tasks performed to keep track of value flows and their sources/destinations.

We plan to enhance the tool by adding some logic to automatically find the solution to specific problems: so far, it is up to the user to apply the right filters to let needed information emerge. For instance, we would like to manually select an island, and then ask to the tool if it contains a mixing service: the result will be a probability value. Moreover, we would like to make BlockChainVis more time-sensitive, in order to shown events: for instance, when value is rapidly moved from one address to another. We are currently extending BlockChainVis

to encompass explicit features that are oriented to digital forensics. For instance, we will try to identify *mixing services* (also called *tumblers*), which can be used to mix value of a ransomware address with other users' value, intending to confuse the trail back to the original source and thus launder money [Kha+15a]. Moreover, we will also characterise other unexpected flows of value, for instance immediately reporting newly created addresses whose incoming balance has rapidly increased: this could help to quickly find addresses linked to ransomware effects. In general, keeping track of anomalous bitcoin flows can help to detect money laundering activities. In particular if such flows connect an address that is a payment endpoint for a *darknet market*: these digital markets primarily are black markets, selling or brokering transactions involving drugs, cyber-arms, weapons, counterfeit currency, stolen credit card details, forged documents, unlicensed pharmaceuticals, steroids, other illicit goods, as well as fully-legal products. In this case, we plan to crawl the Dark Web to collect such addresses and tag them as "sensitive" in BlockChainVis.

We will extend the graphical interface to display one of the most interesting information of Bitcore, the current broadcast transactions (called the *memory pool*). Some examples of what we want to do are in already-existing tools as bitcoind-status[1], MyPHP Bitcoin Node Status[2], Satoshi.info[3]. We plan to build a new module that show informations about miners, e.g., the relationship between miner and hashrate.

We plan also to extend the power of BlockchainVis by making it able to analyse not only Bitcoin, but also other cryptocurrencies, as *Ethereum* for example.

Transaction Information (Chapter 4)

We provided a report on the statistics concerning standard (seven classes) and non-standard (nine classes) transactions in the Bitcoin blockchain, by considering up to block number 550 000, i.e. until November 14th 2018. The most populated class of transactions is P2PKH; the reason is that they represent the default transaction in Bitcoin clients. The second most used class is P2SH, which had a

[1] `https://github.com/craigwatson/bitcoind-status`.

[2] `https://www.reddit.com/r/Bitcoin/comments/2zexq0/my_php_bitcoin_node_status_page/`.

[3] `https://statoshi.info/`.

massive growth of over 40% transactions from [BMS18b].

The presented study can help to understand the compliance of the Bitcoin protocol to the intended purposes, by quantifying past and present deviations. As a result we have obtained that only 0.02% (224 355 out of 968 098 854) of transactions outputs corresponds to non-standard ones: this shows that most of miners and users behave in a standard way. We noticed that only the 0.015% (2 615 out of more than 17 million) of all the circulating bitcoins was burned by non-standard transactions. We saw that the most used transaction inside P2SH transactions is the multi-signature one. We also show that the most used size in bytes of an OP_RETURN transaction is 20 bytes.

We plan to study the distribution of non-standard classes along time, and to relate them with amounts of involved bitcoins (also for standard classes). We will also analyze OnlyHash transactions, the aim is to see if they could be treated as "colored coin" transactions. We plan to use analysis and visualization tools to relate transaction types and topologies together. We also plan to create a module to validate Bitcoin scripts. In [KB18] authors proposed a symbolic verification theory for open SCRIPT, a verifier toolkit, and illustrate examples of use on Bitcoin transactions, including a formalisation of (a fragment of) the language and a novel symbolic approach to SCRIPT verification. Our idea aims to extend this toolkit to verifier also non-standard transaction (particularly with the 2 3 error transactions) and the new standard transaction designed in the last Bitcoin protocol released. Finally, we would like to extend the tool to accept miniscript, a high-level language compiles to script. This might be a boost in terms of the impact/usability of the tool.

Reused Addresses (Chapter 5)

We have analysed Bitcoin addresses that are most frequently reused in transactions. The analysis is a screenshot of the current situation of the blockchain. Such an analysis is important to evaluate how much the privacy of addresses is at risk, and also how much the security of coins is exposed to attacks. We investigated the repetitions of legacy (starting with 1) and hidden addresses (starting with 3, i.e., Nested SegWit transactions); we have also considered legacy addresses hidden in P2SH transactions in order to have a more comprehensive view. What we found is that most of top 100 reused legacy-addresses can be

linked to ransomware payments and scams. Such a metric can be then adopted as a trust indicator for an address.

We plan to check into and translate new Native SegWit addresses (addresses start with *bc1*) into legacy addresses, and integrate such a mapping to the overall analyses on reused usage. Moreover, we would like to analyse Native SegWit addresses and non standard transactions [BMS19]. In addition, our aim is to extend the de-anonymization of reused legacy and nested addresses beyond the top 100, trying to give an identity to as more addresses as possible. Finally, we would also like to design a trust-evaluation module to assess the privacy and security of Bitcoin address, and integrate it with the BlockChainVis suite [BMS18a].

Arbitrage and Bubbles (Chapter 6)

Bitcoin volatility is high, hence it has been claimed as a speculative financial asset rather than a currency. In addition, arbitrage opportunities are indeed possible by trading on different Exchanges, as we prove both via a theoretical model and via an empirical strategy. This study undertook also a *Mixed causal-noncausal* analysis of the BTC/USD exchange rates time series, over the period February-July 2013, to test whether the bubble effect disentangled on observed data may be explained by a *forward looking* behaviour of the economic agents. In the introduction it was noticed that given the system's monetary issuance, the exchange rate of one Bitcoin with respect to a traditional currency should be influenced by agents's future expectations and that classical ARIMA models, *backward looking* by definition, are not suitable to describe the dynamics of the Bitcoin price given the fact that the only time dependence admitted by these model regards the past. *Mixed backward forward looking* MAR models are hence considered both for the BTC/USD exchange rate and for the isolated bubbles.

In the future we plan to evaluate the possibility of proposing cross-evaluation techniques, and propose complementary validation with regression metrics such as RMSE, MAE, RMSD and others. We plan to research to possible theoretical arbitrages by considering the model introduced in [CFP17] as a starting point for a multi-exchange approach.

Stochastic Modelling (Chapter 7)

In this Chapter we analyzed the consensus protocol of Bitcoin by using an extension of the probabilistic model checker PRISM. In particular, we extended PRISM with a library implementing the notion of block of transactions and ledger natively adding the new data types `ledger`, `block` and `set`, and the operations over them. Using this extension, named PRISM+, we defined an abstract model of the Bitcoin protocol where miners' behaviour is described as processes and the whole protocol as the parallel composition of miners. Our model is a faithful abstraction of the Bitcoin PoW protocol. Indeed, in our model each blockchain is selected as the sequence of blocks starting from the pointer to a leaf at maximal depth with root the genesis block. If there are several leaf blocks at maximal depth, the pointer is set to the *first received* leaf at maximal depth.

After defining the model, we performed some probabilistic analyses covering different features of the protocol. The first analysis was instrumental to assess the coherence of our model by verifying that the probability of mining a new block within a given amount of time and that of reaching a fork correspond to those of the real Bitcoin system and coincide with the values available in the literature. The second analysis was concerned with the trade-off between security and the difficulty of the cryptopuzzle. It has been observed that a slight decrease of the difficulty level of the cryptopuzzle leads to a significant increase of the speed of mining at the cost of an almost irrelevant increase of the probability of a fork. Those results are consistent with the ones of Laneve and Veschetti [LV20], which formally demonstrate the probability of a fork in Bitcoin.

We also modelled and analysed networks with churn nodes, which provide a more realistic account of the behaviour of this complex platform. In particular, we pointed out that churn nodes have a strong impact on the way the mining intervals vary with time: indeed, when a node leaves the network frequently, there is an immediate consequence on how the hashing power is distributed in the network. Finally, we simulated the Bitcoin protocol taking into account different kinds of network topologies. The driving question was checking whether the considered alternative topologies have a resistance to forks equal to or greater than the original one of Bitcoin. The results of our simulations clearly pointed

out that the less the nodes are connected, the higher is the probability of reaching a state of fork. Moreover, our result made evident that dynamic participation of nodes affects the process of data propagation in the network. Namely, when a node disconnects, all of its connections are deactivated and network connectivity is reduced and this leads to an increase of the mean number of hops required for a block of transaction to propagate across the network.

The security of blockchains has received much attentions in the last few years and various types of attacks were investigated. In future research, we plan to study these security issues by considering both peer-to-peer network based attacks and mining-based attacks. The first type of attacks, e.g. Eclipse attack [Hei+15] and Sybil attack [Dou02], can be modeled by changing the behavior of the Network process, whereas the second one, e.g. 51% attack, can be analyzed by introducing malicious Miner processes. In this context, we could extend the analysis of Section 7.3. We also plan to extend our PRISM model to faithfully simulate other consensus protocols, as Ethereum one. The current Ethereum's consensus algorithm is based on a PoW that is different from Bitcoin: Ethereum adopts a chain selection rule to include blocks in the main branch that the Bitcoin consensus algorithm would have excluded [Woo14]. The analysis of other types of consensus protocols used in blockchains is also an interesting future work. One of these protocols is the Proof of Stake [BGM16] that would require revisions of the Hasher process, as well as the introduction of a new module that record stakes.

Final remarks

We believe that this thesis can raise awareness that strong anonymity is not a Bitcoin key feature by design. We accomplish that by showing that it is possible to associate addresses with identities using external identifying information. With appropriate techniques, we can also observe the activity of known users in detail. In particular, we show how it easy track the money flow of ransomware (i.e. WannaCry). We also notice that the Bitcoin community (users and miners) behaves strictly to the intended purposes of the Bitcoin protocol regarding the use of standard transactions. In fact, just 0.02% of the transaction outputs are non-standard, involving less than 0.02% of the total Bitcoin capital. On the contrary, the same community do not follow the protocol advice regarding the

address reused. On average, every user uses more than two times his address. It shows how often the privacy of addresses is at risk and how much the security of coins is exposed to attacks.

As we have seen in the last years, Bitcoin volatility is high. However, thanks to it, we see that arbitrage opportunities are possible by trading on different Bitcoin Exchanges. In particular, the higher the volatility more the gain. Finally, we raise the fact that it is possible to speed up the bitcoin protocol without increasing the probability of a fork. It could be possible to have a block every eight minutes with almost the same probability of having forks. That can increase the throughput of bitcoin to 250 transactions per minute (50 more than the actual one).

Appendix A

Appendix Title

A.1 Technologies

The back-end of BlockChainVis is implemented on a machine with 128Gbyte of RAM, 2 processors Intel(R) Xeon(R) CPU E5-2620 v4 2.10GHz 8 core (for a total of 32 threads). In particular, the implementation consists of different Docker virtual machines running: Bitcore, PostgreSQL, MongoDB and all software dedicated to visualization. Figure 94 shows all our docker and their interactions (blue lines). All the web interfaces are manage by a Nginx Proxy[1] and have also a dedicated PHP server. Control panel and Transaction Info have installed also python. All the DB, a part Bitcoin DB, are a MariaDB docker. In particular, the control panel docker contains all the scripts that manage the population of the Bitcoin DB from the Bitcore docker and the MongoDB population from the Bitcoin DB. It also contains the web interface to run this insertion. The visualizer docker contains the web interface to visualize the Bitcoin transactions graphs stored in MongoDB. Instead, the Transaction information docker takes the transactions info directly from the Bitcoin DB and analyses them. It also contains a web interface to show the results. Tradebitcoin put all his data, retrieved from the exchange, in his docker DB. It contains its web interface to show arbitrage opportunities in real-time or in the past. It also allows downloading a CSV file with the historical bitcoin price in a different exchange. The Bitcoin scraper

[1] `https://www.nginx.com/`.

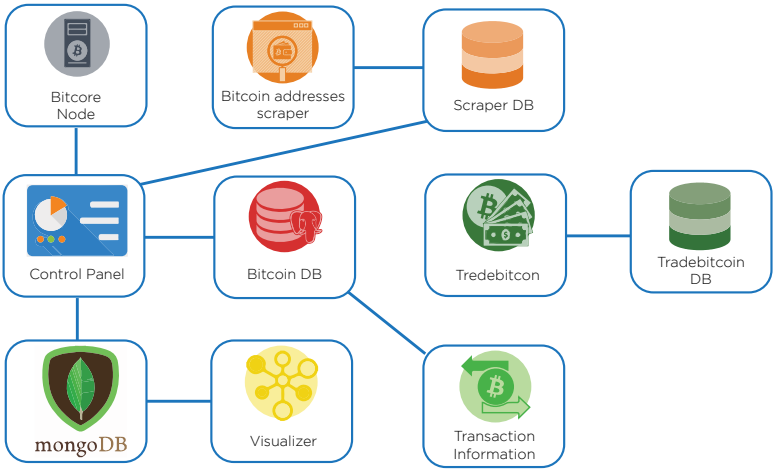

Figure 94: Docker network.

docker get addresses info from the web and store it in his DB. This DB can also be accessed from the control panel to add the addresses info to the visualization info stored in MongoDB. The Scraper container also has the web interface to run the addresses scraping and check the info of a chosen address.

The main technologies used for the back-end are *PostgreSQL*[2], *PHP*, *Python*, and *Bitcore*[3], while the ones for the front-end are *HTML5*, *CSS3*, *Javascript*, and *D3.js*[4] (see Figure 18).

A.2 The Database of Transactions

As database to store Bitcoin Blockchain information we use PostgreSQL[5] database. Postgres is an Object-Relational Database Management System (ORDBMS) emphasising extensibility and standards compliance. As a database server, its primary functions are to store data securely and return it in response to other

[2] `https://www.postgresql.org/`.
[3] `https://bitcore.io/`.
[4] `https://github.com/aaronpowell/db.js/`.
[5] `https://www.postgresql.org/`.

software applications' requests. Initially, we had all blockchain in an OrientDB[6] database. OrientDB is a widely used and open source NoSQL multi-model database. Unlike relational databases, a graph database does not utilise foreign keys or "join" operations. Instead, all relationships are natively stored as vertices of a graph. This results in deep traversal capabilities, increased flexibility and enhanced agility. However, from our tests, this database was quite demanding in terms of RAM usage, which was insufficient to calculate all the islands of transactions present in the blockchain, i.e., the strongly connected components.

Figure 95 shows the Bitcoin DB structure that contains all the Bitcoin blockchain raw data. The six tables to represent all the stored information are:

- *block* table has all the block information: *hash* contains the block hash; *confirmations* is the number of block confirmations; *strippedsize* and *size* are the block dimension in byte; *weight* contains block weight[7]; *version* and *versionhex* contain respectively the block version and the hexadecimal of the block version; *merkleroot* store the block root node; *time* and *mediantime* have the block time information; *nonce* contains the block nonce; *difficulty* is the difficulty to create the block; *previousblockhash* and *previousblockhash* have that hash of the previous and the next block.

- The *tx_input* table contains all the information connected to transaction inputs: *txid_prev* and *vout* store the information of the hash of the spent transaction and his output position; *txid* is the hash of the transaction; *asm* and *hex* contain the script to spent the transaction; *vin* is the imput number.

- The *transaction* table contains all the information present in a Bitcoin transaction except inputs and outputs information that are stored in different tables: *txid* contains the transaction hash; *hex* is the hexadecimal of the transaction; *size* and *vsize* are the transaction dimension in byte; *version* contain the transaction version; *blockhash* store the hash of the block that contains the transaction; *locktime* and *time* have the transaction time information; *coinbase* is the info of the coinbase transaction (it is empty if the transaction is not a coinbase).

[6] `http://orientdb.org/`.

[7] A measurement to compare the size of different transactions to each other in proportion to the block size limit.

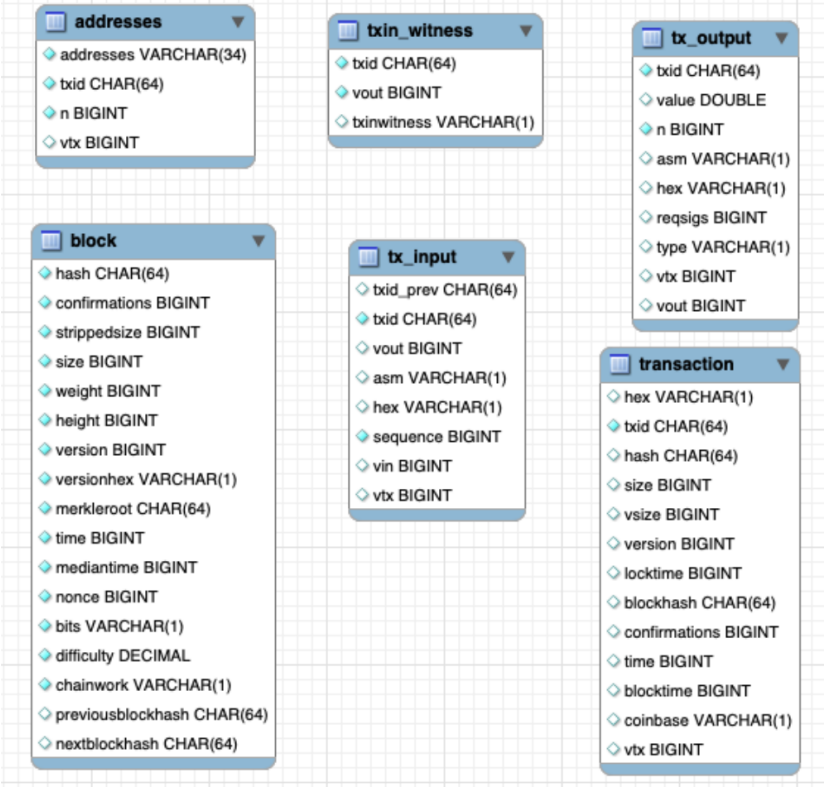

Figure 95: Bitcoin DB schema

- The *Address* table stores the address (*addresses*) and the transaction output where it appears (*txid* and *n*).
- The *tx_output* table contains all the information connected to transaction outputs: *txid* is the hash of the transaction; *asm* and *hex* contain the script to lock the transaction; *n* is the output number; *value* is the bitcoin value in the transaction; *reqsigs* store the number of signature required to unlock the output; *type* is the output type.
- The *tx_witness* table saves all the segregated witness information of all transactions: *txid* and *vout* contain the output information; *txinwitness* is

the data store in the witness field.

We also have a Bitcoin connected components DB created following the idea from [BBT18] (in Figure 96 the schema), which contain the all blockchain transaction graph. We modify their code in order to use it in a Postgres DB. The transaction graph is a bipartite graph with two types of vertices: transactions and outputs. This DB is necessary to calculate the blockchain archipelago (i.e. all the connected components inside the blockchain) in a brief amount of time (less than two days). Having this graph is very helpful in creating the JSON for the visualizer module. In fact, using this DB is very simple to get the flow (addresses transaction path) of a specific address that we want to analyse in the visualizer module. This efficiency in the archipelago computation is due to a small data storage, i.e. we store only compressed version of the transactions ID and his output and addresses [BBT18]. We use the lower 63 bits of the transaction ID as a vertex ID for a transaction. We hash a string representation of an outpoint ($< txid > | < index >$) with SHA-256[8] and use the lower 63 bits of the result as a vertex ID, setting bit 63 to distinguish outpoint vertex IDs from transaction vertex IDs. This makes outpoint vertex IDs negative since the database works with signed integers. Also Bitcoin addresses are represented in the database as an address vertex ID computed by hashing their base-58 encoded string representation with SHA-256, taking the lower 63 bits of the result and setting bit 63. Since the database works with signed integers, this results in a negative number. Transactions are represented in two database tables: *btcin* contains a row for each transaction input (*vin* contains the compressed hash of the previous transaction and *vtx* is the compressed hash of the transaction), *btcout* contains a row for every transaction output: *vtx* is the compressed hash of the transaction; *vout* is the compressed hash of the output; each output has an amount of Bitcoins associated with it (column *btc*); Column *nsigs* contains the number of signatures required to spend this output. If nsigs ≥ 1, the table *btcoutaddr* contains a row for each Bitcoin address that can be used to spend the output, linking these addresses to the outpoint vertex ID (*vout*). Outputs with nonstandard pubkey scripts are recorded with nsigs = 0. For each such output

[8]The SHA-256 algorithm is one flavor of SHA-2 (Secure Hash Algorithm 2), which was created by the National Security Agency in 2001 as a successor to SHA-1. SHA-256 is a patented cryptographic hash function that outputs a value that is 256 bits long.

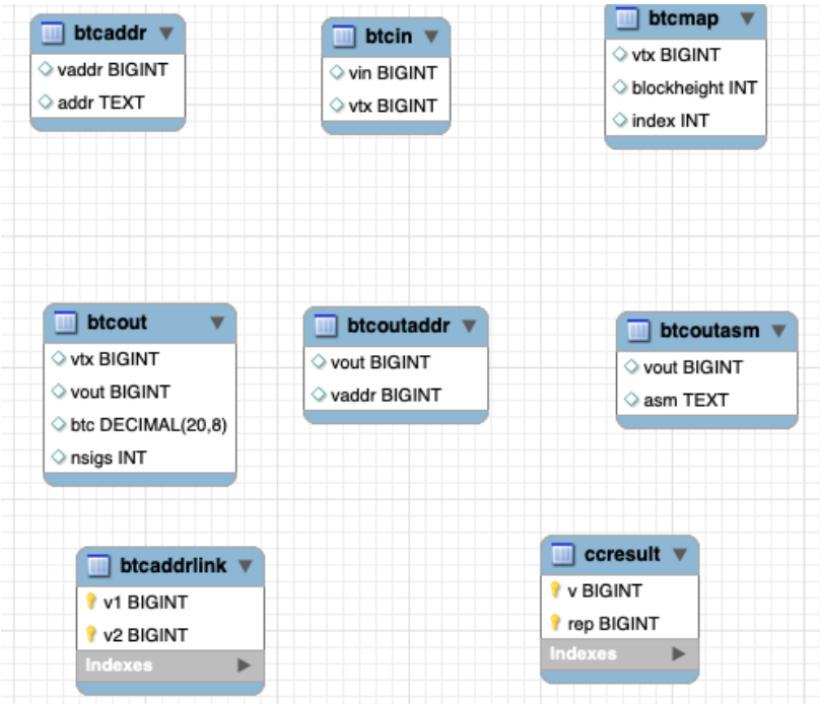

Figure 96: Bitcoin connected components DB schema

there is a row in table *btcoutasm* with the disassembled pubkey script (*asm*); this is intended for manual inspection. Table *btcaddr* maps vertex IDs (*vaddr*) to the string representation (addr). *btcmap* maps transaction vertex IDs (*vtx*) to the height of the block and index within that block where the corresponding transaction is stored.A bipartite graph of addresses and transactions is created using the transaction's inputs as edges. In this graph (archipelago), we found 38,838 distinct connected components (island).

A.3 Web Interface

The design-pattern we have adopted to manage the user-interface is the classical *Model-view-controller* (*MVC*) [KP+88]. Such an architecture divides an appli-

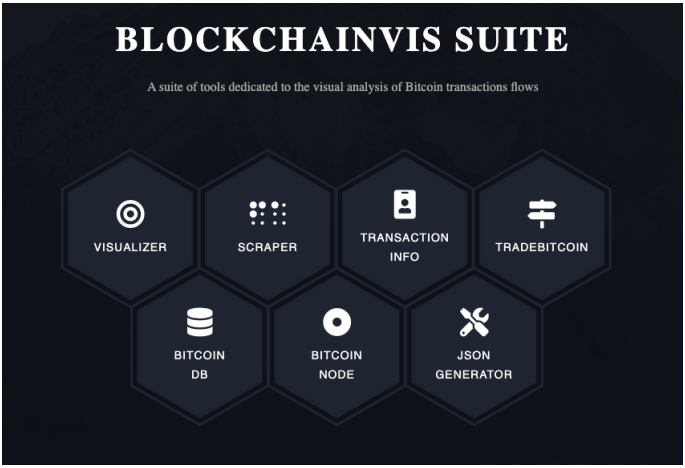

Figure 97: Blockchainvis web interface.

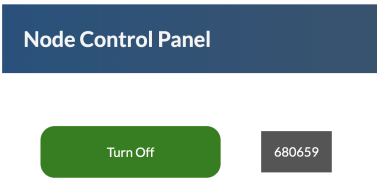

Figure 98: Bitcore control panel.

cation into three interconnected parts, with the purpose to separate the internal representation of information from the ways it is presented to a user. The *Model* manages the data, logic, and rules of the application. The Figure97 show the hub interface of Blockchaivis Suite that allows navigating between all his tool.

We have a control panel[9] that allow us to switch on/of the bitcore node (Figure 98). We can also see, in real-time, the number of blocks stored in the node. This is because Bitcore does not provide a web user interface. Then we developed a control panel[10] (Figure 99) to manage the population of the DB from the

[9] `http://bcviscp.dmi.unipg.it/nodo.html`.
[10] `http://bcviscp.dmi.unipg.it/`.

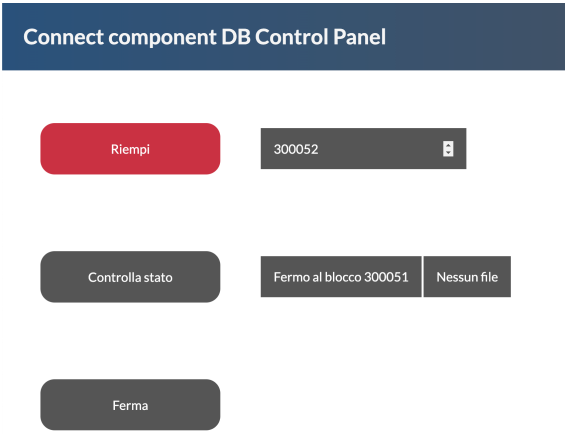

Figure 99: DB population control panel.

bitcore node[11]. This interface allows us to populate DB until the chosen block height. At every moment, it is also possible to check the insertion status (showing the last inserted block height) and terminate it (It will stop after the insertion of all the data related to the last block inserted). The last panel allow us to create a JSON for the visualization. Inserting a bitcoin address or a transaction in the panel will create its flow. This is possible thanks to the function makeJsonView() that gets the Bitcoin DB's information. The structure of the MongoDB, that stores this JSON, is simple. We have one table (called *Visualization*) that has two columns: *ID* that contains the If to identify the JSON and *json* that store the visualization JSON.

[11]the panel are actually two, one for the bitcoin DB and the other for the graph one, but are identical

Bibliography

[And+14] M. Andrychowicz et al. “Secure Multiparty Computations on Bitcoin”. In: *2014 IEEE Symposium on Security and Privacy*. May 2014, pp. 443–458.

[Ant14] Andreas M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O’Reilly Media, Inc., 2014.

[Ant17] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*. 2nd. 41 E University Ave, Champaign, USA: O’Reilly Media, Inc., 2017. ISBN: 1491954388, 9781491954386.

[AW18] A.M. Antonopoulos and G. Wood. *Mastering Ethereum: Building Smart Contracts and DApps*. O’Reilly Media, Incorporated, 2018. ISBN: 978-1-4919-7194-9. URL: https://books.google.it/books?id=SedSMQAACAAJ.

[Azi+96] Adnan Aziz et al. “Verifying Continuous Time Markov Chains”. In: vol. 1102. Lecture Notes in Computer Science. Springer. Computer Aided Verification, 8th International Conference, CAV ’96., 1996, pp. 269–276. DOI: 10.1007/3-540-61474-5_75. URL: https://doi.org/10.1007/3-540-61474-5%5C_75.

[Bac+02] Adam Back et al. “Hashcash-a denial of service counter-measure”. In: (2002).

[Bai+00] Christel Baier et al. “Model Checking Continuous-Time Markov Chains by Transient Analysis”. In: *Computer Aided Verification*. Ed. by E. Allen Emerson and Aravinda Prasad Sistla. Springer Berlin Heidelberg. Berlin, Heidelberg, 2000, pp. 358–372. ISBN: 978-3-540-45047-4.

[Bai+03] C. Baier et al. “Model-checking algorithms for continuous-time Markov chains”. In: *IEEE Transactions on Software Engineering* 29.6 (2003), pp. 524–541. DOI: 10.1109/TSE.2003.1205180.

[Baq+16] Khaled Baqer et al. “Stressing Out: Bitcoin “Stress Testing””. In: *Financial Cryptography and Data Security*. Ed. by Jeremy Clark et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 3–18. ISBN: 978-3-662-53357-4.

[Bar14] J. Barceló. “User Privacy in the Public Bitcoin Blockchain”. In: 2014.

[Bar17] James Andrew Barker. “Triangular Arbitrage with Bitcoin”. PhD thesis. 2017.

[Bas15] Martijn Bastiaan. “Preventing the 51%-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin”. In: 2015.

[Bat+15] Giuseppe Di Battista et al. “Bitconeview: visualization of flows in the bitcoin transaction graph”. In: *2015 IEEE Symposium on Visualization for Cyber Security, VizSec*. IEEE Computer Society, 2015, pp. 1–8. ISBN: 978-1-4673-7599-3.

[BBT18] Harald Bögeholz, Michael Brand, and Radu-Alexandru Todor. “In-database connected component analysis”. In: *CoRR* abs/1802.09478 (2018). arXiv: 1802.09478. URL: http://arxiv.org/abs/1802.09478.

[Ben+96] Johan Bengtsson et al. “UPPAAL—a Tool Suite for Automatic Verification of Real-Time Systems”. In: Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification, Control: Verification, and Control. New Brunswick, NeW Jersey, USA: Springer-Verlag, 1996, pp. 232–243. ISBN: 354061155X.

[BGM16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. “Cryptocurrencies Without Proof of Work”. In: vol. 9604. Lecture Notes in Computer Science. Springer. Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC., 2016, pp. 142–157. DOI: 10.1007/978-3-662-53357-4_10. URL: https://doi.org/10.1007/978-3-662-53357-4%5C_10.

[BH19] Joachim Breitner and Nadia Heninger. “Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2019, pp. 3–20.

[Bia+19] Bruno Biais et al. “The blockchain folk theorem”. In: *The Review of Financial Studies* 32.5 (2019), pp. 1662–1715.

[Bis+] Stefano Bistarelli et al. “Stochastic modeling and analysis of the bitcoin protocol in the presence of block communication delays”. In: *Concurrency and Computation: Practice and Experience* n/a.n/a (), e6749. DOI: https://doi.org/10.1002/cpe.6749. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6749. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6749.

[Bis+17] Stefano Bistarelli et al. “An end-to-end voting-system based on bitcoin”. In: *Proceedings of the Symposium on Applied Computing, SAC 2017*. 2017, pp. 1836–1841.

[Bis+18] Stefano Bistarelli et al. “Is Arbitrage Possible in the Bitcoin Market? (Work-In-Progress Paper)”. In: *GECON*. Vol. 11113. Lecture Notes in Computer Science. Springer, 2018, pp. 243–251.

[Bis+19a] Stefano Bistarelli et al. “Analysis of Ethereum Smart Contracts and Opcodes”. In: *Advanced Information Networking and Applications - Proceedings of the 33rd International Conference on Advanced Information Networking and Applications, AINA 2019, Matsue, Japan, March 27-29, 2019*. Ed. by Leonard Barolli et al. Vol. 926. Advances in Intelligent Systems and Computing. Matsue, Japan: Springer, 2019, pp. 546–558. ISBN: 978-3-030-15031-0. DOI: 10.1007/978-3-030-15032-7_46. URL: https://doi.org/10.1007/978-3-030-15032-7%5C_46.

[Bis+19b] Stefano Bistarelli et al. “End-to-End Voting with Non-Permissioned and Permissioned Ledgers”. In: *J. Grid Comput.* 17.1 (2019), pp. 97–118. DOI: 10.1007/s10723-019-09478-y. URL: https://doi.org/10.1007/s10723-019-09478-y.

[Bis+19c] Stefano Bistarelli et al. “Studying forward looking bubbles in Bitcoin/USD exchange rates”. In: *Proceedings of the 23rd International Database Applications & Engineering Symposium, IDEAS 2019, Athens, Greece, June 10-12, 2019*. Ed. by Bipin C. Desai et al. ACM, 2019, 15:1–15:9. ISBN: 978-1-4503-6249-8. DOI: 10.

1145/3331076.3331106. URL: https://doi.org/10.1145/3331076.3331106.

[Bis+21] Stefano Bistarelli et al. "Highlighting poor anonymity and security practice in the blockchain of Bitcoin". In: *SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021*. Ed. by Chih-Cheng Hung et al. ACM, 2021, pp. 265–272. ISBN: 978-1-4503-8104-8. DOI: 10.1145/3412841.3441909. URL: https://doi.org/10.1145/3412841.3441909.

[BJ90] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. San Francisco, CA, USA: Holden-Day, Inc., 1990. ISBN: 0816211043.

[Bla79] Olivier Blanchard. "Speculative bubbles, crashes and rational expectations". In: *Economics Letters* 3.4 (1979), pp. 387–389. URL: https://EconPapers.repec.org/RePEc:eee:ecolet:v:3:y:1979:i:4:p:387-389.

[BMS18a] Stefano Bistarelli, Ivan Mercanti, and Francesco Santini. "A Suite of Tools for the Forensic Analysis of Bitcoin Transactions: Preliminary Report". In: *Euro-Par Workshops*. Vol. 11339. Lecture Notes in Computer Science. Springer, 2018, pp. 329–341.

[BMS18b] Stefano Bistarelli, Ivan Mercanti, and Francesco Santini. "An Analysis of Non-standard Bitcoin Transactions". In: *Crypto Valley Conference on Blockchain Technology, CVCBT 2018, Zug, Switzerland, June 20-22, 2018*. Zug, Switzerland: IEEE, 2018, pp. 93–96. ISBN: 978-1-5386-7204-4. DOI: 10.1109/CVCBT.2018.00016. URL: https://doi.org/10.1109/CVCBT.2018.00016.

[BMS19] Stefano Bistarelli, Ivan Mercanti, and Francesco Santini. "An Analysis of Non-standard Transactions". In: *Frontiers Blockchain* 2 (2019), p. 7. DOI: 10.3389/fbloc.2019.00007. URL: https://doi.org/10.3389/fbloc.2019.00007.

[Böh+15] R. Böhme et al. "Bitcoin: Economics, technology, and governance". In: *The Journal of Economic Perspectives* 29.2 (2015), pp. 213–238.

[BP17] Massimo Bartoletti and Livio Pompianu. "An analysis of Bitcoin OP_RETURN metadata". In: *CoRR* abs/1702.01024 (2017). arXiv: 1702.01024. URL: http://arxiv.org/abs/1702.01024.

[BPS18] Stefano Bistarelli, Matteo Parroccini, and Francesco Santini. "Visualizing Bitcoin Flows of Ransomware: WannaCry One Week Later". In: *Proceedings of the Second Italian Conference on Cyber Security*. Vol. 2058. CEUR Workshop Proceedings. CEUR-WS.org, 2018.

[Bre+91] F.Jay Breid et al. "Maximum likelihood estimation for noncausal autoregressive processes". In: *Journal of Multivariate Analysis* 36.2 (1991), pp. 175–198. ISSN: 0047-259X. DOI: https://doi.org/10.1016/0047-259X(91)90056-8. URL: http://www.sciencedirect.com/science/article/pii/0047259X91900568.

[Bro05] Daniel RL Brown. "Generic groups, collision resistance, and ECDSA". In: *Designs, Codes and Cryptography* 35.1 (2005), pp. 119–152.

[BS17] Stefano Bistarelli and Francesco Santini. "Go with the -Bitcoin-Flow, with Visual Analytics". In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM, 2017, 38:1–38:6. ISBN: 978-1-4503-5257-4.

[BS73] Fischer Black and Myron Scholes. "The pricing of options and corporate liabilities". In: *The Journal of Political Economy* (1973), pp. 637–654.

[But+13] Vitalik Buterin et al. "Ethereum white paper". In: *GitHub repository* (2013), pp. 22–23.

[CFP17] A. Cretarola, G. Figà-Talamanca, and M. Patacca. "Market attention and Bitcoin price modeling: theory, Estimation and Option Pricing". Available at SSRN: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3042029. 2017.

[Cha+15] Kaylash Chaudhary et al. "Modeling and Verification of the Bitcoin Protocol". In: vol. 196. EPTCS. MARS. 2015, pp. 46–60.

[Con+17] Mauro Conti et al. "A Survey on Security and Privacy Issues of Bitcoin". In: *CoRR* abs/1706.00916 (2017). arXiv: 1706.00916. URL: http://arxiv.org/abs/1706.00916.

[Dav+15] Alexandre David et al. "Uppaal SMC Tutorial". In: *Int. J. Softw. Tools Technol. Transf.* 17.4 (Aug. 2015), pp. 397–415. ISSN: 1433-2779. URL: https://doi.org/10.1007/s10009-014-0361-y.

[DiG+20] Max DiGiacomo-Castillo et al. “Model Checking Bitcoin and other Proof-of-Work Consensus Protocols”. In: 2020 IEEE International Conference on Blockchain (Blockchain). 2020, pp. 351–358. DOI: 10.1109/Blockchain50366.2020.00051.

[Dou02] John R. Douceur. “The Sybil Attack”. In: vol. 2429. Lecture Notes in Computer Science. Springer, IPTPS 2002. Peer-to-Peer Systems, First International Workshop, IPTPS 2002., 2002, pp. 251–260. DOI: 10.1007/3-540-45748-8_24. URL: https://doi.org/10.1007/3-540-45748-8%5C_24.

[DW13] Christian Decker and Roger Wattenhofer. “Information propagation in the Bitcoin network”. In: *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*. IEEE, P2P 2013. 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688704. URL: https://doi.org/10.1109/P2P.2013.6688704.

[Dwy15] Gerald Dwyer. “The economics of Bitcoin and similar private digital currencies”. In: *Journal of Financial Stability* 17.C (2015), pp. 81–91. URL: https://EconPapers.repec.org/RePEc:eee:finsta:v:17:y:2015:i:c:p:81-91.

[ES14] Ittay Eyal and Emin Gün Sirer. “Majority Is Not Enough: Bitcoin Mining Is Vulnerable”. In: vol. 8437. Lecture Notes in Computer Science. Springer. Financial Cryptography and Data Security, FC., 2014, pp. 436–454. DOI: 10.1007/978-3-662-45472-5_28. URL: https://doi.org/10.1007/978-3-662-45472-5%5C_28.

[FK20] Dean Fantazzini and Nikita Kolodin. “Does the Hashrate Affect the Bitcoin Price?” In: *Journal of Risk and Financial Management* 13.11 (2020). ISSN: 1911-8074. DOI: 10.3390/jrfm13110263. URL: https://www.mdpi.com/1911-8074/13/11/263.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. “Impossibility of Distributed Consensus with One Faulty Process”. In: *J. ACM* 32.2 (1985), pp. 374–382. DOI: 10.1145/3149.214121. URL: https://doi.org/10.1145/3149.214121.

[FWC16] Shuqin Fan, Wenbo Wang, and Qingfeng Cheng. “Attacking OpenSSL Implementation of ECDSA with a Few Signatures”. In: *CCS*. ACM, 2016, pp. 1505–1515.

[Gar+14] David Garcia et al. “The digital traces of bubbles: feedback cycles between socio-economic signals in the Bitcoin economy”. In: *CoRR* abs/1408.1494 (2014).

[GJ16] Christian Gourieroux and Joann Jasiak. “Filtering, Prediction and Simulation Methods for Noncausal Processes”. In: *Journal of Time Series Analysis* 37.3 (2016), pp. 405–430. DOI: 10.1111/jtsa.12165. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jtsa.12165. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jtsa.12165.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol: Analysis and Applications”. In: vol. 9057. Lecture Notes in Computer Science. Springer. Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques., 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10. URL: https://doi.org/10.1007/978-3-662-46803-6%5C_10.

[GLL18] Anil Gaihre, Yan Luo, and Hang Liu. “Do Bitcoin Users Really Care About Anonymity? An Analysis of the Bitcoin Transaction Graph”. In: *BigData*. IEEE, 2018, pp. 1198–1207.

[Gra20] Vincent Gramoli. “From blockchain consensus back to Byzantine consensus”. In: *Future Gener. Comput. Syst.* 107 (2020), pp. 760–769. DOI: 10.1016/j.future.2017.09.023. URL: https://doi.org/10.1016/j.future.2017.09.023.

[GZ13] Christian Gouriéroux and Jean-Michel Zakoian. *Explosive Bubble Modelling by Noncausal Process*. Working Papers 2013-04. Center for Research in Economics and Statistics, Feb. 2013. URL: https://ideas.repec.org/p/crs/wpaper/2013-04.html.

[Ham94] James D Hamilton. “Time Series Econometrics”. In: *Princeton U. Press, Princeton* (1994).

[Hay17] Adam Hayes. “Cryptocurrency Value Formation: An empirical study leading to a cost of production model for valuing Bitcoin”. In: *Telematics and Informatics* 34 (Nov. 2017), pp. 1308–1321. DOI: 10.1016/j.tele.2016.05.005.

[HCS17] Julio Hernandez-Castro, Edward J. Cartwright, and Anna Stepanova. "Economic Analysis of Ransomware". In: *CoRR* abs/1703.06660 (2017). arXiv: 1703.06660. URL: http://arxiv.org/abs/1703.06660.

[Hei+15] Ethan Heilman et al. "Eclipse Attacks on Bitcoin's Peer-to-Peer Network". In: ed. by Jaeyeon Jung and Thorsten Holz. 24th USENIX Security Symposium, Washington, D.C., USA, August 12-14, 2015. 2015, pp. 129–144. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman.

[HG15] Andrew Hencic and Christian Gouriéroux. "Noncausal Autoregressive Model in Application to Bitcoin/USD Exchange Rates". In: *Econometrics of Risk*. Vol. 583. Studies in Computational Intelligence. Springer, 2015, pp. 17–40.

[HLT16] Alain Hecq, Lenard Lieb, and Sean Telg. "Identification of Mixed Causal-Noncausal Models in Finite Samples". In: *Annals of Economics and Statistics* 123/124 (2016), pp. 307–331. ISSN: 21154430, 19683863. URL: http://www.jstor.org/stable/10.15609/annaeconstat2009.123-124.0307.

[HS90] Stuart Haber and W. Scott Stornetta. "How to Time-Stamp a Digital Document". In: vol. 537. Lecture Notes in Computer Science. Springer. CRYPTO., 1990, pp. 437–455. DOI: 10.1007/3-540-38424-3_32. URL: https://doi.org/10.1007/3-540-38424-3%5C_32.

[Hua+18] Danny Yuxing Huang et al. "Tracking Ransomware End-to-end". In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2018, pp. 618–631.

[IM20] Javier W Ibáñez and Salvatore Moccia. "Designing the Architecture of a Blockchain Platform: The Case of Alastria, a National Public Permissioned Blockchain". In: *International Journal of Enterprise Information Systems (IJEIS)* 16.3 (2020), pp. 34–48.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)". In: *International Journal of Information Security* 1.1 (2001), pp. 36–63.

[Kal+17] Harry A. Kalodner et al. "BlockSci: Design and applications of a blockchain analysis platform". In: *CoRR* (2017). URL: http://arxiv.org/abs/1709.02489.

[Kan99] Ludwig Kanzler. “Very fast and correctly sized estimation of the BDS statistic”. In: *Available at SSRN 151669* (1999).

[KB18] Rick Klomp and Andrea Bracciali. “On Symbolic Verification of Bitcoin’s script Language”. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*. 2018, pp. 38–56. DOI: 10.1007/978-3-030-00305-0_3. URL: https://doi.org/10.1007/978-3-030-00305-0%5C_3.

[Kha+15a] Amin Kharraz et al. “Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment DIMVA*. Vol. 9148. LNCS. Springer, 2015, pp. 3–24.

[Kha+15b] Amin Kharraz et al. “Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks”. In: *DIMVA*. Vol. 9148. Lecture Notes in Computer Science. Springer, 2015, pp. 3–24.

[Kin13] Sunny King. *Primecoin: Cryptocurrency with prime number proof-of-work. http://primecoin.io/bin/primecoin-paper.pdf*. 2013.

[KNP02] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. “Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach”. In: Lecture Notes in Computer Science 2280 (2002), pp. 52–66. DOI: 10.1007/3-540-46002-0_5. URL: https://doi.org/10.1007/3-540-46002-0%5C_5.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. “PRISM 4.0: Verification of Probabilistic Real-Time Systems”. In: vol. 6806. Lecture Notes in Computer Science. Springer. Computer Aided Verification - 23rd International Conference, CAV 2011., 2011, pp. 585–591. DOI: 10.1007/978-3-642-22110-1_47. URL: https://doi.org/10.1007/978-3-642-22110-1%5C_47.

[KP+88] Glenn E. Krasner, Stephen T. Pope, et al. “A description of the model-view-controller user interface paradigm in the smalltalk-80 system”. In: *Journal of object oriented programming* 1.3 (1988), pp. 26–49.

[Kri15] Ladislav Kristoufek. “What Are the Main Drivers of the Bitcoin Price? Evidence from Wavelet Coherence Analysis”. In: *PLOS ONE* 10.4 (Apr. 2015), pp. 1–15. DOI: 10.1371/journal.pone.0123923. URL: https://doi.org/10.1371/journal.pone.0123923.

[LS08] Markku Lanne and Pentti Saikkonen. “Modeling expectations with noncausal autoregressions”. In: *Available at SSRN 1210122* (2008).

[LS11] Markku Lanne and Pentti Saikkonen. “Noncausal autoregressions for economic time series”. In: *Journal of Time Series Econometrics* 3.3 (2011).

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine Generals Problem”. In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176. URL: http://doi.acm.org/10.1145/357172.357176.

[LV20] Cosimo Laneve and Adele Veschetti. “A Formal Analysis of the Bitcoin Protocol”. In: vol. 86. OASIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. Recent Developments in the Design and Implementation of Programming Languages, 2020, 2:1–2:17. ISBN: 978-3-95977-171-9.

[Mat+18a] Roman Matzutt et al. “A quantitative analysis of the impact of arbitrary blockchain content on bitcoin”. In: *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)*. Nieuwpoort, Curaçao: Springer, 2018.

[Mat+18b] Roman Matzutt et al. “Thwarting Unwanted Blockchain Content Insertion”. In: *2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20, 2018*. Ed. by Abhishek Chandra et al. Orlando, FL, USA: IEEE Computer Society, 2018, pp. 364–370. ISBN: 978-1-5386-5008-0. DOI: 10.1109/IC2E.2018.00070. URL: https://doi.org/10.1109/IC2E.2018.00070.

[McG+16] Dan McGinn et al. “Visualizing Dynamic Bitcoin Transaction Patterns”. In: *Big Data* 4.2 (2016), pp. 109–119.

[Mei+13] Sarah Meiklejohn et al. “A Fistful of Bitcoins: Characterizing Payments Among Men with No Names”. In: *Conference on Internet Measurement Conference*. IMC ’13. ACM, 2013, pp. 127–140. ISBN: 978-1-4503-1953-9.

[MMM] Saeideh Gholamrezazadeh Motlagh, Jelena V. Misic, and Vojislav B. Misic. “Modeling of Churn Process in Bitcoin Network”. In: IEEE, ICNC 2020, pp. 686–691. DOI: 10.1109/ICNC47757.2020.9049704. URL: https://doi.org/10.1109/ICNC47757.2020.9049704.

[MMM20a] Saeideh Gholamrezazadeh Motlagh, Jelena V. Misic, and Vojislav B. Misic. “An analytical model for churn process in Bitcoin network with ordinary and relay nodes”. In: *Peer-to-Peer Networking and Applications* 13.6 (2020), pp. 1931–1942. DOI: 10.1007/s12083-020-00953-y. URL: https://doi.org/10.1007/s12083-020-00953-y.

[MMM20b] Saeideh Gholamrezazadeh Motlagh, Jelena V. Misic, and Vojislav B. Misic. “Impact of Node Churn in the Bitcoin Network”. In: *IEEE Trans. Netw. Sci. Eng.* 7.3 (2020), pp. 2104–2113. DOI: 10.1109/TNSE.2020.2974739. URL: https://doi.org/10.1109/TNSE.2020.2974739.

[Nak08a] S. Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: (2008). [Online; accessed 12-March-2019].

[Nak08b] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. https://bitcoin.org/bitcoin.pdf. 2008.

[Nar+16] Arvind Narayanan et al. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[NJ20] N. Naik and P. Jenkins. “Self-Sovereign Identity Specifications: Govern Your Identity Through Your Digital Wallet using Blockchain Technology”. In: *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. 2020, pp. 90–95. DOI: 10.1109/MobileCloud48802.2020.00021.

[PD18] Pierre-Yves Piriou and Jean-Francois Dumas. “Simulation of Stochastic Blockchain Models”. In: *14th European Dependable Computing Conference, EDCC 2018, Iaşi, Romania, September 10-14, 2018*. IEEE Computer Society, EDCC 2018. 2018, pp. 150–157. DOI: 10.1109/EDCC.2018.00035. URL: https://doi.org/10.1109/EDCC.2018.00035.

[PHD18] Masarah Paquet-Clouston, Bernhard Haslhofer, and Benoit Dupont. “Ransomware Payments in the Bitcoin Ecosystem”. In: *CoRR* abs/1804.04080 (2018). arXiv: 1804.04080. URL: http://arxiv.org/abs/1804.04080.

[PS18] George Pirlea and Ilya Sergey. “Mechanising blockchain consensus”. In: ed. by June Andronick and Amy P. Felty. CPP 2018, Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, Los Angeles, CA, USA, January 8-9, 2018. 2018, pp. 78–90. DOI: 10.1145/3167086. URL: https://doi.org/10.1145/3167086.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. “Analysis of the Blockchain Protocol in Asynchronous Networks”. In: vol. 10211. Lecture Notes in Computer Science. Springer. EUROCRYPT., 2017, pp. 643–673. DOI: 10.1007/978-3-319-56614-6_22. URL: https://doi.org/10.1007/978-3-319-56614-6%5C_22.

[PV15] Gina Pieters and Sofia Vivanco. *Bitcoin arbitrage and unofficial exchange rates*. 2015.

[RCM93] Elizabeth D Rather, Donald R Colburn, and Charles H Moore. “The evolution of Forth”. In: *ACM Sigplan Notices*. Vol. 28. ACM. 1993, pp. 177–199.

[Red11] Martin Reddy. *API Design for C++*. Elsevier, 2011.

[RH11a] F. Reid and M. Harrigan. “An Analysis of Anonymity in the Bitcoin System”. In: *IEEE Third International Conference on Privacy, Security, Risk and Trust and IEEE Third International Conference on Social Computing*. 2011, pp. 1318–1326.

[RH11b] Fergal Reid and Martin Harrigan. “An Analysis of Anonymity in the Bitcoin System”. In: *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust*. IEEE, 2011, pp. 1318–1326.

[RS13] Dorit Ron and Adi Shamir. “Quantitative Analysis of the Full Bitcoin Transaction Graph”. In: *Financial Cryptography and Data Security*. Vol. 7859. LNCS. Springer, 2013, pp. 6–24. ISBN: 978-3-642-39883-4.

[SAA18] Ilia Shumailov, Mansoor Ahmed, and Ross Anderson. “Tendrils of Crime: Visualizing the Diffusion of Stolen Bitcoins”. In: *The Fifth International Workshop on Graphical Models for Security (GramSec)*. LNCS. Springer, 2018.

[SMZ14a] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. “BitIodine: Extracting Intelligence from the Bitcoin Network”. In: *Financial Cryptography*. 2014.

[SMZ14b] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. “BitIodine: Extracting Intelligence from the Bitcoin Network”. In: *Financial Cryptography and Data Security*. Vol. 8437. LNCS. Springer, 2014, pp. 457–468. ISBN: 978-3-662-45471-8.

[SP06] K. Sampigethaya and R. Poovendran. “A Survey on Mix Networks and Their Secure Applications”. In: *Proceedings of the IEEE* 94.12 (Dec. 2006), pp. 2142–2181. ISSN: 0018-9219.

[SVS18] Andrew Sward, Ivy Vecna, and Forrest Stonedahl. “Data Insertion in Bitcoin’s Blockchain”. In: *Ledger* 3 (2018).

[Wan+20] Ziyu Wang et al. “ECDSA weak randomness in Bitcoin”. In: *Future Gener. Comput. Syst.* 102 (2020), pp. 507–513. DOI: 10.1016/j.future.2019.08.034. URL: https://doi.org/10.1016/j.future.2019.08.034.

[Wat+16] H. Watanabe et al. “Blockchain contract: Securing a blockchain applied to smart contracts”. In: *2016 IEEE International Conference on Consumer Electronics (ICCE)*. Jan. 2016, pp. 467–468. DOI: 10.1109/ICCE.2016.7430693.

[WOD18] Rolf Wegberg, Jan-Jaap Oerlemans, and Oskar Deventer. “Bitcoin money laundering: mixed results? An explorative study on money laundering of cybercrime proceeds using bitcoin”. In: *Journal of Financial Crime* 25 (Mar. 2018), pp. 00–00. DOI: 10.1108/JFC-11-2016-0067.

[Woo14] Gavin Wood. *Ethereum: a secure decentralised generalised transaction ledger*. http://gavwood.com/paper.pdf. 2014.

[WT04] Pak Chung Wong and Jim Thomas. “Visual Analytics”. In: *IEEE Comput. Graph. Appl.* 24.5 (2004), pp. 20–21. ISSN: 0272-1716.

[Xia+19] Yang Xiao et al. “A Survey of Distributed Consensus Protocols for Blockchain Networks”. In: *CoRR* abs/1904.04098 (2019). arXiv: 1904.04098. URL: http://arxiv.org/abs/1904.04098.

[YB14] Yuval Yarom and Naomi Benger. “Recovering OpenSSL ECDSA Nonces Using the FLUSH+RELOAD Cache Side-channel Attack”. In: *IACR Cryptol. ePrint Arch.* 2014 (2014), p. 140. URL: http://eprint.iacr.org/2014/140.

[Yer15] D. Yermack. “Is Bitcoin a real currency? An economic appraisal”. In: *Handbook of Digital Currency*. Elsevier, 2015. Chap. second, pp. 31–43.

[Zam+18] Alexei Zamyatin et al. “Flux: Revisiting Near Blocks for Proof-of-Work Blockchains”. In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 415. URL: https://eprint.iacr.org/2018/415.

[ZZP20] Zixuan Zhang, Michael Zargham, and Victor M. Preciado. “On modeling blockchain-enabled economic networks as stochastic dynamical systems”. In: *Appl. Netw. Sci.* 5.1 (2020), p. 19. DOI: 10.1007/s41109-020-0254-9. URL: https://doi.org/10.1007/s41109-020-0254-9.