An Empirical Analysis of Privacy in Cryptocurrencies

George Kappos

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

January 3, 2023

Abstract

Cryptocurrencies have emerged as an important technology over the past decade and have, undoubtedly, become blockchain's most popular application. Bitcoin has been by far the most popular out of the thousands of cryptocurrencies that have been created. Some of the features that made Bitcoin such a fascinating technology include its transactions being made publicly available and permanently stored, and the ability for anyone to have access. Despite this transparency, it was initially believed that Bitcoin provides anonymity to its users, since it allowed them to transact using a pseudonym instead of their real identity. However, a long line of research has shown that this initial belief was false and that, given the appropriate tools, Bitcoin transactions can indeed be traced back to the real-life entities performing them.

In this thesis, we perform a survey to examine the anonymity aspect of cryptocurrencies. We start with early works that made first efforts on analysing how private this new technology was. We analyse both from the perspective of a passive observer with eyes only to the public immutable state of transactions, the blockchain, as well as from an observer who has access to network layer information. We then look into the projects that aimed to enhance the anonymity provided in cryptocurrencies and also analyse the evidence of how much they succeeded in practice.

In the first part of our own contributions we present our own take on Bitcoin's anonymity, inspired by the research already in place. We manage to extend existing heuristics and provide a novel methodology on measuring the confidence we have in our anonymity metrics, instead of looking into the issue from a binary perspective, as in previous research.

In the second part we provide the first full-scale empirical work on measuring anonymity in a cryptocurrency that was built with privacy guarantees, based on a very well established cryptography, Zcash. We show that just building a tool which provides anonymity in theory is very different than the privacy offered in practice once users start to transact with it.

Finally, we look into a technology that is not a cryptocurrency itself but is built on top of Bitcoin, thus providing a so-called layer 2 solution, the Lightning network. Again, our measurements showed some serious privacy concerns of this technology, some of which were novel and highly applicable.

Impact statement

The results presented in this thesis aim to shed light on the anonymity offered by cryptocurrencies in practice. More specifically, this dissertation presents heuristics for breaking the privacy of existing technologies that can be used as a guideline for future designs. Moreover, the presented research can be used to develop tools that assist criminal-related investigations by tracing funds in an efficient and secure way.

In Chapter 4, we propose improved heuristics that identify change outputs in transactions, thus helping trace funds in Bitcoin and any other altcoin from their origin to their final destination. We show how these heuristics are more effective than any previously proposed heuristic aiming to do the same while being the most secure, as in including the lowest ratio of false positives. We also show a novel methodology for evaluating cluster confidence. Those contributions have an impact on the design of investigation tools both in terms of their efficiency as well as measuring their accuracy. At the same time, our findings can be used by the privacy community to design future systems without repeating the same mistakes.

In Chapter 5, we perform a measurement study on the practical anonymity provided by Zcash. Our findings shed some light on the current misconception that users had perfect anonymity just by using Zcash. Our findings show that in the vast majority of the transactions, the anonymity properties of Zcash were not utilised. For the minority of users that did utilise Zcash for privacy, we showed that in more than 67% of the cases, the funds could still be traced. We notified the founders of Zcash regarding our findings, and they responded with updated recommendations regarding the proper usage of the coin. Moreover, we noticed how the founders changed their transaction behaviour soon after the release of our paper, which was also an interesting finding.

In Chapter 6, we perform an empirical analysis of privacy in the Lightning Network. We enhanced the efficiency of existing attacks against Lightning users, as well as designed and empirically estimated the efficacy of new attacks. We showed how these attacks can significantly negatively impact users, mainly due to the fully transparent nature of Lightning's topology and the design of the routing algorithms that aimed to minimise costs until then. Since the release of our paper, several algorithms have been designed to avoid the vulnerabilities we discovered and provide better privacy while maintaining reasonable costs.

All the projects presented in this thesis were published and accepted in open conferences whose proceedings are public. The code of all our implementations is public on Github, and as of June 19th 2022 we have received 213 citations.

Acknowledgements

First and foremost, I would like to thank my primary supervisor, Professor Sarah Meiklejohn for enabling me to do this PhD in so many different ways. I will be forever grateful for her paramount contribution to every project of this thesis but most importantly for making me a better scientist.

I am greatly thankful to each collaborator I had the privilege to work with throughout this journey, Sergi Delgado-Segura, Bernhard Haslhofer, Sanket Kanjalkar, Mary Maller, Andrew Miller, Ania Piotrowska, Sofia Rollet and Rainer Stütz. I want to give a special mention to my closest collaborator, Dr Haaroon Yousaf, an incredible researcher and an even better friend.

This research would have not been possible without the funding and support of the European Commission and The Initiative For Cryptocurrencies & Contracts (IC3), so I would really like to thank both organisations for enabling me to do this.

I would like to thank my parents for everything they have done for me. None of this would have been possible without them since, technically, I would not be in this world in the first place, but also for their invaluable support that bootstrapped me and kept my academic career going. Thanks for feeding me, motivating me to be clean and supporting me in many different ways.

The last acknowledgement goes to Ania, my collaborator, best friend and love of my life. I would have not started this journey if it was not for you and I would not finish it as well.

Το Γιαγια Ποπη

Contents

Abstract **4**

Impact Statement **6**

Acknowledgements **7**

1 Introduction **18**

1.1 Motivation 18

1.2 Thesis Projects 22

1.3 Collaborations and outline 22

1.4 Thesis Contributions 23

2 Background **26**

2.1 Bitcoin 27

2.1.1 Application Layer 27

2.1.2 Network and Consensus Layer 28

2.1.3 Clustering Bitcoin addresses 30

2.2 ZCash 31

2.3 Lightning network 32

2.3.1 Channel management 33

3 Literature Review **39**

3.1 Application-layer privacy 40

3.2 Network layer privacy 44

3.3 Privacy-enhancing solutions 49

3.3.1 E-cash designs with built-in privacy 49
3.3.2 Enhancing privacy with layer 1 solutions 56
3.3.3 Enhancing privacy with layer 2 solutions 60

4 Expanding Bitcoin clustering and tracking 68

4.1 Introduction 68
4.2 Dataset and Methodology 71
4.2.1 Features 72
4.2.2 Creating a cluster dataset 75
4.3 Following Peel Chains 78
4.3.1 Defining a peel chain 78
4.3.2 Identifying inputs and outputs 79
4.3.3 Following transactions 82
4.4 Cluster Validation 83
4.4.1 Cluster classification 85
4.4.2 A validation heuristic 88
4.5 Expanding Clusters 91
4.5.1 Defining the heuristic 91
4.5.2 Evaluating the heuristic 93
4.5.3 Applying the heuristic: Bitcoin Fog 94
4.6 Conclusion 96

5 Measuring the privacy offered in Zcash 97

5.1 Introduction 97
5.2 Participants in the Zcash ecosystem 100
5.3 General Blockchain Statistics 101
5.3.1 Transactions 101
5.3.2 Addresses 102
5.4 T-Address Clustering 104
5.4.1 Clustering addresses 104
5.4.2 Tagging addresses 106

5.4.3 Results 107

5.5 Interactions with the Shielded Pool 108

5.5.1 Founders 113

5.5.2 Miners 115

5.5.3 Other Entities 119

5.6 Interactions within the Shielded Pool 121

5.7 Case Study: The Shadow Brokers 122

5.7.1 Techniques 122

5.7.2 Results 124

5.8 Conclusions 125

6 Analyzing the privacy in the Lightning Network 127

6.1 Introduction 127

6.1.1 Ethical considerations 130

6.2 Blockchain Analysis 130

6.2.1 Data and measurements 130

6.2.2 Private channels 131

6.3 Balance Discovery 134

6.3.1 The attack 134

6.3.2 Attack results 136

6.3.3 Attacker cost 136

6.4 Path Discovery 137

6.4.1 Lightning Network simulator 139

6.4.2 Simulation results 141

6.5 Payment Discovery 143

6.5.1 Payment discovery algorithm 144

6.5.2 Attack simulation 145

6.5.3 Simulated attack results 146

6.6 Conclusions 147

7 Conclusions **148**
7.1 Future works 149
7.2 Closing remarks 150

Bibliography **152**

List of Figures

2.1 A simple diagram illustrating the different types of Zcash transactions. All transaction types are depicted and described with respect to a single input and output, but can be generalised to handle multiple inputs and outputs. In a t-to-t transaction, visible quantities of ZEC move between visible t-addresses ($\mathsf{zIn},\mathsf{zOut} \neq \emptyset$). In a t-to-z transaction, a visible amount of ZEC moves from a visible t-address into the shielded pool, at which point it belongs to a hidden z-address ($\mathsf{zOut} = \emptyset$). In a z-to-z transaction, a hidden quantity of ZEC moves between hidden z-addresses ($\mathsf{zIn},\mathsf{zOut} = \emptyset$). Finally, in a z-to-t transaction, a hidden quantity of ZEC moves from a hidden z-address out of the shielded pool, at which point a visible quantity of it belongs to a visible t-address ($\mathsf{zIn} = \emptyset$). 31

4.1 Balancing the sizes (Figures 4.1a and 4.1b, with the *y*-axis on a log scale) and lifespans (Figures 4.1c and 4.1d) for our true positive (TP) and false positive (FP) clusters. Figures 4.1a and 4.1c represent the cluster sizes and lifespans before balancing, and Figures 4.1c and 4.1d represent these values after balancing. In Figure 4.1c, the vertical line represents the date on which SegWit was introduced. 76

4.2 Variable importance of a conditional random forest model (full dataset). 86

4.3 For our 60 TP and FP clusters, the distribution of $|\mathsf{Val}_\mathsf{C}|$ (the color of the bar), with the clusters ordered from left to right by $|\mathsf{C}_\mathsf{tx}|$ (the height of the bar, on a log scale. 90

4.4 Evaluation of findNext and the modified algorithm findNext2 for the ten TP clusters with the initial highest FDR, with the number of transactions on a log scale. 92

4.5 Transactions representing the link between a withdrawal from Mt. Gox (tx_1) and a deposit into Bitcoin Fog (tx_3), with an additional transaction of interest, tx_2, highlighted in blue. The gray nodes represent multiple UTXOs of the same address and transaction outputs with a dashed outline are unspent. 94

5.1 The total number of each of the different types of transactions over time. 102

5.2 The fraction of the value in each block representing each different type of transaction over time, averaged daily. Here, ‘public’ captures both transparent transactions and the visible components of mixed transactions. 103

5.3 The total value in the shielded pool over time. 104

5.4 Over time, the amount of ZEC put into the shielded pool (in red) and the amount taken out of the pool (in blue). 109

5.5 Over time, the amount of ZEC deposited into the shielded pool by miners, founders, and others. 111

5.6 The addresses that have put more than 10,000 ZEC into the shielded pool over time, where the size of each node is proportional to the value it has put into the pool. The addresses of miners are green, of founders are orange, and of unknown ‘other’ participants are purple. 112

5.7 The z-to-t transactions we associated with miners, founders, and ‘other’, after running some combination of our heuristics. 113

5.8 Over time, the founder deposits into the pool (in red) and withdrawals from the pool (in blue), after running Heuristic 3. 116

5.9 Over time, the value of deposits made by known mining pools into the shielded pool. 118
5.10 The value linked by Heuristic 5, as a function of the block interval required between the deposit and withdrawal transactions. 120
5.11 The number of z-to-z vJoinSplits over time. 122

6.1 Average path length. 142
6.2 Probability of correctly identifying the sender given successful and failed payment. For detailed simulation settings of $\mathsf{lengths}_{\mathsf{long}}$ and $\mathsf{lengths}_{\mathsf{short}}$ see Section 6.4.2 143
6.3 The precision and recall of our payment discovery attack, based on the snapshot interval τ (in log scale). The error bars show 95% confidence intervals over five simulation runs. 145

List of Tables

4.1 Types of addresses found across all clusters. 74

4.2 Performance of our Random Forest model after 5-fold cross-validation. 87

4.3 Comparison of classification results. 87

4.4 The expansion factor and false discovery rate of findNext and findNext2, as evaluated on our 60 TP clusters and as compared with previous change heuristics. Both metrics are averaged across all clusters. 94

5.1 The total number of each transaction type. 102

5.2 The services we interacted with, the identifier of the cluster they were associated with after running Heuristic 1, and the number of deposits and withdrawals we did with them. The first ten are exchanges, ShapeShift is an inter-cryptocurrency exchange, and zcash4win is a Windows-based Zcash client. 107

5.3 The behavior of each of the 14 active founder addresses, in terms of the number of deposits into the pool, the total value deposited (in ZEC), and the number of deposits carrying exactly 249.9999 ZEC in value. 114

5.4 A summary of our identified mining pool activity, in terms of the number of associated addresses used in coingen transactions, and the numbers of each type of transaction interacting with the pool. . . 117

5.5 Amounts charged for TSB monthly dumps, in ZEC. In July and September TSB offered different prices depending on which exploits were being purchased. 123

5.6 Number of clusters that put the required amounts ($\pm$1 ZEC) into the shielded pool. 125

Chapter 1

Introduction

1.1 Motivation

According to most economists, the financial crisis of 2008 was the worst financial disaster since the Great Depression of 1929. Its aftermath is still visible. Undoubtedly, the consequences of the crisis were more prevalent in the US, with the housing market nearly collapsing, stock values depreciating in the magnitude of trillions of dollars and several million Americans becoming unemployed. However, the domino effect of the crisis reached all over the world and affected the financial markets across Europe.

As with every crisis in history, there are many important lessons to be learned, and the 2008 financial collapse revealed major flaws in the financial system. Although the reasons behind the crisis are complicated and outside the scope of this thesis, one crucial factor was the failure of organisations fundamental and central to the whole economy, the financial institutions whose excessive risks and aggressive profit-seeking proved disastrous. After the economic recession, people lost trust in the fundamental pillars of the financial system and started looking for alternative options, those less centralised and more resilient to the failures of individual organisations. Luckily for those people, 2008 was not only the year of the economic crisis but also the year that Bitcoin [1] was created. The main goal of Bitcoin's creators was to build a new type of a financial system, entirely different than the existing one. Its whitepaper was released in October 2008 and according to its author,

Satoshi Nakamoto:

Bitcoin is a purely peer-to-peer version of electronic cash that would allow online payments to be sent directly from one party to another without going through a financial institution.

The decentralised nature of Bitcoin and the ability to perform transactions between entities from across the world, without the intervention of any intermediary, gained immediate attention. As with any emerging technology, it had its ups and downs and there is a long-running dispute between the opposers of Bitcoin and those who believe that it can replace the existing financial system. In any case, its current market cap of over 362 billion dollars and the fact that the underlying blockchain technology is considered one of the hottest computer science topics today shows that it has been one of the most influential new technologies of the past decade.

Besides its decentralisation and universality, another crucial differentiation between Bitcoin and existing systems is the anonymity of transactions. In particular, the privacy of traditional fiscal transactions relies entirely upon financial institutions, mainly banks, who are acting as trusted third parties. However, banks can selectively choose whether to respect the clients' will to remain anonymous. It has been a great point of interest during financial investigations, by either law enforcement agencies, governments or curious individuals, of the level of cooperation financial institutions provide. The main outcome of these investigations is that some countries and banks are much more transparent and some protect their customers' identities at all costs. This effort to protect anonymity has gained the appreciation of privacy advocates, the attraction of legitimate users who wish to remain anonymous but also the attraction of non-legitimate users who choose to perform malicious acts using those banks in order to remain hidden. It's no surprise that the vast majority of financial investigations of suspicious money transfers over the past few years have ended up in banks in Switzerland or the Cayman Islands.

But even in these cases, as long as there is a trusted institution with information regarding the transactions made, no one can ever be sure that confidentiality will be maintained in the presence of an authority coercing the party to cooperate or a malicious actor aiming to steal personal information regarding the transactions. Bitcoin tried to mitigate those obstacles by completely removing trusted parties and by building decentralised anonymity. However, a long line of research [2, 3, 4, 5, 6] prior to this thesis showed that Bitcoin's approach to provide anonymity through the usage of public addresses acting as pseudonyms is poor and users can be de-anonymized.

Although the anonymity of Bitcoin has been heavily investigated, in the early beginning of its deployment in 2009, it was broadly considered granted. No more than two years later, in 2011 the first usage of Bitcoins on a large scale exploited its anonymity by becoming the payment method for the notorious underground marketplace ”Silk Road”, where users could purchase illegal goods from each other in exchange for Bitcoins. Although its operator Ross Ulbricht was caught and the marketplace was shut down by the FBI, illegal activity using Bitcoins certainly did not stop. Numerous cases involving money laundering, like BitInstant, an exchange that knowingly exchanged funds coming from Silk Road, or cases involving Ransomware and purchases of illegal material, continued to arise until criminals realised that Bitcoin is not as anonymous as they thought and they had to use other technologies to transact privately.

Cases like BitInstant or the collapse of Mt.Gox revealed the first flaw of Bitcoin it terms of its goal to achieve decentralisation. In particular, people realised that until Bitcoin becomes a medium to access all necessary goods and therefore eliminates the necessity of holding a national currency, there is a strong need to exchange Bitcoins for dollars or any other national currency. In order to achieve this, companies like BitInstant or any other exchange must be able to perform these transformations and therefore control both the price of Bitcoin and the liquidity of the commodity. Moreover, these companies act as trusted parties in a hypothetical fully decentralised ecosystem and are responsible for rightfully performing operations on behalf of customers. Mt Gox, a Japanese exchange deployed in 2010, was the absolute dominant actor in the Bitcoin ecosystem, and in early 2014 it was handling more than 70% of total Bitcoin transactions. In February 2014, the company seized all operations and filed for bankruptcy claiming they had been hacked and that the vast majority of the coins they owned had been stolen. Up until today, 650,000 Bitcoins of the stolen funds have not been accounted for.

Since the deployment of Bitcoin, the cryptocurrencies ecosystem has rapidly grown and today this multi billion business accounts for more than 10,000 different cryptocurrencies. One of the goals of the evolution of cryptocurrencies has been focusing on providing better efficiency, with the goal of one day being comparable to existing payment networks such as the Visa network. Another goal focuses on the privacy of transactions. The realisation that Bitcoin lacks strong anonymity has led to the development of various solutions which aim to enhance the privacy of users' transactions. One remedy for the lack of privacy in Bitcoin is the system of centralised mixing services, which exchange deposited coins (which in some cases may have a legally questionable background) for clean coins with no prior background. However, such Bitcoin tumblers suffer from inherent weaknesses due to their centralised trust-based model. The mixer acts as a trusted proxy and knows exactly which user sent and received which coins, and thus can easily re-establish the coins' flow. Sharing this information would cost users their privacy. Moreover, such centralised custodial services might face repercussions from the authorities. Another solution is the layer-two protocols, like the Bitcoin Lightning Network, a payment channel network where users can arbitrarily make many off-chain payments between themselves, without the need to broadcast the transactions into the blockchain. Finally, cryptocurrencies such as Zcash, Dash, or Monero were specifically designed to provide better privacy than Bitcoin.

Given that cryptocurrencies are continuously used for illicit purposes and people claim their right to transact privately, it is a very interesting research topic to measure the level of anonymity provided by currencies, the exchange services working with them and how the total ecosystem fits within the needs of Internet today.

1.2 Thesis Projects

The following is the list of the projects that the author has been a part of during this thesis:

- George Kappos, Haaroon Yousaf, Rainer Stütz, Sofia Rollet, Bernhard Haslhofer and Sarah Meiklejohn. *How to Peel a Million: Validating and Expanding Bitcoin Clusters.* In 31st USENIX Security Symposium (USENIX Security 22). This paper is fully presented in Chapter 4

- George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. *An Empirical Analysis of Anonymity in Zcash.* In 27th USENIX Security Symposium (USENIX Security 18). This paper is fully presented in Chapter 5

- George Kappos, Haaroon Yousaf, Ania M. Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. *An Empirical Analysis of Privacy in the Lightning Network.* In International Conference on Financial Cryptography and Data Security (FC 21) This paper is fully presented in Chapter 6

- Haaroon Yousaf, George Kappos, and Sarah Meiklejohn. *Tracing Transactions Across Cryptocurrency Ledgers.* In 28th USENIX Security Symposium (USENIX Security 19)

- George Kappos, Ania M. Piotrowska. *Extending the anonymity of Zcash.* Available at: https://arxiv.org/pdf/1902.07337.pdf

1.3 Collaborations and outline

All the projects presented in this thesis, as well as those not included but listed previously, were done in collaboration with other authors. I was the main contributor of the works described in this thesis.

For the work in Chapter 4, I created the balanced datasets of True Positive (TP) and False Positive (FP) clusters according to the criteria I decided were the most important. For this, the initial dataset provided by Chainalysis and Haaroon Yousaf's code that extracted some of the features were vital. I designed and implemented the Validation and Expansion heuristics as well as performed their evaluation (Sections 6, 7 and 8). The final code was built upon Haaroon Yousaf's original version that utilised transaction and address features to create peelchains. The case studies were possible due to Sarah Meiklejohn's online crawler that provided us with the data.

In Chapter 5, my main focus was the users' interactions with Zcash's shielded pool. I performed the measurements and provided the graphical representations of its usage, but more importantly designed, formally defined and implemented heuristics that were able to de-anonymize a significant percentage of the interactions.

In Chapter 6, I designed and implemented the tracing heuristic that identified private channels on-chain and, in some cases, allowed the identification of the counter-parties of the channels. Moreover, I built a Lightning Network simulator based on publicly available data that was used for the payment discovery attack. Furthermore, I designed and implemented the path discovery attack that utilised the simulator to perform measurements on the probability of intermediate nodes identifying the endpoints of payments.

For the paper *"Tracing transactions across cryptocurrency ledgers"*, I mainly focused on identifying two patterns of activities in Shapeshift that could likely be linked to money laundering (see Section 6 in the paper). I also created a case study that looked into trading bots (see Section 8 in the paper).

For the paper *"Extending The Anonymity Of Zcash"* I proposed the idea of combining mix networks and Zcash in order to enhance the cryptocurrency's anonymity at the network layer. I also presented the proof of concept of how the design would look in practice.

1.4 Thesis Contributions

This thesis is structured as follows. In Chapter 2, we first present to the reader the required fundamental knowledge for the understanding of this project. We describe Bitcoin, both on its application as well as on its network layer, which is knowledge relevant to any alternative cryptocurrency produced from Bitcoin. We then provide a description of the main privacy vulnerability of Bitcoin, the ability to cluster addresses together. Next, we describe Zcash and explain how it provides anonymity to its users. Finally, we outline the design of the Lightning Network.

In Chapter 3, we present a survey of papers relevant to this work. We divide our literature review into three sections; first, we look into the papers whose scope was to analyse privacy on the application layer of Bitcoin through measurement studies. Next, we look into the works that provided privacy measurements by exploiting the network layer. Lastly, we look into the papers that aimed to provide additional privacy to Bitcoin, either by measurements of privacy enhancing coins like Zcash or Monero, or through layer-2 technologies such as tumblers and the Lightning Network.

In Chapter 4, we present our first contribution, which aimed to analyse Bitcoin's privacy at the application layer. To the best of our knowledge, we present the first methodology for validating the correctness of address clusters created by the most popular heuristic used to cluster addresses. Moreover, we provide a change identification heuristic that is different from any prior heuristic, and we empirically evaluate it with the help of ground truth data provided to us by Chainalysis. We show that our heuristic is the most efficient heuristic proposed until now and produces the least number of false positives by order of magnitude.

In Chapter 5, we present our second contribution, which aimed to analyse the application layer privacy of Zcash. Prior to our work Zcash transactions were considered untraceable. Our measurement study showed that this claim was flawed, mainly due to the transactional behaviour of Zcash users. We implemented the co-spend heuristic on the transparent part of the Zcash blockchain, and developed a set of heuristics that traced more than 60% of transactions passing through Zcash's shielded pool.

In Chapter 6, we present our third and final contribution, which looked into the privacy of the Lightning Network. Here, we provided an alternative method for identifying hidden balances of public channels, and designed two heuristics that identified private channels. To the best of our knowledge, we were the first to identify not only those channels but also their participants. Finally, we created a Lightning Network simulator which we used to experimentally evaluate our third attack, which identified with a high probability the payment's endpoints from the perspective of an intermediate node.

As we explained earlier in this section, there are mainly two schools of thought regarding privacy in cryptocurrencies; those that believe that anonymity should be preserved for every user, irrespective of their transaction's motives, and those who believe that transactions should be regulated, hence deterministically traced. We believe that the contributions of this thesis have scientific interest for both those thoughts as our observations could be used to trace the flow of cryptocurrencies and also as a teaching guide for future developments of more private technologies.

Chapter 2

Background

Cryptocurrencies, starting with Bitcoin [1], were designed as global payment systems that do not rely on a centralised authority. Instead, they allow people around the world to send payments to each other without an intermediate authority being able to reject or delay them, or charge fees. All Bitcoin transactions are logged in an append-only public ledger, a copy of which is maintained by each of the peers in the network.

In this chapter, we outline the fundamental background knowledge related to Bitcoin, Zcash and the Lightning Network, which are the key subjects of the works included in this thesis. We start by describing the main components of Bitcoin's blockchain. Next, we briefly touch upon the consensus algorithm which ensures the correctness of the public ledger and describe the Bitcoin protocol at the network layer, i.e., how information is propagated in the peer-to-peer network. Subsequently, we outline the fundamentals of Bitcoin's privacy, which is this thesis' main contribution. Moreover, we describe how Zcash works, an alternative coin (altcoin) that was designed with built-in privacy. Lastly, we present the prerequisite knowledge for the Lightning Network which is a network run separately from Bitcoin's network. Its aim is to provide scalability, lower latency and better anonymity than Bitcoin.

2.1 Bitcoin

Understanding Bitcoin is a pre-requisite knowledge for any scientific work that focuses on cryptocurrencies. In this section we will present the background knowledge that is necessary in order to understand our paper presented in Chapter 4.

2.1.1 Application Layer

Bitcoin users are identified by addresses that act as their pseudonymous. That fact that those pseudonyms are not tied in any way to a physical entity and seem completely random is the main reason that, initially, Bitcoin was thought to provide strong anonymity. Technically, those addresses are the result of a hash function applied to a user's public part of an asymmetric key pair. The private key of this pair enables a user to send Bitcoin from a particular address. Conceptually, Bitcoin transactions transfer value from the input address(es) to the output address(es) of the transaction, and are enabled by the user signing a script that is verified with the user's private key.

In more detail, a Bitcoin transaction tx consists of ordered lists of inputs ($\mathsf{tx.inputs}$) and outputs ($\mathsf{tx.outputs}$). We use $\mathsf{tx.inputs}[i]$ to denote the i-th input and do the same for $\mathsf{tx.outputs}$. Each output output in a transaction is associated with an address $\mathsf{output.addr}$ and a value $\mathsf{output.value}$ representing the amount of coins received by the address in this transaction. Each input to a transaction is similarly associated with an address $\mathsf{input.addr}$ and a value $\mathsf{input.value}$ representing the amount of coins being sent by the address in this transaction. Furthermore, an input is itself a *transaction output* (*TXO*); i.e., an input points to the transaction in which its associated address has received coins. Other peers in Bitcoin's peer-to-peer network can then check that all inputs to a transaction are well formed and are *unspent* (meaning they are *UTXOs*), before propagating the transaction to other peers and eventually enabling its inclusion in the blockchain. This ensures that double-spending is not included in the blockchain, which acts as a global ledger of all transactions.

Concretely, this means that transactions point both *backwards*, in terms of the input UTXOs pointing to past transactions in which they received the coins they are now spending, and *forwards*, in terms of the UTXOs in future transactions that reference any output addresses that get spent. We denote the transaction that an input input points backwards to by input.prev, and the transaction that an output output points forwards to by output.next (if it is spent). We also denote by input.prevIdx the index of an input input in input.prev.outputs; i.e., the index of its transaction output in the transaction in which it was created. For the rest of this work, when we refer to *following* an input to a transaction we mean looking backwards at the past transaction that created this UTXO, and when we refer to following an output we mean looking forwards to any UTXOs that reference it.

Now that we have described how a Bitcoin transaction is created and signed by a user, we will describe how this transaction is verified by the network and added to the blockchain. Initially, the user is responsible for submitting the transaction to the peer-to-peer network. Upon receiving this transaction, a peer is responsible for verifying it, i.e., making sure the signatures correspond to the input addresses, the outputs that are being spent had not been earlier spent and that the sum of coins in the outputs is less or equal than the sum of coins in the inputs. Valid transactions are then added in a queue of transactions that are ready to be included in the blockchain. The nodes that are then responsible for extending the blockchain (the Miners) create a block with as many transactions as possible (in practice, they pick the ones with the highest transactions fees in order to maximise their profit).

2.1.2 Network and Consensus Layer

The Bitcoin network is a peer-to-peer network since all participant nodes are equal and may offer both client and server capabilities. The information within the network flows following a gossip protocol that is based on the announce paradigm. Since nodes are responsible for creating transactions and blocks, the Bitcoin network could have not followed the request paradigm (like for example BitTorrent) since nodes could not know what information to request.

Once a new node joins the network, its first job is to find and connect to other peers. To do so, the node will try to connect to a list of publicly known Bitcoin DNS servers whose job is to offer the network information of other nodes, hence enabling bootstrapping. If all of these hosts fail, a hard-coded DNS server within the Bitcoin client will be used in order to enable connections. Once the node connects to a subset of peers offered by the hosts, it will try to further expand its peers by requesting from every node their own neighbours. Once the connected nodes reply (with up to 1,000 addresses), the newly added node will have a database of known peers, known as the **addrman**. After learning about as many nodes as possible, the newly added node will try to perform 8 outgoing connections by default and 117 incoming ones, all of which it will try to maintain.

Although every node in the network has in theory the same rights and capabilities, in practice there is a distinction between normal peers and miners. The former category can create transactions by spending some of their Bitcoins, verifying the correctness of transactions and blocks created by other peers, as well as relaying them to the rest of the network. The miners can perform every operation that normal peers do, as well as create the blocks of transactions through an operation known as mining. Conceptually, a transaction is valid once it has been added in a block so the goal of a transaction creator is to make it reachable to as many miners as possible. The motivation for the miners to create blocks is the reward they will get upon the completion of a successful block. The blockchain consists of the sequence of those blocks and ensures a common view among all peers, in which transactions are valid and structured timing wise.

Since the blocks are produced in a distributed manner and there is no trivial way of collectively verifying and agreeing that a produced block by a miner is actually a valid one, a consensus mechanism is necessary in order to ensure that the blocks proposed have been created according to the protocol's specifications and that all nodes share the same view of the blockchain. The consensus mechanism used by Bitcoin is called proof of work (PoW) and its purpose is to make it computational infeasible for a potential adversarial miner to extend the blockchain in a different way than the rest of the network (and thus create a fork). In practice, creating a block is very computationally exhaustive for the miner, who needs to perform multiple hash operations before finding a random string that ensures that the final hash is below a certain threshold. This threshold can be tweaked in order to make it easier or harder to find the secret but the goal is to keep the block rate to around 10 minutes. The incentive for a miner to maintain the blockchain by wasting their computational resources is the block reward, which currently consists of a standard fee of 10 Bitcoins as well as the sum of the transaction fees of every transaction they add in the block.

2.1.3 Clustering Bitcoin addresses

A valid Bitcoin transaction needs to be signed using the private keys associated with all its inputs. This has given rise to a common heuristic for clustering together Bitcoin addresses, known as the multi-input or *co-spend* heuristic [7, 8, 5, 9]. This heuristic states that all inputs to a transaction are controlled by the same physical entity, using the fact that they have all signed the transaction as evidence of shared ownership.

Although this heuristic is considered safe in general and has been adopted in practice, it can be invalidated by a specific type of transaction called a *Coinjoin*. When forming a Coinjoin, users work together to create a transaction in which they each control a different input and the outputs likewise represent different recipients. This acts to mix together the coins of these users and thus destroys the link between each individual sender and recipient. Furthermore, it invalidates the co-spend heuristic as it is no longer the case that all inputs are controlled by the same entity.

Beyond the co-spend heuristic, there are a number of proposed *change* heuristics [9, 5, 6]; i.e., heuristics for identifying which output in a transaction the sender uses to send themselves their change (the value of their UTXO subtracted by the amount they are sending to the recipient). As observed by Meiklejohn et al., [9] identifying such outputs not only makes it possible to include this address in the same cluster as the sender and thus enhance the co-spend heuristic, it also makes it possible to identify that the transaction in which this change output is spent is also performed by the same entity. We describe this pattern of following *peel chains* in more detail in Section 4.3, and describe these proposed change heuristics in more

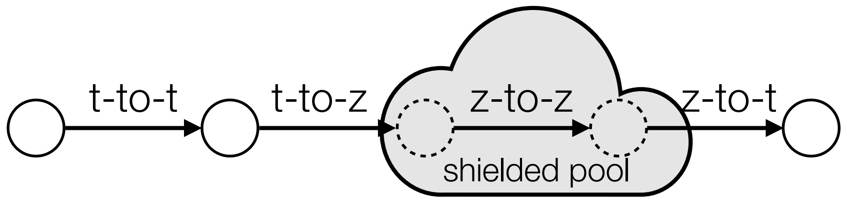


Figure 2.1: A simple diagram illustrating the different types of Zcash transactions. All transaction types are depicted and described with respect to a single input and output, but can be generalised to handle multiple inputs and outputs. In a t-to-t transaction, visible quantities of ZEC move between visible t-addresses ($\mathsf{zIn}, \mathsf{zOut} \neq \emptyset$). In a t-to-z transaction, a visible amount of ZEC moves from a visible t-address into the shielded pool, at which point it belongs to a hidden z-address ($\mathsf{zOut} = \emptyset$). In a z-to-z transaction, a hidden quantity of ZEC moves between hidden z-addresses ($\mathsf{zIn}, \mathsf{zOut} = \emptyset$). Finally, in a z-to-t transaction, a hidden quantity of ZEC moves from a hidden z-address out of the shielded pool, at which point a visible quantity of it belongs to a visible t-address ($\mathsf{zIn} = \emptyset$).

detail in Section 4.5.2 when we compare our own heuristic against them.

2.2 ZCash

In this section we will present the pre-requisite knowledge for the understanding of our work in Chapter 5.

Zcash (ZEC) is an alternative cryptocurrency developed as a (code) fork of Bitcoin that aims to break the link between senders and recipients in a transaction. In Bitcoin, recipients receive funds into addresses (referred to as the vOut in a transaction), and when they spend them they do so from these addresses (referred to as the vIn in a transaction). The act of spending bitcoins thus creates a link between the sender and recipient, and these links can be followed as bitcoins continue to change hands. It is thus possible to track any given bitcoin from its creation to its current owner.

Any transaction which interacts with the so-called shielded pool in Zcash does so through the inclusion of a *vJoinSplit*, which specifies where the coins are coming from and where they are going. To receive funds, users can provide either a transparent address (t-address) or a shielded address (z-address). Coins that are held in z-addresses are said to be in the shielded pool.

To specify where the funds are going, a vJoinSplit contains (1) a list of output t-addresses with funds assigned to them (called *zOut*), (2) two shielded outputs, and (3) an encrypted memo field. The zOut can be empty, in which case the transaction is either *shielded* (t-to-z) or *private* (z-to-z), depending on the inputs. If the zOut list contains a quantity of ZEC not assigned to any address, then we still consider it to be empty (as this is simply the allocation of the miner's fee). Each shielded output contains an unknown quantity of ZEC as well as a hidden double-spending token. The shielded output can be a dummy output (i.e., it contains zero ZEC) to hide the fact that there is no shielded output. The encrypted memo field can be used to send private messages to the recipients of the shielded outputs.

To specify where the funds are coming from, a vJoinSplit also contains (1) a list of input t-addresses (called *zIn*), (2) two double-spending tokens, and (3) a zero-knowledge proof. The zIn can be empty, in which case the transaction is either *deshielded* (z-to-t) if zOut is not empty, or private (z-to-z) if it is. Each double-spending token is either a unique token belonging to some previous shielded output, or a dummy value used to hide the fact that there is no shielded input. The double-spending token does not reveal to which shielded output it belongs. The zero-knowledge proof guarantees two things. First, it proves that the double-spending token genuinely belongs to some previous shielded output. Second, it proves that the sum of (1) the values in the addresses in zIn plus (2) the values represented by the double-spending tokens is equal to the sum of (1) the values assigned to the addresses in zOut plus (2) the values in the shielded outputs plus (3) the miner's fee. A summary of the different types of transactions is provided in Figure 2.1.

2.3 Lightning network

In this section we will present the pre-requisite knowledge for the understanding of our work in Chapter 6.

One of the most promising solutions that has been deployed to address the known issue of Bitcoin's lack of scalability is the creation of the so-called ”layer-two” protocols [10], with the Lightning Network (LN) [11] emerging as the most popular since its launch in March 2018. In Lightning, pairs of participants use the Bitcoin blockchain to open and close *payment channels* between themselves. Within a channel, these two users can arbitrarily make many *off-chain* payments between themselves, without having to use the blockchain. Beyond a single channel, Lightning supports multi-hop payment routing, meaning even participants who are not connected directly can still route payments through a broader *payment channel network* (PCN). Nodes in the network are incentivised to route payments by a fee they can charge for payments they forward.

In more detail, in order to open a Lightning *channel*, two parties deposit bitcoins into a 2-of-2 *multi-signature* address, meaning any transaction spending these coins would need to be signed by both of them. These funds represent the channel *capacity*; i.e., the maximum amount of coins that can be transferred via this channel. Once a channel is established, its participants can use it to arbitrarily exchange many payments, as long as either has a positive balance. They can also close the channel using a Bitcoin transaction that sends them their respective balances from the 2-of-2 multi-signature address.

2.3.1 Channel management

In this subsection, we describe in detail the channel management operations, including its opening and closing, and channel state updates.

2.3.1.1 Opening a channel

For Alice and Bob to create a channel between them, they must fund the channel by forming a *funding transaction* $\mathsf{tx}_{\mathsf{fund}}$. Let's assume that Alice is the one funding the channel.[1] The funding transaction consists of one input, which is a UTXO associated with one of Alice's addresses, and one output, which is a 2-of-2 multisig address representing both Alice's and Bob's public keys. This means that both Alice and Bob must provide their signatures in order to release the funds. The amount sent to the multisig address is the initial capacity C of the channel.

It is important that Alice does not just put this transaction on the Bitcoin blockchain; otherwise, her coins may be locked up if Bob refuses to provide a sig-

[1]In general, either one of the parties funds the channel, or both of them.

nature. Instead, she and Bob first go on to form two *commitment transactions*, one for each of them, which represent their agreement on the current state of the channel. Each commitment transaction has the 2-of-2 multisig as input, and has two outputs: local and remote. The transaction $\mathsf{tx}_{A,i}$, representing Alice's view of state i, essentially sends her current balance to local and sends Bob's current balance to remote. Likewise, $\mathsf{tx}_{B,i}$ sends Bob's balance to local and Alice's balance to remote.

The remote output is simply the address of the other involved party (so addr_B in $\mathsf{tx}_{A,i}$ and addr_A in $\mathsf{tx}_{B,i}$), but the local output is more complicated. Instead, the local output in $\mathsf{tx}_{B,i}$ is encoded with some timeout t, and awaits some potential input σ. If the timeout t has passed, then the funds go to addr_B as planned. Otherwise, if a valid *revocation signature* σ is provided before time t, the funds go to addr_A. This means that the revocation signature allows one party to claim the entire capacity of the channel for themselves. As we explore fully in Section 2.3.1.2, this is used to disincentive bad behavior in the form of publishing old states.

To create a channel, Alice thus creates the transaction $\mathsf{tx}_{B,0}$, sending C to remote and 0 to local (since so far she has supplied all the funds for the channel). She signs this transaction and sends her signature to Bob, who signs it as well. Bob then forms $\mathsf{tx}_{A,0}$, sending 0 to remote and C to local, and sends his signature on this to Alice. Alice then signs it and publishes $\mathsf{tx}_{\mathsf{fund}}$ to the Bitcoin blockchain.

At this point, Alice and Bob both have valid transactions, meaning transactions that are signed by both parties in the input multisig, which is itself the output of a transaction on the blockchain (i.e., the funding transaction). Either of them could thus publish their transaction to the blockchain to close the channel. Alternatively, if they mutually agree to close the channel and want to avoid having one of them wait until the timeout to claim their funds, they could update to a new state without the timeout in local and publish that.

2.3.1.2 Updating a channel state

Once both Alice and Bob have signed $\mathsf{tx}_{A,0}$ and $\mathsf{tx}_{B,0}$, they have agreed on the *state* of the channel, which represents their respective balances B_A and B_B. In particular, the amount sent to local in $\mathsf{tx}_{A,0}$ and to remote in $\mathsf{tx}_{B,0}$ represents Alice's balance, and similarly the amount sent to local in $\mathsf{tx}_{B,0}$ and to remote in $\mathsf{tx}_{A,0}$ represents Bob's balance. The question is now how these transactions are updated when one of them wants to pay the other one, and thus these balances change.

In theory, this should be simple: Alice and Bob can repeat the same process as for the creation of these initial commitment transactions, but with the new balances produced by the payment. The complicating factor, however, is if the old commitment transactions are still valid after the creation of the new ones, then one of them might try to revert to an earlier state by broadcasting an old commitment transaction to the Bitcoin blockchain. For example, if Alice pays Bob in exchange for some service, then it is important that once the service has been provided, Alice cannot publish an old commitment transaction claiming her (higher) balance before she made the payment.

This is exactly the role of the *revocation signature* introduced earlier. In addition to exchanging signatures on the new transactions $\mathsf{tx}_{A,i+1}$ and $\mathsf{tx}_{B,i+1}$, Alice and Bob also exchange the revocation secret keys $\mathsf{sk}_{A,i}$ and $\mathsf{sk}_{B,i}$ that correspond to the public keys $\mathsf{pk}_{A,i}$ and $\mathsf{pk}_{B,i}$ used in $\mathsf{tx}_{A,i}$ and $\mathsf{tx}_{B,i}$. These public keys are derived from base public keys pk_A and pk_B such that (1) both Alice and Bob can compute $\mathsf{pk}_{A,i}$ and $\mathsf{pk}_{B,i}$ for any state i, thus can independently form $\mathsf{tx}_{A,i}$ and $\mathsf{tx}_{B,i}$ as long as they know the right amounts, and (2) the corresponding secret keys $\mathsf{sk}_{A,i+1}$ and $\mathsf{sk}_{B,i+1}$ are completely unknown until one party reveals them to the other. Thus, up until they exchange revocation keys, they can broadcast the transactions for state i to the Bitcoin blockchain as a way to close the channel. Once they exchange the secret keys, however, Bob can form a valid revocation signature and claim all of Alice's funds in local if she broadcasts $\mathsf{tx}_{A,i}$ (it is indeed only Alice who can broadcast a valid $\mathsf{tx}_{A,i}$ as only she has both her and Bob's signatures on it). At this point the new channel state $i+1$ is confirmed and Bob can safely perform his service for Alice.

2.3.1.3 Closing a channel

As long as either Alice or Bob has a positive balance, they can send payments to the other, knowing that if there is a disagreement they can settle their previously agreed channel state onto the blockchain, as described in the previous section. If there is no dispute, and both Alice and Bob agree to close the channel, they perform a *mutual close* of the channel. This means providing a signature that authorises a *settlement transaction*.

2.3.1.4 Hashed time-lock contracts (HTLCs)

The previous section described how the state of a two-party channel can be updated, in which both Alice and Bob are sure that they want a payment to go through, and can thus transition to the new state immediately. In a broader network of channels, however, it may very well be the case that two parties need to prepare their channel for an update that does not happen. To see why, remember from Section 2 that when Alice is picking a path from herself to Bob, the information she has about the channels along the way is their capacity C, which is $\mathsf{C} = \mathsf{C}_{\mathsf{in}} + \mathsf{C}_{\mathsf{out}}$. If $\mathsf{cid}(U_{n-1} \leftrightarrow U_n)$ has capacity $\mathsf{C} \geq \mathsf{amt}$ but $\mathsf{C}_{\mathsf{out}} < \mathsf{amt}$ (i.e., U_{n-1} does not have enough to pay U_n the amount Alice is asking), then the payment fails at this point, so no one along the path should have actually sent any money. Equally, the payment could fail due to a malicious intermediary simply deciding not to forward the onion packet or otherwise follow the protocol.

To thus create an intermediate state, and to unite payments across an entire path of channels, the Lightning Network uses *hashed time-lock contracts*, or *HTLCs* for short. In using HTLCs, Alice and Bob can still transition from $\mathsf{tx}_{A,i}$ and $\mathsf{tx}_{B,i}$ to new transactions $\mathsf{tx}_{A,i+1}$ and $\mathsf{tx}_{B,i+1}$ representing a payment of n coins from Alice to Bob, but the first message that Alice sends includes the hash h and timeout t. This signals to Bob to add an additional output htlc to $\mathsf{tx}_{A,i+1}$, which sends n coins to Alice if the time is greater than t (as a refund) and sends them to him if he provides as input a value x such that $H(x) = h$. Alice and Bob then proceed as usual in exchanging signatures and revocation keys for their respective transactions.

If at some point before t Bob sends such an x to Alice, then it is clear to both parties that Bob could claim the htlc output of $\mathsf{tx}_{A,i+1}$ if it were posted to the blockchain. They thus transition to a new state, $i+2$, in which they go back to two-output commitment transactions, with the additional n coins now added to Bob's balance.

Going back to the third step in the description provided in Section 2 then, U_{i-1} and U_i prepare their channel by updating to this intermediate state $j+1$, using the h and t_i provided in the packet onion_i to form the htlc output. In the fifth and final step, they settle their channel by updating to state $j+2$ by removing the htlc output, once U_i has sent the pre-image x to U_{i-1} and thus demonstrated their ability to claim it. If Bob never provides the pre-image x, then by the pre-image resistance of the hash function no party along the path is able to claim the htlc output, so the payment simply does not happen. If some malicious U_i along the path decides not to continue forwarding x, then all previous U_k, $1 \leq k \leq i$ can still claim the htlc output in the intermediate state.

Most users, however, are not connected directly, so instead need to *route* their payments through the global Lightning Network. Here, nodes are identified by public keys, and edges represent channels, which are publicly associated with a *channel identifier* cid, the channel capacity C, and a *fee* fee that is charged for routing payments via this channel. Privately, edges are also implicitly associated with the *inward* and *outward balances* of the channel. Except for *private* channels, which are revealed only at the time of routing, the topology of this network and its public labels are known to every peer. When routing a payment, the sender (Alice) uses *onion routing* to hide her relationship with the recipient (Bob). Alice selects the entire path to Bob (*source routing*), based on the capacities and fees of the channels between them. The eventual goal is that each intermediate node on this path forwards the payment to its successor, expecting that its predecessor will do the same so its balance will not change. The nodes cannot send the money right away, however, because it may be the case that the payment fails. To thus create an intermediate state, LN uses *hashed time-lock contracts* (*HTLCs*), which allow for time-bound conditional payments. In summary, the protocol follows five basic steps to have Alice pay Bob:

1. Invoicing Bob generates a secret x and computes the hash h of it. He issues an *invoice* containing h and some payment amount amt, and sends it to Alice.

2. Onion routing Alice picks a path $A \rightarrow U_1 \rightarrow \cdots \rightarrow U_n \rightarrow B$. Alice then forms a

Sphinx [12] packet destined for Bob and routed via the U_i nodes. Alice then sends the outermost onion packet onion_1 to U_1.

3. Channel preparation Upon receiving onion_i from U_{i-1}, U_i decodes it to reveal: cid, which identifies the next node U_{i+1}, the amount amt_i to send them, a timeout t_i, and the packet onion_{i+1} to forward to U_{i+1}. Before sending onion_{i+1} to U_{i+1}, U_i and U_{i-1} *prepare* their channel by updating their intermediate state using an HTLC, which ensures that if U_{i-1} does not provide U_i with the pre-image of h before the timeout t_i, U_i can claim a refund of their payment. After this is done, U_i can send onion_{i+1} to

4. Invoice settlement Eventually, Bob receives onion_{n+1} from U_n and decodes it to find (amt, t, h). If amt and h match what he put in his invoice, he sends the invoice pre-image x to U_{n-1} in order to redeem his payment of amt. This value is in turn sent backwards along the path.

5. Channel settlement At every step on the path, U_i and U_{i+1} use x to *settle* their channel; i.e., to confirm the updated state reflecting the fact that amt_i was sent from U_i to U_{i+1} and thus that amt was sent from Alice to Bob.

Chapter 3

Literature Review

When Satoshi Nakamoto published the whitepaper [13] in 2008, it was initially thought by many that Bitcoin provides privacy since it allows users to transact without revealing their real-life identities. Instead, all transactions are conducted between wallet addresses that serve as pseudonyms. Since the publication of the whitepaper, a long line of research has shown that the initial claim of privacy was inaccurate in many different ways. In fact, Bitcoin transactions can be traced by exploiting flaws both at the application as well as the network layer.

In this chapter, we survey the related literature. We first look into the early studies on Bitcoin's anonymity and touch upon the observations they made. Next, we survey other cryptocurrencies as well as privacy-enhancing technologies that build on existing cryptocurrencies. This chapter is structured as follows; in the first section we focus on the papers that looked into Bitcoin's privacy by analysing the public data on its application layer, i.e. the blockchain. Our own work in chapter 4 is an extension, and in some ways, an improvement to these papers. In the second section we focus on papers analysing Bitcoin's anonymity on the network layer. Lastly, we divide our third section into two subsections; first we look into privacy-preserving alternative cryptocurrencies (or altcoins), a research area relevant to our own work in chapter 5. Our last subsection focuses on privacy-enhancing technologies that work on top of Bitcoin and other pre-existing blockchains, which is a research area relevant to our own work in chapter 6.

3.1 Application-layer privacy

The first privacy limitation of Bitcoin was already mentioned by Satoshi Nakamoto [13] in the original Bitcoin whitepaper. Nakamoto observed that transactions with more than one input address may leak important information. Bitcoin's protocol specifies that in order to spend from a public address, knowledge of the corresponding private key is necessary. Therefore, in the case of transactions with multiple inputs one could assume that all the input addresses belong to the same user. This application-layer exploit was extended by an early body of research [4, 8, 14, 15], it was named the *co-spend heuristic* and it is until today the main heuristic used by researchers and blockchain analytic companies [16, 17] to analyse and determine the ownership and flow of Bitcoin and other altcoins.

Ron [8] introduced the notion of an *entity graph* as opposed to the more primitive *address graph*. The entity graph can be constructed by clustering addresses, believed to be operated by the same individual. The merging was performed by running a variant of the Union-Find algorithm on every transaction with more than one distinct input address. One of the paper's main findings was that the Bitcoin network was very centralised, since a handful of entities accounted for the majority of Bitcoin's activity, both in terms of number of transactions and the value of Bitcoins held. The authors observed that the then second biggest Bitcoin entity, in terms of holding value, was the Mt Gox exchange [18]. Interestingly, MtGox was hacked less than a year after the paper was published. Moreover, using a number of heuristics, the authors concluded that the majority of Bitcoins owned by users were held in cold storage, meaning their purpose was to stay idle for a very long period of time rather than being circulated.

Although [8] is known primarily as a measurement study aiming to quantify different properties of the Bitcoin network rather than focusing on privacy, it provided the foundations for extensive literature focusing entirely on Bitcoin's anonymity. First of all, they went beyond just creating clusters of addresses, but actually allocated them to real-world entities such as WikiLeaks, Mt.Gox, Instawallet and Deepbit. They did so by combining the co-spend heuristic with publicly available information. Moreover, they defined several patterns of Bitcoin usage at the application layer which provided the foundation for today's extensive Bitcoin tracing capabilities. As an example, they introduced the notion of *change addresses*, i.e., transaction output addresses that belong to the same user as the input addresses.

Almost concurrently to [8], Reid and Harrigan [4] tried to perform a similar analysis using the Bitcoin blockchain. They introduced a distinction between a *transaction graph* and a *user graph.* Their transaction graph represented transactions as vertices and transaction inputs and outputs as direct edges between the vertices. Similarly, their user graph depicted clusters of addresses as vertices, and coin transfers as edges between the vertices. In both cases, they enhanced their edges with features such as transfer values and timestamps. Next, the authors applied several graph algorithms to both the transaction and user graphs to extract various information. First of all, they introduced an efficient methodology for address clustering, by representing each Bitcoin address as a node and pairwise connecting all the input addresses of each transaction. After doing so, they applied the connected components algorithm to the graph and obtained all the address clusters. Moreover, they showed four different ways of using the user graph in order to deduce secret information. Firstly, they showed how external information could be used in order to correlate nodes to known services or even their IP addresses. They made those correlations by scraping the Internet for voluntary published public keys or network information. Furthermore, they described how information within the Bitcoin network could be exploited in order to allocate user nodes to IP addresses. Lastly, they showed how to directly extract information from the user graph by looking into a node's neighbours within a few hops and how Bitcoin funds can be traced by observing the paths between a sender and a destination node, within those graphs.

While Ron [8] focused mainly on performing measurements in the Bitcoin graph, Reid's [4] analysis was more privacy oriented. However, those two early papers were similar in the sense that the entity graph represented in [8] is identical to the user graph in [4]. Moreover, they both mapped some clusters of addresses to real-life entities using information external to the blockchain. However, the mapping in both works happened for a handful of cases and it was not until later that the potential of the co-spend heuristic was understood.

Meiklejohn et al., [9] were the first to show the full potential of the co-spend heuristic by mapping clusters to most of the biggest, at the time, Bitcoin services, ranging from mining services and exchanges to gambling services or even the early mixing services. Their main source of ground truth data originated from multiple manual interactions with each service, as well as from scraping for online information. In addition to implementing the co-spend heuristic at scale, the authors made general blockchain measurements and compared their results to previous works [8], aiming to focus on the temporal aspect of Bitcoin i.e., how it evolved over time. For instance, they noticed that the lifespan of UTXO's i.e., the block difference between the creation of a coin and how it is spent, significantly decreased compared to when [8] performed it. They showed that this was mainly due to an emerging gambling service, Satoshi Dice, which was the most active cluster at the time. The lifespan of coins sent to Satoshi Dice was significantly low, since the service was immediately paying back their users after they transacted with the service.

Moreover, they designed a second clustering heuristic, complementary to co-spend, that aimed to further collapse Bitcoin addresses to a common operator. To this end, they designed and implemented a *change heuristic* that aimed to identify whether there is a unique output address that belongs to the same entity as the input addresses. The proposed heuristic identifies a *change address* for a transaction if (1) there is an output address that appeared in the blockchain as an output only once (2) there is no intersection between input and output addresses, (3) it is not a coin generation transaction and (4) exactly one output address satisfies (1) and (2).

On top of providing additional clustering of Bitcoin addresses, their change heuristic enabled the development of a transaction pattern in UTXO-based currencies, the *peeling chain* or *peelchain* for short. A peelchain usually begins with a big input coin and the transaction that spends it sends one or more smaller amounts to other addresses. The remainder is transferred to a change address. This transaction pattern is repeated until the input amount becomes small enough that it cannot continue ”peeling off”, at which point it is either spent without generating change, which signals the end of the peelchain, or it is co-spent with a different coin of higher value, in order to continue the same pattern. The authors applied this peelchain heuristic in order to trace known Bitcoin theft cases and hence identified their final destinations.

Prior to [9], Androulaki et al., [14] defined a set of privacy properties for Bitcoin in order to formally evaluate them. Their evaluation used a combination of heuristics that exploited both the implementation details of the then main Bitcoin client and the behaviour of the individuals operating the clients. The former set of heuristics involved the co-spend heuristic, as well as the identification of *shadow address*, which intuitively is similar to Meiklejohn et al.'s change address. They enhanced those two heuristics with additional behaviour-based clustering techniques. Those exploited similarities between different transactions in terms of (1) the timing of transactions, (2) the values of the transactions, and (3) the indices of the addresses within a transaction. In order to evaluate their heuristics, they created a simulated environment that tried to mimic the Bitcoin network within the university the research was conducted.

Ermilov [6] also designed an algorithm that collected address tags in two different ways, either passively through scraping the Internet for available information or from actively interacting with Bitcoin services and obtaining tags, similar to [9]. Those tags would then be used to perform address clustering using two heuristics; the co-spend but with a tweak of ignoring transactions with multiple outputs, since they believed those could include false positives. As a second heuristic they created their own change finding heuristic which they defined as follows: in order to find a change address, (1) a transaction must have exactly 2 outputs, (2) the number of inputs must be different than 2, (3) there should be no intersection between the input and the output addresses, (4) exactly one output address must be freshly created, and (5) the change address's value must have at least 4 decimal points. We implemented all change finding heuristics of [9, 5, 6] and compared them with our own in chapter 4

Spagnuolo [19] used the co-spend heuristic as well as the identification of shadow (or change) addresses implemented by [5] in order to create clusters of addresses. They combined those two heuristics with a scraper that required little to no supervision that scraped various websites for tags. Their main contribution compared to similar works was the creation of an efficient framework that allowed users to make complicated queries for aggregated statistics, such as total value sent and number of interactions between entities on both an address as well as a cluster level. Their framework was one of the earliest practical forensic tools for Bitcoin investigations. They used their framework in order to trace the funds of ransomware cases, such as the CryptoLocker, as well as cases of illicit payments originating from the Silk Road marketplace.

3.2 Network layer privacy

Privacy at the network layer was not taken into consideration in early Bitcoin implementations. As an example, the network traffic was unencrypted by design. Presumably, the rationale behind this decision was that users could instead use Tor [20] or other anonymous communication mechanisms like Network Address Translation (NAT) [21] or a mix network [22, 23, 24]. However, as we later explore, even those network anonymity tools are not sufficient to protect users' privacy.

The first vulnerability on Bitcoin's network layer was reported by Lerner on the BITCOINTALK forum [25]. The goal of the proposed attack was to disclose the real-life identities of Bitcoin users by correlating their transactions with their IP addresses. Lerner exploited a Denial-of-Service (DoS) protection of the bitcoind wallet [26], that prevents the propagation of very low value transactions, i.e., ”penny-flooding” transactions. Importantly, this mechanism would be used by a user's wallet as long as the transaction in question was from an address they controlled. This conditional check would be then used as an oracle by an attacker in order to determine which addresses were controlled by a particular wallet.

At the same time, Koshy et al., [27] also tried to correlate Bitcoin public keys to IP addresses by creating their own Bitcoin client and optimising it for sniffing all

Bitcoin traffic. They next configured their client to not ban other nodes or ignore malfunctioned transactions, because even those network packets were important for their work. First, they connected their wallet to every available peer and recorded all traffic continuously for 5 months. Their goal was to guess the origin (i.e., the owner of the input addresses) of every incoming transaction. They did so by observing the behaviour of the node from which they received the transaction. In total, they managed to correlate around 1,000 Bitcoin addresses to IP addresses by exploiting particular transaction-relaying behaviour from the sender node. They were not able to identify the origin of transactions that followed the most common relay patterns and also they could not handle the cases where the origin was behind a NAT or Tor. Nevertheless, their work showcased the consequences of having network traffic unencrypted and, more importantly, with no metadata protection in place.

The first sophisticated attack exploiting Bitcoin's network layer data was designed by Biryukov [28]. The authors were able to map Bitcoin transactions to nodes using a modest adversarial model. Moreover, they did so while at the same time being able to target people using NAT and Tor. The first step for their attack aimed to prevent their targets from using Tor. In order to do that, the authors exploited a native DoS protection of the Bitcoin client that would ban peer nodes that send malformed packets. To this end, they used Tor themselves and sent malformed packets to the destination servers that in return banned the nodes from which the malformed packets were received, i.e., the Tor exit nodes. That way, any victim node that used Tor from there on would be unable to access the Bitcoin network at all, since every Tor exit node would be banned. The next goal was to learn about the network topology and, in particular, about the entry nodes that their victims were connected to. To this end, the authors connected to as many entry nodes as possible and listened to the traffic originating from them. They exploited the address propagation mechanism, and were able to at least determine an important fraction of the victims' entry nodes. They showed that learning only three of the victim's entry nodes was enough to map the triplet of nodes to a unique target node. After learning about the victims' entry nodes the mapping of transactions to victim nodes was performed in the following way: they listened to all of their connections for incoming messages signaling a new transaction and, according to the set of peers from which they heard it from, they were able to estimate from which of the victim nodes the transaction originated.

Miller et al., [29] tried to infer the topology of the Bitcoin network by exploiting its address propagation mechanism. In particular, the mechanism had a peculiar feature at that time: whenever node A learned from its neighbour N about the existence of node T, A would add in their list of known nodes (*addrMan*) node T, alongside a timestamp t, which was the time they heard about it, plus 2 hours of "penalty". The authors exploited this feature by sending *getAddr* messages to all their neighbours and observing the timestamps accompanying each connected pair. Intuitively, by counting how many times the difference of 2 hours was present, the authors could determine whether two nodes shared a link and when the link was created. The authors named this probing technique *AddressProbe* and included it in their custom build wallet *CoinScope* optimised for this attack.

The technique by Biryukov [28] required disturbing the network, by flooding a node's *addrMan* list with malfunctions entries. The *AddressProbe* technique, on the other hand, allowed the node to act as a passive observer that collected and analysed information that nodes would provide anyway. Hence, the attack proposed in [29] was more robust and scalable.

Neudecker et al., [30] also tried to infer Bitcoin's network topology and achieved a precision and recall of approximately 40%. Instead of exploiting the address propagation mechanism as in [29], the authors utilised the flooding mechanism used for transaction propagation in Bitcoin. In particular, they performed a timing analysis on the transactions a node receives from different sources. The main advantage of their method over [28] was that the attack required a passive adversary who does not flood the network in any way. Compared to [29], they did not exploit a particular detail of a protocol that could later be patched but instead their attack was more generic and could be applied in any P2P network that used flooding as a propagation mechanism.

As a countermeasure to the timing analysis attack in [30], the Bitcoin core client added the following feature: each time a client received a transaction which was validated, it was not immediately forwarded to neighbouring peers. Instead, it was held in a queue and then forwarded in a batch after a random time interval. Interestingly, this transaction batching feature was later exploited by Grundmann et al., [31] who designed the following attack: an adversary would connect to a subset of all reachable nodes and then create one transaction for each of their neighbours. After observing the batches of transactions the adversarial node would receive from each neighbour, they could learn with a high probability some links within this subset of nodes. One of the drawbacks of this technique was that it could not target non-reachable nodes (as opposed to [28]). Moreover, the authors could not target a specific node and learn about its neighbours, since the links they learned were random.

Neudecker [30] designed and experimentally evaluated a second attack that exploited the double spending prevention mechanism that Bitcoin implemented. In particular, during this attack, the adversary would create a set of unique transactions that would all spend from the same output and send one to each of their peers. Afterwards, according to which of the unique transactions they would hear of from a separate target node, they could determine which of their neighbours the target node shared a link with. This attack had the advantage over the first, in that it could determine the links of a specific node rather than finding a set of random links. However, it still required the attacker to connect to every single node in the network, which made it hard to implement at scale.

Delgado [32] aimed to perform a full scale network topology inference by exploiting the core client's feature of handling *orphan* transactions. In its basic form, the attack worked as follows; an attacker A connects to nodes B and C, and her goal is to determine whether B and C share a link. To this end, the attacker sends two conflicting double-spending transactions to B and C and subsequently sends a third transaction to B. Due to the handling of orphan transactions, the attacker could determine whether B heard about the third transaction (which means that B and C share a link) or it did not hear about it, which means that they were not connected.

This attack variant was useful to determine whether B and C were connected but it could not scale out for the entire network. In order to get a full network topology inference, the authors took several additional steps. They used the CoinScope node software built in [29] in order to achieve synchronicity within their attack, i.e., to ensure all transactions would arrive at their destinations at the same time and to ensure isolation, i.e., meaning transactions sent by the adversary will remain in the destination node and only there. In order to extend the link inference to multiple nodes the authors exploited the handling of the *mapOrphanTransactions* pool management by the clients. The authors achieved a precision of over 90% on their inference technique and created a full snapshot of the Bitcoin TestNet graph.

Neudecker et al., [33] were the first to develop a privacy breaking heuristic that combined information from both the application and network layer of Bitcoin. In particular, they implemented all the clustering heuristics developed until then and their goal was to investigate whether the mapping of a cluster to a unique IP address was possible. To this end, they deployed two nodes for monitoring purposes and connected them to every reachable node in the network. Furthermore, they correlated an IP address to a particular transaction if (1) the same IP address was the first to inform both of the monitored nodes about the transaction, (2) both of the monitored nodes heard about the transaction from this IP address within 100 ms, and (3) no other IP address informed them about this transaction within a time threshold calculated according to each node's location and estimated latency. By combining the clustering heuristics with the IP address identification, the authors were able to perform a validation of their results that prior network-layer works were unable to do. In particular, by mapping several transactions to the same IP address, they could then check if those transactions were mapped to the same cluster according to the application layer heuristics as well. Additionally, they were able to perform some measurements regarding the consistency of IP addresses within a cluster and their main observation was that a handful of IP addresses were responsible for a large portion of all incoming transactions.

One of the main criteria used to evaluate network layer papers is their ability to target non-reachable nodes, potentially using Tor. In general, up until 2015, the common belief was that using Tor provided protection regarding the users' network identity. Biryukov et al., [34] showed that the usage of Tor not only did not provide any additional anonymity to the users, but also harmed it by adding a potential attack surface. In particular, their attack performed the following steps; initially the attacker would force Tor clients to route their traffic through attacker-controlled Tor exit nodes. To achieve this, the attacker would abuse the native DoS protection that Bitcoin clients were using and force benign Tor exit nodes to be banned, thus making attacker-controlled Tor exit nodes exclusively available. After forcing Tor clients to use the adversarial exit nodes, the attacker would flood the address lists of the victim with malfunctioned Bitcoin addresses that would not be white-listed at that time and would act as a unique identifier (similar to cookies in browsers). In the last step, whenever the victim would connect to the Bitcoin network without using Tor, those cookies could be used to correlate the set of transactions already allocated to the cookie, with the sender's actual IP address.

3.3 Privacy-enhancing solutions

In this section, we shift from empirical works that focused on the analysis of Bitcoin's privacy and instead focus on papers that tried to enhance the privacy of cryptocurrency technologies.

3.3.1 E-cash designs with built-in privacy

Most of the initial e-cash proposals to solve Bitcoin's privacy problems relied on a trusted issuer to provide the currency, usually referred to as a "*bank*" [35, 36]. This could be a centralised entity explicitly appointed by the system, or the responsibility of issuance could be distributed among several nodes, assuming some threshold would behave benignly and reliably. In this thesis we will not review these works but only focus on proposals that enhance Bitcoin's privacy without adding a central authority.

3.3.1.1 Zcash

Miers et al., introduced Zerocoin [37], a distributed e-cash system that used cryptographic techniques to provide transaction unlinkability, i.e., obfuscate the link between the creation of an output and its spending, without adding any additional trust to the network. To this end, the authors extended Bitcoin's functionality by adding new transaction types: the *mint* and the *spend* transactions. The former type of transaction is used when a user wishes to exchange part of their Bitcoins to a Zerocoin, whereas the latter is used in order to spend one Zerocoin and transfer back the same amount of Bitcoins to the user. All Zerocoins have the same fixed value and are represented by a Pedersen commitment value, C, to a serial number s and an opening value r, which are both secret and known only to the Zerocoin's owner. When a user, Alice, wishes to spend one of her Zerocoins, she reveals the secret s and creates a non-interactive zero-knowledge proof π that (1) she knows secret r used alongside s to produce C, and (2) C belongs to a list of previously logged and publicly known commitments that represent all the Zerocoins in the system.

Zerocoin's implementation required the creation of a separate cryptocurrency or a network's hard-fork, which required nearunanimity to be attempted in practice. Moreover, the cryptographic techniques used by Zerocoin led to high verification times and large proof sizes, which made the protocol impractical. Furthermore, the construction of Zerocoin allowed only for the creation and spending of fixed-value coins. Thus it did not support flexible transaction payments, such as the ability to create a change output. Lastly, although the *mint* and the *spend* transactions hid the sender and the recipient, respectively, other crucial information was still revealed. For those reasons, the Zerocoin protocol could be used in practice more as a decentralised tumbler rather than a standalone cryptocurrency.

Danezis et al., [38] proposed an improvement of the Zerocoin protocol to reduce the verification time of the zero-knowledge proofs and their sizes. In particular, instead of using the Strong RSA assumption and the double-discrete logarithm proofs used in Zerocoin, they used elliptic curves and bilinear pairings. In order to improve upon those, they used the Pinocchio [39] construction, which is a pairing-based proof system, an alternative to the double-discrete logarithm proof but with better performance. Although [38] improved Zerocoin's performance, there were several issues that were still unsolved. First of all, although faster and lighter, Danezis's proposal did not scale out, which meant that there could be only a limited number of Zerocoins in circulation before the system would become slow. Moreover, the Zerocoin value would still be fixed, not allowing for flexible transactions. Finally, the anonymity was preserved only for the transaction's sender, while the recipient and the transferred value would still be visible.

Eli Ben-Sasson [40] tried to tackle all the issues identified in [37, 38] and created a full-fledged anonymous altcoin. They benefited from the contemporary performance improvements of the zk-SNARKs [41, 42, 43, 44] and used additional cryptographic primitives to increase the system's functionality. In total, they managed to reduce the transaction size to less than 1 kB and made the proofs verifiable in under 6 ms, which was a significant improvement compared to the original Zerocoin idea. Their protocol ended up becoming the privacy altcoin, Zcash [45]. In terms of functionality, their main improvement over Zerocoin, was to incorporate in the commitments, which were still used to represent coins, information regarding the value of the coin and the address to which it belongs. By doing so, they allowed for the creation (minting) of coins with fluctuating values that could be transferred, by tweaking the address field accordingly.

Quesnelle [46] was the first to empirically measure the privacy offered in Zcash. His first contribution was to measure the extent to which Zcash users were making use of the optional privacy features. He noticed that only 19.6% of the transactions at the time were utilising the *shielded address* feature, which meant that the remaining 80.4% were susceptible to all the attacks possible in Bitcoin. Out of the 19.6%, only 1.9% were using shielded addresses for both the sender and the recipient of the transaction (hence hiding the amount transferred as well), which meant that in the vast majority of the transactions, some information was revealed.

Moreover, the author noticed that pool users frequently sent their transparent coins to shielded addresses and quickly returned them back to transparent addresses, calling this pattern a *round-trip transaction*. He then tried to exploit this, by linking each shielded transaction to its corresponding de-shielded transaction, based on timing and value. To this end, for each shielded transaction, he looked for a de-shielded transaction occurring within a short time window and carried the exact same value, and this value was unique across the whole blockchain. This simple heuristic was able to trace Zcash going through its privacy pool, accumulating up to 31% of the total value.

Biryukov et al., [34] also worked on empirically evaluating privacy in Zcash. Their first contribution was to build their own Zcash blockchain parser that was an extension to the Blocksci parser build in [47]. The authors were able to parse through all transactions interacting with the shielded pool in 5.6 seconds. In terms of their de-anonymization heuristics, they focused entirely on de-anonymizing the interactions performed by the mining pools since those were the ones mainly using it. To this end, Biryukov et al., proposed two heuristics for each of the most common patterns mining pools exhibited utilising mining pool addresses they scraped from their respective websites.

3.3.1.2 Monero

Nicolas van Saberhagen [48] proposed an alternative solution to Bitcoin, CryptoNote, which would enhance its privacy. The first problem CryptoNote tried to tackle was the linkability of Bitcoin payments every time an address was reused. To this end, [48] introduced the notion of unlinkable payments in the form of one-time (or *stealth*) addresses. In particular, whenever a user wishes to receive payments, he can publish his public key, which potential payers can use to derive stealth addresses.

The second improvement of [48] was untraceable payments. In Bitcoin, to spend an output coin, a user must sign it with his private key. Hence the spending of an output can be trivially connected to its creation. In order to fix this, CryptoNote's transactions use a cryptographic primitive called ring signatures, in particular a scheme proposed by Fujisaki et al., [49]. In the context of CryptoNote, the ring signatures worked in the following way. When Alice wishes to spend an output sent to her stealth address, instead of directly signing it, she looks for *mixins*, i.e., other same value outputs not belonging to her. She then creates a ring signature where the group consists of the owners of all the selected common-valued outputs. Intuitively, the more common-valued outputs she selects, the greater the anonymity set of her transaction, hence the anonymity she gains.

Monero was built from CryptoNote and, soon after its deployment, a research bulletin published by the Monero Research Lab (MRL) and in particular by Noether [50] warned about a potential vulnerability in Cryptonote's untraceability feature. They noticed that by not adding mixins, a transaction creator not only harms the privacy of the particular transaction but also creates a domino effect, harming other entities' privacy as well, known as the "chainreaction" effect. They also noticed that this observation could be used by an active attacker who could create as many unspent outputs as possible and monitor their spends. A few months later, MRL published a second research bulletin [51], which focused on Monero's traceability limitations. The authors reiterated the problem of zero-mixin transactions and claimed that making it mandatory for each transaction to have at least one mixin would eventually solve the problem. Moreover, they focused on the probability distribution of the choice of mixins within a transaction. In particular, they noticed that a transaction spending 2 outputs that were created at exactly the same block was a strong indicator that those outputs are real spendings and not mixins. This was especially accurate if the two spending outputs were created not only during the same block but on the exact same transaction.

In case of transactions with inputs created at different blocks, the authors described a methodology to distinguish probabilistically which of the inputs was the real spending. To this end, they focused on the age distribution of the outputs and noticed that if a client picked mixins uniformly at random, it meant that all outputs were equally probable to be selected; hence, the older an output was, the more times it would be picked as a mixin. Moreover, users tend to spend their coins fast, so in a transaction that used mixins, the oldest input was the least likely to be a real spend, and the newest had the highest probability of being the correct one.

Noether et al., [52] introduced *ring confidential transactions* (*RingCTs*) that aimed to patch the discovered vulnerabilities of Monero related to public values. Möser et al., [53] provided the first empirical work on Monero's anonymity by measuring the impact of the two vulnerabilities mentioned earlier. They noticed that when a Monero client picked mixins, it did not check whether those mixins had already been spent ambiguously, i.e., spent with no other mixins involved. Their measurements showed that 64% of all transactions did not have any mixins involved, hence no anonymity. More importantly, out of the remaining 36% of transactions, in 63% of those the real inputs could be distinguished just by simple elimination.

In terms of exploiting Monero's mixins selection vulnerability, the authors made the following contributions; they created a ground-truth dataset by looking into the transactions that either used no mixins or used mixins but the real input could be deduced using the first vulnerability. By doing so, they obtained a set of transactions whose input was known and the age of the spent coins could be simply calculated. They also enhanced this ground-truth dataset with transactions they performed using the Monero wallet with default mixin-selection parameters and chose the time interval between the output-creating and output-spending transactions using three different distributions. They noticed that users tend to spend their Monero coins relatively fast, hence the spending outputs were very new. For those transactions, they also looked into the age distribution of the mixins that were excluded and noticed that those followed a very different distribution which resembled a triangular one. Having this ground-truth dataset allowed them to assume with high confidence that in transactions in which the real input could not be deduced by simple exclusion, they could assume it was the ”newest” one, according to their observation with the ground truth transactions. According to their estimates, when using the *newest coin* rule, they could identify the actual input in 80% of the total transactions with an accuracy of 92.33%.

Kumar et al., [54] also presented an empirical analysis of Monero's anonymity. They implemented the vulnerabilities identified by [50] and [51] and achieved almost identical results as [53]. Additionally to [53], they measured the frequency of output values and observed that 93% of those were unique. This showed that even if users wanted to add mixins to their transactions, this would be impossible since there were not other outputs of identical values. Moreover, they noticed that 99.98% of the outputs were not denomination compliant, which showed how infeasible the suggestion by [51] to enforce the output denomination restraint on Monero was.

The authors also evaluated another vulnerability described in [51], namely the co-spending of outputs created in the same transaction. As [51] suggested, this resulted in a strong indication that those transaction inputs would have a much higher probability of not being mixins compared to other inputs. After implementing this vulnerability they discovered that 43% of the transactions in their dataset were affected, i.e., they contained inputs that were outputs in the same transaction. Moreover, they noticed that several *RingCT* transactions had the same problem, which showed that although *RingCTs* solved the problems of mixins' unavailability due to values, it did not solve every problem.

3.3.1.3 Other private cryptocurrencies

Fauzi et al., [55] built Quisquis, a cryptocurrency with built-in privacy guarantees. Their construction is based on updatable keys and efficient zero-knowledge arguments, and its security relies on the decisional Diffie–Hellman (DDH) assumption. In a nutshell, when Alice wishes to pay Bob with Quisquis, she uses as inputs her UTXO, of value x, as well as other users' UTXOs carrying the same value. Then as outputs, Alice adds Bob's public key to facilitate the payment, and for each of the other inputs, she uses an output address produced by the respective input address by updating its corresponding public key.

In terms of the practical anonymity provided in Quisquis, there has been no research investigating it, presumably because the project was never deployed in practice. However, there are some features of potential interest that could be studied. For instance, similar to Monero, Quisquis requires the sender to find other UTXOs carrying exactly the same value, which proved very challenging for Monero before the introduction of *RingCTs*, so it is unclear how this would work with Quisquis.

Moreover, the ability of a user to update other people's public keys introduces the challenge of notifying those users that their addresses have been updated, and perhaps introduces a race condition; it is unclear what would happen if Alice spends her UTXO at the same time that another user, Charlie, uses this UTXO as a decoy.

All the cryptocurrencies we have discussed so far work with the UTXO model, similar to Bitcoin, however private solutions for the account model have been proposed as well. Bunz et al., [56] designed Zether, a smart contract that provides certain additional anonymity compared to the native token of the underlying blockchain, like ETH on the Ethereum blockchain. Zether produces the token ZTH, the holders of which can conceal their balances and also obfuscate the sender and the recipient of a ZTH transaction using cryptographic proofs. Additionally, it can be implemented in all smart contract-based blockchains, including Ethereum, without any change in the undergoing consensus mechanism, hence we consider it as a stand-alone cryptocurrency.

3.3.2 Enhancing privacy with layer 1 solutions

In this section, we look into the technologies that aim to enhance the on-chain privacy of existing cryptocurrencies without the need to execute a hard fork. In contrast to the previous section, in which we described altcoins designed with privacy by default, we now look into solutions that can be used as "plug-in" mechanisms.

Zether [BAZB20] is a system that enables publicly known smart contracts to reason about homomorphic commitments in zero knowledge, and in particular enables these to transact in a manner that hides transaction amounts; it does not hide the identities of parties involved in the transaction, beyond a small anonymity set. Furthermore, the cost of verifying a transaction scales linearly with the size of the anonymity set, whereas in Zexe this cost scales logarithmically with the size of anonymity set.

3.3.2.1 CoinJoin-based technologies

CoinJoin was first described in detail by one of Bitcoin's early developers, Gregory Maxwell, at the BitcoinTalk forum [57]. Maxwell, motivated by the contemporary line of research [2, 5, 6, 3], expressed the importance of enhancing Bitcoin's transaction privacy since, at that moment, the probability that all transaction inputs belonged to the same user was extremely high. To this end, he suggested that multiple participants participate in the same transaction by providing their inputs and outputs, obfuscating the transaction graph and invalidating the co-spend heuristic.

Ruffing et al., built CoinShuffle [58] as an alternative to Maxwell's CoinJoin based on Chaum's credentials [59], that would operate without the need of a third party and achieve better security properties and performance. CoinShuffle runs in three rounds: *announcement*, were the participants of the coinjoin generate an ephemeral encryption and decryption key pair. Next, they publish to the rest of the participants their public encryption key. The second step, *shuffling*, is the round that ensures CoinShuffle's unlinkability. Each participant picks their output Bitcoin address, which for the purposes of the shuffling round acts as their public key. In each round, a participant encrypts their Bitcoin address with all the public keys of the participants ahead of them and appends this ciphertext to a list of ciphertexts they receive from their predecessor. The layered decryption and the fact that each participant shuffles randomly ensures that only the last participant obtains in plaintext the list of all the output addresses, which they can then broadcast to the rest of the network. In the third and last round, each of the participants confirms that their Bitcoin address is included in the list of published outputs. In that case, they can craft a Bitcoin transaction that includes all inputs and outputs, sign it with their private key and publish it to the rest of the participants.

The advantages of CoinShuffle over Maxwell's Coinjoin are the following; CoinShuffle can operate without the necessity of a semi-trusted third party. In Maxwell's approach, the server cannot steal the coins or learn the mapping between inputs and outputs, but it can disrupt the whole procedure by going offline or refusing to cooperate. In CoinShuffle, not only is there no such central entity, but also, thanks to the Dissent protocol [60] on which it is based, dishonesty can be detected by anyone in the third round, since accountable information can be discovered and published to the blockchain. Moreover, Dissent's key advantage over

Tor, which was Maxwell's suggestion for protecting network anonymity, is that it is secure against a global passive adversary who can intercept all communication between participants and network nodes.

The same authors built CoinShuffle++ [61], which was an improvement over the previous design, mainly enhancing the performance and making it a practical extension to Bitcoin. Although CoinShuffle is built on the Dissent anonymous communication network, CoinShuffle++ was built on the novel P2P mixing protocol, DiceMix, that the authors showed was optimal for the purpose of coin mixing in Bitcoin. The protocol extends the original idea of Dining Cryptographers network (DC-net) [62] by adding the accountable information in the case where one participant misbehaves, and the automatic exclusion of such entities. In the original version, misbehaving peers of the DC-net would cause the disruption of a protocol. The authors managed to achieve a significant improvement of a mix round of 50 participants achieved in 8 seconds, compared to the equivalent round in CoinShuffle which took three minutes

CoinJoins are widely used in practice today [63, 64, 65] in UTXO-based currencies, especially Bitcoin, which shows the significance of those works. Wasabi and Joinmarket are the most well-known CoinJoin-based Bitcoin mixing services, and both of those wallets' software are based on Chaumian's CoinJoins. This direction has shown that although decentralisation is a desirable property to enhance privacy, the fact that the implementation of a service with a trusted entity, like Chaumian's coinjoin, is much easier has probably played a crucial role.

3.3.2.2 Tumbler technologies

Bonneau et al., [66] designed MixCoin, one of the earliest proposals of a centralised tumbler that can run on top of Bitcoin or any other similar UTXO-based currency. Their protocol relies on a central entity facilitating atomic swaps of coins with users that wish to obtain fresh, untraceable coins. In the first step, the user decides on certain parameters of the exchange, i.e., the addresses of the user and the mix, the delay between the deposit from the user to the mix and the withdrawal from the mixer to the user, the time that the mix has available in order to facilitate the payment etc.

One important thing for both the mixer and the user is to choose Bitcoin addresses that have not been associated with any past activities to minimise the risk of exposure. The user then encodes all this information in a message, which he sends to the mix before the swap happens, which the mixer signs and gives back to the user. The signed messages acts as a swap's warranty.

Valenta et al., [67] built Blindcoin, an extension to MixCoin that inherited most of its desirable features while hiding the correlation between users' deposits and withdrawals even from the mixer itself. To this end, the authors used the Fuchsbauer [68] blind signature scheme that required only one message from the user and one from the signer (mixer) in order to work. Blindcoin's scheme also required an append-only public log for the signed warranty to be published and the necessary information for the user to prove ownership of a token in a blind manner. Although Blindcoin improved upon MixCoin in terms of the threat model, this came at a cost. In particular, adding the append-only public log increased the protocol's complexity and execution time as it now required two additional transactions, one to publish the blind token and one to redeem it. Moreover, it required a universal public key for the mix that would be consistent over time, hence the anonymity set of each transaction was restricted to the number of transactions with this mix's public key.

Heilman et al., [69] designed TumbleBit, which could facilitate multiple payments between two sets of users, *Payers* and *Payees*, within a predefined epoch period, but also in its simplest form act as a tumbler for single payments. Thanks to the use of standard RSA assumption and ECDSA signatures, TumbleBit improved upon the designs outlined above, as it satisfied all the security properties a tumbler should have while at the same time achieving high performance. The protocol operates in three phases, *Escrow*, *Payment* and *Cash-out*. The core components of TumbleBit, the puzzle-promise protocol and the puzzle-solver protocol, ensure two security properties that some of the mixers we saw in this section did not satisfy. In particular, the blinding operations ensure that even the Tumbler would be unable to identify, in our example that the payment was from Alice to Bob. On the other hand, the atomicity of the fair-exchanges ensured that either both of the payments (from

Alice to the Tumbler and from the Tumbler to Bob) succeeded or none; hence the Tumbler could not cheat and steal any coins. Moreover, due to the pairwise mixing, the system is scalable and does not have the restrictions that Coinjoin-based mixing services have.

3.3.3 Enhancing privacy with layer 2 solutions

In this final section, we look into the Lightning Network [70], a layer-2 solution for Bitcoin based on payment channels (see Chapter 2). Although, there are other layer-2 solutions based on payment channels, like the Raiden Network for ETH [71] or the Bolt network for Zcash [72], Lightning is the one most relevant to this thesis. In particular, in Chapter 6 we present our own empirical analysis of the anonymity offered by the Lightning network.

We consider as related all research that focuses on Lightning, particularly as it relates to privacy. Most of the previous research however has focused on the scalability, utility and crypto-economic aspects of Lightning. Branzei et al., [73] looked into the economic equilibrium that resulted from the introduction of fast payment networks in Bitcoin and, in particular, Lightning. Their main argument was that introducing several fast off-chain payment networks will indeed significantly increase the transaction throughput; however, it will lower the fees of the actual Bitcoin transactions. Although this initially may seem like a welcomed effect, the authors argued that the decreased revenue from fee payments could harm Bitcoin's security as a majority attack would become more feasible.

Khalil et al., [74] looked into the problem of payment channels getting one-sided, i.e., the entire balance getting into the possession of only one participant. To this end, they designed REVIVE, a protocol that allowed mutually distrustful network participants to re-balance their channels without the need of any on-chain transaction, except in the case of a dispute. Although REVIVE was built as a proof of concept on a payment network based on Ethereum, it could be applied to any payment network, including Lightning.

Nida Khan et al., [75] made an extensive analysis of transaction fees on various payment networks, including Lightning, comparing each other but also comparing to equal valued on chain transactions. Their main finding was that Lightning was cheaper compared to Raiden, whereas Stellar was the one offering the fastest transfers.

Stefano Martinazzi [76] and Beres et al., [77] looked into the graph properties of Lightning. They presented some general statistics on the number of nodes, channels and balances highlighting the great adoption that the system had. In [77] the authors implemented several general-purpose graph metrics like average degree, connected components or identification of network's bridges. Their measurements showed that the system had, already, a great degree of centralisation, i.e., a handful of nodes controlled the majority of the channels and acted as payment hubs in the network. This feature had consequences in the robustness of the network, as specific entities could either prevent a large portion of the payments or enforce censorship. In terms of privacy, those nodes could collect invaluable information regarding the payments they were participating in. Moreover, [77] also made an important observation that the graph properties of the network would inevitably lead to short payment paths.

Rohrer et al., [78] also focused on the topology of Lightning but went beyond implementing standard graph metrics and made a similar observation that the network is very centralised. They showed that since the network resembles a small-world network [79], it is not only vulnerable to random failures but is also very susceptible to targeted attacks. In particular, they showed how adversaries can exhaust the victim nodes' balances and therefore isolate them from the rest of the network. This is done by opening strategic channels, and carefully creating payments through specific paths that will cause subsequent channels to be one-directed. Two drawbacks of their attacks are that they require the adversary to directly open channels with the victim nodes and also have high liquidity (at least as much as the victims), which makes these attacks less practical as the attacker could easily be spotted but also has to lock a lot of their own funds. Pérez-Solà et al., [80] designed a variation of the same purposed attack that managed to reduce the cost for the attacker, as it no longer required her to pay the payment fees. Mizrahi et al., [81] also designed their own isolation attack that did not require the attacker to create direct channels to victims or even lock significant amounts of money.

Tochner et al., [82] made an analysis on how the implementation of payment path finding of the main wallet implementations can cause severe availability issues. In particular, their measurements showed that only the top 10 central nodes could disrupt roughly 80% of all paths, just by refusing to forward the payment they were asked to. The fact that the network was very centralised and also the way the clients chose their paths, trying to minimise their cost, led to a handful of nodes being constantly picked. To exhibit the scale of this attack, their measurements showed that a strategic attacker who charged minimum fees can disrupt 65% of the total payments by opening only 5 new channels whereas a more resourceful attacker that opens 30 channels can disrupt up to 80%. Moreover, they noticed that the majority of the clients were using the default settings, in terms of the charging fees, which made such DoS attacks even easier to implement. In order to overcome the path finding vulnerabilities, they made their own suggestions on how the payment routes should be formed, which didn't take only the fees into consideration.

Although all the papers mentioned above made significant contributions to the research around Lightning, by pinpointing its security vulnerabilities, the following papers that focused on privacy are closer to our work. Malavolta et al., [83] were the first to perform a full-scale privacy analysis on payment channel networks and their findings were applicable to Lightning as well. Beyond privacy, they identified potential drawbacks on all existing PCN's in terms of concurrency, since there was no built-in method to tackle it. They designed two protocols, Fulgor and Rayo, that could provably handle concurrency, using two different strategies. In terms of privacy, they were the first to formalise it by defining *balance security*, *value privacy* and *sender/receiver anonymity*, similar to the universal composability (UC) framework [84]. Interestingly, they showed that concurrency and privacy, their two main research topics, exhibit a trade-off as enforcing non-blocking progress and therefore improving concurrency lowered the anonymity for the sender or the receiver of the payment.

They also designed a Multi-Hop Hash Time-Lock Contract (Multi-Hop HTLC), which could be run on the Ethereum smart contract platform and provide better privacy guarantees than the HTLCs implemented at that moment by Lightning. In terms of privacy, they added a feature of blinding the hash produced by the receiver, which could be used up until that moment for participants of the payment to identify the fact that they were part of the same path. Given the earlier research that showed how centralised the network was, it became obvious that two colluding nodes with very high connectivity had a very strong probability of being in the same path and have a chance to identify the sender and/or the receiver of the payment.

Malavolta et al., [85] improved on those HTLCs by designing Anonymous Multi-Hop Locks (AMHLs) that would provide privacy and security under a defined UC framework and improve the efficiency of the previous Multi-Hop HTLCs. They evaluated their design and showed that it required at most 60 milliseconds to be constructed and its size would be a few bytes in the average case and 500 bytes in the worst case. They showed that AMHLs were a practical design, deployable in all the main blockchains. The importance of their work was exhibited by the fact that Lightning incorporated their ECDSA-based AMHLs to their own PCN. Besides being more practical and efficient, their design patched a current vulnerability on HTLCs that allowed intermediate, colluding users to steal other participants' fees.

Nowatowski and Tøn [86] studied various heuristics in order to identify Lightning transactions on the Bitcoin blockchain. Their heuristics only targeted public channels and their main contribution was the observation that at least 75% of all Bitcoin transactions using the P2WSH script are related to Lightning activity, hence many of them can be deduced by observing the blockchain.

Concurrently to our work in Chapter 6, Romiti et al., [87] developed several heuristics to link Bitcoin wallets to Lightning entities. One of their heuristics is similar to the tracing heuristic we develop in Chapter 6, but their goal is to create augmented Bitcoin clustering methods rather than to identify private channels. However, their work, compared to ours, goes beyond analysing the privacy offered in Lightning, since they correlate Lightning entities to Bitcoin ones, hence allocating actual network information to Bitcoin nodes, which can have severe privacy outcomes.

Herrera-Joancomarti et al., [88] were the first to propose a novel attack that could identify the secret balances within payment channels that an adversary is not involved in. They showed how an attacker can establish a payment channel with a potential victim and obtain the hidden balances between the victim and all of her neighbours. In particular, this attack requires the attacker to send 1-hop payments to all of the victim's neighbours. In the case that the payment went forward, that meant that the balance between the victim and her neighbour was enough, hence the adversarial sender could learn that the real balance was higher than the payment itself. In case of a failure, the hidden balance was less than that. The attacker could then strategically perform those payments using carefully picked values and identify the hidden balance with the least possible efforts.

Importantly, this attack costs nothing in terms of the fees for the attacker since she performs payments that the recipient did not produce a valid hash for, as is required as described in Chapter 2. To this end, when a payment reaches its destination it will fail since the recipient did not ask for it and fees will not be charged along the path. The authors evaluated their attack in the Lightning TestNet (for ethical reasons) and showed that it is indeed very possible in practice and, more importantly, it can leverage the centralised nature of the network and be very efficient. In particular, they showed that is possible to retrieve the hidden balances for 50% of the channels just by targeting 18 nodes, for 80% by targeting 78 nodes, and for 90% by connecting to 141 nodes, which was just 8.1% of the total nodes at that moment.

Although this attack was fundamental to the privacy property of balance secrecy, and inspired subsequent works to improve upon it, it did not come without limitations. First of all, the authors assumed it was trivial to connect to nodes, in order to attack them, when multiple works showed that is far from it, since a large percentage of the nodes are unreachable because they are offline or have misconfigured nodes. Moreover, there are some features that all Lightning clients implement, that really limit the feasibility of such attacks, with the most important ones being MAX_PAYMENT_ALLOWED and MAX_FUNDING, which limit the maximum value a node allows to be routed through it, and the maximum value it has for opening a channel. Lastly, but most importantly, this attack relied on the error messages that the nodes broadcast in case of a failed payment which would make it oblivious in a patch of the clients that would no longer create such error messages. We show in our work in Chapter 6 how we bypassed this important last limitation.

Nisslmueller et al., [89] implemented the same balance discovery probing attack but did a more thorough analysis of all the possible error messages a sender node may receive. After identifying the hidden balances, they took a step further and showed that by repeatedly probing particular channels and learning the balances, one can also infer, with very high accuracy, the payments that happened involving those channels, both in terms of value and timing. The trade-off here is that the more frequent the probings are, the more accurate the results will be.

Moreover, they also introduced a second timing attack that took into consideration the block difference between when the node is asked to forward an HTLC and when the block is resolved. By analysing this time difference, a node could guess with a fairly high accuracy the position it belongs to within a path, which is very crucial information. In the case the attacking node is either the first or the last hop, they could infer who is the sender or the receiver, accordingly.

Tikhomirov et al., [90] improved upon the original balance discovery attack from [88], by reducing the excessive discovery attempts and scaling the attack. In particular, prior to the balance discovery payments, the attacker's first goal is to identify which nodes and which channels are available for probing and which aren't. This is done by either directly connecting to the nodes and checking their reachability or sending tiny payments of insignificant value and checking whether they succeed. This optimisation is very important since, as we also show in Chapter 6, a large portion of the network is unavailable and checking in advance the attack scope significantly increases the performance of the attack itself. Moreover, they showed how balances of channels can be inferred without even connecting to any of the participant nodes. Instead, they incrementally built paths of arbitrary length (up to 20 hops) and can discover balances one hop at a time. Lastly, they proposed countermeasures which, interestingly, although require revealing some information regarding balances, increase the privacy of the information hidden and simultaneously make payments easier and less prone to random errors.

Perhaps an even more important privacy property of Lightning is the obfuscation of the sender and/or the receiver of a payment from any node in the network, even from those involved in the payment route. [77] looked briefly at the question of finding the sender and recipient of a payment, from a point of view of a passive adversary involved as an intermediate node. Similar to our work in Chapter 6, they developed an LN traffic simulator based on publicly available network snapshots and information published by certain node owners. They used their simulator in order to infer important information regarding the profits of nodes forwarding payments and their main conclusion was that for the majority of the network, the participation is irrational as their profits from routing are negligible. More relevant to our work, they used their simulator in order to argue that in 16-34% of the transactions, according to their simulator parameters, the payment paths consist of only 1 hop. This has immediate effects on privacy since it is equivalent to saying that an intermediate node has a 16-34% probability of knowing who the sender and the receiver of the payment are (the predecessors and successor nodes, accordingly).

We followed a similar approach in our work in Chapter 6, where we designed our own simulator in order to argue about payment privacy, however our simulators differ in many perspectives. In general, while Beres et al., [77] tried to build their simulator with the logic of trying to find the most realistic parameters, we claim that those parameters are highly debatable and rely on secret information that is impossible to be known. Instead, we followed a best case and worse case scenario based on the few available public sources, such as the published logs by LNBig. Moreover, in their simulations they assume for simplicity that each payment is carrying the same value, which is unrealistic. Instead, we again followed the best case and worse case approach and defined the limits of the payments within a range of possible probabilities. Furthermore, their work considered only single-hop payments while we take a more general approach and calculate the probabilities of intermediate nodes guessing the endpoints, even for longer paths. Lastly, we use our simulator to look into two privacy properties, whereas they looked only into one.

There are a number of other Lightning Network studies that use network simulators [73, 91, 92, 93]. Several of these simulators were used to perform economic analysis of the Lightning network [73, 92, 93], while the CLoTH simulator [91] provides performance statistics (e.g., time to complete a payment, probability of payment failure, etc.). However, all of those simulators make several simplifying assumptions about the topology, path selection algorithm, and distribution of payments. As such, they are not suitable for an analysis of LN's privacy properties.

Chapter 4

Expanding Bitcoin clustering and tracking

In this chapter, we introduce our work *"How to Peel a Million: Validating and Expanding Bitcoin Clusters"*, which takes a deep look into the Privacy offered by the layer 1 of Bitcoin. We design a new heuristic that is designed to track a certain type of flow, called a *peel chain*, that represents many transactions performed by the same entity; in doing this, we implicitly cluster these transactions and their associated pseudonyms together. We then use this heuristic to both validate the results of existing clustering heuristics and to expand them. We also develop a machine learning-based validation method and, using a ground-truth dataset, evaluate all our approaches and compare them with the state of the art. Ultimately, our goal is to not only enable more powerful tracking techniques but also call attention to the limits of anonymity in these systems.

4.1 Introduction

Since its introduction in 2008, Bitcoin has used a pseudonymous system for transferring coins, with entities forming transactions in which they and their recipient(s) are identified using just a set of *pseudonyms* or *addresses* that have no inherent link to their identity. It has been demonstrated by now, however, that this use of pseudonyms does not make Bitcoin anonymous. This has in large part been driven by the development of various *clustering heuristics* that identify multiple pseudonyms operated by the same entity [7, 8, 5, 9, 94, 6], with research also showing that de-anonymization is possible at the network layer [27, 28]. These clustering heuristics use patterns of usage present in the Bitcoin blockchain as evidence of the shared ownership of the pseudonyms they cluster together; one heuristic that has been particularly widely adopted is the so-called *co-spend* heuristic, which says that all addresses used as input to the same transaction belong to the same entity. This heuristic has been so effective that companies such as Chainalysis now use it — and heuristics derived from it — to provide Bitcoin tracking as a service to both law enforcement agencies and financial institutions, such as cryptocurrency exchanges, looking to comply with anti-money laundering (AML) regulations. These heuristics can be used not only to cluster together pseudonyms operated by the same entity but, as a consequence, to track flows of bitcoins as they are transferred from one entity to another.

This ability to track flows of bitcoins has been used in several high-profile investigations, such as the indictment of Ross Ulbricht as the operator of the Silk Road marketplace [95]; the blocked movement of funds paid to the WannaCry ransomware operators [96]; the takedown of one of the largest websites hosting child sexual abuse material [97]; and the takedown of several terrorist financing campaigns [98]. More recently, Roman Sterlingov was arrested based on allegations that he served as the operator of the Bitcoin Fog mixing service for ten years [99]. Despite the arrest taking place in April 2021, the allegations were supported by evidence taken from the Bitcoin blockchain as early as 2011 [100]. This investigation thus makes clear just what is possible when all transactions are stored in a globally visible and immutable ledger.

In this paper, we extend known heuristics for tracking flows of bitcoins by formalizing in Section 4.3 the notion of a *peel chain*, which is a set of linked transactions that are all initiated by the same entity, and presenting heuristics for identifying peel chains and following them forwards and backwards. As compared with previous heuristics, ours are based not on properties of individual transactions or addresses within transactions, but rather on general features associated with a cluster formed by the co-spend heuristic. In particular, we describe in Section 4.2 how we assign features to a cluster based on the transactions and addresses it contains.

We also describe in Section 4.2 our main dataset, which consists of 120 clusters formed by the co-spend heuristic. Using data provided to us by Chainalysis, we know that 60 of these clusters are *true positives*, meaning all addresses they contain really do belong to the same entity, and that 60 of them are *false positives*, meaning they contain one or more *Coinjoin* transactions and thus all addresses do not actually belong to the same entity. This access to ground-truth data provides us with the rare ability to evaluate the accuracy of our heuristics, as well as ones that were previously proposed.

In particular, we argue how our basic heuristic for identifying and following peel chains can be used to both *validate* and *expand* traditional clustering heuristics such as the co-spend heuristic. In this first usage, presented in Section 4.4.2, we argue that we can use the ability to identify peel chains to increase our confidence in the results of the co-spend heuristic. We also present in Section 4.4.1 a machine learning approach for validating a cluster as a whole; i.e., for classifying it as either a true positive or a false positive. Using our ground-truth dataset, we are able to evaluate this classifier and show that it achieves an accuracy of 89%.

In the second usage as an *expansion* heuristic, we demonstrate how the ability to identify peel chains can be used to expand the results of the co-spend heuristic. In particular, following a peel chain forwards means identifying the *change output* in each transaction in the chain, so our algorithm includes an implicit change heuristic. As compared with previous change heuristics [5, 9, 94, 6], we demonstrate using our ground-truth dataset that our change heuristic achieves a false discovery rate of only 0.02%; the next-best heuristic achieves a false discovery rate of 12.7%. We then apply this expansion heuristic to track the funds withdrawn from an exchange account associated with Roman Sterlingov to their deposit into the Bitcoin Fog mixing service, showing that our heuristic is able to link these two transactions whereas all previous heuristics would be unable to do so.

To summarize, we make the following contributions:

- We provide a heuristic, based on a robust set of features, for both identifying peel chains and following them forwards and backwards.
- We present a machine learning-based classifier and a validation heuristic, both of which can be used to influence the confidence we can have in the results of the co-spend heuristic.
- We present a heuristic for expanding co-spend clusters, and evaluate it using a custom-built ground-truth dataset. Our comparison with previous heuristics shows that ours is significantly more effective and significantly safer.

The techniques we develop are directly applicable in cryptocurrency investigations, and thus have the potential to be adopted and used in them. Above all, however, we hope that our work helps to correct the misperception of Bitcoin [101, 102, 103] that persists even in June 2021 as "anonymous and almost untraceable" [104] and a way to allow "people [to] receive digital payments without revealing their identity" [105].

4.2 Dataset and Methodology

To start, we were given 241 Bitcoin addresses and 20,016 Bitcoin transactions by Chainalysis, a company that provides blockchain data and analysis to businesses and government agencies.[1] The addresses represented *true positive* clusters, in the sense that Chainalysis had manually verified that all the addresses in the same co-spend cluster as this address really did belong to the same service (typically by confirming directly with the service). The transactions were all Coinjoins and thus represented *false positive* clusters, meaning all of the addresses in the resulting co-spend cluster would not actually belong to the same service. This ground-truth dataset was necessary for evaluating our heuristics, and would not have been possible to get at this scale without working with Chainalysis or directly with the services themselves. None of the clusters represented individual users, and we had no additional information about the entities represented by the clusters (e.g., the name of the service).

[1] https://www.chainalysis.com/

From this initial dataset, we created clusters using the co-spend heuristic and represented each cluster C as a tuple $(\mathsf{C}_{\mathsf{addr}}, \mathsf{C}_{\mathsf{tx}})$ where $\mathsf{C}_{\mathsf{addr}}$ is the set of all addresses in the cluster and C_{tx} is the set of all transactions initiated by one or more addresses in $\mathsf{C}_{\mathsf{addr}}$. This resulted in 241 true positive (TP) clusters and 16,974 false positive (FP) clusters. We describe in Section 4.2.2 how from this initial dataset we created a more balanced dataset of 60 true positive (TP) and 60 false positive (FP) clusters that we then used in the remainder of our analysis. In order to do so, we first describe the features we defined for transactions, addresses, and the overall cluster.

4.2.1 Features

We consider features of three different types of objects within Bitcoin: transactions, addresses, and clusters. These features are largely defined by the wallet software used by a given entity and the decisions they make in scripting their transactions, and as we will see the set of possible features is largely stable within even large clusters. This consistency is crucial in the algorithms we develop for following *peel chains* in Section 4.3. Our set of chosen features is based on Bitcoin usage today, but we stress that new features can be incorporated as Bitcoin evolves without changing our overall approach.

4.2.1.1 Transaction features

Every Bitcoin transaction has a different set of features, according to both the action it is performing and the wallet program and version used to generate it. We consider the following four features.

Replace-by-fee/sequence number. At any given point in time, there can be multiple versions of the same transaction in the Bitcoin network; for example, if a user broadcasts a transaction to the network but it never gets included in a block, they may broadcast a new version with an increased fee in the hopes of increasing its chances. The *sequence number* helps identify different versions of a transaction, with a higher sequence number indicating that the transaction is more recent. If a user does not want transactions to be able to be replaced they can thus set the sequence number to be the maximum value (`0xffffffff`). The sequence number is set for each transaction input, and the transaction is considered to be *replaceable* if any of its inputs have a sequence number less than this maximum value [106]. We thus set this feature to be true for a transaction if it is replaceable and false if it is not.

Locktime. A transaction can set a *locktime* (or time lock) to indicate that it cannot be spent before a block at some height has been mined. We set this feature to be true if a locktime has been set and false if not.

Version. The *version* of a transaction, which is either 1 or 2, determines the rules used to validate the transaction [107].

SegWit. *SegWit* (Segregated Witness) [108] allows a transaction to be separated into its semantic data (i.e., information about who is sending and receiving bitcoins) and its signature data. A transaction can indicate if it uses SegWit by setting its fifth byte to `0x00`. We set this feature to be true if SegWit is enabled and false if not.

We thus represent the features of a transaction tx as a 4-tuple containing binary values (1/2 for the version and true/false for the rest) in each entry. We denote by $\mathsf{features}_{\mathsf{tx}}$ the function used to extract these features from a transaction.

4.2.1.2 Address features

The BlockSci tool [109] categorizes Bitcoin scripts into ten generic types: pubkey, pubkey hash, witness pubkey hash, multisig, multisig pubkey, script hash, witness script hash, witness unknown, non-standard, and nulldata (with the witness prefix indicating that it uses SegWit). Some of these categories can be further broken down according to whether the address is *compressed* or *uncompressed*. To briefly explain some of the more common types, the pubkey format allows users to send coins to a public key. Both pubkey hash and witness pubkey hash allow users to instead send coins to the hash of a public key. The script hash and witness script hash formats allow users to send coins to the hash of an arbitrary script. These coins can then be spent only by the owner(s) of the underlying script.

Address type	TP (%)	FP (%)
pubkey hash (compressed)	41.15	1.19
pubkey hash (uncompressed)	0.0	0.010
witness pubkey hash (compressed)	5.76	37.44
witness pubkey hash (uncompressed)	37.04	61.36
multisig (2/2)	5.6	0.0
multisig (2/3)	2.8	0.0
multisig (3/4)	0.1	0.0
multisig (2/6)	0.24	0.0
SegWit multisig (2/2)	2.22	0.0
SegWit multisig (2/3)	5.12	0.0

Table 4.1: Types of addresses found across all clusters.

A common script in Bitcoin is an m-of-n multisig. Using a multisig address, a user or set of users can require that at least m of the n available keys specified in the script must sign a transaction in order to spend the coins from that address.

Across our set of 246,600 addresses, we identified 10 distinct combinations of these categories that were used, as summarized in Table 4.1. We thus represent the features of an address as its address type, which takes one of these ten values. We denote by $\mathsf{features}_{\mathsf{addr}}$ the function used to extract the feature from an address.

4.2.1.3 Cluster features

Each cluster contains a set of addresses and a set of transactions. From these sets C_{tx} and $\mathsf{C}_{\mathsf{addr}}$, we can extract the relevant features (which we did using BlockSci) to build the sets TF_{C} and AF_{C} of all transaction and address features present in the cluster.

In addition to these sets of features, we define for each cluster a *change strategy*, which we denote by $\mathsf{change}_{\mathsf{C}}$. This cluster-level feature considers the pattern, if any, the transactions in this cluster exhibit when forming change outputs. We identify a change output in a transaction in C_{tx} if there is exactly one output whose address belongs to the cluster (i.e., is in $\mathsf{C}_{\mathsf{addr}}$). If there are zero or multiple such addresses then we ignore this transaction for the purposes of setting the change strategy.

Intuitively, some wallet software may send the change in a transaction to a specific output index by default; e.g., the first or last output. As many entities are likely to use scripts or other automated methods to form transactions, it may be the case that their transactions thus have patterns in terms of the index of the change output. To this end, we define four different values for the cluster's $\mathsf{change}_{\mathsf{C}}$:

$\mathsf{change}_{\mathsf{C}} = -1$. For every transaction in C_{tx} with a single identified change output, it was always at the last index.

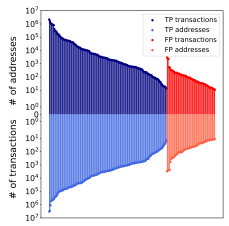
$\mathsf{change}_{\mathsf{C}} = 0$. For every transaction in C_{tx} with a single identified change output, it was always at the first index.

$\mathsf{change}_{\mathsf{C}} = 1$. For every transaction in C_{tx} with a single identified change output, it was always at either the first or the last index.

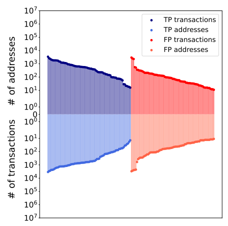
$\mathsf{change}_{\mathsf{C}} = \mathrm{None}$. There was at least one transaction in C_{tx} that did not follow the patterns above; i.e., with a single identified change output that was at neither the first nor the last index.

4.2.2 Creating a cluster dataset

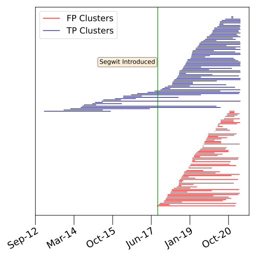
In building a dataset of clusters, our goal was to have it be as balanced as possible, in terms of the features introduced in the previous section. This was particularly important in our validation of the co-spend heuristic in Section 4.4, in which we differentiate between TP and FP clusters based on their features. Concretely, we focused on creating a balance between true and false positives for the following three parameters: (1) the number of clusters in each category, (2) the sizes of both $\mathsf{C}_{\mathsf{addr}}$ and C_{tx}, and (3) the period of time in which the cluster was active. We refer to the last property as the cluster's *lifespan*. The first two properties are generally important in creating a balanced dataset, and this last property is also essential as behavior in Bitcoin transactions has changed significantly over time. Ensuring that the lifespans of TP and FP clusters had a significant overlap was thus the only way to ensure a fair comparison; e.g., making it so we could not trivially distinguish because all TP clusters had one transaction feature set to true and all FP clusters had it set to false.



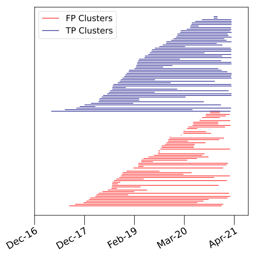
(a)



(b)



(c)



(d)

Figure 4.1: Balancing the sizes (Figures 4.1a and 4.1b, with the y-axis on a log scale) and lifespans (Figures 4.1c and 4.1d) for our true positive (TP) and false positive (FP) clusters. Figures 4.1a and 4.1c represent the cluster sizes and lifespans before balancing, and Figures 4.1c and 4.1d represent these values after balancing. In Figure 4.1c, the vertical line represents the date on which SegWit was introduced.

To start, we set a minimum threshold of 10 for both C_{tx} and $\mathsf{C}_{\mathsf{addr}}$ and discarded all clusters that were smaller. This left us with 183 true positive clusters (out of 241) but only 75 false positive clusters (out of 16,974). This happened because the vast majority of the inputs to the Coinjoin transactions were one-time addresses, meaning the resulting cluster was such that $|\mathsf{C}_{\mathsf{tx}}| = 1$. After obtaining these 183 TP and 75 FP clusters, we further observed that they were highly imbalanced in the parameters we considered, as shown in Figure 4.1a.

This figure shows that not only are there more TP clusters, but also they are much larger on average than the FP clusters. For example, in our initial dataset, TP clusters had up to 3.2M transactions (with an average of 56K) whereas FP clusters had only up to 283K (with an average of 19K). Since the TP clusters were on average so much bigger than the FP ones, we decided to remove the 108 biggest TP clusters from our analysis (in terms of $|\mathsf{C}_{\mathsf{addr}}| + |\mathsf{C}_{\mathsf{tx}}|$). This created a dataset of 75 TP and 75 FP clusters, each of comparable size (in terms of both $\mathsf{C}_{\mathsf{addr}}$ and C_{tx}), as we see in Figure 4.1b.

This approach created a balance in terms of the first property, but did not address the issue of having overlapping lifespans: as we see in Figure 4.1c, there are several TP clusters whose lifespan ended before any FP clusters even began. We can also see this has a direct impact on our transaction features, as several of our TP clusters existed before SegWit was introduced but none of our FP clusters did. To address this imbalance, we removed the TP clusters whose lifespan didn't overlap with the lifespan of any FP clusters. We ended up with 60 TP and 60 FP clusters that were balanced in all three properties, as shown in Figure 4.1b and Figure 4.1d. In the end, all clusters had between 15 and 3415 addresses (with an average of 258.6 for FP clusters and 642.6 for TP ones), between 11 and 3448 transactions (with an average of 307.7 for FP clusters and 692.4 for TP ones), and operated at some point between April 2017 and April 2021.

Feature statistics. In terms of transaction features, we found each of the possible 16 4-tuples in at least one of our clusters. Most clusters (76 out of 120) used only a single combination of transaction features, however, and all clusters used six or fewer. The average number of features was 1.55 for FP clusters and 1.67 for TP clusters. This suggests that clusters are largely consistent in their transaction behavior. We found a similar level of consistency when looking at address features: 56.7% of TP clusters and 53.3% of FP clusters used only one address type, and all clusters used three or fewer.

We found that 18 of our TP clusters had a completely consistent change strategy ($\mathsf{change}_\mathsf{C} = 0$ or -1) and 30 had $\mathsf{change}_\mathsf{C} = 1$; this left 12 with no change strategy. As might be expected, FP clusters were less consistent (given that they actually contained different sets of users): 34 had no identifiable change strategy, 8 had a completely consistent change strategy, and 18 had $\mathsf{change}_\mathsf{C} = 1$.

To understand the overlap in features across different clusters, we looked at the Jaccard similarity between the sets of 5-tuples representing their combined transaction and address features. We found that the average Jaccard similarity across all pairs of clusters was 0.13, which suggests that they are relatively dissimilar in these features. There were several notable exceptions, however, and in particular cases where the features were not only overlapping but in fact identical. For example, there were 26 clusters whose transactions all had the same combination of features (Version 1 and SegWit-enabled transactions that had no locktime and were not replaceable) and whose addresses were all of the same type (uncompressed witness pubkey hash). This is unsurprising as it represents the default setting of the standard Bitcoin wallet software.

4.3 Following Peel Chains

In this section, we define the concept of a peel chain and present our heuristics for identifying them, according to the features defined in the previous section.

4.3.1 Defining a peel chain

The concept of a peel chain was first introduced by Meiklejohn et al. [9] as a series of transactions originating from a transaction with a relatively large UTXO as input (i.e., a UTXO with a large associated value). When this UTXO is spent it creates two outputs: a small one representing a payment to an external entity and a larger one representing the change. This pattern can then be repeated many times, with each hop in the chain slowly "peeling" smaller values from the original UTXO, until the remaining change amount is small (at which point it can be combined with other small UTXOs to create a large one and start the process over again). In this work we consider more general peel chains in which transactions might have more than one input and more than two outputs. In fact, our only requirement is that each adjacent hop in the peel chain is connected by a change output.

Peel chains are in some sense fundamental to UTXO-based cryptocurrencies, which do not allow the partial spending of transaction outputs. Given that it is highly unlikely for an entity to have the exact amount they want to pay someone associated with a UTXO, their payment inevitably forms a change output that can in turn be used as input to a subsequent payment. A second reason that peel chains are very common in Bitcoin is more specific to bigger services, as we explore in Section 4.4.2. In particular, it would be time-consuming, error-prone, and inefficient for big services to craft thousands of transactions manually. For these reasons, they naturally perform transactions using scripts. This automated behavior not only creates long peel chains but also creates patterns that make these peel chains easier to identify and follow.

4.3.2 Identifying inputs and outputs

In order to follow peel chains, the first step is to link together transactions according to their change outputs. Concretely, this means introducing two algorithms: $\mathsf{findNext}$, which aims to identify the unique change output in a transaction, and $\mathsf{findPrev}$, which aims to identify the input(s) in a transaction that originate from transactions conducted by the same entity (as opposed to transactions in which that entity received coins from another one). In its goal, $\mathsf{findNext}$ is comparable to previous work that developed change identification heuristics [5, 9, 94, 6]. As we describe in more detail in Section 4.5.2, however, these previous works identify change outputs based on the freshness of output addresses and the output values. In contrast, $\mathsf{findNext}$ focuses on the index of the output (according to $\mathsf{change}_\mathsf{C}$), the cluster's features (according to TF_C and AF_C), and the next hops of every output that has been spent.

We formally specify $\mathsf{findNext}$ and $\mathsf{findPrev}$ in Algorithms 1 and 2. Intuitively, both start with a transaction $\mathsf{tx} \in \mathsf{C}_\mathsf{tx}$, and aim to output either the next hop in this transaction's peel chain ($\mathsf{findNext}$) or the previous hops ($\mathsf{findPrev}$), according to the features exhibited by the cluster.

$\mathsf{findNext}$. For $\mathsf{findNext}$, we first identify the set of outputs that might represent the change output, according to the cluster's change strategy $\mathsf{change}_\mathsf{C}$ (lines 1–6 in Algorithm 1). If the change strategy is associated with a single output index *i* (either 0 or -1) then we include only that output in this candidate set, while if it is 1 we include both the first and last outputs and if it is None we include all outputs. Next, for each candidate change output output we check to see if:

1. The output is spent, i.e., $\mathsf{output.next} \neq \perp$.

2. The address has a type that exists within the cluster, i.e., $\mathsf{features}_\mathsf{addr}(\mathsf{output.addr}) \in \mathsf{AF}_\mathsf{C}$.

3. The next hop of the output has features that exist within the cluster, i.e., $\mathsf{features}_\mathsf{tx}(\mathsf{output.next}) \in \mathsf{TF}_\mathsf{C}$.

If each of these checks pass, we add the transaction in which this output is

Algorithm 1: findNext

```
Result: nextTx
Input: tx, $\mathsf{change_C}$, $\mathsf{TF_C}$, $\mathsf{AF_C}$
1  if $\mathsf{change_C} \in \{0, -1\}$ then
2      candidates $\leftarrow$ {tx.outputs[$\mathsf{change_C}$]}
3  else if $\mathsf{change_C} = 1$ then
4      candidates $\leftarrow$ {tx.outputs[0], tx.outputs[$-1$]}
5  else
6      candidates $\leftarrow$ tx.outputs
7  nextTx $\leftarrow \emptyset$
8  for output $\in$ candidates do
9      $b_{\mathsf{next}} \leftarrow$ (output.next $\neq \perp$)
10     $b_{\mathsf{addr}} \leftarrow$ ($\mathsf{features_{addr}}$(output.addr) $\in \mathsf{AF_C}$)
11     $b_{\mathsf{tx}} \leftarrow$ ($\mathsf{features_{tx}}$(output.next) $\in \mathsf{TF_C}$)
12     if $b_{\mathsf{addr}} \wedge b_{\mathsf{next}} \wedge b_{\mathsf{tx}}$ then
13         nextTx $\leftarrow$ nextTx $\cup$ {output.next}
14 if |nextTx| $= 1$ then
15     return nextTx[0]
16 else
17     return $\perp$
```

spent to a set representing the possible next hops in the peel chain (line 13). At the end, if there is only one candidate transaction then we output it as the next hop. If there are zero or multiple choices then we output $\perp$ to indicate that we are unsure of the change output and thus the next hop.

We experimentally evaluate the accuracy of findNext in Section 4.5.2, where we see it produces a very low number of false positives. To see why, we consider that findNext incorrectly identifies the next hop in a peel chain only if two things happen simultaneously: (1) the transaction either doesn't produce a change output or the cluster deviates from its known address and transaction features only in spending the change output in tx, and (2) exactly one output that meaningfully receives coins in tx has the same address features as C, and produces the same transaction features when it spends the received coins. For the first point, as we discussed above it is unlikely for a transaction to have no change output given that one bitcoin is highly divisible (to the eighth decimal place) and an entity would have to have the exact amount (plus fees) that they wanted to pay someone associated with a UTXO.

As we saw in Section 4.2.2, clusters are highly consistent in both their address and transaction features, which also makes deviations in their behavior unlikely. For the second point, we also saw in Section 4.2.2 that clusters are largely non-overlapping in their behavior (with some exceptions).

In terms of false negatives, $\mathsf{findNext}$ fails to identify the change output if either (1) it finds no suitable candidate or (2) it finds more than one candidate. In the first case, the cluster would need to either use a different change strategy $\mathsf{change}_\mathsf{C}$ (putting the change output at a different index from expected) or use a different set of features in both the address and the next transaction. In the second case, there needs to be at least one receiving output that behaves in the same way as C, in terms of having the same address and transaction features. It also needs to be the case that C has $\mathsf{change}_\mathsf{C} = 1$ or $\mathsf{change}_\mathsf{C} = \text{None}$, because in the case where $\mathsf{change}_\mathsf{C} = 0$ or $\mathsf{change}_\mathsf{C} = -1$, there is no chance of $\mathsf{findNext}$ finding multiple candidates since only one will be investigated. As with false positives, the consistent and distinct qualities of cluster features thus suggest that false negatives are relatively unlikely to occur as well.

$\mathsf{findPrev}$. Our second algorithm, $\mathsf{findPrev}$, looks at the inputs to a transaction rather than at its outputs. In particular, while the co-spend heuristic tells us that each input belongs to the same entity, it may be the case that some of these inputs represent coins received from other entities. Our goal is to be able to follow peel chains (which are created by a single entity) backwards, which means $\mathsf{findPrev}$ must thus isolate the previous transactions in which these inputs were used as change outputs. This means that we first map each input to a transaction tx to the transaction in which it was created ($\mathsf{input.prev}$) and to its index in the output list of that transaction ($\mathsf{input.prevIdx}$). We next filter out all previous transactions that do not match the transaction features of the cluster (line 3 in Algorithm 2), and then within this filtered set keep track of all transactions ($\mathsf{candidates}$), in addition to all transactions in which one of the inputs was created at either the first or last index ($\mathsf{candidates}_0$ and $\mathsf{candidates}_{-1}$ respectively). Then, as with $\mathsf{findNext}$, we consider the change strategy defined by the cluster and use it to decide which of these candidate sets to

Algorithm 2: findPrev

```
Result: prevTxs
Input: tx, $\mathsf{change}_C$, $\mathsf{TF}_C$, $\mathsf{AF}_C$
1  $\mathsf{candidates}_0, \mathsf{candidates}_{-1}, \mathsf{candidates} \leftarrow \emptyset$
2  for input $\in$ tx.inputs do
3      if $\mathsf{features}_{tx}(\mathsf{input.prev}) \in \mathsf{TF}_C$ then
4          $i \leftarrow$ input.prevIdx
5          if $i \in \{0, -1\}$ then
6              $\mathsf{candidates}_i \leftarrow \mathsf{candidates}_i \cup \{\mathsf{input.prev}\}$
7          $\mathsf{candidates} \leftarrow \mathsf{candidates} \cup \{\mathsf{input.prev}\}$
8  if $\mathsf{change}_C \in \{0, -1\}$ then
9      return $\mathsf{candidates}_{\mathsf{change}_C}$
10 else if $\mathsf{change}_C = 1$ then
11     return $\mathsf{candidates}_0 \cup \mathsf{candidates}_{-1}$
12 else
13     return candidates
```

return (lines 8–13).

The potential for false positives in findPrev is significantly higher than for findNext, as the algorithm returns multiple transactions rather than a single one. Thus, false positives can occur if any of the entities sending coins in a previous hop exhibits the same transaction features and follows the same change strategy. In other words, we rely more heavily on cluster features being distinct (as compared to findNext where we also could count on their consistency), which as we saw in Section 4.2.2 is not always the case. We discuss this further in Section 4.5.

In terms of false negatives, findPrev fails to identify a input.prev originating from the same cluster only if that input.prev deviates in its transaction features or follows a different change_C. Here again we can rely on the consistency of cluster transaction features to argue that this is relatively unlikely to happen.

4.3.3 Following transactions

With findNext and findPrev in place, we can define algorithms for following peel chains forwards (followFwd) and backwards (followBkwd). The ability to follow a transaction both forwards and backwards allows us to capture the full peel chain, regardless of the position of our starting transaction. These algorithms are defined in Algorithms 3 and 4.

Algorithm 3: followFwd

```
    Result: $\mathsf{fwdTxs}_{\mathsf{tx,heur}}$
    Input: $\mathsf{tx}$, $\mathsf{heur}$, $\mathsf{C}_{\mathsf{tx}}$, $\mathsf{change}_{\mathsf{C}}$, $\mathsf{TF}_{\mathsf{C}}$, $\mathsf{AF}_{\mathsf{C}}$
1   $\mathsf{fwdTxs}_{\mathsf{tx,heur}} \leftarrow \emptyset$
2   $\mathsf{tx}_{\mathsf{cur}} \leftarrow \mathsf{tx}$
3   while $\mathsf{tx}_{\mathsf{cur}} \neq \bot$ do
4       if $\mathsf{heur} = \mathsf{validation} \wedge \mathsf{tx}_{\mathsf{cur}} \notin \mathsf{C}_{\mathsf{tx}}$ then
5           break
6       $\mathsf{fwdTxs}_{\mathsf{tx,heur}} \leftarrow \mathsf{fwdTxs}_{\mathsf{tx,heur}} \cup \{\mathsf{tx}_{\mathsf{cur}}\}$
7       $\mathsf{tx}_{\mathsf{cur}} \leftarrow \mathsf{findNext}(\mathsf{tx}_{\mathsf{cur}}, \mathsf{change}_{\mathsf{C}}, \mathsf{TF}_{\mathsf{C}}, \mathsf{AF}_{\mathsf{C}})$
8   return $\mathsf{fwdTxs}_{\mathsf{tx,heur}}$
```

followFwd**.** To follow a transaction tx forwards, followFwd continues going to the next hop in the peel chain, as identified by findNext, until the peel chain ends or findNext otherwise cannot identify a next hop. Along the way it adds the hops to a set of transactions $\mathsf{fwdTxs}_{\mathsf{tx}}$, which it outputs at the end. Line 4 of this algorithm includes a check that is specific to our validation heuristic; we describe this modification in Section 4.4.2 when we present that heuristic.

followBkwd**.** Following transactions backwards is more involved than following them forwards, as findPrev outputs a set of transactions rather than a single one. We can think of followBkwd as performing a breadth-first search: it defines a set of transactions to follow, which is initially set to be just the starting transaction (line 2 of Algorithm 4). As long as there are transactions left to follow, it picks the first of these, adds it to the set, and looks at its previous hops according to findPrev (line 6). It then adds these previous hops to the set of transactions (line 9) and continues. Again, this algorithm contains an additional check in the case of the validation heuristic (line 7), which we describe in Section 4.4.2.

4.4 Cluster Validation

Currently, the clusters output by the co-spend heuristic are largely treated as ground truth, despite the fact that there exist techniques such as Coinjoin that invalidate them. In this section, we thus investigate ways to improve one's confidence in the results of this heuristic. In particular, we explore two approaches, each of which is applicable in a different scenario.

Algorithm 4: followBkwd

```
   Result: $\mathsf{bkwdTxs}_{\mathsf{tx,heur}}$
   Input: tx, heur, $\mathsf{C}_{\mathsf{tx}}$, $\mathsf{change}_{\mathsf{C}}$, $\mathsf{TF}_{\mathsf{C}}$, $\mathsf{AF}_{\mathsf{C}}$
1  $\mathsf{bkwdTxs}_{\mathsf{tx,heur}} \leftarrow \emptyset$
2  $\mathsf{bkwdScope} \leftarrow \{\mathsf{tx}\}$
3  while $|\mathsf{bkwdScope}| > 0$ do
4      $\mathsf{tx}_{\mathsf{cur}} \leftarrow \mathsf{bkwdScope}[0]$
5      $\mathsf{bkwdTxs}_{\mathsf{tx,heur}} \leftarrow \mathsf{bkwdTxs}_{\mathsf{tx,heur}} \cup \{\mathsf{tx}_{\mathsf{cur}}\}$
6      $\mathsf{prevTxs} \leftarrow \mathsf{findPrev}(\mathsf{tx}_{\mathsf{cur}}, \mathsf{change}_{\mathsf{C}}, \mathsf{TF}_{\mathsf{C}}, \mathsf{AF}_{\mathsf{C}})$
7      if $\mathsf{heur} = \mathsf{validation}$ then
8          $\mathsf{prevTxs} \leftarrow \mathsf{prevTxs} \cap \mathsf{C}_{\mathsf{tx}}$
9      $\mathsf{bkwdScope} \leftarrow \mathsf{bkwdScope} \cup \{\mathsf{prevTxs}\}$
10 return $\mathsf{bkwdTxs}_{\mathsf{tx,heur}}$
```

Our first approach, described in Section 4.4.1, is a classifier for co-spend clusters that attempts to distinguish between TP and FP clusters. This type of classifier can implicitly be realized by a Coinjoin detection mechanism, such as the one implemented in BlockSci, and indeed when we implement this approach we find it achieves 87.5% accuracy. Our classifier, which is based on Random Forest, achieves 89.2% accuracy. It achieves, however, a much lower false negative rate (10% as compared to 20%), which in turn lowers the risk of an investigator or researcher making an incorrect assumption about the results of the co-spend heuristic. Furthermore, our classifier is more robust as it depends on the behavior of entities in general rather than just the characteristics of a single Coinjoin transaction (which can be changed by a Coinjoin service such as JoinMarket to avoid detection).

Our second approach, described in Section 4.4.2, links together transactions within the same co-spend cluster that our heuristics from Section 4.3 identify as belonging to the same peel chain. In doing so, we increase our confidence that these transactions were indeed performed by the same entity. Moreover, if we run it for every transaction in the cluster then we see that TP and FP clusters have different behaviors in terms of how many distinct peel chains they contain.

4.4.1 Cluster classification

4.4.1.1 Methodology

Based on the transaction characteristics defined in Section 4.2.1.1, we computed aggregated cluster-level features. For the SegWit and locktime features, we calculated the fraction of transactions in the cluster that had this value set to true (`prop_segwit_enabled` and `prop_locktime_enabled` respectively). For the version feature, we calculated the proportion of version 1 transactions (`prop_v1`). We did not compute a feature column for version 2 transactions to avoid multicollinearity issues, since `prop_v2` is given by $1 - \texttt{prop\_v1}$. Within each cluster we also determined the proportion of all available input address types (as defined in Section 4.2.1.2). These values were aggregated to a single feature using the maximal value (`address_type_max_prop`). Finally, the cluster feature (defined in Section 4.2.1.3) was transformed into two classes (`change_strategy`): no change strategy ($\mathsf{change}_\mathsf{C}$ = None), or an identified change strategy on either the first or last output ($\mathsf{change}_\mathsf{C} \in \{-1, 0, 1\}$).

Due to the small sample size (60 FP and 60 TP samples) we selected classification models that do not require extensive hyper-parameter tuning and tend to perform very well in a default setting [110]. We applied Random Forest (RF) [111, 110], which is a popular and powerful machine learning method. RF is an advancement of single classification and regression trees (CART [112]), which recursively partition the feature space into rectangular regions. Observations with similar response values are grouped together and, for classification tasks, the majority response is used as a prediction within each region. As compared to CART, RF can handle a large number of covariates effectively without overfitting and are able to account for correlation as well as interactions among features. Another important property of RF is that it immediately provides internal variable importance measures that can be used to rank covariates. For fitting of the CART-based RF approach, we used the implementation in the R-package *ranger* [113]. As an alternative, we also applied the *cforest* implementation from the package *party* [114, 115].

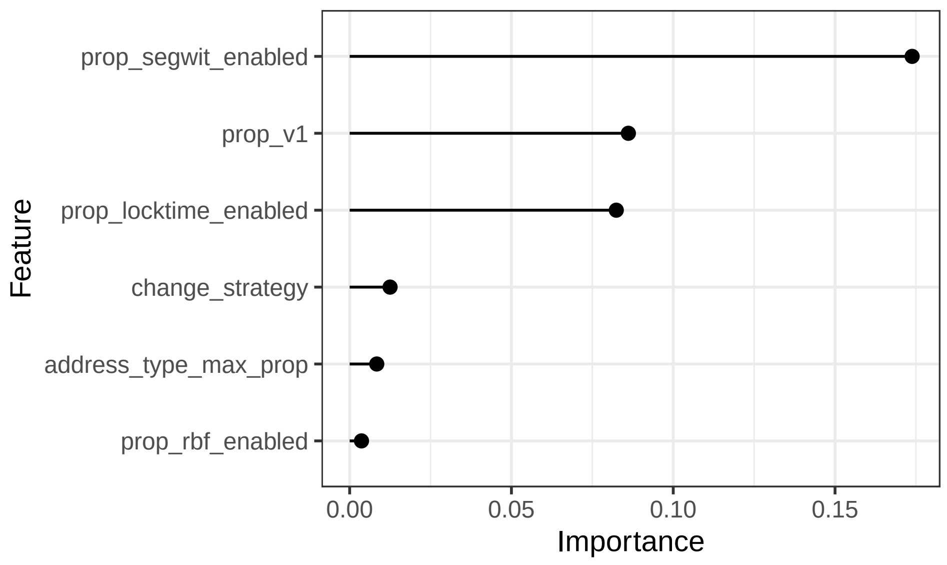


Figure 4.2: Variable importance of a conditional random forest model (full dataset).

4.4.1.2 Classification results

We fit RF models with 500 trees to the dataset consisting of the features described above and using the cluster type (TP/FP) as the target variable. First, we fit the models to the full dataset and analyzed the intrinsic variable importance measures. Figure 4.2 shows the (conditional) variable importance for the *cforest* model fitted on the full dataset. The three most important features are the proportion of SegWit transactions, the proportion of version 1 transactions, and the proportion of transactions with enabled locktime. This is also in agreement with the (permutation) variable importance obtained from the CART-based RF.

The model performance was evaluated using resampling techniques. For training and testing of the models we implemented a cross-validation (CV) procedure. Accuracy, meaning the proportion of correctly classified instances, was chosen as a model performance evaluation metric. The mean accuracy values and their associated standard errors after a 5-fold CV are shown in Table 4.2, and a confusion matrix is in Table 4.3a. We obtain a mean accuracy between 84% and 89%. According to the standard errors, these values are also relatively stable on the CV-testing folds.

Overall, while our classifier would of course benefit from extended experimentation with a larger dataset, these results and the high level of accuracy suggest that

Statistic	RF	Conditional RF
Mean accuracy	0.892	0.842
Standard error	0.017	0.031

Table 4.2: Performance of our Random Forest model after 5-fold cross-validation.

	Truth		Class error
Prediction	FP	TP	
FP	54	7	11.5 %
TP	6	53	10.2 %

(a) Confusion matrix of our RF model (summed values after 5-fold CV).

	Truth		Class error
Heuristic	FP	TP	
FP	48	3	5.9 %
TP	12	57	17.4 %

(b) Confusion matrix for the BlockSci classifier.

Table 4.3: Comparison of classification results.

it would be possible to deploy this method in the manner suggested earlier in this section; i.e., for an investigator to use it to gain some confidence in the results of the co-spend heuristic at an early stage in an investigation.

4.4.1.3 Comparison with BlockSci

To compare our statistical approach to the current state of the art, we explore the Coinjoin detection feature available in BlockSci [109] as the function `isCoinjoin`. This function works at the level of transactions, meaning given a transaction it outputs 0 or 1 according to whether or not it seems to be a Coinjoin. It first checks if a transaction has (1) more than two inputs or more than three outputs. It then defines the number of *participants* in a Coinjoin as half the number of outputs plus one ($\lfloor (n+1)/2 \rfloor$ where n is the number of outputs). It checks that (2) the number of participants is at most twice the number of inputs, following the logic that each participant should split their coins at most twice, and (3) the number of participants does not exceed the number of unique input addresses. Finally, it checks that (4) the count of the most common output value is equal to the number of participants, meaning participants are mixing the same number of coins, and that (5) no outputs receive a "dust" amount of BTC (defined as 546 or 2730 satoshis). The transaction is marked as a Coinjoin if and only if all the conditions are met.

Using this function, we define a cluster-level classifier by saying that if any transaction in the cluster is flagged as a Coinjoin, the entire cluster is a false positive. We tested this classifier on our full dataset of TP and FP clusters; the results are in Table 4.3b. As we can see, both BlockSci and our classifier have high accuracy (87.5% and 89.2% respectively), with BlockSci having twice as many false negatives and our classifier having more false positives (7 as compared to 3). As argued earlier, the false negative rate is more crucial in our imagined use case, as a false negative could cause an investigator to believe that a cluster represents a single entity when in fact it does not. Furthermore, the fact that our classifier is based on all of the features within a cluster makes it more robust than our constructed BlockSci classifier, which depends only on the features of a single transaction and can thus be easily evaded by constructing Coinjoins without these specific features.

4.4.2 A validation heuristic

Our second method for increasing the confidence we have in a cluster focuses less on the cluster as a whole and more on the connections between individual transactions. In particular, we utilize the heuristics defined in Section 4.3 to partition the transactions of a cluster into peel chains.

4.4.2.1 Defining the heuristic

Our starting point is a co-spend cluster C, represented by the tuple $(\mathsf{C}_{\mathsf{addr}}, \mathsf{C}_{\mathsf{tx}})$. Next, we run $\mathsf{followFwd}$ and $\mathsf{followBkwd}$ with the parameter $\mathsf{heur} = \mathsf{validation}$ for every $\mathsf{tx} \in \mathsf{C}_{\mathsf{tx}}$. Crucially, this parameter means that we do not follow any transactions that are not already in the cluster. For a given tx this gives us the sets $\mathsf{fwdTxs}_{\mathsf{tx},\mathsf{validation}}$ and $\mathsf{bkwdTxs}_{\mathsf{tx},\mathsf{validation}}$, which collectively represent all transactions within the cluster that lie along the same peel chain as the starting one. We denote the union of these two sets as $\mathsf{Pchain}_{\mathsf{V}}(\mathsf{tx})$.

After obtaining the set $\{\mathsf{Pchain}_{\mathsf{V}}(\mathsf{tx})\}_{\mathsf{tx}} \in \mathsf{C}_{\mathsf{tx}}$ of all such peel chains, we noticed that some peel chains contained overlapping but not identical sets of transactions, according to the starting transaction tx. We thus merged these overlapping peel chains in a transitive fashion to end up with a set of distinct peel chains, $\mathsf{Pchain}_\mathsf{V}(\mathsf{C})$, that is a partition of all transactions in the cluster. To measure the overall tendency for a cluster to form peel chains, we use the value $\mathsf{Val}_\mathsf{C} = \frac{|\mathsf{Pchain}_\mathsf{V}(\mathsf{C})|}{|\mathsf{C}_\mathsf{tx}|}$, which is closer to 1 if a cluster consists of many peel chains and closer to 0 if it consists of fewer.

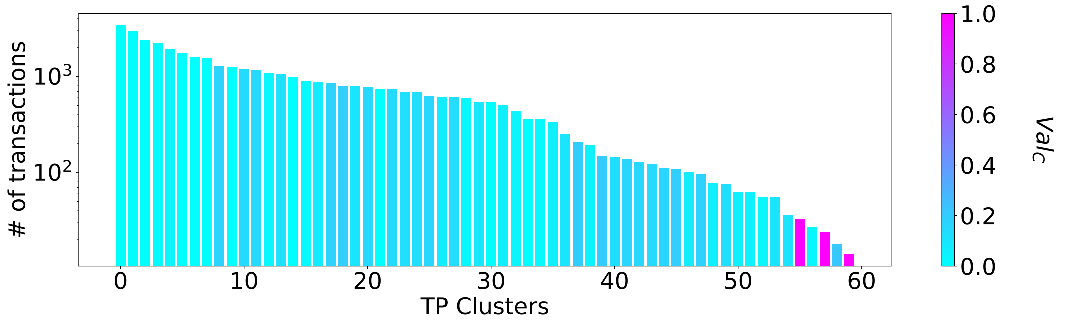
Our *validation heuristic* then says that for any two transactions $\mathsf{tx}_1, \mathsf{tx}_2 \in \mathsf{Pchain}_\mathsf{V}$ for $\mathsf{Pchain}_\mathsf{V} \in \mathsf{Pchain}_\mathsf{V}(\mathsf{C})$ (i.e., two transactions that are part of the same cluster and part of the same peel chain), we can have higher confidence that they were performed by the same entity than for two transactions $\mathsf{tx}_1, \mathsf{tx}_2 \in \mathsf{C}_\mathsf{tx}$.

4.4.2.2 Applying the heuristic

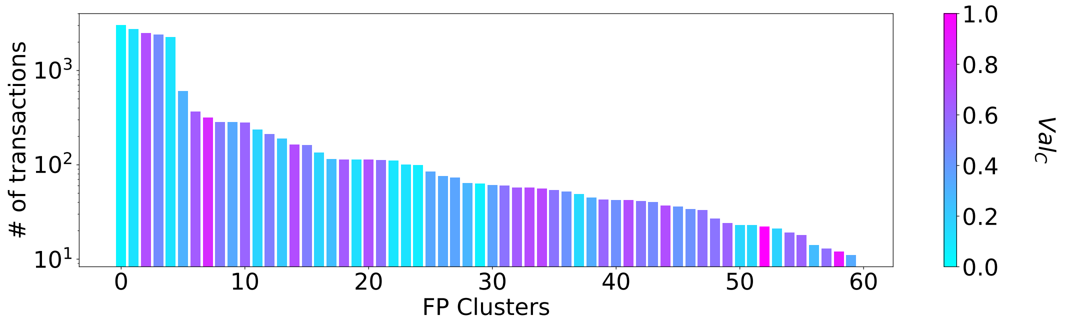
It is not possible to assess the accuracy of our validation heuristic directly, as we do not have the relevant ground-truth data. For our FP clusters, for example, we knew that they contained at least one Coinjoin but did not have any information about the other transactions (and indeed for some FP clusters it was clear they contained other Coinjoins beyond the ones we were given). For our TP clusters, we knew that all transactions were performed by the same entity but not if they represented the same peel chain.

Instead, we used our validation heuristic to understand the behavior of our TP and FP clusters. To this end, we ran the validation heuristic for each of our clusters and looked at the resulting value of Val_C. As our results in Figure 4.3 show, the FP clusters had significantly higher values of Val_C on average: 0.43 as compared to 0.14 for TP clusters. Overall, Val_C did not increase with the size of the cluster, despite the possible expectation that clusters with a higher number of transactions would form a higher number of peel chains. This suggests instead that bigger clusters tend to be more predictable in terms of their behavior, which is perhaps not surprising if we consider that the operators of these big clusters use automated scripts in order to form their transactions.

In terms of using this heuristic in practice, there are currently several *nested*



(a) TP clusters



(b) FP clusters

Figure 4.3: For our 60 TP and FP clusters, the distribution of $|\mathsf{Val}_\mathsf{C}|$ (the color of the bar), with the clusters ordered from left to right by $|\mathsf{C}_\mathsf{tx}|$ (the height of the bar, on a log scale.

services [116] that operate using accounts maintained at a variety of different exchanges. Using just the co-spend heuristic, these nested services would thus appear to be operated by the same entity as the exchange they use. Using our validation heuristic, however, it would be possible to separate out the activities of these nested services (which would likely form their own peel chains) from the activities of the exchange itself.

Furthermore, while we defined our validation heuristic for pairs of transactions, as we can see in Figure 4.3 a lower value of Val_C can also increase our confidence in the cluster overall. This is difficult to quantify given our current source of ground truth, however, so we leave as open work a more thorough evaluation of the impact of this heuristic on overall cluster confidence.

4.5 Expanding Clusters

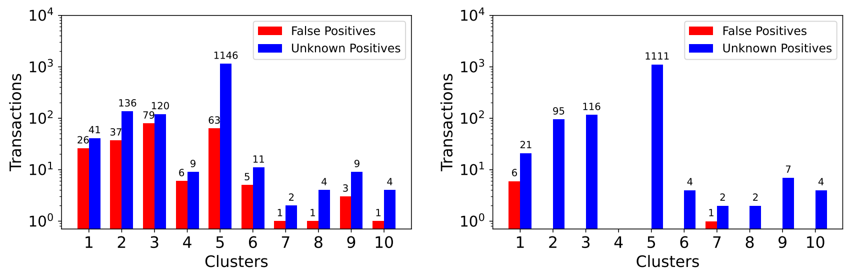
Our expansion heuristic is structurally similar to our validation heuristic, in that it is also based on the ability to identify peel chains, but it has a different goal: to identify new transactions that were not already in the cluster but for which we nevertheless have high confidence that they were formed by the same entity. In this approach, this heuristic more closely resembles previous change heuristics in the literature.

4.5.1 Defining the heuristic

As with the validation heuristic, our starting point is a co-spend cluster C represented by a tuple $(\mathsf{C}_{\mathsf{addr}}, \mathsf{C}_{\mathsf{tx}})$. Next, we run $\mathsf{followFwd}$ and $\mathsf{followBkwd}$ with the parameter $\mathsf{heur} = \mathsf{expansion}$ for every $\mathsf{tx} \in \mathsf{C}_{\mathsf{tx}}$. This gives us the sets $\mathsf{fwdTxs}_{\mathsf{tx},\mathsf{expansion}}$ and $\mathsf{bkwdTxs}_{\mathsf{tx},\mathsf{expansion}}$ for every tx, which still represent the set of all transactions that lie along the same peel chain as tx but crucially may contain transactions that are not already in the cluster. We denote the union of these sets, representing all identified transactions, as $\mathsf{Txs}_{\mathsf{expansion}}$ and denote by $\mathsf{expansion}_{\mathsf{C}}$ the set of newly identified transactions; i.e., the ones that weren't already in the cluster ($\mathsf{Txs}_{\mathsf{expansion}} \setminus \mathsf{C}_{\mathsf{tx}}$). Our *expansion* heuristic then says that all transactions in $\mathsf{expansion}_{\mathsf{C}}$ were carried out by the same entity represented by the cluster.

After defining this heuristic, our goal was to identify its accuracy and its effectiveness. To measure effectiveness we defined the *expansion factor* Expsn as the increase of a cluster's coverage in terms of its number of transactions $(100 \cdot \frac{\mathsf{expansion}_{\mathsf{C}}}{\mathsf{C}_{\mathsf{tx}}})$. To measure accuracy, we treated as ground truth the set of tags provided to us in the Chainalysis Reactor tool,[2] which are tags that are gathered either internally by Chainalysis or from public websites and documents. In particular, for each transaction in $\mathsf{expansion}_{\mathsf{C}}$, if Chainalysis had tagged it as belonging to an entity then we considered it a false positive. If it had no tag for the transaction, we considered it an *unknown positive*. In other words, while we could not be sure that the transaction was formed by the same entity, there was at least no evidence to the contrary. We then considered the *false discovery rate* FDR as the number of false positives divided by the size of $\mathsf{expansion}_{\mathsf{C}}$ (which is the standard definition for false discovery

[2] https://www.chainalysis.com/chainalysis-reactor/



(a) findNext **(b)** findNext2

Figure 4.4: Evaluation of findNext and the modified algorithm findNext2 for the ten TP clusters with the initial highest FDR, with the number of transactions on a log scale.

rate if we treat unknown positives as true positives).

In running the heuristic in its basic form, however, we encountered two problems. First, because previous transactions were not filtered out in followBkwd in the way they were for the validation heuristic, the set bkwdScope was significantly larger and it became computationally infeasible to run the algorithm for longer peel chains. Second, including so many transactions also increased the possibility of encountering a false positive, as discussed in Section 4.3.2, and thus made the algorithm more prone to error. For both of these reasons we decided to limit our expansion heuristic and only follow peel chains forwards using followFwd.

After running this version of our heuristic, we achieved an FDR of 0.62%, which was already quite low relative to the other heuristics (as we see below in Section 4.5.2). Nevertheless, after manually inspecting some of the false positive transactions, we observed that the main cause was following outputs to transactions with multiple inputs that were in turn part of a bigger cluster. We thus added an extra condition into findNext requiring that the inputs in the next hop next represented the entirety of the addresses in their cluster. In other words, we added only those transactions whose inputs were only ever co-spent with each other. We call this modified change heuristic findNext2.

To illustrate the effect of this modification, Figure 4.4a shows the ten TP clusters for which findNext had the highest FDR and thus performed the least well. We then re-ran expansion using findNext2; Figure 4.4b shows the results for the same ten clusters. As we can see, in eight of the clusters, findNext2 eliminated all false positives, at the cost of missing a relatively small number of unknown positives.

4.5.2 Evaluating the heuristic

In order to best evaluate our expansion heuristic, we sought to compare it with previous change heuristics.

Androulaki et al. [5] identify the change output in a transaction tx if (1) the transaction has exactly two outputs, and (2) it has the only *fresh* address in tx.outputs, meaning output.addr is the only one appearing for the first time in the blockchain.

Meiklejohn et al. [9] identify the change output in a transaction tx if (1) it has the only fresh address in tx.outputs; (2) tx is not a coin generation; and (3) there is no *self-change address* in tx.outputs, meaning no address used as both an input and an output.

Goldfeder et al. [94] use the same conditions as the one by Meiklejohn et al. but additionally require that (4) the transaction tx is not a Coinjoin.

Ermilov et al. [6] were the first to consider not only the behavior of the outputs and their addresses but also the value they received. They identify the change output in a transaction tx if (1) the transaction has exactly two outputs; (2) the transaction does not have two inputs; (3) there is no self-change address; (4) the output has the only fresh address in tx.outputs; and (5) the output's value is significant to at least the fourth decimal place.

We implemented each of these heuristics and ran the expansion heuristic on each of our 60 TP clusters using these algorithms as well as our own algorithms findNext and findNext2. The results, in terms of false discovery rate (FDR) and expansion factor (Expsn), are in Table 4.4.

As Table 4.4 shows, our heuristics achieve both a significantly lower false positive rate than all previous heuristics and a significantly higher expansion rate. The heuristics from Androulaki et al. and Meiklejohn et al. have the highest false discovery rates, which is somewhat expected given that Bitcoin has changed considerably

Heuristic	Expsn	FDR
findNext	147.43	0.62
findNext2	124.46	0.02
Androulaki et al. [5]	93.03	64.19
Meiklejohn et al. [9]	79.94	51.64
Goldfeder et al. [94]	73.7	48.7
Ermilov et al. [6]	28.6	12.7

Table 4.4: The expansion factor and false discovery rate of findNext and findNext2, as evaluated on our 60 TP clusters and as compared with previous change heuristics. Both metrics are averaged across all clusters.

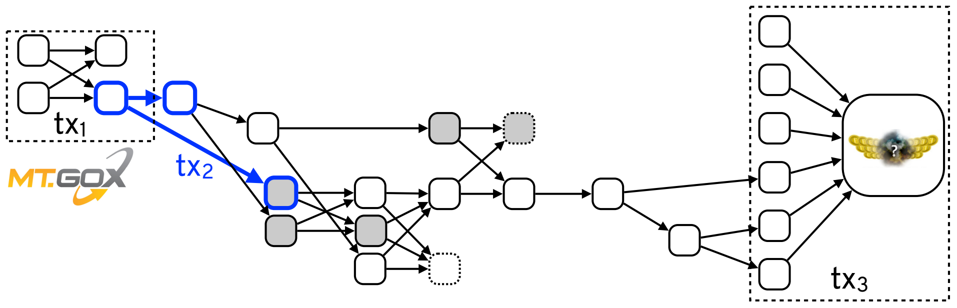


Figure 4.5: Transactions representing the link between a withdrawal from Mt. Gox (tx_1) and a deposit into Bitcoin Fog (tx_3), with an additional transaction of interest, tx_2, highlighted in blue. The gray nodes represent multiple UTXOs of the same address and transaction outputs with a dashed outline are unspent.

since they were introduced in 2013. As might also be expected, the heuristic from Goldfeder et al. achieved similar results to the one from Meiklejohn et al., with the extra Coinjoin requirement reducing both the expansion and the false positive rates by a small amount. Finally, Ermilov et al. achieved the lowest FDR of the four because of the stricter conditions of their heuristic, but this came at the expense of having the lowest expansion rate.

4.5.3 Applying the heuristic: Bitcoin Fog

To test our expansion heuristic in practice, we sought to run it for a cluster that had been associated with known illicit activities. In particular, we looked at a recent case against Roman Sterlingov, who was accused of being the operator of the Bitcoin Fog mixing service for almost 10 years [99]. According to the affidavit of an IRS special agent [100], one of the main pieces of evidence against Sterlingov was the connection of a deposit made in 2011 to the Bitcoin Fog cluster (tx_3 in Figure 4.5) with a withdrawal from the Mt. Gox exchange cluster (tx_1), in which Sterlingov had an account using his real name.

We first checked whether or not the cluster that received the coins from Mt. Gox was the same as the cluster that sent the coins to Bitcoin Fog, in order to check if it would be possible to link tx_1 with tx_3 based solely on the co-spend heuristic. This was not the case, however, meaning it was necessary to take intermediate transactions into account.

We then followed the funds from the Mt. Gox withdrawal forwards, using followFwd, to see if we would reach the deposit to Bitcoin Fog. Both findNext and findNext2 failed after only one hop, however, as the two outputs in tx_2 had the same address features and were spent in transactions with the same features. Our algorithms were thus unable to isolate the change output. These outputs were both furthermore *fresh*, meaning it was their first appearance in the blockchain, so the other change heuristics described in Section 4.5.2 also would have been unable to follow the transaction forwards.

We thus worked backwards instead; i.e., we started with the deposit to Bitcoin Fog and followed the funds backwards to see if we would eventually find the withdrawal from Mt. Gox. While running followBkwd for all transactions in a cluster was computationally infeasible, it was possible to do it here as we started with only a single transaction. Indeed, after seven hops, we ended up with tx_1 in our set $\mathsf{bkwdTxs}_{\mathsf{tx}_3,\mathsf{expansion}}$. While this same analysis was likely done manually by the IRS agent, this would quickly become infeasible if there were more intermediate hops; furthermore, manual analysis is arguably more error-prone as it is subject to human judgment.

While successful in this case, it is important to remember our discussion in Section 4.3.2 that followBkwd is more prone to false positives than followFwd. Indeed, in 2011 all transactions had the same features (as the ones we use were not introduced until years later), meaning followBkwd could continue backwards indefinitely without ever registering a change in the entity performing the transactions. In our case, the paths from the last three deposits in tx_3 all led back to the same origin (the second output in tx_1), which in turn sent all its money to these three inputs and two unspent UTXOs, meaning we could be more sure in the link between the two. While care must thus be taken when using followBkwd in this way, applying it to a present-day scenario would likely be more safe as transactions would be expected to have a more diverse set of features.

4.6 Conclusion

In this paper, we presented heuristics to expand and validate the applicability of widely used Bitcoin clustering heuristics, using a balanced ground-truth dataset to evaluate them. While this research arguably further reduces anonymity in Bitcoin, we believe that it ultimately benefits the developers and users of this project in revealing the extent to which tracking flows of bitcoins is possible and motivating further research into improved anonymity protocols. As an immediate countermeasure, users who are concerned about their privacy can switch to more privacy-focused cryptocurrencies such as Zcash and Monero, although previous work has shown that even these are subject to some degree of de-anonymization [117, 118, 119, 120].

Chapter 5

Measuring the privacy offered in Zcash

In this chapter, we introduce ”The Empirical Analysis of Anonymity in Zcash”, which takes a deep look into the Privacy offered by the layer 1 of Zcash.

Among the now numerous alternative cryptocurrencies derived from Bitcoin, Zcash is often touted as the one with the strongest anonymity guarantees, due to its basis in well-regarded cryptographic research. In this work, we examined the extent to which anonymity is achieved in the deployed version of Zcash. We investigated all facets of anonymity in Zcash's transactions, ranging from its transparent transactions to the interactions with and within its main privacy feature, a shielded pool that acts as the anonymity set for users wishing to spend coins privately. We concluded that while it is possible to use Zcash in a private way, it is also possible to shrink its anonymity set considerably by developing simple heuristics based on identifiable patterns of usage.

5.1 Introduction

Since the introduction of Bitcoin in 2008 [1], cryptocurrencies have become increasingly popular to the point of reaching a near-mania, with thousands of deployed cryptocurrencies now collectively attracting trillions of dollars in investment. While the broader positive potential of "blockchain" (i.e., the public decentralized ledger underlying almost all cryptocurrencies) is still unclear, despite the growing number of legitimate users there are today still many people using these cryptocurrencies for less legitimate purposes. These range from the purchase of drugs or other illicit goods on so-called dark markets such as Dream Market, to the payments from victims in ransomware attacks such as WannaCry, with many other crimes in between. Criminals engaged in these activities may be drawn to Bitcoin due to the relatively low friction of making international payments using only pseudonyms as identifiers, but the public nature of its ledger of transactions raises the question of how much anonymity is actually being achieved.

Indeed, a long line of research [4, 8, 121, 15**?**] has by now demonstrated that the use of pseudonymous addresses in Bitcoin does not provide any meaningful level of anonymity. Beyond academic research, companies now provide analysis of the Bitcoin blockchain as a business [122]. This type of analysis was used in several arrests associated with the takedown of Silk Road [123], and to identify the attempts of the WannaCry hackers to move their ransom earnings from Bitcoin into Monero [124].

Perhaps in response to this growing awareness that most cryptocurrencies do not have strong anonymity guarantees, a number of alternative cryptocurrencies or other privacy-enhancing techniques have been deployed with the goal of improving on these guarantees. The most notable cryptocurrencies that fall into this former category are Dash [125] (launched in January 2014), Monero [126] (April 2014), and Zcash [45] (October 2016). At the time of this writing all have a market capitalization of over 1 billion USD [127], although this figure is notoriously volatile, so should be taken with a grain of salt.

Even within this category of privacy-enhanced cryptocurrencies, and despite its relative youth, Zcash stands somewhat on its own. From an academic perspective, Zcash is backed by highly regarded research [37, 40], and thus comes with seemingly strong anonymity guarantees. Indeed, the original papers cryptographically prove the security of the main privacy feature of Zcash (known as the *shielded pool*), in which users can spend shielded coins without revealing which coins they have spent. These strong guarantees have attracted at least some criminal attention to Zcash: the underground marketplace AlphaBay was on the verge of accepting it before their shutdown in July 2017 [128], and the Shadow Brokers hacking group started accepting Zcash in May 2017 (and in fact for their monthly dumps accepted exclusively Zcash in September 2017) [129].

Despite these theoretical privacy guarantees, the deployed version of Zcash does not require all transactions to take place within the shielded pool itself: it also supports so-called *transparent* transactions, which are essentially the same as transactions in Bitcoin in that they reveal the pseudonymous addresses of both the senders and recipients, and the amount being sent. It does require, however, that all newly generated coins pass through the shielded pool before being spent further, thus ensuring that all coins have been shielded at least once. This requirement led the Zcash developers to conclude that the anonymity set for users spending shielded coins is in fact all generated coins, and thus that "the mixing strategies that other cryptocurrencies use for anonymity provide a rather small [anonymity set] in comparison to Zcash" and that "Zcash has a distinct advantage in terms of transaction privacy" [130].

In this paper, we provide the first in-depth empirical analysis of anonymity in Zcash, in order to examine these claims and more generally provide a longitudinal study of how Zcash has evolved and who its main participants are. We begin in Section 5.3 by providing a general examination of the Zcash blockchain, from which we observe that the vast majority of Zcash activity is in the transparent part of the blockchain, meaning it does not engage with the shielded pool at all. In Section 5.4, we explore this aspect of Zcash by adapting the analysis that has already been developed for Bitcoin, and find that exchanges typically dominate this part of the blockchain.

We then move in Section 5.5 to examining interactions with the shielded pool. We find that, unsurprisingly, the main actors doing so are the founders and miners, who are required to put all newly generated coins directly into it. Using newly developed heuristics for attributing transactions to founders and miners, we find that 65.6% of the value withdrawn from the pool can be linked back to deposits made by either founders or miners. We also implement a general heuristic for linking together other types of transactions, and capture an additional 3.5% of the value using this. Our relatively simple heuristics thus reduce the size of the overall anonymity set by 69.1%.

In Section 5.6, we then look at the relatively small percentage of transactions that have taken place within the shielded pool. Here, we find (perhaps unsurprisingly) that relatively little information can be inferred, although we do identify certain patterns that may warrant further investigation. Finally, we perform a small case study of the activities of the Shadow Brokers within Zcash in Section 5.7, and in Section 5.8 we conclude.

All of our results have been disclosed, at the time of the paper's submission, to the creators of Zcash, and discussed extensively with them since. This has resulted in changes to both their public communication about Zcash's anonymity as well as the transactional behavior of the founders. Additionally, all the code for our analysis is available as an open-source repository.[1]

5.2 Participants in the Zcash ecosystem

In this section we describe four types of participants who interact in the Zcash network.

Founders took part in the initial creation and release of Zcash, and will receive 20% of all newly generated coins (currently 2.5 ZEC out of the 12.5 ZEC block reward). The founder addresses are specified in the Zcash chain parameters [131].

Miners take part in the maintenance of the ledger, and in doing so receive newly generated coins (10 out of the 12.5 ZEC block reward), as well as any fees from the transactions included in the blocks they mine. Many miners choose not to mine on their own, but join a mining pool; a list of mining pools can be found in Table 5.4. One or many miners win each block, and the first transaction in the block is a *coin generation* (coingen) that assigns newly generated coins to their address(es), as well as to the address(es) of the founders.

[1] https://github.com/manganese/zcash-empirical-analysis

Services are entities that accept ZEC as some form of payment. These include exchanges like Bitfinex, which allow users to trade fiat currencies and other cryptocurrencies for ZEC (and vice versa), and platforms like ShapeShift [132], which allow users to trade within cryptocurrencies and other digital assets without requiring registration.

Finally, users are participants who hold and transact in ZEC at a more individual level. In addition to regular individuals, this category includes charities and other organizations that may choose to accept donations in Zcash. A notable user is the Shadow Brokers, a hacker group who have published several leaks containing hacking tools from the NSA and accept payment in Zcash. We explore their usage of Zcash in Section 5.7.

5.3 General Blockchain Statistics

We used the `zcashd` client to download the Zcash blockchain, and loaded a database representation of it into Apache Spark. We then performed our analysis using a custom set of Python scripts equipped with PySpark. We last parsed the block chain on January 21 2018, at which point 258,472 blocks had been mined. Overall, 3,106,643 ZEC had been generated since the genesis block, out of which 2,485,461 ZEC went to the miners and the rest (621,182 ZEC) went to the founders.

5.3.1 Transactions

Across all blocks, there were 2,242,847 transactions. A complete breakdown of the transaction types is in Table 5.1, and graphs depicting the growth of each transaction type over time are in Figures 5.1 and 5.2.[2] The vast majority of transactions are public (i.e., either transparent or a coin generation). Of the transactions that do interact with the pool (335,630, or 14.96%, in total), only a very small percentage are private transactions; i.e., transactions within the pool. Looking at the types of transactions over time in Figure 5.1, we can see that the number of coingen, shielded, and deshielded transactions all grow in an approximately linear fashion. As we explore in Section 5.5.2, this correlation is due largely to the habits of the

[2]We use the term 'mixed' to mean transactions that have both a vIn and a vOut, and a vJoinSplit.

Type	Number	Percentage
Transparent	1 648 745	73.5
Coingen	258 472	11.5
Deshielded	177 009	7.9
Shielded	140 796	6.3
Mixed	10 891	0.5
Private	6934	0.3

Table 5.1: The total number of each transaction type.

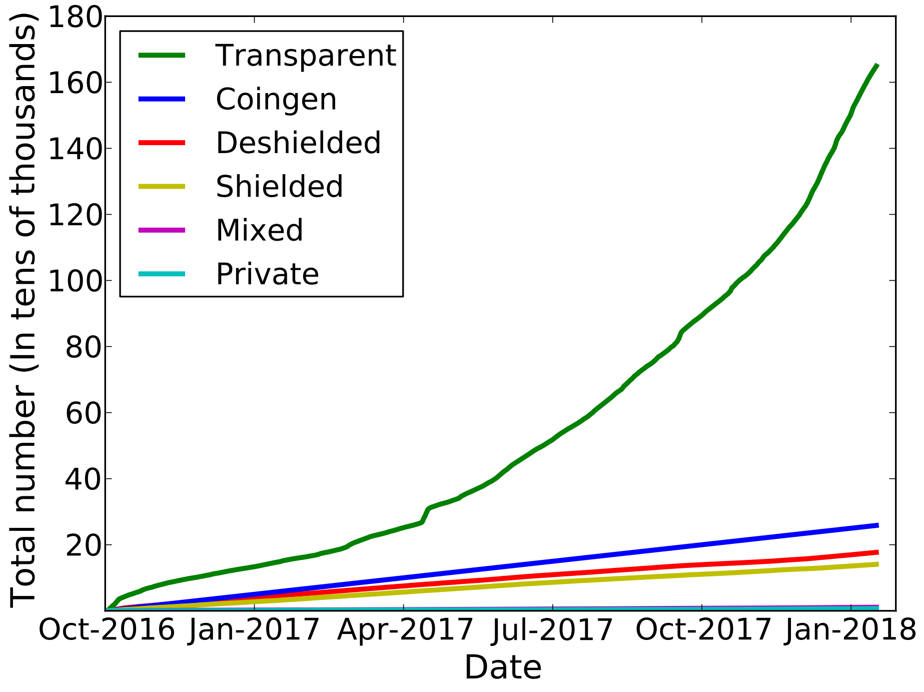


Figure 5.1: The total number of each of the different types of transactions over time.

miners. Looking at both this figure and Figure 5.2, we can see that while the number of transactions interacting with the pool has grown in a relatively linear fashion, the value they carry has over time become a very small percentage of all blocks, as more mainstream (and thus transparent) usage of Zcash has increased.

5.3.2 Addresses

Across all transactions, there have been 1,740,378 distinct t-addresses used. Of these, 8,727 have ever acted as inputs in a t-to-z transaction and 330,780 have ever acted as outputs in a z-to-t transaction. As we explore in Section 5.5.2, much of this asymmetry is due to the behavior of mining pools, which use a small number of

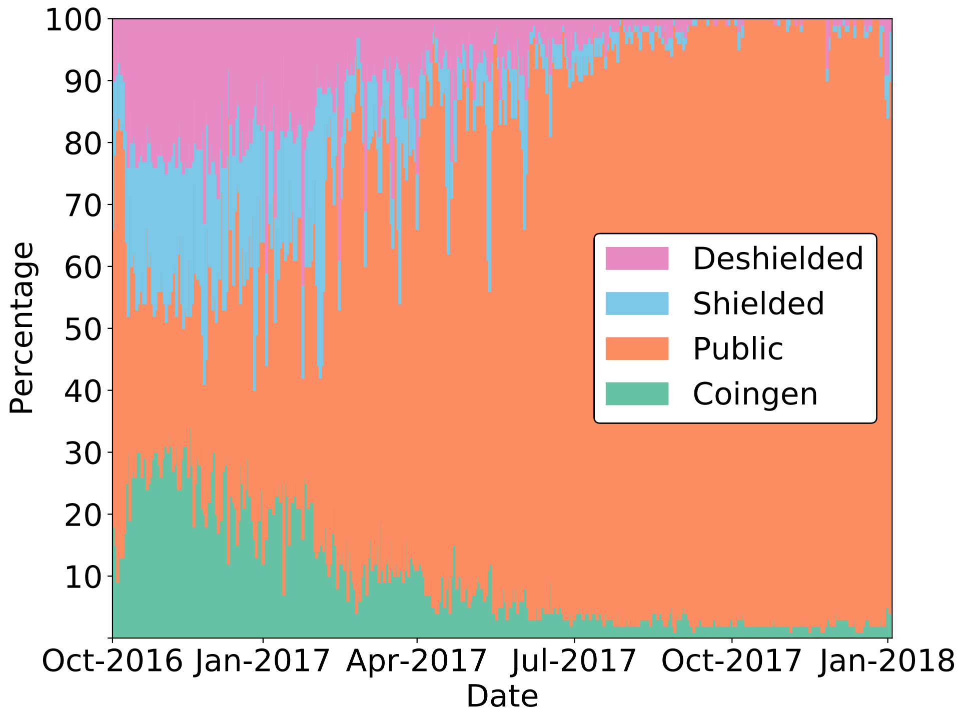


Figure 5.2: The fraction of the value in each block representing each different type of transaction over time, averaged daily. Here, 'public' captures both transparent transactions and the visible components of mixed transactions.

addresses to collect the block reward, but a large number of addresses (representing all the individual miners) to pay out of the pool. Given the nature of the shielded pool, it is not possible to know the total number of z-addresses used.

Figure 5.3 shows the total value in the pool over time. Although the overall value is increasing over time, there are certain shielding and de-shielding patterns that create spikes. As we explore in Section 5.5, these spikes are due largely to the habits of the miners and founders. At the time of writing, there are 112,235 ZEC in the pool, or 3.6% of the total monetary supply.

If we rank addresses by their wealth, we first observe that only 25% of all t-addresses have a non-zero balance. Of these, the top 1% hold 78% of all ZEC. The address with the highest balance had 118,257.75 ZEC, which means the richest address has a higher balance than the entire shielded pool.

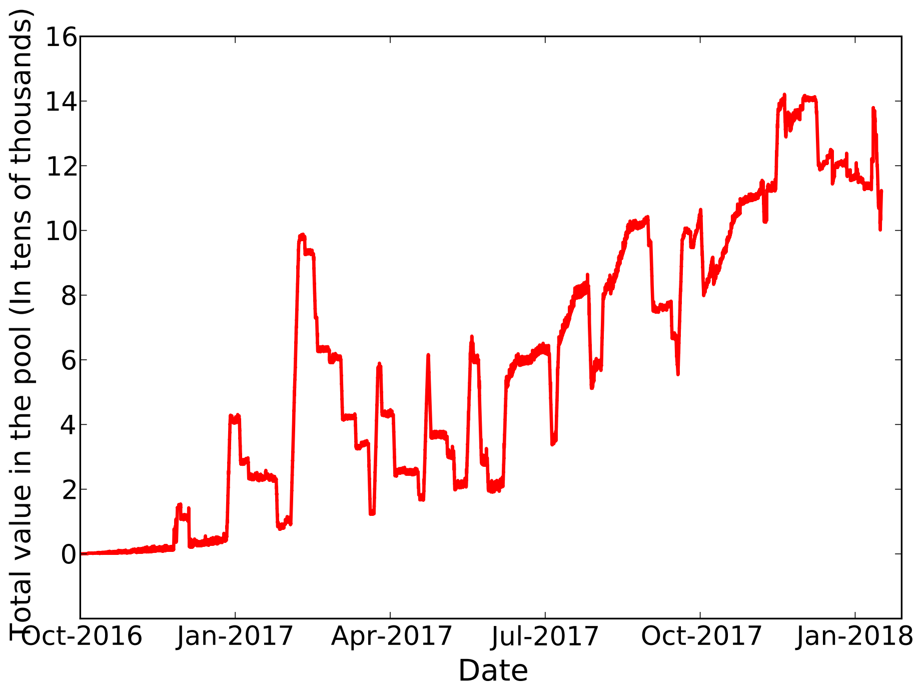


Figure 5.3: The total value in the shielded pool over time.

5.4 T-Address Clustering

As discussed in Section 5.3, a large proportion of the activity on Zcash does not use the shielded pool. This means it is essentially identical to Bitcoin, and thus can be de-anonymized using the same techniques discussed for Bitcoin in Section 3.

5.4.1 Clustering addresses

To identify the usage of transparent addresses, we begin by recalling the "multi-input" heuristic for clustering Bitcoin addresses. In this heuristic, addresses that are used as inputs to the same transaction are assigned to the same cluster. In Bitcoin, this heuristic can be applied to all transactions, as they are all transparent. In Zcash, we perform this clustering as long as there are multiple input t-addresses.

Heuristic 1. If two or more t-addresses are inputs in the same transaction (whether that transaction is transparent, shielded, or mixed), then they are controlled by the same entity.

In terms of false positives, we believe that these are at least as unlikely for Zcash as they are for Bitcoin, as Zcash is a direct fork of Bitcoin and the standard client has the same behavior. In fact, we are not aware of any input-mixing techniques like CoinJoin [57] for Zcash, so could argue that the risk of false positives is even lower than it is for Bitcoin. As this heuristic has already been used extensively in Bitcoin, we thus believe it to be realistic for use in Zcash.

We implemented this heuristic by defining each t-address as a node in a graph, and adding an (undirected) edge in the graph between addresses that had been input to the same transaction. The connected components of the graph then formed the clusters, which represent distinct entities controlling potentially many addresses. The result was a set of 560,319 clusters, of which 97,539 contained more than a single address.

As in Bitcoin, using just this one heuristic is already quite effective but does not capture the common usage of *change addresses*, in which a transaction sends coins to the actual recipient but then also sends any coins left over in the input back to the sender. Meiklejohn et al. [15] use in their analysis a heuristic based on this behavior, but warn that it is somewhat fragile. Indeed, their heuristic seems largely dependent on the specific behavior of several large Bitcoin services, so we chose not to implement it in its full form. Nevertheless, we did use a related Zcash-specific heuristic in our case study of the Shadow Brokers in Section 5.7.

Heuristic 2. If one (or more) address is an input t-address in a vJoinSplit transaction and a second address is an output t-address in the same vJoinSplit transaction, then if the size of zOut is 1 (i.e., this is the only transparent output address), the second address belongs to the same user who controls the input addresses.

To justify this heuristic, we observe that users may not want to deposit all of the coins in their address when putting coins into the pool, in which case they will have to make change. The only risk of a false positive is if users are instead sending money to two separate individuals, one using a z-address and one using a t-address. One notable exception to this rule is users of the zcash4win wallet. Here, the address of the wallet operator is an output t-address if the user decides to pay the developer fee, so would produce exactly this type of transaction for users putting money into the shielded pool. This address is identifiable, however, so these types of transactions can be omitted from our analysis. Nevertheless, due to concerns about the safety of this heuristic (i.e., its ability to avoid false positives), we chose not to incorporate it into our general analysis below.

5.4.2 Tagging addresses

Having now obtained a set of clusters, we next sought to assign names to them. To accomplish this, we performed a scaled-down version of the techniques used by Meiklejohn et al. [15]. In particular, given that Zcash is still relatively new, there are not many different types of services that accept Zcash. We thus restricted ourselves to interacting with exchanges.

We first identified the top ten Zcash exchanges according to volume traded [127]. We then created an account with each exchange and deposited a small quantity of ZEC into it, tagging as we did the output t-addresses in the resulting transaction as belonging to the exchange. We then withdrew this amount to our own wallet, and again tagged the t-addresses (this time on the sender side) as belonging to the exchange. We occasionally did several deposit transactions if it seemed likely that doing so would tag more addresses. Finally, we also interacted with ShapeShift, which as mentioned in Section 2 allows users to move amongst cryptocurrencies without the need to create an account. Here we did a single "shift" into Zcash and a single shift out. A summary of our interactions with all the different exchanges is in Table 5.2.

Finally, we collected the publicized addresses of the founders [131], as well as addresses from known mining pools. For the latter we started by scraping the tags of these addresses from the Zchain explorer [133]. We then validated them against the blocks advertised on some of the websites of the mining pools themselves (which we also scraped) to ensure that they were the correct tags; i.e., if the recipient of the coingen transaction in a given block was tagged as belonging to a given mining pool, then we checked to see that the block had been advertised on the website of that mining pool. We then augmented these sets of addresses with the addresses tagged as belonging to founders and miners according to the heuristics developed in Section 5.5. We present these heuristics in significantly more detail there, but

Service	Cluster	# deposits	# withdrawals
Binance	7	1	1
Bitfinex	3	4	1
Bithumb	14	2	1
Bittrex	1	1	1
Bit-z	30	2	1
Exmo	4	2	1
HitBTC	18	1	1
Huobi	26	2	1
Kraken	12	1	1
Poloniex	0	1	1
ShapeShift	2	1	1
zcash4win	139	1	2

Table 5.2: The services we interacted with, the identifier of the cluster they were associated with after running Heuristic 1, and the number of deposits and withdrawals we did with them. The first ten are exchanges, ShapeShift is an inter-cryptocurrency exchange, and zcash4win is a Windows-based Zcash client.

they resulted in us tagging 123 founder addresses and 110,918 miner addresses (belonging to a variety of different pools).

5.4.3 Results

As mentioned in Section 5.4.1, running Heuristic 1 resulted in 560,319 clusters, of which 97,539 contained more than a single address. We assigned each cluster a unique identifier, ordered by the number of addresses in the cluster, so that the biggest cluster had identifier 0.

5.4.3.1 Exchanges and wallets

As can be seen in Table 5.2, many of the exchanges are associated with some of the biggest clusters, with four out of the top five clusters belonging to popular exchanges. In general, we found that the top five clusters accounted for 11.21% of all transactions. Identifying exchanges is important, as it makes it possible to discover where individual users may have purchased their ZEC. Given existing and emerging regulations, they are also the one type of participant in the Zcash ecosystem that might know the real-world identity of users.

In many of the exchange clusters, we also identified large fractions of addresses that had been tagged as miners. This implies that individual miners use the addresses of their exchange accounts to receive their mining reward, which might be expected if their goal is to cash out directly. We found some, but far fewer, founder addresses at some of the exchanges as well.

Our clustering also reveals that ShapeShift (Cluster 2) is fairly heavily used: it had received over 1.1M ZEC in total and sent roughly the same. Unlike the exchanges, its cluster contained a relatively small number of miner addresses (54), which fits with its usage as a way to shift money, rather than hold it in a wallet.

5.4.3.2 Mining pools and founders

Although mining pools and founders account for a large proportion of the activity in Zcash (as we explore in Section 5.5), many re-use the same small set of addresses frequently, so do not belong to large clusters. For example, Flypool had three single-address clusters while Coinotron, coinmine.pl, Slushpool and Nanopool each had two single-address clusters. (A list of mining pools can be found in Table 5.4 in Section 5.5.2). Of the coins that we saw sent from clusters associated with mining pools, 99.8% of it went into the shielded pool, which further validates both our clustering and tagging techniques.

5.4.3.3 Philanthropists

Via manual inspection, we identified three large organizations that accept Zcash donations: the Internet Archive, torservers.net, and Wikileaks. Of these, torservers.net accepts payment only via a z-address, so we cannot identify their transactions (Wikileaks accepts payment via a z-address too, but also via a t-address). Of the 31 donations to the Internet Archive that we were able to identify, which totaled 17.3 ZEC, 9 of them were made anonymously (i.e., as z-to-t transactions). On the other hand, all of the 20 donations to Wikileak's t-address were made as t-to-t transactions. None of these belong to clusters, as they have never sent a transaction.

5.5 Interactions with the Shielded Pool

What makes Zcash unique is of course not its t-addresses (since these essentially replicate the functionality of Bitcoin), but its shielded pool. To that end, this section

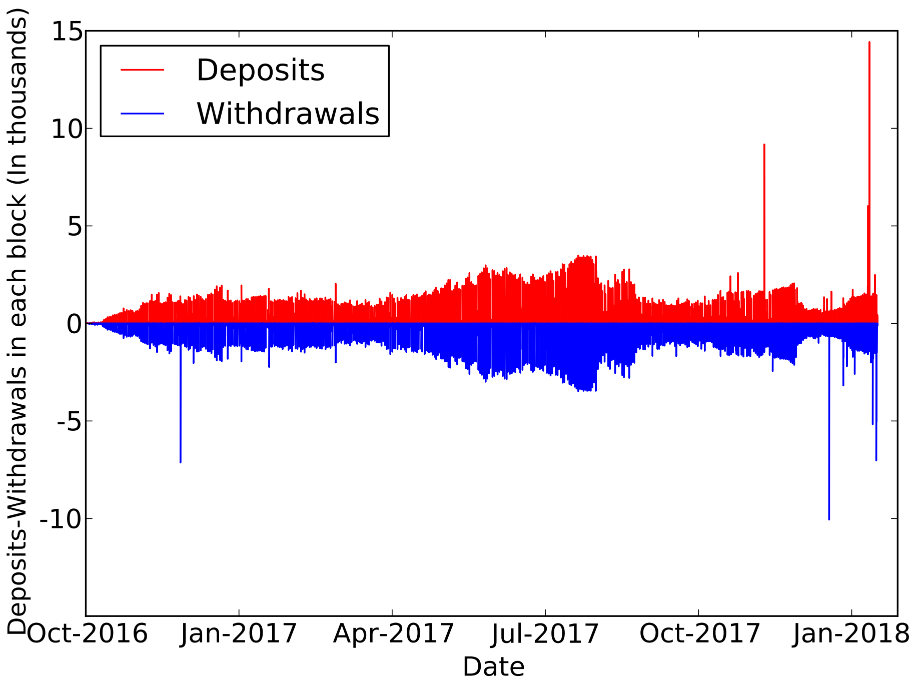


Figure 5.4: Over time, the amount of ZEC put into the shielded pool (in red) and the amount taken out of the pool (in blue).

explores interactions with the pool at its endpoints, meaning the deposits into (t-to-z) and withdrawals out of the pool (z-to-t). We then explore interactions within the pool (z-to-z transactions) in Section 5.6.

To begin, we consider just the amounts put into and taken out of the pool. Over time, 3,901,124 ZEC have been deposited into the pool,[3] and 3,788,889 have been withdrawn. Figure 5.4 plots both deposits and withdrawals over time.

This figure shows a near-perfect reflection of deposits and withdrawals, demonstrating that most users not only withdraw the exact number of ZEC they deposit into the pool, but do so very quickly after the initial deposit. As we see in Sections 5.5.1 and 5.5.2, this phenomenon is accounted for almost fully by the founders and miners. Looking further at the figure, we can see that the symmetry is broken occasionally, and most notably in four "spikes": two large withdrawals, and two large deposits. Some manual investigation revealed the following:

"The early birds" The first withdrawal spike took place at block height 30,900,

[3]This is greater than the total number of generated coins, as all coins must be deposited into the pool at least once, by the miners or founders, but may then go into and out of the pool multiple times.

which was created in December 2016. The cause of the spike was a single transaction in which 7,135 ZEC was taken out of the pool; given the exchange rate at that time of 34 USD per ZEC, this was equivalent to 242,590 USD. The coins were distributed across 15 t-addresses, which initially we had not tagged as belonging to any named user. After running the heuristic described in Section 5.5.1, however, we tagged all of these addresses as belonging to founders. In fact, this was the very first withdrawal that we identified as being associated with founders.

"Secret Santa" The second withdrawal spike took place on December 25 2017, at block height 242,642. In it, 10,000 ZEC was distributed among 10 different t-addresses, each receiving 1,000 ZEC. None of these t-addresses had done a transaction before then, and none have been involved in one since (i.e., the coins received in this transaction have not yet been spent).

"One-man wolf packs" Both of the deposit spikes in the graph correspond to single large deposits from unknown t-addresses that, using our analysis from Section 5.4, we identified as residing in single-address clusters. For the first spike, however, many of the deposited amounts came directly from a founder address identified by our heuristics (Heuristic 3), so given our analysis in Section 5.5.1 we believe this may also be associated with the founders.

While this figure already provides some information about how the pool is used (namely that most of the money put into it is withdrawn almost immediately afterwards), it does not tell us who is actually using the pool. For this, we attempt to associate addresses with the types of participants identified in Section 5.2: founders, miners, and 'other' (encompassing both services and individual users).

When considering deposits into the shielded pool, it is easy to associate addresses with founders and miners, as the consensus rules dictate that they must put their block rewards into the shielded pool before spending them further. As described in Section 5.4.2, we tagged founders according to the Zcash parameters, and tagged as miners all recipients of coingen transactions that were not founders. We then used these tags to identify a founder deposit as any t-to-z transaction using one or more founder addresses as input, and a miner deposit as any t-to-z transaction

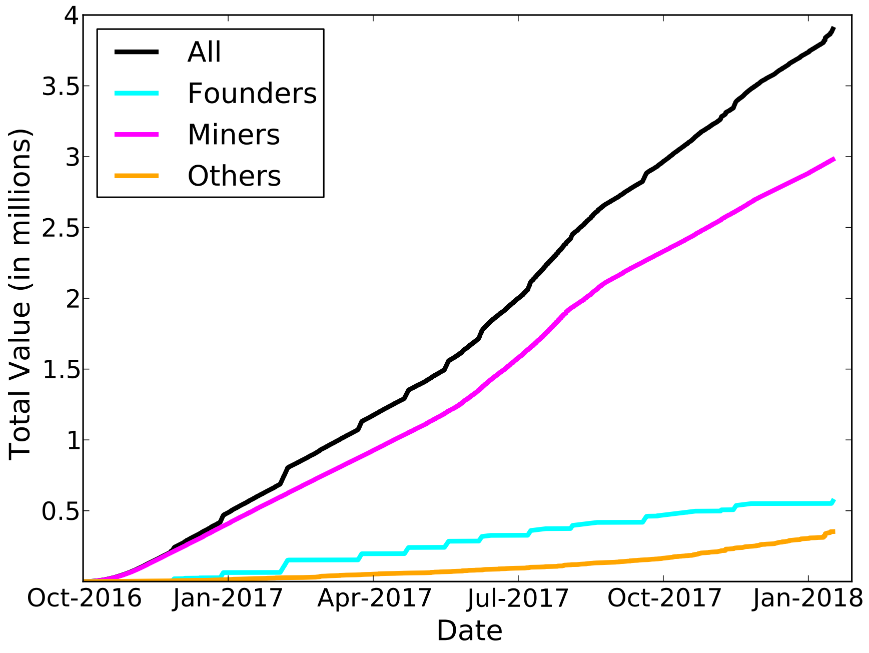


Figure 5.5: Over time, the amount of ZEC deposited into the shielded pool by miners, founders, and others.

using one or more miner addresses as input. The results are in Figure 5.5.

Looking at this figure, it is clear that miners are the main participants putting money into the pool. This is not particularly surprising, given that all the coins they receive must be deposited into the pool at least once, so if we divide that number of coins by the total number deposited we would expect at least 63.7% of the deposits to come from miners. (The actual number is 76.7%.) Founders, on the other hand, don't put as much money into the pool (since they don't have as much to begin with), but when they do they put in large amounts that cause visible step-like fluctuations to the overall line.

In terms of the heaviest users, we looked at the individual addresses that had put more than 10,000 ZEC into the pool. The results are in Figure 5.6.

In fact, this figure incorporates the heuristics we develop in Sections 5.5.1 and 5.5.2, although it looked very similar when we ran it before applying our heuristics (which makes sense, since our heuristics mainly act to link z-to-t transactions). Nevertheless, it demonstrates again that most of the heavy users of the pool are miners, with founders also depositing large amounts but spreading them over a

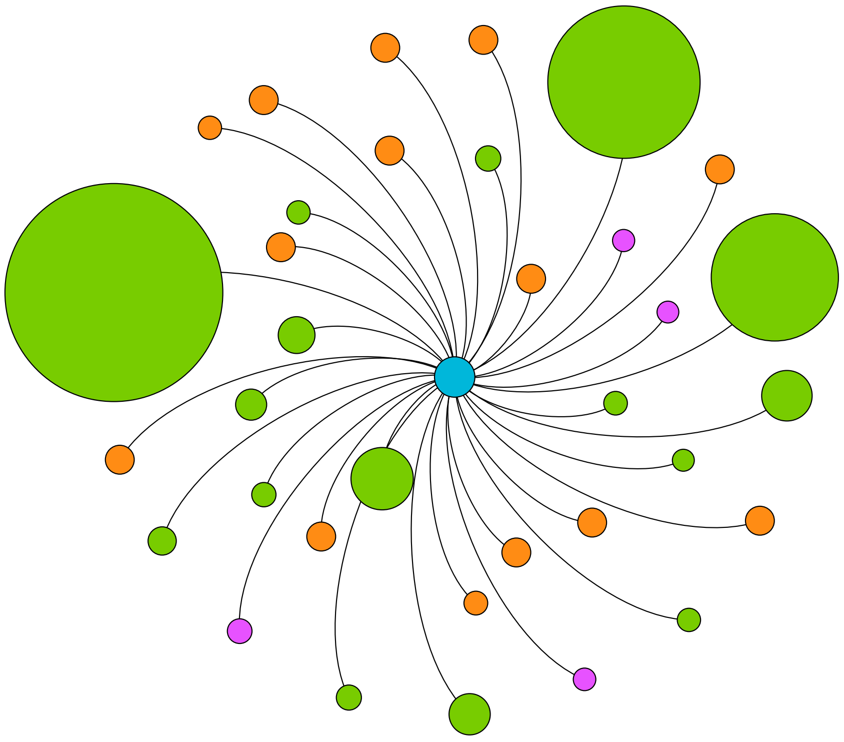
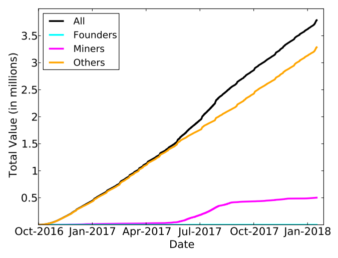


Figure 5.6: The addresses that have put more than 10,000 ZEC into the shielded pool over time, where the size of each node is proportional to the value it has put into the pool. The addresses of miners are green, of founders are orange, and of unknown 'other' participants are purple.

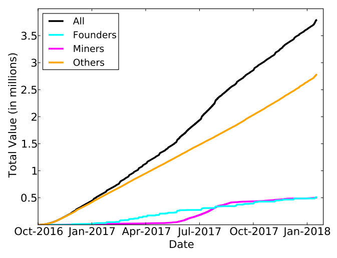
wider variety of addresses. Of the four 'other' addresses, one of them belonged to ShapeShift, and the others belong to untagged clusters.

While it is interesting to look at t-to-z transactions on their own, the main intention of the shielded pool is to provide an anonymity set, so that when users withdraw their coins it is not clear whose coins they are. In that sense, it is much more interesting to link together t-to-z and z-to-t transactions, which acts to reduce the anonymity set. More concretely, if a t-to-z transaction can be linked to a z-to-t transaction, then those coins can be "ruled out" of the anonymity set of future users withdrawing coins from the pool. We thus devote our attention to this type of analysis for the rest of the section.

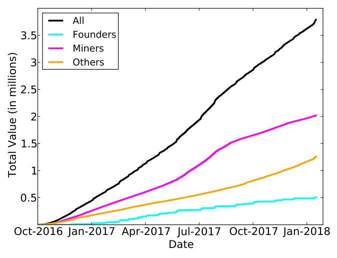
The most naïve way to link together these transactions would be to see if the same addresses are used across them; i.e., if a miner uses the same address to withdraw their coins as it did to deposit them. By running this simple form of linking, we see the results in Figure 5.7a. This figure shows that we are not able to identify



(a) No heuristics



(b) Founder heuristic



(c) Founder and miner heuristics

Figure 5.7: The z-to-t transactions we associated with miners, founders, and 'other', after running some combination of our heuristics.

any withdrawals as being associated with founders, and only a fairly small number as associated with miners: 49,280 transactions in total, which account for 13.3% of the total value in the pool.

Nevertheless, using heuristics that we develop for identifying founders (as detailed in Section 5.5.1) and miners (Section 5.5.2), we are able to positively link most of the z-to-t activity with one of these two categories, as seen in Figures 5.7b and 5.7c. In the end, of the 177,009 z-to-t transactions, we were able to tag 120,629 (or 68%) of them as being associated with miners, capturing 52.1% of the value coming out of the pool, and 2,103 of them as being associated with founders (capturing 13.5% of the value). We then examine the remaining 30-35% of the activity surrounding the shielded pool in Section 5.5.3.

5.5.1 Founders

After comparing the list of founder addresses against the outputs of all coingen transactions, we found that 14 of them had been used. Using these addresses, we were able to identify founder deposits into the pool, as already shown in Figure 5.5. Table 5.3 provides a closer inspection of the usage of each of these addresses.

	# Deposits	Total value	# Deposits (249)
1	548	19 600.4	0
2	252	43 944.6	153
3	178	44 272.5	177
4	192	44 272.5	176
5	178	44 272.5	177
6	178	44 272.5	177
7	178	44 272.5	177
8	178	44 272.5	177
9	190	44 272.5	176
10	188	44 272.5	176
11	190	44 272.5	176
12	178	44 272.5	177
13	191	44 272.5	175
14	70	17 500	70
Total	2889	568 042.5	2164

Table 5.3: The behavior of each of the 14 active founder addresses, in terms of the number of deposits into the pool, the total value deposited (in ZEC), and the number of deposits carrying exactly 249.9999 ZEC in value.

This table shows some quite obvious patterns in the behavior of the founders. At any given time, only one address is "active," meaning it receives rewards and deposits them into the pool. Once it reaches the limit of 44,272.5 ZEC, the next address takes its place and it is not used again. This pattern has held from the third address onwards. What's more, the amount deposited was often the same: exactly 249.9999 ZEC, which is roughly the reward for 100 blocks. This was true of 74.9% of all founder deposits, and 96.2% of all deposits from the third address onwards. There were only ever five other deposits into the pool carrying value between 249 and 251 ZEC (i.e., carrying a value close but not equal to 249.9999 ZEC).

Thus, while we were initially unable to identify any withdrawals associated with the founders (as seen in Figure 5.7a), these patterns indicated an automated use of the shielded pool that might also carry into the withdrawals. Upon examining the withdrawals from the pool, we did not find any with a value exactly equal to 249.9999 ZEC. We did, however, find 1,953 withdrawals of exactly 250.0001 ZEC (and 1,969 carrying a value between 249 and 251 ZEC, although we excluded the extra ones from our analysis).

The value alone of these withdrawals thus provides some correlation with the deposits, but to further explore it we also looked at the timing of the transactions. When we examined the intervals between consecutive deposits of 249.9999 ZEC, we found that 85% happened within 6-10 blocks of the previous one. Similarly, when examining the intervals between consecutive withdrawals of 250.0001 ZEC, we found that 1,943 of the 1,953 withdrawals also had a proximity of 6-10 blocks. Indeed, both the deposits and the withdrawals proceeded in step-like patterns, in which many transactions were made within a very small number of blocks (resulting in the step up), at which point there would be a pause while more block rewards were accumulated (the step across). This pattern is visible in Figure 5.8, which shows the deposit and withdrawal transactions associated with the founders. Deposits are typically made in few large steps, whereas withdrawals take many smaller ones.

Heuristic 3. Any z-to-t transaction carrying 250.0001 ZEC in value is done by the founders.

In terms of false positives, we cannot truly know how risky this heuristic is, short of asking the founders. This is in contrast to the t-address clustering heuristics presented in Section 5.4, in which we were not attempting to assign addresses to a specific owner, so could validate the heuristics in other ways. Nevertheless, the high correlation between both the value and timing of the transactions led us to believe in the reliability of this heuristic.

As a result of running this heuristic, we added 75 more addresses to our initial list of 48 founder addresses (of which, again, only 14 had been used). Aside from the correlation showed in Figure 5.8, the difference in terms of our ability to tag founder withdrawals is seen in Figure 5.7b.

5.5.2 Miners

The Zcash protocol specifies that all newly generated coins are required to be put into the shielded pool before they can be spent further. As a result, we expect that a large quantity of the ZEC being deposited into the pool are from addresses

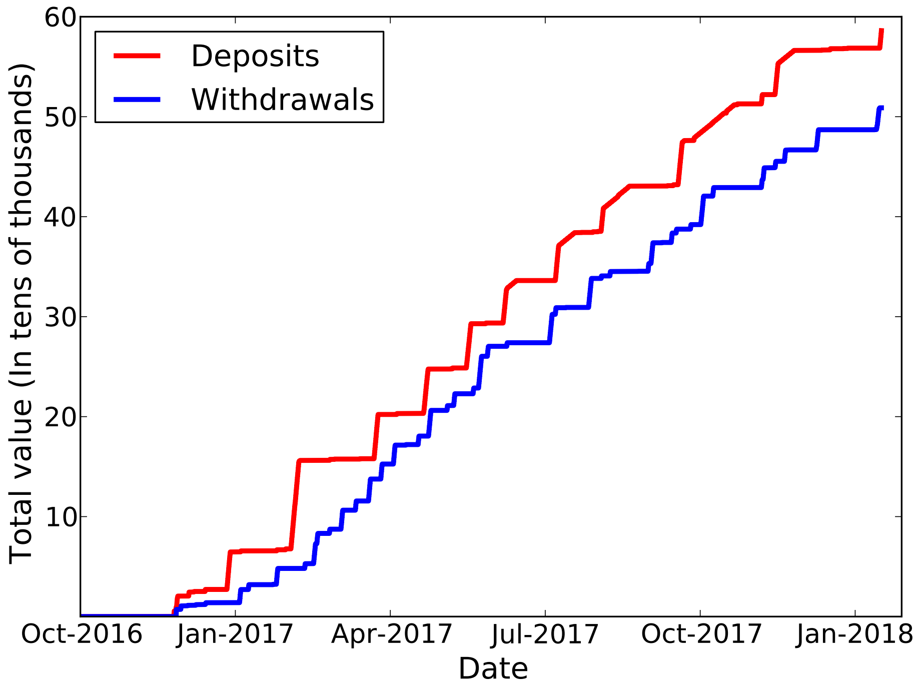


Figure 5.8: Over time, the founder deposits into the pool (in red) and withdrawals from the pool (in blue), after running Heuristic 3.

associated with miners.

5.5.2.1 Deposits

As discussed earlier and seen in Figure 5.5, it is easy to identify miner deposits into the pool due to the fact that they immediately follow a coin generation. Before going further, we split the category of miners into individual miners, who operate on their own, and mining pools, which represent collectives of potentially many individuals. In total, we gathered 19 t-addresses associated with Zcash mining pools, using the scraping methods described in Section 5.4.2. Table 5.4 lists these mining pools, as well as the number of addresses they control and the number of t-to-z transactions we associated with them. Figure 5.9 plots the value of their deposits into the shielded pool over time.

In this figure, we can clearly see that the two dominant mining pools are Flypool and F2Pool. Flypool consistently deposits the same (or similar) amounts, which we can see in their linear representation. F2Pool, on the other hand, has bursts of large deposits mixed with periods during which it is not very active, which

Name	Addresses	t-to-z	z-to-t
Flypool	3	65 631	3
F2Pool	1	742	720
Nanopool	2	8319	4107
Suprnova	1	13 361	0
Coinmine.pl	2	3211	0
Waterhole	1	1439	5
BitClub Pool	1	196	1516
MiningPoolHub	1	2625	0
Dwarfpool	1	2416	1
Slushpool	1	941	0
Coinotron	2	9726	0
Nicehash	1	216	0
MinerGate	1	13	0
Zecmine.pro	1	6	0

Table 5.4: A summary of our identified mining pool activity, in terms of the number of associated addresses used in coingen transactions, and the numbers of each type of transaction interacting with the pool.

we can also see reflected in the graph. Despite their different behaviors, the amount deposited between the two pools is similar.

5.5.2.2 Withdrawals

While the withdrawals from the pool do not solely re-use the small number of mining addresses identified using deposits (as we saw in our naïve attempt to link miner z-to-t transactions in Figure 5.7a), they do typically re-use some of them, so can frequently be identified anyway.

In particular, mining pool payouts in Zcash are similar to how many of them are in Bitcoin [15, 134]. The block reward is often paid into a single address, controlled by the operator of the pool, and the pool operator then deposits some set of aggregated block rewards into the shielded pool. They then pay the individual reward to each of the individual miners as a way of "sharing the pie," which results in z-to-t transactions with many outputs. (In Bitcoin, some pools opt for this approach while some form a "peeling chain" in which they pay each individual miner in a separate transaction, sending the change back to themselves each time.) In the payouts for some of the mining pools, the list of output t-addresses sometimes includes

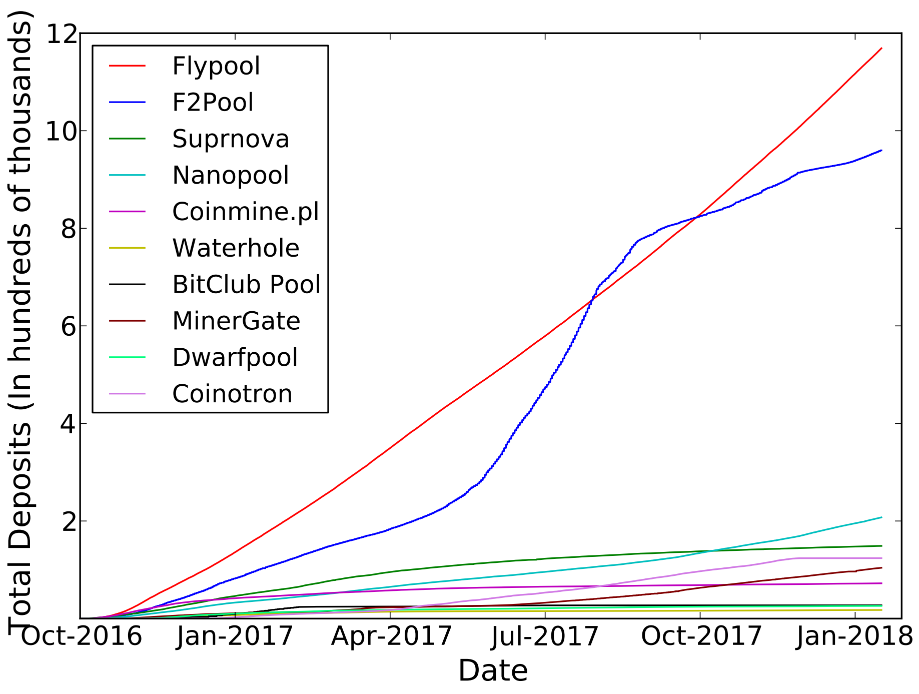


Figure 5.9: Over time, the value of deposits made by known mining pools into the shielded pool.

one of the t-addresses known to be associated with the mining pool already. We thus tag these types of payouts as belonging to the mining pool, according to the following heuristic:

Heuristic 4. If a z-to-t transaction has over 100 output t-addresses, one of which belongs to a known mining pool, then we label the transaction as a mining withdrawal (associated with that pool), and label all non-pool output t-addresses as belonging to miners.

As with Heuristic 3, short of asking the mining pool operators directly it is impossible to validate this heuristic. Nevertheless, given the known operating structure of Bitcoin mining pools and the way this closely mirrors that structure, we again believe it to be relatively safe.

As a result of running this heuristic, we tagged 110,918 addresses as belonging to miners, and linked a much more significant portion of the z-to-t transactions, as seen in Figure 5.7c. As the last column in Table 5.4 shows, however, this heuristic captured the activity of only a small number of the mining pools, and the large jump in linked activity is mostly due to the high coverage with F2Pool (one of the two richest pools). This implies that further heuristics developed specifically for other pools, such as Flypool, would increase the linkability even more. Furthermore, a more active strategy in which we mined with the pools to receive payouts would reveal their structure, at which point (according to the 1.1M deposited by Flypool shown in Figure 5.9 and the remaining value of 1.2M attributed to the 'other' category shown in Figure 5.7c) we would shrink the anonymity set even further.[4]

5.5.3 Other Entities

Once the miners and founders have been identified, we can assume the remaining transactions belong to more general entities. In this section we look into different means of categorizing these entities in order to identify how the shielded pool is being used.

In particular, we ran the heuristic due to Quesnelle [46], which said that if a unique value (i.e., a value never seen in the blockchain before or since) is deposited into the pool and then, after some short period of time, the exact same value is withdrawn from the pool, the deposit and the withdrawal are linked in what he calls a *round-trip transaction*.

Heuristic 5. [46] For a value v, if there exists exactly one t-to-z transaction carrying value v and one z-to-t transaction carrying value v, where the z-to-t transaction happened after the t-to-z one and within some small number of blocks, then these transactions are linked.

In terms of false positives, the fact that the value is unique in the blockchain means that the only possibility of a false positive is if some of the z-to-z transactions split or aggregated coins in such a way that another deposit (or several other deposits) of a different amount were altered within the pool to yield an amount identical to the initial deposit. While this is possible in theory, we observe that of the 12,841 unique values we identified, 9,487 of them had eight decimal places

[4]It is possible that we have already captured some of the Flypool activity, as many of the miners receive payouts from multiple pools. We thus are not claiming that all remaining activity could be attributed to Flypool, but potentially some substantial portion.

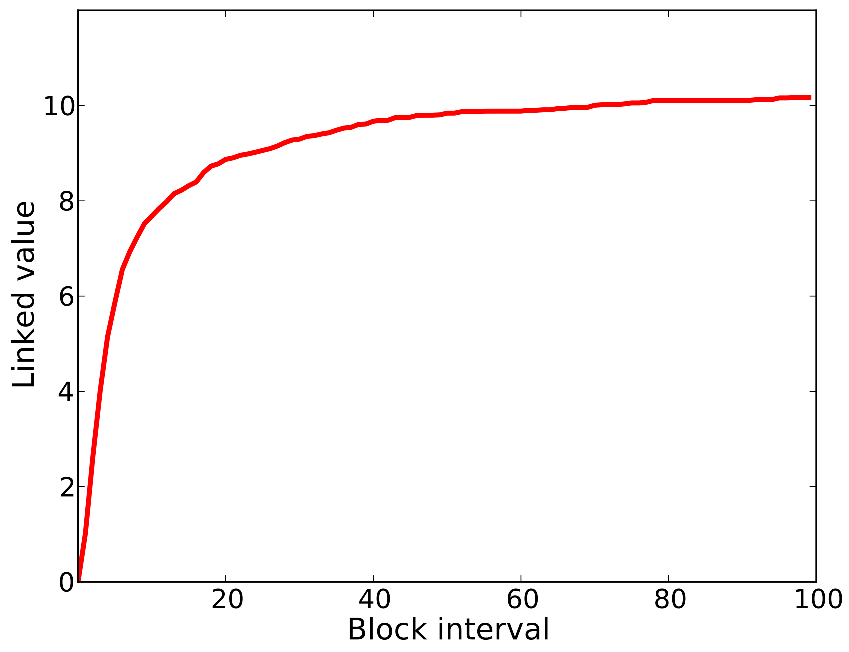


Figure 5.10: The value linked by Heuristic 5, as a function of the block interval required between the deposit and withdrawal transactions.

(the maximum number in Zcash), and 98.9% of them had more than three decimal places. We thus view it as highly unlikely that these exact values were achieved via manipulations in z-to-z transactions.

By running this heuristic, we identified 12,841 unique values, which means we linked 12,841 transactions. The values total 1,094,513.23684 ZEC and represent 28.5% of all coins ever deposited in the pool. Interestingly, most (87%) of the linked coins were in transactions attributed to the founders and miners, so had already been linked by our previous heuristics. We believe this lends further credence to their soundness. In terms of the block interval, we ran Heuristic 5 for every interval between 1 and 100 blocks; the results are in Figure 5.10.

As this figure shows, even if we assume a conservative block interval of 10 (meaning the withdrawal took place 25 minutes after the deposit), we still capture 70% of the total value, or over 700K ZEC. If we require the withdrawal to have taken place within an hour of the deposit, we get 83%.

5.6 Interactions within the Shielded Pool

In this section we consider private transactions; i.e., z-to-z transactions that interact solely with the shielded pool. As seen in Section 5.3.1, these transactions form a small percentage of the overall transactions. However, z-to-z transactions form a crucial part of the anonymity core of Zcash. In particular, they make it difficult to identify the round-trip transactions from Heuristic 5.

Our analysis identified 6,934 z-to-z transactions, with 8,444 vJoinSplits. As discussed in Section 2, the only information revealed by z-to-z transactions is the miner's fee, the time of the transaction, and the number of vJoinSplits used as input. Of these, we looked at the time of transactions and the number of vJoinSplits in order to gain some insight as to the use of these operations.

We found that 93% of z-to-z transactions took just one vJoinSplit as input. Since each vJoinSplit can have at most two shielded outputs as its input, the majority of z-to-z transactions thus take no more than two shielded outputs as their input. This increases the difficulty of categorizing z-to-z transactions, because we cannot know if a small number of users are making many transactions, or many users are making one transaction.

In looking at the timing of z-to-z transactions, however, we conclude that it is likely that a small number of users were making many transactions. Figure 5.11 plots the cumulative number of vJoinSplits over time. The occurrences of vJoinSplits are somewhat irregular, with 17% of all vJoinSplits occurring in January 2017. There are four other occasions when a sufficient number of vJoinSplits occur within a sufficiently short period of time as to be visibly noticeable. It seems likely that these occurrences belong to the same group of users, or at least by users interacting with the same service.

Finally, looking back at the number of t-to-z and z-to-t transactions identified with mining pools in Table 5.4, it is possible that BitClub Pool is responsible for up to 1,300 of the z-to-z transactions, as it had 196 deposits into the pool and 1,516 withdrawals. This can happen only because either (1) the pool made extra z-to-z transactions, or (2) it sent change from its z-to-t transactions back into the shielded

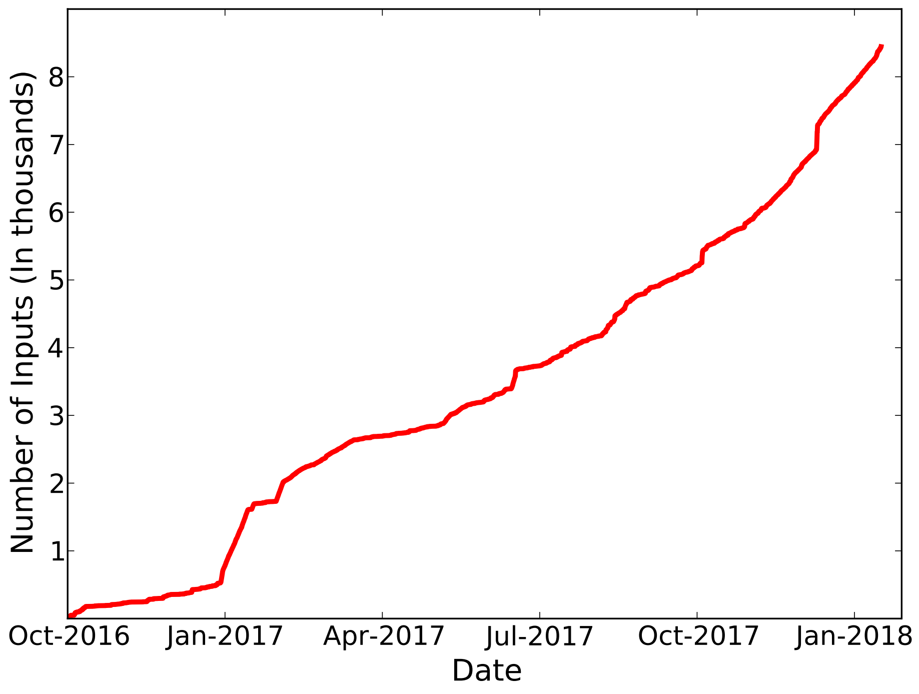


Figure 5.11: The number of z-to-z vJoinSplits over time.

pool. As most of BitClub Pool's z-to-t transactions had over 200 output t-addresses, however, we conclude that the former explanation is more likely.

5.7 Case Study: The Shadow Brokers

The Shadow Brokers (TSB) are a hacker collective that has been active since the summer of 2016, and that leaks tools supposedly created by the NSA. Some of these leaks are released as free samples, but many are sold via auctions and as monthly bundles. Initially, TSB accepted payment only using Bitcoin. Later, however, they began to accept Zcash for their monthly dump service. In this section we discuss how we identified t-to-z transactions that could represent payments to TSB. We identified twenty-four clusters (created using our analysis in Section 5.4) matching our criteria for potential TSB customers, one of which could be a regular customer.

5.7.1 Techniques

In order to identify the transactions that are most likely to be associated with TSB, we started by looking at their blog [129]. In May 2017, TSB announced that they would be accepting Zcash for their monthly dump service. Throughout the summer

May/June	July	August	September	October
100	200	500	100	500
	400		200	
			500	

Table 5.5: Amounts charged for TSB monthly dumps, in ZEC. In July and September TSB offered different prices depending on which exploits were being purchased.

(June through August) they accepted both Zcash and Monero, but in September they announced that they would accept only Zcash. Table 5.5 summarizes the amount they were requesting in each of these months. The last blog post was made in October 2017, when they stated that all subsequent dumps would cost 500 ZEC.

To identify potential TSB transactions, we thus looked at all t-to-z transactions not associated with miners or founders that deposited either 100, 200, 400, or 500 ZEC $\pm$ 5 ZEC. Our assumption was that users paying TSB were not likely to be regular Zcash users, but rather were using it with the main purpose of making the payment. On this basis, addresses making t-to-z transactions of the above values were flagged as a potential TSB customer if the following conditions held:

1. They did not get their funds from the pool; i.e., there were no z-to-t transactions with this address as an output. Again, if this were a user mainly engaging with Zcash as a way to pay TSB, they would need to buy their funds from an exchange, which engage only with t-addresses.

2. They were not a frequent user, in the sense that they had not made or received more than 250 transactions (ever).

3. In the larger cluster in which this address belonged, the total amount deposited by the entire cluster into the pool within one month was within 1 ZEC of the amounts requested by TSB. Here, because the resulting clusters were small enough to treat manually, we applied not only Heuristic 1 but also Heuristic 2 (clustering by change), making sure to weed out false positives. Again, the idea was that suspected TSB customers would not be frequent users of the pool.

As with our previous heuristics, there is no way to quantify the false-positive risks associated with this set of criteria, although we see below that many of the transactions matching it did occur in the time period associated with TSB acceptance of Zcash. Regardless, given this limitation we are not claiming that our results are definitive, but do believe this to be a realistic set of criteria that might be applied in the context of a law enforcement investigation attempting to narrow down potential suspects.

5.7.2 Results

Our results, in terms of the number of transactions matching our requirements above up until 17 January 2018, are summarized in Table 5.6. Before the first TSB blog post in May, we found only a single matching transaction. This is very likely a false positive, but demonstrates that the types of transactions we were seeking were not common before TSB went live with Zcash. After the blog post, we flagged five clusters in May and June for the requested amount of 100 ZEC. There were only two clusters that was flagged for 500 ZEC, one of which was from August. No transactions of any of the required quantities were flagged in September, despite the fact that TSB switched to accepting only Zcash in September. This is possible for a number of reasons: our criteria may have caused us to miss transactions, or maybe there were no takers. From October onwards we flagged between 1-6 transactions per month. It is hard to know if these represent users paying for old data dumps or are simply false positives.

Four out of the 24 transactions in Table 5.6 are highly likely to be false positives. First, there is the deposit of 100 ZEC into the pool in January, before TSB announced their first blog post. This cluster put an additional 252 ZEC into the pool in March, so is likely just some user of the pool. Second and third, there are two deposits of 200 ZEC into the pool in June, before TSB announced that one of the July dump prices would cost 200 ZEC. Finally, there is a deposit of 400 ZEC into the pool in June before TSB announced that one of the July dump prices would cost 400 ZEC.

Of the remaining clusters, there is one whose activity is worth discussing.

Month	100	200	400	500
October (2016)	0	0	0	0
November	0	0	0	0
December	0	0	0	0
January (2017)	1	0	0	0
February	0	0	0	0
March	0	0	0	0
April	0	0	0	0
May (before)	0	0	0	0
May (after)	3	1	0	0
June	2	1	1	0
July	1	2	0	0
August	1	0	0	1
September	0	0	0	0
October	2	0	0	0
November	1	0	0	0
December	2	3	0	1
January (2018)	0	1	0	0

Table 5.6: Number of clusters that put the required amounts (± 1 ZEC) into the shielded pool.

From this cluster, there was one deposit into the pool in June for 100 ZEC, one in July for 200 ZEC, and one in August for 500 ZEC, matching TSB prices exactly. The cluster belonged to a new user, and most of the money in this user's cluster came directly from Bitfinex (Cluster 3).

5.8 Conclusions

This paper has provided the first in-depth exploration of Zcash, with a particular focus on its anonymity guarantees. To achieve this, we applied both well-known clustering heuristics that have been developed for Bitcoin and attribution heuristics we developed ourselves that take into account Zcash's shielded pool and its unique cast of characters. As with previous empirical analyses of other cryptocurrencies, our study has shown that most users are not taking advantage of the main privacy feature of Zcash at all. Furthermore, the participants who do engage with the shielded pool do so in a way that is identifiable, which has the effect of significantly eroding the anonymity of other users by shrinking the overall anonymity set.

Future work: Our study was an initial exploration, and thus left many avenues open for further exploration. For example, it may be possible to classify more z-to-z transactions by analyzing the time intervals between the transactions in more detail, or by examining other metadata such as the miner's fee or even the size (in bytes) of the transaction. Additionally, the behavior of mining pools could be further identified by a study that actively interacts with them.

Suggestions for improvement: Our heuristics would have been significantly less effective if the founders interacting with the pool behaved in a less regular fashion. In particular, by always withdrawing the same amount in the same time intervals, it became possible to distinguish founders withdrawing funds from other users. Given that the founders are both highly invested in the currency and knowledgeable about how to use it in a secure fashion, they are in the best place to ensure the anonymity set is large.

Ultimately, the only way for Zcash to truly ensure the size of its anonymity set is to require all transactions to take place within the shielded pool, or otherwise significantly expand the usage of it. This may soon be computationally feasible given emerging advances in the underlying cryptographic techniques [135], or even if more mainstream wallet providers like Jaxx roll out support for z-addresses. More broadly, we view it as an interesting regulatory question whether or not mainstream exchanges would continue to transact with Zcash if it switched to supporting only z-addresses.

Chapter 6

Analyzing the privacy in the Lightning Network

In this chapter, we present our paper, *"An Empirical analysis of Anonymity in the Lightning Network"*, which looked into the privacy properties of the Bitcoin layer 2 solution, the Lightning Network.

Payment channel networks, and the Lightning Network in particular, seem to offer a solution to the lack of scalability and privacy offered by Bitcoin and other blockchain-based cryptocurrencies. Previous research has focused on the scalability, availability, and crypto-economics of the Lightning Network, but relatively little attention has been paid to exploring the level of privacy it achieves in practice. This paper presents a thorough analysis of the privacy offered by the Lightning Network, by presenting several attacks that exploit publicly available information about the network in order to learn information that is designed to be kept secret, such as how many coins a node has available or who the sender and recipient are in a payment routed through the network.

6.1 Introduction

Since its introduction in 2008, Bitcoin [13] has become the most widely adopted cryptocurrency. The decentralized and permissionless nature of Bitcoin allows all users to join the network and avoids the need for intermediaries and authorities who control the flow of money between them. Instead, the validity of each transaction is verified by a consensus decision made by the network participants themselves; valid transactions are then recorded in the public blockchain. The blockchain thus acts as a ledger of all transactions that have ever taken place.

The need to broadcast transactions to all peers in the network and store them in a permanent ledger, however, presents two problems for the longevity of blockchain-based cryptocurrencies. First, it imposes severe scalability limitations: the Bitcoin blockchain today is over 300 GB, and Bitcoin can achieve a throughput of only ten transactions per second. Other cryptocurrencies achieve somewhat higher throughputs, but there is an inherent tradeoff in these broadcast-based systems between throughput and security [136, 137]. Second, the transparent nature of the ledger means anyone can observe the flow of coins, identify the counterparties to a transaction, and link different transactions. This has been shown most decisively for Bitcoin [4, 3, 14, 2, 19], but this type of analysis extends even to cryptocurrencies that were explicitly designed with privacy in mind [54, 138, 120, 53, 139, 46, 34].

The most promising solutions that have been deployed today to address the issue of scalability are so-called "layer-two" protocols [10], with the Lightning Network (LN) [11] emerging as the most popular one since its launch in March 2018. In Lightning, pairs of participants use the Bitcoin blockchain to open and close *payment channels* between themselves. Within a channel, these two users can make arbitrarily many *off-chain* payments between themselves, without having to use the blockchain. Beyond a single channel, Lightning supports multi-hop payment routing, meaning even participants who are not connected directly can still route payments through a broader *payment channel network* (PCN). Nodes in the network are incentivized to route payments by a fee they can charge for payments they forward.

In addition to the promise it shows in improving scalability, Lightning also seems to address the issue of privacy. As we elaborate on in Section 2.3, the nodes in the network and most of the channels in the network are publicly known in order to build up the PCN (although some channels may be kept private), as is the *capacity* of a given channel, meaning the maximum payment value that can be routed through it. The individual *balances* associated with the channel, however, are kept secret. Furthermore, payments are not broadcast to all peers and are not stored in a public ledger. Even if a payment passes through multiple channels, onion routing is used to ensure that each node on the path can identify only its immediate predecessor and successor.

As is the case with ledger-based cryptocurrencies, however, the gap in Lightning between the potential for privacy and the reality is significant, as we show in this work. In particular, we consider four main privacy properties promised by LN [70, 140]:

Private channels should allow two nodes to share a channel but keep its existence, along with all of its information (capacity, participants, etc.), hidden from the rest of the network. We explore this property in Section 6.2.2 by presenting a heuristic that identifies on-chain funding of private channels and one or even both of the participants.

Third-party balance secrecy says that although the capacity of the channel is public, the respective balances of the participants should remain secret. We explore this property in Section 6.3 by presenting and evaluating a generic method by which an active attacker (i.e., one opening channels with nodes in the network) can discover channel balances.

On-path relationship anonymity says that intermediate nodes routing the payment should not learn which other nodes, besides their immediate predecessor or successor, are part of the payment's route. We explore this property in Section 6.4, where we leverage an LN simulator we developed (described in Section 6.4.1) to evaluate the ability of an intermediate node to infer the sender and recipient in payments that it routes.

Off-path payment privacy says that any node not involved in routing a payment should not infer any information regarding the routing nodes or the payment value. We explore this property in Section 6.5 by presenting and evaluating a method by which an active attacker can use the ability to discover balances to form *net-*

work snapshots. By comparing consecutive network snapshots, the attacker can infer payments by identifying where and by how much the balances of channels changed.

6.1.1 Ethical considerations

The attacks presented in Sections 6.4 and 6.5 are evaluated on a simulated network rather than the live one, but our attack in Section 6.3 is evaluated on the live test network. As in related active attacks on Bitcoin [27, 28, 141], we made every effort to ensure that our attacks did not interfere with the normal functioning of the network: the messages sent during the attack have no abnormal effect and do not cost any money to process, and their volume is relatively modest (we sent at most 24 messages per node we attacked). We thus believe that they did not have any long- or short-term destructive effect on the nodes that processed them. We disclosed the results of this paper to the developers of the three main LN clients and the liquidity provider Bitrefill in February 2020, and have discussed the paper with the Lightning developers since then.

6.2 Blockchain Analysis

6.2.1 Data and measurements

The Lightning network can be captured over time by periodic snapshots of the public network graph, which provide ground-truth data about nodes (identifiers, network addresses, status, etc.) and their channels (identifiers, capacity, endpoints, etc.). To obtain a comprehensive set of snapshots, we used data provided to us by (1) our own lnd client, (2) one of the main c-lightning developers, and (3) scraped user-submitted (and validated) data from 1ML[1] and LN Bigsun.[2] To analyze on-chain transactions, we also ran a full Bitcoin node, using the BlockSci tool [47] to parse and analyze the raw blockchain data.

Our LN dataset included the hash of the Bitcoin transaction used to open each channel. By combining this with our blockchain data, we were thus able to

[1] https://1ml.com/
[2] https://ln.bigsun.xyz/

identify when channels closed and how their funds were distributed. In total, we identified 174,378 channels, of which 135,850 had closed with a total capacity of 3315.18 BTC. Of the channels that closed, 69.22% were claimed by a single output address (i.e., the channel was completely unbalanced at the time of closure), 29.01% by two output addresses, and 1.76% by more than two outputs.

6.2.2 Private channels

Private channels provide a way for two Lightning nodes to create a channel but not announce it to the rest of the network. In this section, we seek to understand the extent to which private channels can nevertheless be identified, to understand their privacy limitations as well as the scope of our remaining attacks on the public network.

We first provide an upper bound on the number of private channels using a *property heuristic*, which identifies Bitcoin transactions that seem to represent the opening and closing of channels but for which we have no public channel identifier.

6.2.2.1 Property Heuristic

To align with our LN dataset, we first looked for all Bitcoin transactions that (1) occurred after January 12, 2018 and (2) before September 7, 2020, and (3) where one of the outputs was a P2WSH address (which LN channels have to be, according to the specification). We identified 3,500,312 transactions meeting these criteria, as compared with the 174,378 public channels opened during this period. We then identified several common features of the known opening transactions identified from our dataset: (i) 99.91% had at most two outputs, which likely represents the funder creating the channel and sending themselves change; (ii) 99.91% had a single P2WSH output address; (iii) 99.85% had a P2WSH output address that received at most 16,777,215 satoshis, which at the time of our analysis is the maximum capacity of an LN channel; (iv) 99.99% had a P2WSH output that appeared at most once as both an input and output, which reflects its "one-time" usage as a payment channel and not as a reusable script; and (v) 99.99% were funded with either a WitnessPubKeyHash address or ScriptHash address.

By requiring our collected transactions to also have these features and excluding any transactions involved in opening or closing public channels, we were left with 267,674 potential transactions representing the opening of private channels. If the outputs in these transactions had spent their contents (i.e., the channel had been closed), then we were further able to see how they did so, which would provide better evidence of whether or not they were associated with the Lightning Network. Again, we identified the following features based on known closing transactions we had from our network data: (i) 100% had a non-zero sequence number, as required by the Lightning specification [70]; (ii) 100% had a single input that was a 2-of-2 multisig address, again as required by the Lightning design; and (iii) 98.24% had at most two outputs, which reflects the two participants in the channel.

By requiring our collected opening transactions to also have a closing transaction with these three features, we were left with 77,245 pairs of transactions that were potentially involved in opening and closing private channels. Again, this is just an upper bound, since there are other reasons to use 2-of-2 multisigs in this way that have nothing to do with Lightning.

We identified 77,245 pairs of transactions that were potentially involved in opening and closing private channels, but likely has a high false positive rate. We thus developed a *tracing heuristic*, which follows the "peeling chain" [2] initiated at the opening and closing of public channels to identify any associated private channels.

6.2.2.2 Tracing heuristic

We next look not just at the properties of individual transactions, but also at the flow of bitcoins between transactions. In particular, we observed that it was common for users opening channels to do so in a "peeling chain" pattern [2]. This meant they would (1) use the change in a channel opening transaction to continue to create channels and (2) use the outputs in a channel closing transaction to open new channels. Furthermore, they would often (3) co-spend change with closing outputs; i.e., create a channel opening transaction in which the input addresses were the change from a previous opening transaction and the output from a previous closing one.

By systematically identifying these operations, we were able to link together channels that were opened or closed by the same Lightning node by following the peeling chain both forwards and backwards. Going backwards, we followed each input until we hit a transaction that did not seem to represent a channel opening or closure, according to our property heuristic. Going forwards, we identified the change address in a transaction, again using the property heuristic to identify the channel creation address and thus isolate the change address as the other output, and continued until we hit one of the following: (1) a transaction with no change output or one that was unspent, meaning we could not move forwards, or (2) a transaction that did not satisfy the property heuristic. We also did this for all of the outputs in a known channel closing transaction, to reflect the second pattern identified above.

We started with the 174,378 public channels identified in our LN dataset. By applying our tracing heuristic, we ended up with 27,386 additional channel opening transactions. Of these, there were 27,183 that fell within the same range of blocks as the transactions identified by our property heuristic.

Using the tracing heuristic, however, not only identified private channels but also allowed us to cluster together different channels (both public and private), according to the shared ownership of transactions within a peeling chain [2]. To this end, we first clustered together different channels according to their presence in the same peeling chain, and then looked at the public channels within each cluster and calculated the common participant, if any, across their endpoints. If there was a single common participant, then we could confidently tag them as the node responsible for opening all of these channels.

In order to find the other endpoint of each private channel, we followed the closing outputs of the channel's closing transaction, whenever applicable, leveraging the second and third observed patterns in the tracing heuristic. In particular, when a closing output was spent in order to open a new channel, we performed the same clustering operation as earlier. We failed to identify the second participant in each channel only when the channel was still open, the channel was closed but the closing output was still unspent, or the closing output was used for something other than Lightning.

Out of the 27,183 transactions we identified as representing the opening of private channels, we were able to identify both participants in 2,035 (7.5%), one participant in 21,557 (79.3%), and no participants in 3,591 (13.2%). Our identification method applies equally well, however, to public channels. We were able to identify the opening participant for 155,202 (89.0%) public channels. Similarly, for the public channels that were already closed, which represent 185,860 closing outputs, we were able to associate 143,577 (77.25%) closing outputs with a specific participant.

6.3 Balance Discovery

Previous attacks designed to discover the balances associated with individual channels (as opposed to just their capacity) [88, 89, 90] exploited debug information as an *oracle*. In these attacks, an attacker opens a channel with a node and routes a fake payment hash, with some associated amount amt, through its other channels. Based on the error messages received and performing a binary search on amt (i.e., increasing amt if the payment went through and decreasing it if it failed), the attacker efficiently determines the exact balance of one side of the channel. In this section we perform a new *generic* attack on the LN testnet. As compared to previous attacks, our attacker must run two nodes rather than one. If error messages are removed or made generic in the future, however, our attack would continue to work whereas previous attacks would not.

6.3.1 The attack

In our attack, an attacker running nodes A and D needs to form a path $A \rightarrow B \rightarrow C \rightarrow D$, with the goal of finding the balance of the channel $B \rightarrow C$. This means our attacker needs to run two nodes, one with a channel with outgoing balance (A), and one with a channel with incoming balance (D). Creating the channel $A \rightarrow B$ is easy, as the attacker can just open a channel with B and fund it themselves. Opening the channel $C \rightarrow D$ is harder though, given that the attacker must create incoming balance.

Today, there are two main options for doing this. First, the attacker can open the channel $C \to D$ and fund it themselves, but assign the balance to C rather than to D (this is called funding the "remote balance"). This presents the risk, however, that C will immediately close the channel and take all of its funds. We call this approach unassisted channel opening. The second option is to use a *liquidity provider* (e.g., Bitrefill[3] or LNBIG[4]), which is a service that sells channels with incoming balance.[5] We call this assisted channel opening.

Once the attacker has created the channels $A \to B$ and $C \to D$, they route a random payment hash H to D, via B and C, with some associated amount $\mathtt{amt}$. If D receives H, this means the channel from B to C had sufficient balance to route a payment of amount $\mathtt{amt}$. If D did not receive H after some timeout, the attacker can assume the payment failed, meaning $\mathtt{amt}$ exceeded the balance from B to C. Either way, the attacker can (as in previous attacks) repeat the process using a binary search on $\mathtt{amt}$. Eventually, the attacker discovers the balance of the channel as the maximum value for which D successfully receives H.

To a certain extent, this attack generalizes even to the case in which there is more than one intermediate channel between the two attacker nodes. In this more general case, however, the above method identifies the bottleneck balance in the entire path, rather than the balance of an individual channel. In the event of a payment failure though, the current C-lightning and LND clients return an *error index*, which is the position of the node at which the payment failed. This means that an attacker would know exactly where a payment failed along a longer path. We chose not to use this index when implementing our attack, in order to keep it fully generic and to test just the basic version, but leave an attack that does use this index as interesting future research.

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

[4] http://lnbig.com/

[5] Bitrefill, for example, sells a channel with an incoming balance of 5000000 satoshis (the equivalent at the time of writing of 493.50 USD) for 8.48 USD.

6.3.2 Attack results

We performed this attack on testnet on September 3 2020. We ran two LN nodes and funded all our channels (unassisted), both locally and remotely, which required a slight modification of the client (as fully funding a remote channel is restricted by default). We opened channels with every accessible node in the network. At the time of the attack there were 3,159 nodes and 9,136 channels, of which we were able to connect to 103 nodes and attack 1,017 channels. We were not able to connect to a majority of the overall nodes, which happened for a variety of reasons: some nodes did not publish an IPv4 address, some were not online, some had their advertised LN ports closed, and some refused to open a channel.

Of these 1,017 channels, we determined the balance of 568. Many (65%) of the channels were fairly one-sided, meaning the balance of the attacked party was 70% or more of the total capacity. We received a variety of errors for the channels where we were unsuccessful, such as *TemporaryChannelFailure*, or we timed out as the client took more than 30 seconds to return a response.

We did not carry out the attack on mainnet due to cost and ethical considerations, but believe it likely that the attack would perform better there. This is because there is no cost for forgetting to close open channels on testnet or maintain a node, whereas on mainnet a user is incentivized by an opportunity cost (from fees) to ensure a node is maintained and its channels are active.

6.3.3 Attacker cost

In our experiment we used testnet coins, which are of essentially no value, so the monetary cost for us to perform this attack was negligible. To understand the practical limitations of this attack, however, we estimate the minimum cost on mainnet. When creating the outgoing channel $A \rightarrow B$, the attacker must pay for the opening and closing transaction fees on the Bitcoin blockchain. At the time of our attack, this was 0.00043 BTC per transaction. They must also remotely fund the recipient node with enough *reserve satoshis* to allow the forwarding of high payments, which at present are 1% of the channel capacity. To create the incoming channel $C \rightarrow D$, the attacker can use liquidity providers like Bitrefill, who at the time of writing allow users to buy channels with 0.16 BTC incoming capacity for 0.002604 BTC.

Purchasing the cheapest incoming liquidity available today would cost the attacker 0.00086 BTC and 0.005 BTC on hold, enabling routes to 4,811 channels (with a total capacity of 45 BTC). This would require opening 2,191 channels with a maximum channel capacity of 0.04998769 BTC. In total, this would require the attacker to spend 1.097 BTC and put 109.53 BTC on hold.

6.4 Path Discovery

We now describe how an *honest-but-curious* intermediate node involved in routing the payment can infer information regarding its path, and in particular can identify the sender and recipient of the payment. Our strategy is similar to a passive variant of a predecessor attack [142] proposed against the Crowds [143] anonymous communication network. Our strategy can be further extended by analyzing the sparse network connectivity and limited number of potential paths due to channel capacity.

In contrast to previous work [77], we consider not only single-hop routes but also routes with multiple intermediate nodes. The only assumption we make about the adversary's intermediate node is that it keeps its channel balanced, which can be easily done in practice.

We define $\Pr_S$ and $\Pr_R$ as the probability that the adversary successfully discovers, respectively, the sender and recipient in a payment. Following our notation, Béres et al. claim, based on their own simulated results, that $\Pr_S = \Pr_R$ ranges from 0.17 to 0.37 depending on parameters used in their simulation. We show that this probability is actually a lower bound, as it does not take into account multiple possible path lengths or the chance that a payment fails (their simulation assumes that all payments succeed on the first try).

The strategy of our honest-but-curious adversary is simple: they always guess that their immediate predecessor is the sender. In other words, if we define H as the adversary's position along the path, they always assume that $H = 1$. Similarly, they always guess that their immediate successor is the recipient. We focus on the probability of successfully guessing the sender; the probability of succesfully guessing the recipient can be computed in an analogous way.

Successful payments: We start by analyzing the success probability of this adversary in the case of a successful payment, which we denote as $\Pr_S^{\mathsf{succ}}$. We define as $\Pr[L=\ell]$ the probability of a path being of length ℓ, and as $\Pr[H=h \mid L=\ell]$ the probability that the adversary's node is at position h given that the path length is ℓ. According to the Lightning specification [70], the maximum path length is 20. By following the strategy defined above, we have that

$$\begin{aligned}\Pr_S^{\mathsf{succ}} &= \sum_{n=3}^{20} \Pr[L=\ell \mid \mathsf{succ}] \cdot \Pr[H=1 \mid L=\ell, \mathsf{succ}] \\ &= \Pr[L=3 \mid \mathsf{succ}] \\ &\quad + \sum_{n=4}^{20} \Pr[L=\ell \mid \mathsf{succ}] \cdot \Pr[H=1 \mid L=\ell, \mathsf{succ}]\end{aligned}$$

since $\Pr[H=1 \mid L=3, \mathsf{succ}] = 1$ given that the adversary is the only intermediate node in this case. Hence, $\Pr[L=3 \mid \mathsf{succ}]$ is a lower bound on $\Pr_S$.

To consider the overall probability, we focus on the conditional probabilities $\Pr[H=1 \mid L=\ell, \mathsf{succ}]$. If all nodes form a clique,[6] then it would be almost equally probable for any node to be in any hop position $H=h$. (The only reason the distribution is not entirely uniform is that some channels may be chosen more often than others, depending on the relative fees they charge, but an adversary could choose fees to match its neighbors as closely as possible.) In this case then, the probability that $H=1$ is just $1/(\ell-2)$.

Failed payments: Similarly, in case the payment fails, we define the probability $\Pr_S^{\mathsf{fail}}$ as

$$\Pr_S^{\mathsf{fail}} = \sum_{\ell=3}^{20} \Pr[L=\ell \mid \mathsf{fail}] \cdot \Pr[H=1 \mid L=\ell, \mathsf{fail}].$$

[6]This would rather be a clique excluding a link between the sender and recipient, since otherwise they would presumably use their channel directly.

This is the same formula as for $\Pr_S^{\mathsf{succ}}$ so far, but we know that $\Pr[L = 3 \mid \mathsf{fail}] = 0$, since if the adversary is the only intermediate node the payment cannot fail. Furthermore, the conditional probability $\Pr[H = 1 \mid L = \ell, \mathsf{fail}]$ is different from the probability $\Pr[H = 1 \mid L = \ell, \mathsf{succ}]$, as the fact that a payment failed reveals information to the adversary about their role as an intermediate node. In particular, if an intermediate node successfully forwards the payment to their successor but the payment eventually fails, the node learns that their immediate successor was not the recipient and thus that the failed path was of length $L \geq 4$ and their position is not $L-1$. This means that $\Pr[L = \ell \mid \mathsf{fail}]$ becomes $\Pr[L = \ell \mid \mathsf{fail}, \ell \geq 4]$. We thus get

$$\begin{aligned}\Pr_S^{\mathsf{fail}} &= \Pr[L = 4 \mid \mathsf{fail}, \ell \geq 4] \\ &\quad + \sum_{\ell=5}^{20} \Pr[L = \ell \mid \mathsf{fail}] \cdot \Pr[H = 1 \mid L = \ell, \mathsf{fail}].\end{aligned}$$

This gives $\Pr[L = 4 \mid \mathsf{fail}, \ell \geq 4]$ as a lower bound in the case of a failed payment. As we did in the case of successful payments, we assume a clique topology as the best case for this adversary's strategy, in which their chance of guessing their position is $1/(\ell-3)$ (since they know they are not the last position). We thus obtain

$$\Pr_S^{\mathsf{fail}} = \Pr[L = 4 \mid \mathsf{fail}, \ell \geq 4] + \sum_{\ell=5}^{20} \Pr[L = \ell \mid \mathsf{fail}] \cdot \frac{1}{\ell-3}.$$

6.4.1 Lightning Network simulator

In order to investigate the success of this on-path adversary, we need measurements that it would require significant resources to obtain from the live network, such as the average path length for a payment. Given the financial and ethical concerns this would raise, we make the same decision as in previous work [77, 73, 91, 92, 93] to develop a Lightning network simulator to perform our analysis. We implemented our simulator in 2,624 lines of Python 3 and will release it as open-source software.

Network topology: As mentioned in Section 2.3, we represent the network as a graph $G = (V, E)$. We obtain the information regarding V and E from the snapshots we collected, as described in Section 6.2.1, which also include additional information such as capacities and fees. The network topology view of our simulator is thus an exact representation of the actual public network.

Geolocation: Nodes may publish an IPv4, IPv6 or .onion address, or some combination of these. If a node advertised an IPv4 or IPv6 address then we used it to assign this node a corresponding geolocation. This enabled us to accurately simulate TCP delays on the packets routed between nodes, based on their distance and following previous studies [136] in using the global IP latency from Verizon.[7] For nodes that published only a .onion, we assign delays according to the statistics published by Tor metrics, given the higher latency associated with the Tor network.[8]

Path selection: As discussed in Section 2.3, the route to the destination in LN is constructed solely by the payment sender. All clients generally aim to find the shortest path in the network, meaning the path with the lowest amount of fees. As shown by Tochner et al. [82], however, both the routing algorithm and the fee calculation differ across the three main choices of client software: lnd, c-lightning, and eclair. We could not easily extract or isolate the routing algorithms from these different implementations, so chose to implement all three versions of the path finding algorithm ourselves. We did this using Yen's *k*-shortest path algorithm [144] and the networkx Dijkstra's SPF algorithm.[9]

Software versions: Our collected snapshots did not include information about software versions, so we scraped the Owner Info field for each node listed on the 1ML website. Although in 91% of the cases this field is empty, the results allow us to at least estimate the distribution of the client software. We obtained information about 370 nodes and found that 292 were lnd, 54 were c-lightning, and 24 were eclair. We randomly assign software versions to the remaining nodes in the network according to this distribution, and then modify the weight function in the path finding algorithm according to the software version.

[7] https://enterprise.verizon.com/terms/latency/

[8] https://metrics.torproject.org/onionperf-latencies.html

[9] https://networkx.github.io/documentation/stable/reference/algorithms/shortest_paths.html

6.4.1.1 Payment parameters

Our first parameter, $\mathsf{t_{pay}}$, represents the total daily number of payments happening in LN. For this, we use an estimate from LNBIG [145], the largest node that holds more than 40% of the network's total capacity at the time of writing. According to LNBIG, the total number of routed transactions going through the network is 1000-1500 per day, but this does not take into account the payments performed via direct channels. Given this estimation, we use two values for $\mathsf{t_{pay}}$: 1000, representing a slight underestimate of today's volume, and 10,000, representing a potential estimate for the future of LN.

We also define as endpoints the parameter that determines the sender and the recipient of a payment. We define two values for this parameter: *uniform*, which means that the payment participants are chosen uniformly at random, and *weighted*, which means the participants are chosen randomly according to a weighted distribution that takes into account their number of direct channels (i.e., their degree). Similarly, we use values to determine the values of payments. When values is *cheap*, the payment value is the smallest value the sender can perform, given its current balances. When values is *expensive*, the payment value is the biggest value the sender can send.

6.4.2 Simulation results

Given the parameters $\mathsf{t_{pay}}$, endpoints, and values, we ran two simulation instances, with the goal of finding the worst-case and best-case scenarios for the on-path adversary. Based on the respective probabilities for $\Pr_S^{\mathsf{succ}}$ and $\Pr_S^{\mathsf{fail}}$, we can see that the worst case is when the path is long and the payment is likely to succeed, while the best case is when the path is short and the payment is likely to fail. Since the total volume $\mathsf{t_{pay}}$ does not affect the path length, we use $\mathsf{t_{pay}} = 1000$ for both instances. Each simulation instance was run using the network and node parameters scraped on September 1, 2020.

In our first simulation, $\mathsf{lengths_{long}}$, our goal was to capture the adversary's worst case. This meant we chose endpoints = *uniform*, so that the choice of sender and receiver was not biased by connectivity, and thus paths were not short due to

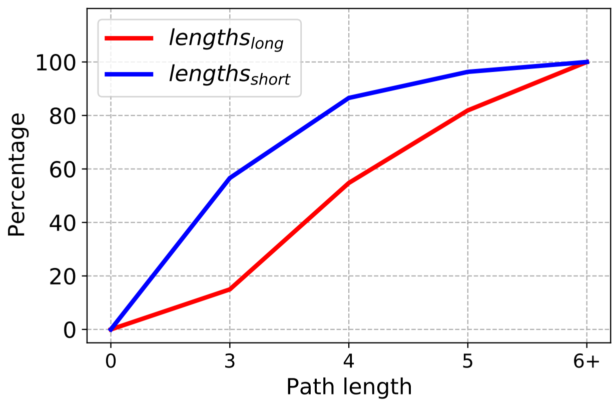


Figure 6.1: Average path length.

their potentially high connectivity. Similarly, we chose values $=$ *cheap* to minimize the probability of having a payment fail. For our second simulation, $\text{lengths}_{\text{short}}$, our goal was to capture the adversary's best case, so we chose endpoints $=$ *weighted* to ensure highly connected nodes were picked more often and thus paths were shorter. We also chose values $=$ *expensive*, leading to many balance failures.

As shown in Figure 6.1, even when we attempted to maximize the path length in $\text{lengths}_{\text{long}}$, 14.98% of paths still consist of only one hop. In $\text{lengths}_{\text{short}}$, 56.65% of paths consisted of a single hop. This interval agrees with recent research, which argues that 17-37% of paths have only one intermediate node [77]. The main reason the paths are short even in $\text{lengths}_{\text{long}}$ is that the network topology and the client path finding algorithm have a much larger effect on the path length than endpoints or values.

Beyond the results in Figure 6.1, running our simulator enabled us to estimate the probabilities $\Pr[L = \ell]$ for $3 \leq \ell \leq 20$ for both the best- and worst-case scenario for the adversary. We now use those results to compute the probabilities $\Pr_S^{\text{succ}}$ and $\Pr_S^{\text{fail}}$ for the case where our adversary is succesful only when it is impossible to be wrong (LowerBound) as well as in the case of a clique topology (clique), as shown in Figure 6.2. Here the clique topology is the worst possible topology for the adversary, since a less complete topology would allow the adversary to rule out nodes that cannot be involved in the payment and thus increase their confidence.

$\Pr_S^{\text{succ}}$ is bounded from below when $L = 3$, since in that case the adversary can

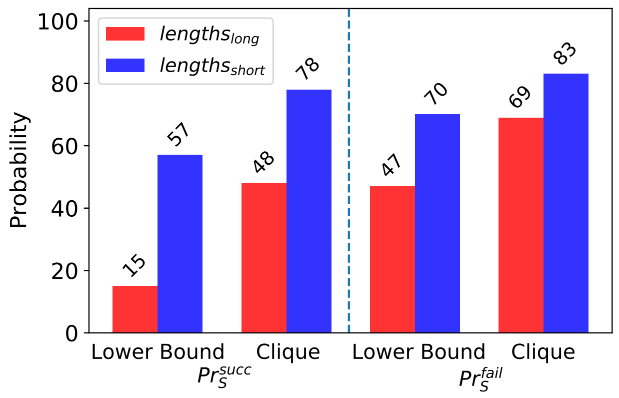


Figure 6.2: Probability of correctly identifying the sender given successful and failed payment. For detailed simulation settings of $\mathsf{lengths}_{\mathsf{long}}$ and $\mathsf{lengths}_{\mathsf{short}}$ see Section 6.4.2

never be wrong. Similarly, $\Pr_S^{\mathsf{fail}}$ is bounded from below when $L = 4$. In the case of a successful payment, the lower bound on $\Pr_S^{\mathsf{succ}}$ ranges from 15% ($\mathsf{lengths}_{\mathsf{long}}$) to 57% ($\mathsf{lengths}_{\mathsf{short}}$). On the other hand, the lower bound of $\Pr_S^{\mathsf{fail}}$ increases with the percentage of unsuccessful attempts, up to 83% ($\mathsf{lengths}_{\mathsf{short}}$), which is significantly higher than any previously recorded experiment. This is also not just a theoretical result: according to recent measurements, 34% of payments fail on the first try [146].

Our measurements show that even an adversary following an extremely simple strategy can have a high probability of inferring the sender of a payment routed through their node, especially in the case in which the payment fails. This is likely due to LN's highly centralized topology, which means paths are short and often involve the same intermediate nodes, as well as the fact that clients are designed to find the cheapest—and thus shortest—paths. Without changes in the client or the network topology, it is thus likely that intermediate nodes will continue to be able to violate on-path relationship anonymity.

6.5 Payment Discovery

In this section, we analyze the off-path payment privacy in Lightning, in terms of the ability of an attacker to learn information about payments it did not participate in routing.

Informally, our attack works as follows: using the balance discovery attack described in Section 6.3, the attacker constructs a *network snapshot* at time t consisting of all channels and their associated balances. It then runs the attack again at some later time $t+\tau$ and uses the differences between the two snapshots to infer information about payments that took place by looking at any paths that changed. In the simplest case that only a single payment took place between t and $t+\tau$ (and assuming all fees are zero), the attacker can see a single path in which the balances changed by some amount amt and thus learn everything about this payment: the sender, the recipient, and the amount amt. More generally, two payments might overlap in the paths they use, so an attacker would need to heuristically identify such overlap and separate the payments accordingly.

6.5.1 Payment discovery algorithm

We define τ to be the interval in which an attacker is able to capture two snapshots, S_t and $S_{t+\tau}$, and let $G_{\mathsf{diff}} = S_{t+\tau} - S_t$ be the difference in balance for each channel. Our goal is then to decompose G_{diff} into paths representing distinct payments. More specifically, we construct paths such that (1) each edge on the path has the same amount (plus fees), (2) the union of all paths results in the entire graph G_{diff}, and (3) the total number of paths is minimal. This last requirement is to avoid splitting up multi-hop payments: if there is a payment from A to C along the path $A \rightarrow B \rightarrow C$, we do not want to count it as two (equal-sized) payments of the form A to B and B to C.

We give a simple algorithm that solves the above problem under the assumption the paths are disjoint. This assumption may not always hold, but we will see in Section 6.5.3 that it often holds when the interval between snapshots is relatively short. Our algorithm proceeds iteratively by "merging" payment paths. We initially consider each non-zero edge in G_{diff} as a distinct payment. We then select an arbitrary edge with difference amt, and merge it with any adjacent edges with the same amount (plus the publicly known fee f) until no edge of weight amt can be merged.

$$A \xrightarrow{\mathsf{amt}+f_{A,B}+f_{B,C}} B,\ B \xrightarrow{\mathsf{amt}+f_{B,C}} C \ \Rightarrow\ \text{infer payment A to C}$$

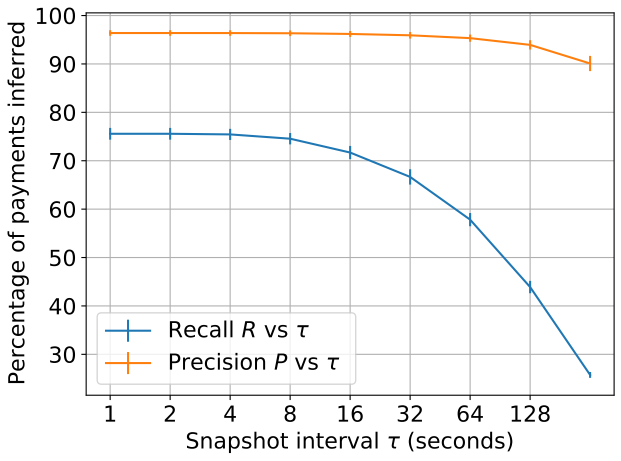


Figure 6.3: The precision and recall of our payment discovery attack, based on the snapshot interval τ (in log scale). The error bars show 95% confidence intervals over five simulation runs.

We then remove this path from G_{diff} and continue with another edge. Asymptotically the running time of this algorithm is $O(|E|^2)$ for E edges; given the size and sparsity of Lightning Network today this means it runs in under a second.

There are several ways this algorithm can make incorrect inferences. First, it would incorrectly merge two same-valued payments A to B and B to C occurring end-to-end. Second, our algorithm does not attempt to resolve the case that a single channel is used for multiple payments in an interval. Looking ahead to Section 6.5.3, our experiments show this happens infrequently when the snapshot intervals are short enough. Finally, as we saw in Section 6.3, balance discovery may fail for some (or many) channels in the network. Our algorithm takes a conservative approach designed to minimize the false positive rate: as a final filtering step, it suppresses any pairs of inferred payments with approximately the same amount (within a small threshold of two satoshis).

6.5.2 Attack simulation

We denote the attacker's precision by P (the number of correctly detected payments divided by the total number of detected payments) and recall by R (the number of correctly detected payments divided by the number of actual payments). We are primarily interested in understanding how these performance metrics depend on the interval at which an attacker takes snapshots (τ).

To answer these questions we leverage the simulator we developed in Section 6.4.1, and extend it to include the balance discovery attack from Section 6.3. Due to the fact that 98% of the errors in this attack were because a node was not online or did not participate in any payments, we set a 0.05 probability of it failing on a functional channel in which both nodes are online. In keeping with the discussion in Section 6.4.1.1, we use $\mathsf{t}_{\mathsf{pay}} = 2000$ as the total number of payments per day and sample the senders and recipients randomly from all nodes in the network. In terms of payment value, our simulation is a pessimistic scenario where the payment amounts are very small (1000 satoshis on average) but fluctuate uniformly within a small range around this average (± 10 satoshis). This is pessimistic because it is close to the worst-case scenario, in which all payments have identical amounts, but two factors make it likely that real-world payments would have much greater variation: (1) payments are denominated in fine-grained units (1 satoshi = 10^{-8} BTC), and (2) wallets typically support generating payment invoices in units of fiat currency by applying the real-time Bitcoin exchange rate, which is volatile.

6.5.3 Simulated attack results

In order to figure out the effect of the snapshot interval τ on P and R, we take balance inference snapshots of the entire network for varying time intervals, ranging from $\tau = 1$ second to $\tau = 2^8$ seconds. Each time, we run the simulator for a period of 30 days, amounting to 60,000 payments in total. Figure 6.3 shows the relationship between τ and the number of payments inferred and confirms the intuition that the attack is less effective the longer the attacker waits between snapshots, as this causes overlap between multiple payments. At some point, however, sampling faster and faster offers diminishing returns; e.g., for $\tau = 32$ seconds, the attacker has a recall R of 66%, which increases slowly to 74.1% for $\tau = 1$ second. With a realistic minimum of $\tau = 30$ seconds, which is the time it took us and others to run the balance discovery attack on a single channel [88]), the attacker has a recall of more than 67%. Because of our final filtering step in our discovery algorithm, we have a precision P very close to 95% for smaller values of τ.

6.6 Conclusions

In this paper, we systematically explored the main privacy properties of the Lightning Network and showed that, at least in its existing state, each property is susceptible to attack. Unlike previous work that demonstrated similar gaps between theoretical and achievable privacy in cryptocurrencies, our research does not rely on patterns of usage or user behavior. Instead, the same interfaces that allow users to perform the basic functions of the network, such as connecting to peers and routing payments, can also be exploited to learn information that was meant to be kept secret. This suggests that these limitations may be somewhat inherent, or at least that avoiding them would require changes at the design level rather than at the level of individual users.

Chapter 7

Conclusions

The main focus of this thesis was the analysis of privacy offered by cryptocurrencies. In particular, we analysed the privacy of Bitcoin, Zcash and the Lightning Network. To this end, we designed and implemented novel methodologies and attacks, as well as applied previously proposed heuristics in a new manner, in order to perform our measurement studies. Our results indicate that cryptocurrencies, including privacy-focused ones and layer-2 solutions, do not sufficiently defend against the tracing of coins.

In Chapter 4, we designed and implemented an improved heuristic for identifying the change output addresses in Bitcoin. Our change identification heuristic was the main building block of our *expansion heuristic*, which uses transaction and address features in order to fingerprint user wallets. Compared to previous change identification methods, our expansion heuristic is by far the most efficient, while at the same time provides the lowest false positive ratio.

Furthermore, we also designed a novel heuristic for validating the correctness of address clustering. We implemented our heuristic on the sets of true- and false-positive clusters, and showed that our heuristic allows to successfully distinguish between those. To the best of our knowledge, this heuristic is the first that aims to increase confidence on address clusters.

In Chapter 5, we presented an in-depth empirical evaluation of the anonymity offered by Zcash, a private alternative to Bitcoin, through the usage of both previously proposed heuristics and newly developed ones. First, we implemented the

Bitcoin's co-spent heuristic, and showed that it has similar effects in Zcash, which was expected since transactions not involving the shielded pool are identical to Bitcoin. However, our study showed that even transactions involving the shielded pool are also subject to de-anonymization. To this end, we developed a tracing heuristic, which allowed us to successfully correlate more than 60% of transactions. Moreover, we performed the first full-scale anonymity measurement of Zcash by developing individual heuristics for each of the cryptocurrency's user groups; the miners, the founders and the regular users.

In Chapter 6, we went beyond layer-1 solutions and looked into the layer-2 Lightning Network, which promises to enhance Bitcoin's scalability and privacy. Our study in Chapter 6 was the first to collectively look into all of the privacy properties promised by the Lightning Network.

We first extended upon the previously proposed probing attack that aimed to identify hidden channel balances. Our enhanced probing technique does not require relying on the error messages returned by clients, thus cannot be easily prevented by simply removing or changing the error messages.

Furthermore, we proposed two new heuristics for the identification of private channels. Our tracing heuristic was the first one to not only identify private channels, but also their participants.

Moreover, we also investigated the information leaked to the intermediate routing nodes. In particular, we analysed how successful the nodes routing the payment are in inferring the endpoints of a payment. To this end, we developed a comprehensive Lightning Network simulator which we used to perform our analysis.

7.1 Future works

Extending the Expansion heuristic. In Chapter 4 we presented our *expansion heuristic* and applied it to several cryptocurrency related crimes. It would be interesting to apply the heuristic to exchange services whose transactions are scripted, hence would probably exhibit even more persistent features. Moreover, it would be also interesting to investigate how the new privacy enhancing feature, Taproot [147], will impact our heuristic.

Current state of Zcash ecosystem. Since our work in Chapter 4, several upgrades have been deployed for Zcash clients. The old sprout pool has been replaced with a more modern sapling and orchard pools that offer much better performance. An interesting follow-up analysis would be to check whether our heuristics are still applicable and effective on the new pools and whether better performance has successfully incentivised the users to use the pools more. Moreover, it would be interesting to see a full-scale design and implementation of our short paper [148] that combined Zcash with network layer protection mechanisms, such as mix networks. We argue, that in order to provide strong privacy guarantees to its users, cryptocurrencies must defend against network analysis.

Network analysis of Lightning Network. In Chapter 6, we performed a measurement study of Lightning's privacy properties by analysing the application layer data. However, a potential extension could focus on its network layer privacy. Although Lightning uses multi-hop onion routing, we showed in our analysis (through our simulator) that the current volume of traffic is low. Low traffic combined with lack of timing obfuscation make network traffic analysis easy [149, 150]. It would be interesting to analyse how successful attacks previously deployed against Tor or other anonymous communication networks, are in the case of the Lightning payment channels.

7.2 Closing remarks

Throughout this thesis, we developed and evaluated heuristics in order to perform anonymity measurements in cryptocurrencies. We showed that the anonymity achieved in practice differs a lot from the one that was expected at the design stage. One of the main reasons for this discrepancy is human behaviour and, in particular, the way those technologies are used. We showed that tracing funds is possible not only in Bitcoin but also in alternative technologies that were built with privacy in mind. Our tools can be used both for investigation purposes, in the context of stolen or lost funds, but also by the privacy community as guidelines for avoiding future mistakes.

Bibliography

[1] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. bitcoin.org/bitcoin.pdf.

[2] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. *Commun. ACM*, 2016.

[3] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan*, 2013.

[4] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), Boston, MA, USA*, 2011.

[5] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, 2013.

[6] Dmitry Ermilov, Maxim Panov, and Yury Yanovich. Automatic Bitcoin address clustering. In *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications (ICMLA 2017)*, pages 461–466, 2018.

[7] Fergal Reid and Martin Harrigan. An analysis of anonymity in the Bitcoin system. *Security and Privacy in Social Networks*, pages 197–223, 2013.

[8] Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, 2013.

[9] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the Internet Measurement Conference - IMC '13*, number 6, pages 127–140, 2013.

[10] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Off the chain transactions. *IACR Cryptol. ePrint Arch.*, 2019.

[11] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable off-chain instant payments, 2016.

[12] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *30th IEEE Symposium on Security and Privacy (S&P), Oakland, California, USA*, 2009.

[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[14] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan*, 2013.

[15] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins:

characterizing payments among men with no names. In *Proceedings of the 2013 Internet Measurement Conference (IMC)*, pages 127–140, 2013.

[16] Chainalysis. https://www.chainalysis.com/.

[17] Elliptic. https://www.elliptic.co/.

[18] Mtgox hack. https://en.wikipedia.org/wiki/Mt._Gox.

[19] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. Bitiodine: Extracting intelligence from the bitcoin network. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados*, 2014.

[20] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[21] Network address translation. https://en.wikipedia.org/wiki/Network_address_translation.

[22] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security, Vancouver, BC, Canada*, 2017.

[23] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[24] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *2003 Symposium on Security and Privacy, 2003.*, pages 2–15. IEEE, 2003.

[25] Sergio Lerner. New vulnerability: know your peer public addresses in 14 minutes. https://bitcointalk.org/?topic=135856%202014, 2013.

[26] Bitcoind wallet. https://github.com/bitcoin/bitcoin.

[27] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anoymity in Bitcoin using P2P network traffic. In *International Conference on Financial Cryptography and Data Security (FC)*, 2014.

[28] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in Bitcoin P2P network. In *Proceedings of ACM CCS*, 2014.

[29] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering bitcoin's public topology and influential nodes. *et al*, 2015.

[30] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. Timing analysis for inferring the topology of the bitcoin peer-to-peer network. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France*, 2016.

[31] Matthias Grundmann, Till Neudecker, and Hannes Hartenstein. Exploiting transaction accumulation and double spends for topology inference in bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 2018.

[32] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. Txprobe: Discovering bitcoin's network topology using orphan transactions. In *International Conference on Financial Cryptography and Data Security*, pages 550–566. Springer, 2019.

[33] Till Neudecker and Hannes Hartenstein. Could network information facilitate address clustering in bitcoin? In *International conference on financial cryptography and data security*, pages 155–169. Springer, 2017.

[34] Alex Biryukov, Daniel Feher, and Giuseppe Vitto. Privacy aspects and subliminal channels in zcash. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng

Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS, London, UK*, 2019.

[35] Ben Laurie. An efficient distributed currency. *Practice*, 100, 2011.

[36] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. *arXiv preprint arXiv:1505.06895*, 2015.

[37] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. pages 397–411, 2013.

[38] George Danezis, Cedric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*, pages 27–30, 2013.

[39] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.

[40] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. pages 459–474, 2014.

[41] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 321–340. Springer, 2010.

[42] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography Conference*, pages 169–189. Springer, 2012.

[43] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography Conference*, pages 315–333. Springer, 2013.

[44] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[45] Zcash. https://z.cash.

[46] Jeffrey Quesnelle. On the linkability of zcash transactions. *CoRR*, abs/1712.01210, 2017.

[47] Harry A. Kalodner, Steven Goldfeder, Alishah Chator, Malte Möser, and Arvind Narayanan. Blocksci: Design and applications of a blockchain analysis platform. *arXiv:1709.02489*, 2017.

[48] Nicolas van Saberhagen. Cryptonote v2.0, 2013. https://cryptonote.org/whitepaper.pdf.

[49] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *International Workshop on Public Key Cryptography*, pages 181–200. Springer, 2007.

[50] Surae Noether, Sarang Noether, and Adam Mackenzie. A note on chain reactions in traceability in cryptonote 2.0. *Research Bulletin MRL-0001. Monero Research Lab*, 1:1–8, 2014.

[51] Adam Mackenzie, Surae Noether, and Monero Core Team. Improving obfuscation in the cryptonote protocol. *Monero research lab report MRL-0004*, 2015.

[52] Shen Noether, Adam Mackenzie, and the Monero Research Lab. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[53] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, et al. An empirical analysis of traceability in the Monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018.

[54] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS - 22nd European Symposium on Research in Computer Security, Oslo, Norway*, 2017.

[55] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In *International conference on the theory and application of cryptology and information security*, pages 649–678. Springer, 2019.

[56] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*. Springer, 2020.

[57] Gregory Maxwell. CoinJoin: Bitcoin privacy for the real world. bitcointalk.org/index.php?topic=279249, August 2013.

[58] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical decentralized coin mixing for Bitcoin. pages 345–364, 2014.

[59] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[60] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 340–350, 2010.

[61] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. P2p mixing and unlinkable bitcoin transactions. Cryptology ePrint Archive, Report 2016/824, 2016. https://ia.cr/2016/824.

[62] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1988.

[63] Wasabi wallet. https://github.com/zkSNACKs/WalletWasabi.

[64] Samurai wallet. https://samouraiwallet.com/.

[65] Joinmarket wallet. https://github.com/JoinMarket-Org/joinmarket-clientserver.

[66] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. Mixcoin: Anonymity for Bitcoin with accountable mixes. pages 486–504, 2014.

[67] Luke Valenta and Brendan Rowan. Blindcoin: Blinded, accountable mixes for Bitcoin. pages 112–126, 2015.

[68] Georg Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. *IACR Cryptol. ePrint Arch.*, 2009. http://eprint.iacr.org/2009/320.

[69] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: an untrusted Bitcoin-compatible anonymous payment hub. In *Proceedings of NDSS 2017*, 2017.

[70] Lightning network specifications. https://github.com/lightningnetwork/lightning-rfc.

[71] Fast, cheap, scalable token transfers for Ethereum. https://raiden.network/.

[72] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[73] Simina Brânzei, Erel Segal-Halevi, and Aviv Zohar. How to charge lightning. *CoRR*, abs/1712.10222, 2017.

[74] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 439–453, 2017.

[75] Nida Khan et al. Lightning network: A comparative review of transaction fees and data analysis. In *International Congress on Blockchain and Applications*, pages 11–18. Springer, 2019.

[76] Stefano Martinazzi. The evolution of lightning network's topology during its first year and the influence over its core values. *arXiv preprint arXiv:1902.07307*, 2019.

[77] Ferenc Béres, István András Seres, and András A. Benczúr. A cryptoeconomic traffic analysis of bitcoins lightning network. *CoRR*, abs/1911.09432, 2019.

[78] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 347–356. IEEE, 2019.

[79] Small-world network. https://en.wikipedia.org/wiki/Small-world_network.

[80] Cristina Pérez-Sola, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. Lockdown: Balance availability attack against lightning network channels. In *International Conference on Financial Cryptography and Data Security*. Springer, 2020.

[81] Ayelet Mizrahi and Aviv Zohar. Congestion attacks in payment channel networks. In *International Conference on Financial Cryptography and Data Security*. Springer, 2021.

[82] Saar Tochner, Stefan Schmid, and Aviv Zohar. Hijacking routes in payment channel networks: A predictability tradeoff. *CoRR*, abs/1909.06890, 2019.

[83] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. *Cryptology ePrint Archive*, 2018.

[84] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001.

[85] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous Multi-Hop Locks for blockchain scalability and interoperability. In *Proceedings of NDSS*, 2018.

[86] Mariusz Nowostawski and Jardar Tøn. Evaluating methods for the identification of off-chain transactions in the Lightning Network. *Applied Sciences*, 2019.

[87] Matteo Romiti, Friedhelm Victor, Pedro Moreno-Sanchez, Bernhard Haslhofer, and Matteo Maffei. Cross-layer deanonymization methods in the lightning protocol, 2020.

[88] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquin Garcia-Alfaro. On the difficulty of hiding the balance of Lightning Network channels. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (CCS)*, 2019.

[89] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. Toward active and passive confidentiality attacks on cryptocurrency off-chain networks, 2020.

[90] Sergei Tikhomirov, Rene Pickhardt, Alex Biryukov, and Mariusz Nowostawski. Probing channel balances in the lightning network, 2020.

[91] Marco Conoscenti, Antonio Vetrò, Juan Carlos De Martin, and Federico Spini. The cloth simulator for HTLC payment networks with introductory lightning network performance results. 2018.

[92] Felix Engelmann, Henning Kopp, Frank Kargl, Florian Glaser, and Christof Weinhardt. Towards an economic analysis of routing in payment channel net-

works. In *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, SERIAL@Middleware 2017, Las Vegas, NV, USA*. ACM, 2017.

[93] Yuhui Zhang, Dejun Yang, and Guoliang Xue. Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks. In *2019 IEEE International Conference on Communications, ICC 2019, Shanghai, China*, 2019.

[94] Steven Goldfeder, Harry Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *arXiv preprint arXiv:1708.04748*, 2017.

[95] Andy Greenberg. Prosecutors Trace $13.4M in Bitcoins From the Silk Road to Ulbricht's Laptop, January 2015. https://www.wired.com/2015/01/prosecutors-trace-13-4-million-bitcoins-silk-road-ulbrichts-laptop/.

[96] Jesse Dunietz. The Imperfect Crime: How the WannaCry Hackers Could Get Nabbed, August 2017. https://www.scientificamerican.com/article/the-imperfect-crime-how-the-wannacry-hackers-could-get-nabbed/.

[97] United States Department of Justice. South Korean national and hundreds of others charged worldwide in the takedown of the largest darknet child pornography website, which was funded by Bitcoin. https://www.justice.gov/opa/pr/south-korean-national-and-hundreds-others-charged-worldwide-takedown-largest-darknet-child, 2019.

[98] United States Department of Justice. Global disruption of three terror finance cyber-enabled campaigns. https://www.justice.gov/opa/pr/global-disruption-three-terror-finance-cyber-enabled-campaigns, August 2020.

[99] Andy Greenberg. Feds Arrest an Alleged $336M Bitcoin-Laundering Kingpin, April 2021. https://www.wired.com/story/bitcoin-fog-dark-web-cryptocurrency-arrest/.

[100] Devon Beckett. Statement of facts, April 2021. https://storage.courtlistener.com/recap/gov.uscourts.dcd.230456/gov.uscourts.dcd.230456.1.1_1.pdf.

[101] Katharina Krombholz, Aljosha Judmayer, Matthias Gusenbauer, and Edgar Weippl. The other side of the coin: User experiences with Bitcoin security and privacy. In *International Conference on Financial Cryptography and Data Security*, 2016.

[102] Alexandra Mai, Katharina Pfeffer, Matthias Gusenbauer, Edgar Weippl, and Katharina Krombholz. User mental models of cryptocurrency systems - a grounded theory approach. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, 2020.

[103] Svetlana Abramova, Artemij Voskobojnikov, Konstantin Beznosov, and Rainer Böhme. Bits under the mattress: Understanding different risk perceptions and security behaviors of crypto-asset users. In *CHI Conference on Human Factors in Computing Systems (CHI)*, 2021.

[104] Benoit Faucon, Ian Talley, and Summer Said. Israel-Gaza Conflict Spurs Bitcoin Donations to Hamas, June 2021. https://www.wsj.com/articles/israel-gaza-conflict-spurs-bitcoin-donations-to-hamas-11622633400.

[105] Rachel Monroe. How to negotiate with ransomware hackers, June 2021. https://www.newyorker.com/magazine/2021/06/07/how-to-negotiate-with-ransomware-hackers.

[106] David A. Harding and Peter Todd. BIP 125: Opt-in Full Replace-by-Fee Signaling, 2015. https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki.

[107] Mark Friedenbach, BtcDrak, Nicolas Dorier, and kinoshitajona. BIP 68: Relative lock-time using consensus-enforced sequence numbers, 2015. https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki.

[108] Eric Lombrozo, Johnson Lau, and Pieter Wuille. BIP 141: Segregated

Witness (Consensus layer), 2015. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.

[109] Harry Kalodner, Malte Möser, Kevin Lee, Steven Goldfeder, Martin Plattner, Alishah Chator, and Arvind Narayanan. Blocksci: Design and applications of a blockchain analysis platform. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2721–2738. USENIX Association, August 2020.

[110] Andy Liaw and Matthew Wiener. Classification and regression by random forest. *R News*, 2(3):18–22, 2002.

[111] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[112] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[113] Marvin N. Wright and Andreas Ziegler. `ranger`: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017.

[114] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(25), 2007.

[115] Carolin Strobl, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis. Conditional variable importance for random forests. *BMC Bioinformatics*, 9(307), 2008.

[116] Chainalysis Team. 270 service deposit addresses drive 55% of money laundering in cryptocurrency, February 2021. https://blog.chainalysis.com/reports/cryptocurrency-money-laundering-2021.

[117] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in Zcash. In William Enck and Adri-

enne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security, Baltimore, MD, USA8*, 2018.

[118] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of linkability in the Monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2017.

[119] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of Monero's blockchain. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 153–173, Cham, 2017. Springer International Publishing.

[120] Zuoxia Yu, Man Ho Au, Jiangshan Yu, Rupen Yang, Quiliang Xu, and Wang Fat Lau. New empirical traceability analysis of CryptoNote-style blockchains. In *Proceedings of the 23rd International Conference on Financial Cryptography and Data Security (FC)*, 2019.

[121] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan*, 2013.

[122] Yaya J. Fanusie and Tom Robinson. Bitcoin laundering: An analysis of illicit flows into digital currency services, January 2018. A memorandum by the Center on Sanctions and Illicit Finance and Elliptic.

[123] Cyrus Farivar and Joe Mullin. Stealing bitcoins with badges: How Silk Road's dirty cops got caught, August 2016. https://arstechnica.com/tech-policy/2016/08/stealing-bitcoins-with-badges-how-silk-roads-dirty-cops-got-caught/.

[124] Jesse Dunietz. The Imperfect Crime: How the WannaCry Hackers Could Get Nabbed, August 2017. https://www.scientificamerican.com/article/the-imperfect-crime-how-the-wannacry-hackers-could-get-nabbed/.

[125] Dash. https://www.dash.org.

[126] Monero. https://getmonero.org.

[127] Cryptocurrency market capitalizations. https://coinmarketcap.com/.

[128] Alphabay will accept Zcash starting July 1st, 2017. DarkNetMarkets Reddit post, 2017. https://www.reddit.com/r/DarkNetMarkets/comments/6d7q81/alphabay_will_accept_zcash_starting_july_1st_2017/.

[129] JP Buntinx. The Shadow Brokers only accept ZCash payments for their monthly dump service, May 2017. https://themerkle.com/the-shadow-brokers-only-accept-zcash-payments-for-their-monthly-dump-service/.

[130] Zcash faqs. https://z.cash/support/faq.html.

[131] Zcash Chain Parameters. https://github.com/zcash/zcash/blob/v1.0.0/src/chainparams.cpp#L135-L192.

[132] Shapeshift. https://shapeshift.io.

[133] Zchain explorer. http://explorer.zcha.in/.

[134] Ittay Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA*, 2015.

[135] What is Jubjub? https://z.cash/technology/jubjub.html.

[136] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria*, 2016.

[137] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün

Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains - (A position paper). In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados*, 2016.

[138] Abraham Hinteregger and Bernhard Haslhofer. Short paper: An empirical analysis of monero cross-chain traceability. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC, Frigate Bay, St. Kitts and Nevis*, 2019.

[139] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in zcash. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security, Baltimore, MD, USA8*, 2018.

[140] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017.*

[141] Ivan Bogatyy. Linking 96% of Grin transactions. https://github.com/bogatyy/grin-linkability.

[142] Matthew K. Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Trans. Inf. Syst. Secur.*, 7(4):489–522, 2004.

[143] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[144] Jin Y Yen. An algorithm for finding shortest routes from all source nodes to

a given destination in general networks. *Quarterly of Applied Mathematics*, 1970.

[145] Person behind 40% of LN's capacity: "I have no doubt in Bitcoin and the Lightning Network". https://www.theblockcrypto.com/post/41083/person-behind-40-of-lns-capacity-i-have-no-doubt-in-bitcoin-and-the-lightning-network.

[146] The Lightning Conference: Of channels, flows and icebergs talk by Christian Decker. https://www.youtube.com/watch?v=zk7hcJDQH-I.

[147] Taproot. https://www.investopedia.com/bitcoin-taproot-upgrade-5210039.

[148] George Kappos and Ania M. Piotrowska. Extending the anonymity of Zcash. 2019. http://arxiv.org/abs/1902.07337.

[149] Paul F. Syverson, Gene Tsudik, Michael G. Reed, and Carl E. Landwehr. Towards an analysis of onion routing security. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA*, 2000.

[150] Andrei Serjantov and Peter Sewell. Passive-attack analysis for connection-based anonymity systems. *Int. J. Inf. Sec.*, 2005.