Master Thesis

Data-Driven De-Anonymization in Bitcoin

Author(s):
Nick, Jonas David

Publication Date:
2015

Permanent Link:

https://doi.org/10.3929/ethz-a-010541254 →

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Data-Driven De-Anonymization in Bitcoin

Master's Thesis

Jonas David Nick

jonasd.nick@gmail.com

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Christian Decker
Prof. Dr. Roger Wattenhofer

August 9, 2015

Acknowledgements

I thank Christian Decker for his active support. Our weekly meetings shaped the direction and were essential for the success of this thesis. I am especially thankful for the encouragement during the course of the project. Christian gave me access to his Bitcoin infrastructure which saved a lot of time and allowed me to focus on the topic. Also, his continued advice greatly helped to improve my academic writing and presentation techniques.

I thank Prof. Dr. Roger Wattenhofer for his feedback and providing the resources to enable this research. In particular, without the SGE Arton cluster I would not have been able to use 70,000 CPU hours and the scale of this project would have been much smaller.

Abstract

We analyse the performance of several clustering algorithms in the digital peer-to-peer currency Bitcoin. Clustering in Bitcoin refers to the task of finding addresses that belongs to the same wallet as a given address.

In order to assess the effectiveness of clustering strategies we exploit a vulnerability in the implementation of Connection Bloom Filtering to capture ground truth data about 37,585 Bitcoin wallets and the addresses they own. In addition to well-known clustering techniques, we introduce two new strategies, apply them on addresses of the collected wallets and evaluate precision and recall using the ground truth. Due to the nature of the Connection Bloom Filtering vulnerability the data we collect is not without errors. We present a method to correct the performance metrics in the presence of such inaccuracies.

Our results demonstrate that even modern wallet software can not protect its users properly. Even with the most basic clustering technique known as multi-input heuristic, an adversary can guess on average 68.59% addresses of a victim. We show that this metric can be further improved by combining several more sophisticated heuristics.

Contents

Chapter 1

Introduction

Bitcoin is a peer-to-peer currency which maintains a global history of transactions – the blockchain [Nak08]. Transactions are used to transfer bitcoins between users with the help of asymmetric cryptography. We use the term Bitcoin with a capital "B" to denote the payment system and bitcoin to denote currency units. Users of the system are represented by public keys and authorize transactions using their private keys. They use software applications that are called *wallets* and are mainly responsible for holding private keys and signing transactions. Also, they often broadcast, receive and validate transactions.

The question we address in this project is about the privacy properties of Bitcoin. Privacy is a natural requirement for a financial system which handles sensitive payments like salaries, debts and medical bills. Neither the parties involved nor the transaction value should be public information. An additional prerequisite for a successful currency is fungibility. A bitcoin currency unit should be substitutable for any other bitcoin and thus each bitcoin should have the same value. For example, there have been reports that some bitcoins are not accepted at certain exchanges because they have been allegedly involved in criminal activity at some point of their history. Obviously, this problem can be only solved with proper transactional privacy. It is evident that privacy aspects have strongly shaped Bitcoin's history and will continue to do so. Especially in Bitcoin's early days, online drug markets that use Bitcoin drove Bitcoin's adoption and as part of their advertising they claimed to be anonymous. On the other hand regulatory agencies also attribute anonymity to Bitcoin and therefore have a motive to enhance supervision.

However, Bitcoin's original design was never intended to guarantee anonymity and even privacy can only be achieved in narrow limits as previous research has already shown. This publication further sheds light on the amount of information an attacker observing the blockchain can obtain. Our contribution is the first large-scale performance evaluation of several clustering strategies on the live Bitcoin network. Clustering strategies attempt to determine which public keys belong to the same persons using public information in the blockchain. With that knowledge it is possible to identify the transactions of the person. In order

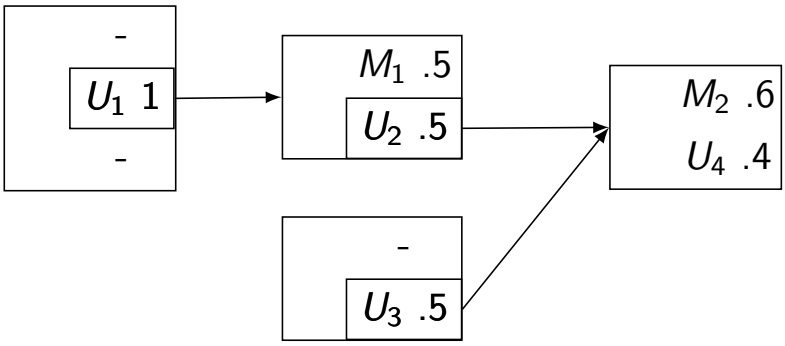

Figure 1.1: Exemplary part of a transaction graph where nodes represent transactions and edges connect an output with the transaction that spends it.

to quantify the harm that can be done to a user's privacy in practice we collect the public keys of 37,585 wallets. This data is used to evaluate the performance of various clustering algorithms and shows that an attacker can learn 69.34% of a wallet's public keys.

1.1 Bitcoin Transactions

Transactions transfer values between entities that are represented by their public keys, or *pubkeys*. Thus, one can think of pubkeys as account numbers in the traditional banking system. The corresponding private key is used to prove ownership of the account.

For example, assume there is a user with pubkey U_1 and a balance of 1 bitcoin. This is represented in Bitcoin as a transaction moving 1 bitcoin to U_1. Now, the user wants to pay half a bitcoin to a merchant with pubkey M_1. She therefore creates a transaction that redeems a transaction *output* of a previous transaction (Figure 1.1). An output is a tuple of a *Bitcoin script* and a value, whereby the script specifies the condition for spending. The condition usually consists of providing a valid signature for a pubkey, which requires access to the corresponding private key. So, the balance of a user is the sum of values of outputs she can provide the necessary conditions for. A transaction can be understood as claiming outputs and creating new ones.

Accordingly, the user redeems her output by proving her ownership of U_1 by including a signature over the complete transaction. The reference to the spent output and the signature is the *input* of the transaction. In order to pay the merchant, she adds an output to her transaction with pubkey M_1 and the value they agreed upon. Because a spent output can never be spent again, she adds another output U_2 to her transaction that sends the surplus value to a new pubkey she created. This output is called *change*. The difference between

the sum of inputs and outputs are fees that are going to be collected by Bitcoin *miners*. Miners are responsible for ordering transactions and process transactions quicker which have more fees per byte of serialized transaction.

Among other less relevant conditions, for a transaction to be valid the sum of input values must exceed the sum of output values and the referenced outputs must not be previously claimed by any other transaction in the blockchain. Additionally, the script evaluation has to succeed which in most cases requires providing a valid signature.

One technical detail about transactions that will be important later is that, initially, regular transaction outputs contained the plain pubkeys of the receiving parties, whereas nowadays outputs usually only consist of the hash of a pubkey. The former transaction output is called *pay-to-pubkey*, the latter is the newer *pay-to-pubkey-hash* scheme. The only difference for the redeemer is then that she has to provide signature and pubkey. A pubkey hash is usually communicated in the form of a Bitcoin address, which encodes the hash, an address version number and a checksum in base58. The terms address and pubkey are mostly interchangeable for our purposes.

The example proceeds with the user wanting to pay a merchant M_2 0.6 Bitcoin. With her pubkey U_2 she has only 0.5 bitcoin available but luckily she has a another spare output with 0.5 bitcoin. Because the transaction exceeds the value of every single output the user can redeem, she has to create a so called multi-input transaction. This transaction therefore contains signatures for multiple pubkeys.

1.2 Blockchain Privacy

The fundamental problem of Bitcoin is that the sending pubkeys, the receiving pubkeys and values of all transactions are publicly recorded in the blockchain. An attacker that obtained the public key of a victim can scan the transaction history and find a transaction the pubkey was involved in, which reveals the sending and receiving party. On the other hand the link between real identities and pubkeys is in general unknown. This is often referred to as *pseudonymity*.

If an attacker knows that a pubkey belongs to the user and the merchant address is publicly known, she can conclude that the user paid the merchant. Moreover, an attacker might not only find a single transaction with that pubkey but the knowledge of the pubkey could reveal the whole transaction history of the user. This is why users are strongly encouraged not to reuse their pubkeys, but create a new key pair for each receiving transaction. However, in many cases this is inconvenient for use cases such as publishing a tipping or donation address or to use your pubkey for identification.

The possibility of creating a practically infinite number of pubkeys triggered

research interest in the discipline of *clustering*. Given a pubkey, find pubkeys that belong to the same user using the transaction history. There are several ways to measure the success of a clustering strategy. We are concerned with the *precision* and *recall* metrics. Precision is the number of correctly identified pubkeys as a fraction of the total number of pubkeys found by the strategy. For example, a precision of 0.5 means that half of the pubkeys found by the strategy were correct and the other half were not. Recall is the number of correctly identified pubkeys as a fraction of the total number of pubkeys that belong to the wallet. A recall of 0.5 means that the strategy has found half of the wallets pubkeys.

1.3 Related Work

Nakamoto [Nak08] already mentions a clustering technique we call multi-input heuristic in his original Bitcoin paper. This heuristic is implemented in an open source analysis framework [SMZ14] and has been studied by various research groups [AKR+13, MPJ+13, OKH13]. Meiklejohn et al. applied clustering and assigned names to clusters by interacting with various services and therefore acquiring their addresses. Androulaki et al. [AKR+13] formulated a clustering strategy called shadow heuristic. In order to estimate the success of clustering they simulated participants of the Bitcoin network to generate transaction data. They concluded that Bitcoin does not protect the privacy of its users sufficiently.

In contrast to previous studies, we capture actual data about consumer wallets and their entire set of pubkeys. Then we cluster the collected pubkeys and compute precision and recall using the ground truth. We think that analysing real world data is superior to simulations because the clustering strategies are very sensitive to the actual spending behaviour of the users. The data itself is collected using a vulnerability that is discussed in detail in [GCKG14].

These attacks on privacy rely on Blockchain analysis and have to be distinguished from de-anonymization techniques that use the peer-to-peer network [KKM14, BKP14].

Chapter 2

Address Clustering

The general idea behind clustering is to have a pubkey p from the set of all pubkeys $\mathbb{P}$ and to apply a clustering algorithm $h : \mathbb{P} \to 2^{\mathbb{P}}$ to get a set of pubkeys that are likely to belong to the same wallet. Then the clustering algorithm is applied recursively to each pubkey in the set to get the complete cluster for p. Note that there is no guarantee that the following clustering algorithms give useful results. They merely exploit how typical wallet softwares create transactions, which is why they are also called *heuristics*. To our knowledge, the consumer and optimal change heuristic have not been used in academic research before.

2.1 Multi-Input Heuristic

This heuristic uses the fact that wallets are usually solely responsible for creating transactions. If there is a transaction spending a victim's output, an attacker can conclude that the pubkeys of other redeemed outputs also belong to the victim. In our example in Figure 1.1, if an attacker knows that address U_3 belongs to the user, she could be quite certain that U_2 belongs to the same user. Therefore, the corresponding clustering algorithm $h_I(p)$ searches for all transactions that redeem p and returns all pubkeys which are also redeemed in the same transactions.

The success of this heuristic depends on the number and size of a wallet's multi-input transactions. For example, if a user usually receives small amounts and sends large amounts there will be more multi-input transactions.

2.2 Shadow Heuristic

This heuristic exploits how most wallets handle change. These wallets generate a fresh key pair for each change output and do not expose it to the user such that a change pubkey is by default not used multiple times. Coming back to our example in Figure 1.1 we assume that M_1 is constantly used as the payments

merchant address. We would not expect this for a change address, so U_2 must be the change ergo U_1 and U_2 are in the same wallet.

In detail, the heuristic algorithm $h_S(p)$ searches for all transactions where p is redeemed. Then it returns for each transaction with two or more outputs the unique change candidate if it exists. A shadow change candidate is pubkey that appears at most in two transactions, of which one is a receiving and one a spending transaction.

The success of heuristics that try to detect the change is dependent on the recipient of the transaction. If a user's wallet follows best practice and uses a fresh change, the other party could reuse their pubkeys and therefore make the change distinguishable.

2.3 Consumer Heuristic

In contrast to previous research on clustering we are only concerned with clustering addresses from a consumer wallet. By consumer wallet we understand a wallet that by default only allows to send bitcoins to a single address. Every popular wallet falls under this category, such as Bitcoin Core, Electrum, MultiBit, Armory, Android Bitcoin Wallet, etc. A transaction generated by a consumer wallet will always have one or two outputs. Therefore, we can make use of this to find the change of a transaction.

The heuristic algorithm $h_C(p)$ searches for all transaction where p is redeemed. For each transaction it returns the consumer change candidate if it exists and is unique. A consumer change candidate is a pubkey whose transactions always have two or less outputs.

2.4 Optimal Change Heuristic

This heuristic is based on the assumption that wallet software does not spend unnecessary outputs. In general, when a user requests to send a certain number of bitcoins, her wallet searches for outputs it can spend in the transaction. The transaction can only be valid if the sum of redeemed output values exceeds the value the user intends to send. Furthermore, it should hold that when any of selected outputs is omitted, the sum of the remaining outputs does not reach the desired value. If this does not hold and such an output exists, the behavior is suboptimal because spending it has no effect except increasing the size of the transaction and therefore requiring more fees.

This implies that the change value is smaller than any of the spent outputs. If the change was larger than one output then this output can be left out and the change is reduced by the output's value. If a transaction has a unique output

with a smaller value than any of the inputs, it is very likely to be the true change output, so we call it *optimal change output*. In the rightmost transaction of Figure 1.1, for example, it is reasonable to think that U_4 is the change address because the associated value, 0.4 bitcoin, is uniquely smaller than both 0.5 bitcoin inputs. If the 0.6 bitcoin output was the change then a wallet should have omitted one of the inputs.

Similar to the other heuristics, the optimal change heuristic $h_O(p)$ searches for all transaction where p is redeemed. For each transaction it returns the optimal change output if it exists and is unique.

2.5 Wallets

Instead of trying to cluster every pubkey in the blockchain we only analyse a subset of wallet implementations. Therefore, we can apply heuristics that exploit specific wallet behaviors and would often fail when applied to a random key in the blockchain. In order to assess the applicability of a clustering heuristic, we have to understand some of the inner workings of the wallets we are dealing with. It should become evident that we can not simply generalize the clustering performance from one wallet implementation to another.

For example, the multi-input heuristic can be easily defeated by a mechanism called a coinjoin transaction. It is in principle possible that a transaction has been created by multiple parties which each sign for one pubkey and then send the transaction to the next person. In its simplest form, the coinjoin works by multiple users creating a single transaction. This is in conflict with the central assumption of the multi-input heuristic. The multi-input heuristic would mark every transaction input as belonging to the same wallet, which is clearly not correct. Moreover, as long as the output values are the same all mappings from input to output of a coinjoin are equiprobable. We could have attempted to design a heuristic to detect coinjoins but coinjoin transactions should be extremely rare in the wallets we study. None of them perform coinjoins automatically or at the users request. Rather, a user would have to export her private keys and import it into specialized software, which have only a small number of users. However, we can not exclude the possibility of coinjoins completely.

The process by which a wallet collects outputs to construct a transaction that satisfies the given amount and fee requirements is called *coin selection*. There are different variables a wallet can attempt to optimize during coin selection. For example, the Bitcoin Core wallet tries to maximize the number of outputs a transaction redeems and minimizes the change. Reducing the size of the unspent outputs is favorable because this is the data that is replicated among all validating Bitcoin nodes. The BitcoinJ library on the other hand optimizes the fees a transaction has to pay in order to be timely confirmed by the Bitcoin miners.

The fees are dependent on the size of the serialized transaction and therefore BitcoinJ's objective is the opposite of Bitcoin Core's unspent outputs minimization. In general, these implementations follow a complex logic, which is not guaranteed to give the optimal result because of the computational complexity of the problem in question. In addition to implying that the clustering results are very specific to a wallet implementation, this also means that the optimal change heuristic might fail occasionally.

There are still legacy wallets in wide use, for example MultiBit, which do not generate fresh change pubkeys. They use the first pubkey in the wallet as the receiver of the change. This behaviour is clearly problematic from a privacy perspective, because this key is going to be the input of many future transactions. Also, it contradicts the expectation of the shadow heuristic which therefore can not be applied.

In some newer wallets like Android Bitcoin Wallet 4, address reuse is discouraged by automatically creating and displaying a new address when they notice that the current address received something. In other wallets such as Bitcoin Core and MultiBit a user has to manually request a new address, which typically results in more transactions with the same key.

We expect our heuristics to be quite precise – they will only rarely label a pubkey as being in the wallet when in reality it is not. This is because the heuristics are conservatively designed and specifically tied to the wallet software so that the precision should in theory be perfect. However, the multi-input heuristic fails when it encounters coinjoin transactions and the shadow heuristic returns a wrong result if the wallet reuses a change address. Since the wallet by itself never reuses the change and the address is not exposed in the user interface, a user would have to find out her pubkey with manual blockchain inspection which is unlikely to happen. In order for the consumer heuristic to return a wrong result, the wallet software would have to be able to create transactions with more than one recipient which is not possible in the wallets we are going to discuss. The optimal change heuristic can indeed fail in a few cases even when the wallet is normally used because coin selection may not be able to find the optimal solution. The fraction of wallet pubkeys the clustering strategy returns, the recall, on the other hand is mostly determined by the behavior of the users. In particular, the recall is affected by the number of transactions, the number of bitcoins the user usually receives and the number of bitcoins the user usually spends.

Chapter 3

Bloom Filter Attack

Our approach to studying the real effect of clustering involves a collection of real wallets, i.e. sets of pubkeys from the blockchain that belong to the same wallet. We apply the heuristic and use the wallet data to measure the performance on ground truth.

Getting the data is usually a nontrivial undertaking because users will not deliberately give up their privacy and wallet software is expected to prevent such privacy leaks by all means. We used an exploit for a wallet implementation called BitcoinJ which allows to reconstruct all of its pubkeys after connecting via the peer-to-peer network. The vulnerability lies in the application of Bloom filters in Bitcoin's network protocol and BitcoinJ's specific implementation. In this chapter we first give a short introduction to Bloom Filters and how they are used in Bitcoin before we discuss the actual vulnerability.

3.1 Bloom Filters

A Bloom filter is a probabilistic data structure that is used to check whether an element is a member of a set [Blo70]. The advantage of applying Bloom filters is that they require less space than the full data set. The two operations on Bloom filters are *insert* and *query*. Insert puts a piece of data into the filter and a data query returns true if the data has been inserted before. Bloom filters are characterized by their *false positive rate* which is the probability that a query is positive although the element was never inserted into the filter. There is a trade-off between the false positive rate and the size of the filter – the smaller the acceptable false positive, the bigger the filter. On the other hand Bloom filters never return false negatives, so an element put into the filter will always be recognized as such.

Bloom filters are used in Bitcoin *simplified payment verification* (SPV) wallets. In contrast to *full nodes*, SPV clients do not validate the whole blockchain. Validation requires knowledge about every single transaction in the blockchain, whereas SPV nodes can save bandwidth by requesting only the subset of trans-

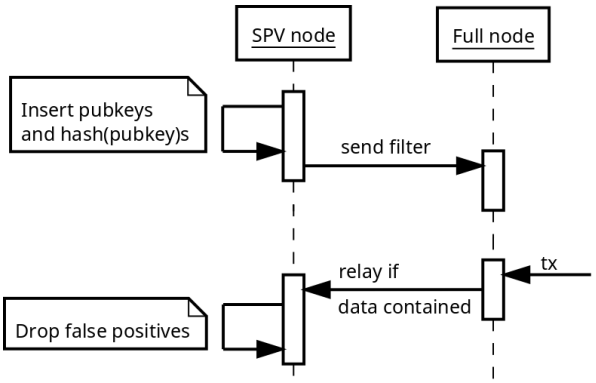

Figure 3.1: Sequence diagram of the Connection Bloom filtering protocol.

actions the SPV node is interested in. One mechanism to achieve that was suggested in Bitcoin Improvement Proposal BIP 37 "Connection Bloom Filtering" [HC12] and subsequently implemented. The proposal introduces an extension to the Bitcoin network protocol that uses Bloom filters to allow SPV nodes to learn only about transactions that affect their balances. It works as follows (Figure 3.1):

The SPV node stores a Bloom filter that contains all pubkeys and hashes of pubkeys of the wallet. Upon connecting to a full node, the SPV and full node exchange initial handshake messages. Immediately afterwards, the SPV node sends its filter through the connection. Whenever the full node receives a transaction it queries the filter with parts of the transaction data. If there is a match it relays the transaction to the SPV node which can then check if one of its pubkeys is really of interest and potentially discards the transaction.

Using a Bloom filter instead of the trivial protocol which just sends all pubkeys and pubkey-hashes to the full node has a disadvantage. It produces false positives and therefore leads to unnecessary network communication. However, using a Bloom filter is advantageous to the full note: the filter reduces the space requirements and allows quick queries. Hence, Bloom filters also help the full node to be more robust against denial of service attacks that target resource exhaustion. More importantly, Bloom filters are intended to prevent leakage of information about the wallet's pubkeys. If the false positive rate is sufficiently high, the large amount of possible data contained does not give an attacker substantial information about the actual data contained. But these privacy benefits are achieved only at the cost of some wasted bandwidth. Therefore, the false positive rate controls a bandwidth/privacy trade-off which has to be carefully tuned.

At first glance, Connection Bloom Filtering appears to have attractive privacy properties. For example, a common false positive rate for filters we see in the Bitcoin network is 0.0146%, which enables acceptable performance on wallets

including those connected via mobile networks. Further, this means that an attacker querying the whole blockchain consisting of 60 million pubkeys would expect $0.0146\% \cdot 60\text{E}6 = 8760$ false positives. Most wallets truly contain only a relatively small number of keypairs, so the attacker has only a small chance of identifying a true positive among those 8760 positives.

3.2 Vulnerability

The attack targets the most used SPV wallet library that uses Bloom filters called BitcoinJ, but there may be more vulnerable implementations.

The main issue is that BitcoinJ puts both pubkey and pubkey-hash into the Bloom filter. This implies that if a pubkey is really owned by the wallet, both pubkey and hash must be in the filter. Because these are independent events, an algorithm that queries the filter with both and returns the logical AND, lowers the false positive rate to its square. When applying this method to our previous example with false positive rate 0.0146% we expect only 1.27 false positives when scanning the whole blockchain. Therefore, an attacker can guess true positives in the set of all positive pubkeys with high probability.

After reporting this issue to the developers of BitcoinJ, we thought about possible fixes but the issues involved turned out to be quite complex. One might suggest to increase the false positive rate such that even its square provides a certain amount of privacy. But this would ramp up the bandwidth usage which, according to the BitcoinJ maintainers, is not acceptable for their users. It is reasonable to ask if it is really necessary that the BitcoinJ node inserts pubkeys in addition to pubkey-hashes. If BitcoinJ would not insert pubkeys then the wallet would not notice that it received a pay-to-pubkey transaction and BitcoinJ naturally wants to remain compatible to all standard transaction types.

Even if this is somehow fixed there is another problem which occurs when the filter gets *dirty*. A filter is dirty when a lot of additional elements are inserted after the creation of the filter. Due to the inner workings of the Bloom filter, the false positive rate increases as more elements are inserted and the bandwidth costs become more and more significant. In order to match the desired false positive rate, the filter has to be recomputed and send to the peers. An attacker with two separate filters from the same wallet is able to remove false positives by taking the intersection of both filters' matches.

There are in fact a couple of additional problems in the Bloom filter protocol that have no simple fixes[1]. It appears that substantial changes in the protocol design are necessary to improve privacy. Unfortunately, this means that the situation will not change soon.

[1] BitcoinJ developer Mike Hearn's response to the vulnerability report: https://groups.google.com/forum/#!msg/bitcoinj/Ys13qkTwcNg/9qxnhwnkeoIJ

3.3 Filter Collection

We have seen how to get all pubkeys from the filter with only a small number of false positives but we still need to collect a sufficient number from the Bitcoin network. SPV wallets usually do not accept incoming connections. They ask hard-coded *seed nodes* for random peers and open outbound connections to the network addresses they learn about. Therefore, a crawler has to be picked up by the seed nodes first.

Our implementation is written in the Go programming language and extensively uses the btcwire library[2] for Bitcoin peer-to-peer communication. To clearly state that it is not a regular node, the crawler uses the user agent string "btcwire" in its application-layer handshake. When exchanging a handshake with a seed node, a node needs to announce to relay blocks and to have more blocks than a threshold in order to be reliably recognized by the seed. After being noticed by seed nodes the crawler waits for incoming connections and records the message when it contains a filter. The connection is closed when a filter is received or when two minutes pass to free resources on both ends.

Beginning with the 12th of December 2014, we connected custom nodes to the Bitcoin peer-to-peer network and collected filters until we were satisfied with the number of collected filters. Over time we successively added up to 20 nodes. We ended the experiment at the 10th of February 2015 when 70,078 unique filters were captured.

Interestingly, the number of filters collected per day varied strongly within and between crawlers. Some crawlers received 0 to 20 filters every day, whereby others consistently collected 50 to 600 filters daily. This shows that the seed nodes selection of good peers is far from being uniformly random. At the end of the experiment, the probability that the nodes suggested by a seed (ranging from 21 to 27 at the time) include at least one of our crawlers was around 4.3%.

[2] btcwire package by Conformal: https://github.com/btcsuite/btcd

Chapter 4

Data Preparation

Using the collected Bloom filters we exploit their vulnerability to find actual wallet pubkeys. The term *match-set* refers to the pubkeys we extract from the filter and, accordingly, for each filter there is a match-set and an underlying wallet. In order to find the match-sets we create a set with all pubkeys that occur in the blockchain and use them to query each of the filters. When there is a match for both pubkey and hash the pubkey is regarded to be in the wallet.

Instead of querying all pubkeys from the blockchain we only use the pubkeys that were contained in the blockchain up to the moment of the collection of the individual filter. We obtain the blockchain pubkeys by finding all pay-to-pubkey or pay-to-pubkey-hash outputs and selecting the associated pubkey. Note that if an output is pay-to-pubkey-hash, we can find its destination pubkey only when it has been revealed by spending the associated output. Because almost all transactions of our wallets are pay-to-pubkey-hash, we can not reconstruct the current state of the user's wallet. Only pubkeys that occur in outputs that have been spent can be extracted from the filter. Our data set contains 60.88 million pubkeys using the blockchain state at the 10th February 2015 – the date of the collection of the last filter. We use the btcutil library[1] to load the serialized filter and query the set of pubkeys. This process takes approximately one hour per filter on a single processor core.

Apart from the false positive pubkeys that match the filter even though they were not put into the filter before, there are other reasons why our extracted match-sets could differ from the real wallet. First, putting a foreign pubkey into the filter allows tracking the balance of arbitrary addresses. This feature is called watch-only addresses and is implemented in BitcoinJ, but none of the actual wallet implementations we are going to analyse expose this feature in their graphical user interface. Second, a wallet implementation might shard its keys to multiple Bloom filters and send each peer a separate filter with only a subset of keys. While theoretically possible, to our knowledge there is no such implementation. Third, there could be nodes which deliberately broadcast

[1] btcutil package: https://github.com/btcsuite/btcutil

bogus filters to make Bloom filter analysis more difficult. However, in our view the public awareness of the vulnerability was fairly small at the time of filter collection. There is no protection against these issues other than considering only a subset of known wallet implementations and limiting the influence a single wallet can have on the final performance result. The only difference between wallet and match-sets we can estimate are the false positives that stem from the filter query, because we know the false positive rate of the filter.

At this point we have effectively obtained the match-sets from underlying wallets, but still need to clean the data from experimentation artifacts. First, we discarded all empty match-sets from the data set reducing its size to 55,111. Then we removed what we view as duplicate wallets which brought the number further down to 38,009.

As part of the initial handshake, clients exchange version messages containing a *user agent* string that identifies the client software. The user agents in our data are dominated by BitcoinJ-based software. Still, exact client versions were quite diverse and suggest a lot of custom software. We discarded wallets which did not present the user agent of a well known BitcoinJ wallet. So, we only retained Bitcoin Wallet, MultiBit, KnC, Hive and Green Bitcoin Wallet.

There are some match-sets that are clearly outliers because they have an enormous size, most likely caused by bugs or misconfiguration. Therefore, we have to pick an arbitrary threshold to remove the exceeding match-sets. A threshold of 1000 was considered suitable and affected 15 match-sets. Eleven of them were from from MultiBit and one from Bitcoin Wallet 4. Two of the wallets contained between 1000 and 2000 pubkeys, two contained between 2000 and 3000 and the rest had far more than 10,000 pubkeys. Finally, after outlier removal we ended up with 37,585 wallets.

In the rest of this chapter we are going to review the duplicate detection step.

4.1 Duplicate Detection

Each filter has a nonce which makes the filter unique even when the same elements are inserted. It is likely that two filters with the same nonce were issued by the same wallet. If two filters have the same nonce but a different content, then new pubkeys have probably been added to the wallet and therefore also to the filter. To account for that during the filter collection, we retained filters that had the same nonce but different content as an already existing filter. Furthermore, it is also very likely that some of our filters belong to the same wallets, even though they have a different nonce, size or content and thus also show up multiple times in our collection. This can happen if the wallet software restarts and generates a new filter with a fresh nonce. In order to improve the quality of the data we want to find and remove such duplicate match-sets that

have been issued by the same wallet. We apply a conservative approach which means that we prefer to throw away overly many match-sets over having the same wallet represented multiple times, which would bias the results. Therefore, we use an algorithm that treats all pairs of match-sets that overlap as duplicates and chooses to retain the one with the greater size because it is probably the most up to date.

Assuming the absence of key sharding, this method has the property that if two wallets are indeed duplicates it will also treat the match-sets as duplicates. The converse is not true, because a false positive in querying the filter can make a pair appear as duplicate. However, this does not harm our data, because we just throw away more than necessary and the probability of this event is reasonably small as the following calculation shows.

Denote a wallet and its corresponding match-set with w_i and m_i respectively, which are both sets of pubkeys, and the false positive probability of the wallet extraction $\Pr(p \in m_i \mid p \notin w_i)$ with φ. Let $\mathbb{P}$ be the set of all pubkeys in the blockchain and let $\approx$ be an equivalence relation for overlapping sets, so $x \approx y \iff |x \cap y| > 0$. Then we have that $\Pr(\neg(m_i \approx m_j) \mid w_i \approx w_j) = 0$ because as previously discussed Bloom filter queries do not result in false negatives. Also, we have

$$\begin{aligned}
\Pr(m_i \approx m_j | \neg(w_i \approx w_j)) &= 1 - \Pr\left(|m_i \cap m_j| = 0 \;\middle|\; |w_i \cap w_j| = 0\right) \\
&= 1 - \Pr\left(\cap_{p \in \mathbb{P}} \neg(p \in m_i \wedge p \in m_j) \;\middle|\; |w_i \cap w_j| = 0\right) \\
&= 1 - (1 - \varphi_i \varphi_j)^{|\mathbb{P}| - |w_i| - |w_j|}
\end{aligned}$$

For example, with the size of $\mathbb{P}$ being around 60E6 and the common parameter $\varphi = 0.000147^2$ it holds that the probability for erroneous de-duplication per pair is around 2.66×10^{-8}.

Chapter 5

Evaluation

At this point we have extracted a collection of match-sets, each corresponding to a wallet, and can use this data to evaluate the performance of the previously discussed clustering algorithms. Before we do that, we will have a more detailed look at the data we have collected.

One of the most important steps in the data preparation phase is to select only match-sets from well-known BitcoinJ wallets. But we have to look even more closely and actually distinguish two types of wallets because they have vastly different behaviors. The first type consists of wallet implementations using recent versions of BitcoinJ. Their filters commonly have false positive rates very similar to the previously assumed 0.0147%. We estimated the false positive rate by querying ten billion distinct strings and computing the ratio of matches. The second type uses older BitcoinJ versions where the false positive rate is negligible and we expect virtually no false positives. Since this change was introduced in BitcoinJ version 0.12, we call wallets using previous versions *legacy* wallets.

Legacy versions of BitcoinJ do not only have a much lower false positive rate but they function fundamentally different. First, they use always the first key in the wallet for the change output instead of a fresh pubkey. Second, they do not encourage using a fresh pubkey for each receiving transaction – a user has to manually request a new address. One of the effects is that the distribution of the number of pubkeys differs considerably between legacy and *modern* wallets. Figure 5.1 demonstrates that, in general, a modern wallet has much more pubkeys. In order to produce the figure, the distribution of pubkeys in modern wallets has been adjusted to incorporate false positives by subtracting for each individual wallet the expected number of false positives from the size of the match-set. Then we discarded match-sets with less than zero expected true pubkeys and applied the ceiling function to the remaining counts.

Table 5.1 shows the number of distinct wallet implementations in the data. We can see that Bitcoin Wallet for Android is by far the most common, followed by MultiBit, while KnC, Hive and Green Bitcoin Wallet play only a minor role. With the exception of Bitcoin Wallet 3, at the time, even the most recent versions of the client software listed as legacy wallets use an outdated BitcoinJ library.

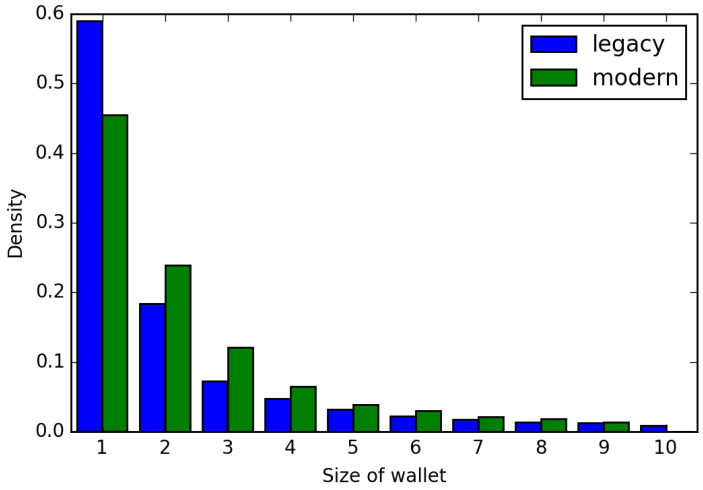

Figure 5.1: Distribution of the number of pubkeys in legacy vs. modern wallets.

Despite legacy wallets having the previously discussed critical privacy issues, they still represent 21.6% of the data.

Another interesting aspect is the balance of the wallets we captured. It is not possible to reconstruct the balance of modern wallets because their false positive rate would lead to an unpredictable bias. There are some legacy wallets which have a high false positive rate as well, so we select only wallets with a false positive rate smaller than 0.001%. Because most wallets have a much smaller false positive rate we expect no false positives in this selection, which turned out to contain 7777 wallets. In total, the sum of bitcoins received by the selected wallets is 321,825 with an average of 41 bitcoins. The distribution of total received bitcoins can be seen in Figure 5.2. It shows that more than half of the wallets received less than one bitcoin and the rest follows an extremely long tail. The maximum value that has been received by a single is 62,560 bitcoins.

In the following we assume that an attacker knows if her victim's wallet software is legacy or modern. This information can be acquired via side-channel information or in many cases simply by inspecting the blockchain. For example, the attacker could search for all transactions that spend the victim's outputs and then check if these transactions always send to the same pubkey. This is likely the change address of the victim's wallet and therefore the victim is running a legacy wallet.

We can not simply apply a clustering algorithm to a modern wallet because our match-sets contain enough false positives to substantially bias the result. Therefore, we will now first analyse legacy wallets, then learn how we deal with false positives and examine modern wallets.

Table 5.1: Number of wallets per useragent

(a) Legacy wallets

Wallet	Number
MultiBit 0.5.18	5621
MultiBit <0.5.18	1500
Bitcoin Wallet 3	619
Bitcoin Wallet <3	35
KnC	157
Hive	154
Green Bitcoin Wallet	38
total	8124

(b) Modern wallets

Wallet	Number
Bitcoin Wallet 4.18	5859
Bitcoin Wallet 4.17	4071
Bitcoin Wallet 4.16	15796
Bitcoin Wallet <4.16	3735
total	29461

In order to efficiently query the blockchain we created two key-value databases from the raw blockchain data. The first is an address index that maps an address to the identifiers of transactions where that address appears. Second, we have a transaction index that returns the transaction corresponding to a transaction id. For each application of a clustering algorithm we use the blockchain state at the moment of the filter collection. This prevents finding pubkeys via clustering that legitimately stem from the wallet, but were not found during wallet extraction because they were only later added to the wallet. We completely ignore inputs and outputs that do not follow the pay-to-pubkey or pay-to-pubkey-hash standard such as pay-to-script-hash. Also, we do not make use of the sighash flag, which could indicate a non-standard transaction that affects the clustering strategies in a negative way.

5.1 Metrics

The function $h : \mathbb{P} \to 2^{\mathbb{P}}$ denotes a clustering algorithm that maps a pubkey to a set of pubkeys which are likely to belong to the same wallet. h can be viewed as a binary classifier on the set of pubkeys which assigns positive to all pubkeys in $h(p)$ and negative otherwise. Thus, we can use standard metrics from binary classification: *precision* and *recall*. We define that $p \notin h(p)$, so the result of the clustering algorithm does not contain the input pubkey.

First, we want to compute the precision $\pi_h(p, w)$ of heuristic h on a pubkey p of wallet w to determine the fraction of new pubkeys found via h that really

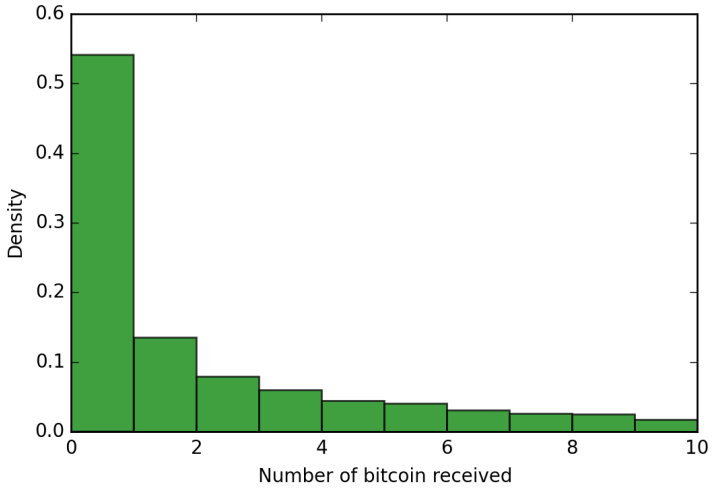

Figure 5.2: Distribution of total received bitcoins for a subset of legacy wallets.

belong to the wallet.

$$\pi_h : \mathbb{P} \times 2^{\mathbb{P}} \to \mathbb{Q}$$
$$\pi_h(p, w) := \frac{|h(p) \cap w|}{|h(p)|}$$

Because $p \notin h(p)$ we choose to compute the precision only over the pubkeys that were newly discovered with the heuristic. Otherwise, the result would be systematically biased since p would always be contained in $h(p)$ and w. This also means that $|h(p)|$ can be zero, so the precision is only defined for pubkeys for which the heuristic returns a non-empty result. When averaging over the keys of a wallet we get the per wallet precision π_h^ω.

$$\pi_h^\omega : 2^{\mathbb{P}} \to \mathbb{Q}$$
$$\pi_h^\omega(w) := \frac{1}{|\tau(w)|} \sum_{p \in \tau(w)} \pi_h(p, w)$$
$$\tau : 2^{\mathbb{P}} \to 2^{\mathbb{P}}$$
$$\tau(w) := \{p : p \in w \wedge |h(p)| > 0\}$$

The function τ filters the pubkeys of a wallet such that only pubkeys are kept which have a defined precision. When averaging over all wallets we get a final

precision score Π for the full data set.

$$\Pi_h : 2^{2^{\mathbb{P}}} \to \mathbb{Q}$$
$$\Pi_h(W) := \frac{1}{|T^\pi(W)|} \sum_{w \in T^\pi(W)} \pi_h^\omega(w)$$
$$T^\pi : 2^{2^{\mathbb{P}}} \to 2^{2^{\mathbb{P}}}$$
$$T^\pi(W) = \{w : w \in W \wedge |\tau(w)| > 0\}$$

If there are no pubkeys in the wallet with $|h(p)| > 0$ then the per wallet precision is undefined. So, we need a function T to only include wallets in the average we actually learn something from. Π can be interpreted as the probability that a pubkey returned by the application of h on a uniformly random pubkey chosen from a uniformly random wallet correctly belongs to the wallet.

The second metric is the recall $\rho_h(p, w)$ which represents the fraction of pubkeys of the wallet that are returned by the clustering algorithm h on p.

$$\rho_h(p, w) = \frac{|(h(p) \cup \{p\}) \cap w|}{|w|}$$
$$\rho_h^\omega(w) = \frac{1}{|w|} \sum_{p \in w} \rho_h(p, w)$$
$$P_h(W) := \frac{1}{|W|} \sum_{w \in W} \rho_h^\omega(w)$$

Note the subtle difference to the precision: for the recall it is more meaningful to add the pubkey p we started with to $h(p)$ because we are interested in the actual fraction of pubkeys the attacker obtained after application of the heuristic. This removes the need for a threshold filter τ that was used in the precision. Averaging over all wallets yields the probability that a pubkey belonging to the same wallet is returned by the heuristic.

5.2 Legacy Wallets

Legacy wallets are wallets that use a BitcoinJ version earlier than 0.12. They do not use fresh change pubkeys and users have to manually request new pubkeys to prevent address reuse. Applying the multi-input heuristic results in an average precision Π of 90.79%, and an average recall P of 79.76%. It's not surprising to see that the precision is pretty large, because the heuristics are designed conservatively and they are only applied to wallets the heuristics are made for.

Figure 5.3 shows the recall for different wallet sizes. It limits the x-axis to wallets with 20 pubkeys because the amount of data with bigger wallet sizes is

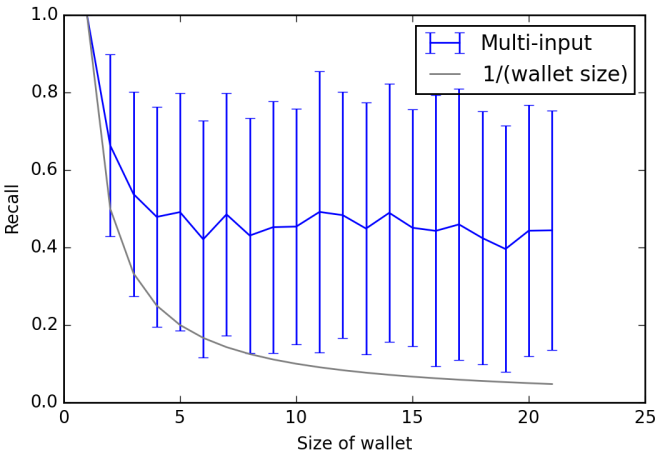

Figure 5.3: Mean recall and standard deviation of the multi-input heuristic on legacy wallets. The x-axis represents the number of pubkeys in the wallet.

not sufficient. The precision is not shown in the plot because it more or less remains constant for all wallet sizes.

In order to have a reference point to compare the multi-input heuristic to, we use a recall of 1/(wallet size) as a baseline. This represents the case when no heuristic is applied, so the attacker does not find more pubkeys than the single one she already has. The multi-input heuristic clearly beats the baseline and, interestingly, the recall does not really decrease for larger wallets. This can be explained by the general lack of fresh pubkeys in these wallets. Not only can we find a good deal of pubkeys that belong to the wallet, due to pubkey reuse there are more transactions that use the same pubkey compared to modern wallets. This means that even when the multi-input heuristic discovers only a few pubkeys, an attacker may learn about a lot of transactions that are received or sent by the wallet.

Note that there are a few wallets which have a false positive rate large enough to contain false positives. The total expected number of false positives in all 41,079 matched pubkeys is around 1605 and there are 332 match-sets which have more than 1% chance of having one or more false positives. However, the results are stable even when excluding these wallets.

It is not sensible to apply the shadow heuristic to legacy wallets because the heuristic assumes that the change pubkey has never been used before. Further, the results of the multi-input are satisfying enough so that we do not try to apply more heuristics to the legacy wallets. It should be clear that legacy wallets have a negative impact on privacy and users should use modern wallets instead.

5.3 Modern Wallets

We are mostly interested in analysing modern wallets. They represent the majority of the data, implement best practices and are what almost all users are going to use in the future. The relatively high number of false positives has prevented us so far from simply applying the clustering algorithm similarly to the legacy wallet case.

Having false positives vastly increases the computing time because the heuristic may involve a large number of transactions or return large sets of pubkeys, each of which is going to be recursively fed into the heuristic. There are, for example, pubkeys in the blockchain which occur in more than three million transactions. In order to achieve an acceptable performance, we apply caching to look up as few transaction as possible and use early stopping criteria to avoid spending a lot of time on one of those popular false positives.

The effects of false positives on both precision and recall are quite diverse. In the following example we can see how the recall changes in the face of false positives. Assume we apply the heuristic to a true positive of the wallet. Then, in the presence of false positives, the true recall would be greater or equal than the recall we compute naively. On the other hand, applying the heuristic to a false positive will very likely not find any new pubkeys of the wallet, because the false positive is just a random key in the blockchain. This means that the recall is 1/(wallet size) and thus lowers the average recall of the wallet. Additionally, it is very likely that we have wallets in our data that do not contain any pubkey, but the filter match some false positives. Then we average over a wallet that really should not be included at all and depending on the number of false positives can lower or increase the total average over all wallets.

We are going to focus on the recall and will not compute the precision for modern wallets. This is because the precision is not expected to decrease substantially due to the design of the heuristics and because the false positive correction would be more complicated than for the recall.

5.3.1 Dealing with false positives

We created a statistical model to deal with the false positives found in modern wallets. The problem is that it is not possible to tell which exact pubkeys are false positives. The only thing we know is the false positive rate of the filter which determines the distribution of the number of false positives in the match-set. So, we use some assumptions to be able to correct the recall if the number of false positives is known. We then apply a Monte Carlo method to sample the number of false positives for the whole collection and compute the corrected recall.

As previously, let each $w_i \in W$ be a set of pubkeys which represents the

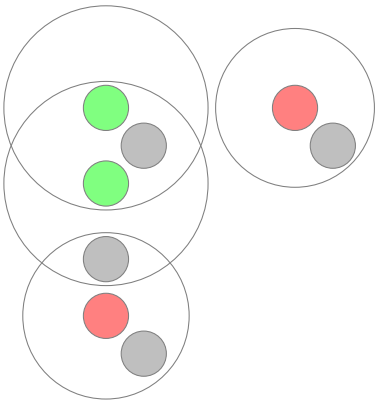

Figure 5.4: Visualization an exemplary wallet to which the assumptions apply. A green circle represents a true positive, a red circle represents a false positive. Grey circles represents pubkeys not contained in the match-set. The big empty circles include the pubkeys that were found by applying a clustering algorithm to the filled circle in the center.

i-th wallet in our dataset. However, w_i is unknown – we only know the set of pubkeys $m_i \in M$ that matched the corresponding filter. Due to the nature of Bloom filters, match-sets have no false negatives and we can write $m_i = w_i \cup \phi_i$, $w_i \cap \phi_i = \emptyset$ where ϕ_i is the set of false positives of the i-th match-set. The goal is to estimate $P_h(W)$, but we only know M instead of W. We do not know the set of false positives but only the distribution of its size.

Our solution is, first, to make some assumptions to be able to represent $P_h(W)$ as a function of M and $|\phi|$. Then, we repeatedly sample from the probability distribution of $|\phi_i|$ and compute the recall.

The assumptions we are going to make are the following:

$$\forall p \in m_i \, \forall p' \in \phi_i \backslash \{p\} \quad p \notin \left(h(p') \cup p'\right) \qquad \text{Assumption 1}$$
$$\forall p \in \phi_i \, \forall p' \in m_i \backslash \{p\} \quad p \notin \left(h(p') \cup p'\right) \qquad \text{Assumption 2}$$

This is means that (1) the result of applying the heuristic to a false positive pubkey does not contain any other match and (2) the heuristic does not find a false positive if applied to any other match (see also Figure 5.4). These assumptions are justified by the fact that there are only a few false positives and they are uniformly distributed over the complete set of pubkeys $\mathbb{P} \backslash w_i$. Therefore, the probability that they are somewhere 'close' to each other or to a pubkey in the true wallet w_i is extremely small.

Note that the validity of the assumptions depends on the actual instantiation of the heuristic. A heuristic that for any input simply returns all pubkeys of the blockchain would obviously violate these assumptions. The heuristics we are

going to explore are quite conservative and in practice only return a very small fraction of the possible pubkeys.

Using the assumptions we can compute the per wallet recall ρ_h^ω of wallet w_i only with m_i and $|\phi_i|$. We define $\hat{\rho}_h^\omega(m_i, |\phi_i|)$ to compute ρ_h^ω with the known arguments.

$$\rho_h^\omega(w_i) = \frac{|m_i|^2}{(|m_i| - |\phi_i|)^2} \left(\rho_h^\omega(m_i) - \frac{|\phi_i|}{|m_i|^2} \right) := \hat{\rho}_h^\omega(m_i, |\phi_i|)$$

Proof.

$$\begin{aligned}
\rho_h^\omega(m_i) &= \frac{1}{|m_i|} \sum_{p \in m_i} \rho_h(p, m_i) \\
&= \frac{1}{|m_i|} \left(\sum_{p \in w_i} \rho_h(p, m_i) + \sum_{p \in \phi_i} \frac{|\,(h(p) \cup p) \cap m_i|}{|m_i|} \right) \\
&= \frac{1}{|m_i|} \left(\sum_{p \in w_i} \rho_h(p, m_i) + \frac{|\phi_i|}{|m_i|} \right) && \text{Assumption 1} \\
&= \frac{1}{|m_i|} \left(\sum_{p \in w_i} \left(\frac{|\,(h(p) \cup p) \cap w_i|}{|m_i|} + \frac{|\,(h(p) \cup p) \cap \phi_i|}{|m_i|} \right) + \frac{|\phi_i|}{|m_i|} \right) \\
&= \frac{1}{|m_i|} \left(\sum_{p \in w_i} \frac{|\,(h(p) \cup p) \cap w_i|}{|m_i|} + \frac{|\phi_i|}{|m_i|} \right) && \text{Assumption 2} \\
&= \frac{|w_i|^2}{|m_i|^2} \rho_h^\omega(w_i) + \frac{|\phi_i|}{|m_i|^2} && \text{assuming } |w_i| > 0 \\
\rho_h^\omega(w_i) &= \frac{|m_i|^2}{|w_i|^2} \left(\rho_h^\omega(m_i) - \frac{|\phi_i|}{|m_i|^2} \right)
\end{aligned}$$

□

At this point we want to average over all wallets to get $P_h(W)$. Note that we previously assumed that there is at least one true positive in w_i, because otherwise $\rho_h^\omega(w_i)$ is not defined. This means that when averaging over the match-sets M we have to skip those that only consist of false positives. We introduce the function $T^\rho(M) := \{m_i : m_i \in M \wedge |\phi_i| < |m_i|\}$ to select the match-sets that do not exclusively consist of false positives.

$$P_h(W) = \frac{1}{|W|} \sum_{w_i \in W}^{n} \rho_h^{\omega}(w_i)$$
$$= \frac{1}{|T^{\rho}(M)|} \sum_{m_i \in T^{\rho}(M)} \hat{\rho}_h(m_i, |\phi_i|)$$

Since $|\phi_i|$ are random variables and therefore also $T^{\rho}(M)$ we are only able to compute a probabilistic estimate for the recall from our data.

$|\phi_i|$ is following a binomial distribution $\mathbb{B}$ with $|\mathbb{P}|$ number of trials, where P is the set of all pubkeys in the blockchain. The success probability of a trial is determined by the false positive rate of the filter φ_i. Using the pubkey extraction method of section 3.2 the probability of having a single false positive is the filter's squared false positive rate. Technically, we would have to use $|\mathbb{P}| - |w_i|$ as the number of trials, but $|w_i|$ is negligible in relation to $|\mathbb{P}|$.

$$|\phi_i| \sim \mathbb{B}(\varphi_i^2, |\mathbb{P}|)$$

Fortunately, we can place upper bounds on $|\phi_i|$. It is clear that $|\phi_i|$ can not exceed $|m_i|$, but we can also apply our assumptions to eventually find a tighter bound. Every pubkey p in ϕ_i must fulfill the following conditions: No other pubkey in m_i will be returned by applying the heuristic to p and applying the heuristic to any key in m_i does not return p (see also Figure 5.4). Using the assumptions we can find a set of false positive candidates $c(m_i)$ and use its size as an upper bound for $|\phi_i|$.

$$c(m_i) := \{p : p \in m_i \wedge (\forall p' \in m_i \backslash \{p\} \;\; p \notin (h(p') \cup p') \wedge p' \notin (h(p) \cup p))\}$$

Thus we have for the probability distribution of $|\phi_i|$:

$$\Pr\left(|\phi_i| = k \;\middle|\; |\phi_i| \leq |c(m_i)|\right) = \frac{\Pr\left(|\phi_i| \leq |c(m_i)| \;\middle|\; |\phi_i| = k\right) \Pr(|\phi_i| = k)}{P(|\phi_i| \leq |c(m_i)|)}$$
$$= \frac{b(k, |\mathbb{P}|, \varphi_i^2)}{B(|c(m_i)|, |\mathbb{P}|, \varphi_i^2)} \qquad \text{for } k \leq |c(m_i)|$$

where $b(k, n, p)$ the probability mass function and $B(k, n, p)$ the cumulative density function for k successes in n Bernoulli trials with probability p.

We've seen that knowing the distribution of $|\phi_i|$ allows computing a probability distribution for the true recall. This would involve lengthy operations like inverting the Poisson binomial distribution of $T(M)$ and dealing with dependent variables. Therefore, we resort to a Monte Carlo method by sampling from $|\phi_i|$, correcting the measured P and recording their distribution.

5.3.2 Results

Table 5.2 shows the results of applying the heuristics to modern wallets. We correct for false positives using the method described in the previous section by sampling false positives a thousand times, computing for each the average recall P and taking their mean. The confidence interval is created by taking the 0.5-percentile and 99.5-percentile of P.

We compare against the baseline 1/(wallet size) which essentially means applying no heuristic such that the recall for each pubkey is 1/(wallet size). It is apparent that the multi-input heuristic is a substantial improvement over the baseline. The observation that it is much more effective on legacy wallets is consistent with our previous discussions on legacy and modern wallets.

Table 5.2: Mean corrected recall and 99% confidence intervals of applying the clustering algorithms to modern wallets and correcting for false positives. See section 5.3.2 for a description of the heuristics.

Heuristic	mean recall	99% confidence interval
1/(wallet size)	66.27%	[65.83%, 66.80%]
Multi-input	68.59%	[68.17%, 69.05%]
Shadow	69.16%	[68.77%, 69.58%]
Consumer	69.26%	[68.85%, 69.73%]
Optimal	69.34%	[68.93%, 69.80%]
Best	70.94%	[70.55%, 71.30%]

The change heuristics are always applied in combination with the multi-input heuristic. This means that for every transaction that is looked up with the multi-input heuristic, we also check if we can find the change. Instead of applying each change heuristics only with the multi-input heuristic, we use the following combinations:

Shadow shadow only

Consumer shadow and consumer heuristic

Optimal shadow, consumer and optimal heuristic

They are combined so that if the first heuristic is not able to determine the change, the second heuristic is applied and so on. One can see that the shadow heuristic is able to improve the result, but the other two change heuristics only help in some cases. However, when we applied the consumer heuristic without the shadow heuristic in test evaluations on smaller data, we saw a significant effect. So, it is reasonable to assume that the minor role of the consumer heuristic in combination with the shadow heuristic can be explained by a correlation between wallets which create transactions with more than two outputs and wallets that

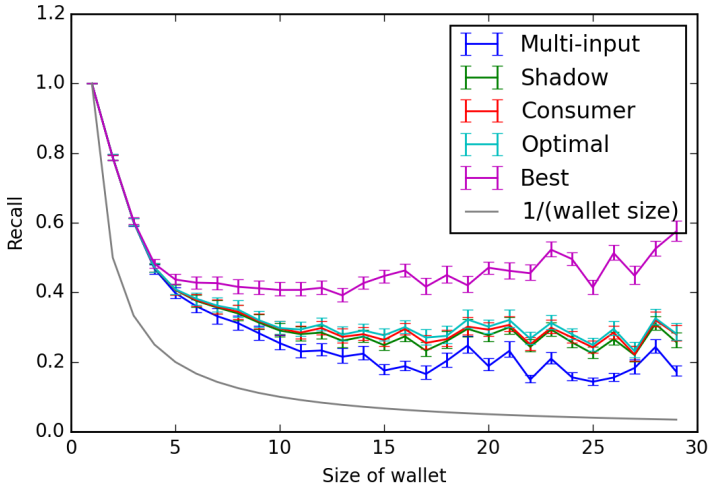

Figure 5.5: 99% confidence intervals for the mean recall after false positive correction of various heuristics on modern wallets. See section 5.3.2 for a description of the heuristics.

reuse addresses. In total, the Consumer heuristic improved the result for 1027 wallets and the Optimal heuristic helped in another 1009 cases.

The "Best" heuristic in the table refers to the result that is theoretically possible with the multi-input heuristic and a perfect change detection. This is the upper bound on the recall we can achieve with our heuristics.

In practice, we can obtain the best recall using all heuristics at once. It shows that on average an attacker equipped with one pubkey will find 69.34% of pubkeys of the wallet. The distribution of wallet sizes is exponential – almost half of the wallets only contain one pubkey and the recall on these wallets is always 1, independent of the heuristic. Therefore, the mean recall does not appear to be a significant improvement over multi-input alone. Figure 5.5 demonstrates that for many wallet sizes the improvements are substantial. We can see that the heuristics are drastically better than the 1/(wallet size) baseline and that there is still quite some room to the upper bound. Interestingly, in contrast to legacy wallets the recall decreases with larger wallets.

Chapter 6

Discussion

There is significant interest in blockchain analysis both from a research and commercial standpoint. On the other hand, Bitcoin developers constantly develop new ideas to increase the privacy of users. We are going to show how users can defend themselves against the heuristics discussed in the project and how the heuristics can be improved.

There are two further ways to enhance our clustering strategies. We use the change heuristics – shadow, consumer and optimal – only in one direction. Given a known pubkey, only transactions were considered where an output for the starting pubkey is redeemed. What can additionally be done is to find transactions which send to an output with the pubkey. Using the heuristic assumption it is checked if the pubkey is the change and if so, all inputs of the transaction are returned. By being able to step both in forward and backward direction, such a modification can increase the recall considerably as further experiments have shown. The problem is, however, that the heuristic assumptions are applied to the wallet implementation that creates the transactions and we do not know which exact implementation it is. For example, assume there is a transaction from a legacy wallet that sends to a pubkey from a modern wallet, which we use to apply the bidirectional shadow heuristic to. Legacy wallets do not satisfy the shadow assumption because their change address is reused. It is likely that the modern wallet's pubkey will be considered change and all transaction inputs are returned even though they belong to the legacy wallet. So, applying the heuristic without information about the sending wallet implementation seriously harms the precision. It is possible to mitigate that effect by trying to learn about the wallet's behaviour. One strategy is to test if all input pubkeys satisfy the heuristic assumption. If there is, for example, a transaction from that wallet without a shadow output then we would not assume a shadow output in the original transaction and stop the heuristic from going backwards. Furthermore, it might be possible to classify the wallet implementation by finding identifying behaviour in the blockchain or acquiring side-channel information.

The second idea to enhance the clustering strategy is to apply the underlying assumptions recursively to potential change. This is possible with the shadow

and consumer assumption, where we can clearly conclude that a wallet does not satisfy the assumption if one of the pubkeys is used to sign a transaction that has no shadow output or no consumer candidate. In general, with high probability one of the outputs of a transaction is the change and the wallet software spending the change is the same as the one creating the original transaction. By recursively applying the assumptions, we check if the wallet implementations that are spending the outputs have the same behaviour. If this only applies to a single output then it is very likely the change. Assume, for example, that an attacker obtains a pubkey from a modern wallet. She applies the shadow heuristic and finds that there is an output for the pubkey and it is spent in a transaction, which has itself two outputs, o_1 and o_2. She can not determine which output of the transaction is change, because the associated pubkeys never showed up in the blockchain before. But she can examine the transactions spending the outputs o_1 and o_2 and check if there is at least one shadow output. For instance, if the transaction redeeming o_1 does not have a shadow output then o_2 must have been the change output because o_1 was clearly spent from another wallet. Similarly, the shadow or consumer assumption could have been applied further if o_1 had a shadow output.

6.1 Mitigation

Blockchain analysis clearly affects all Bitcoin wallets. But it has been confirmed by our results that a simple measure to defend against clustering strategies is to use a modern wallet. While it is often difficult to determine if wallet software reuses the change address from the documentation or user interface, everybody can check his transactions using web interfaces to the blockchain. It is often recommended to not reuse addresses by creating a new address for each received payment. If this is not done, an attacker can find multiple transactions when looking up an address, learns a lower bound of the victims balance and can try to identify all parties involved. Without doubt, it is helpful to maintain the discipline to not reuse addresses by having wallet software that automatically displays a new address upon receiving a transaction to the previously displayed address. The modern wallet we discussed before, namely Android Bitcoin Wallet 4, has this feature. Still, with the current Bitcoin protocols address reuse can not be prevented when an address is published, for example, to receive tips or donations. Adjusting the coin selection strategy can help to reduce the negative effects of address reuse. The coin selection could ensure that all outputs that are addressed to the same pubkey are spent in the same transaction.

What this work has clearly shown is that individual privacy is not only affected by own actions, but also by the sending and receiving parties involved. If they would not reuse addresses, the shadow heuristic would not be able to distinguish the change output from the intended receiver. But we have seen that

the shadow heuristic is a substantial improvement over the multi-input heuristic alone. This shows that address reuse is prevalent which is problematic because the individual can not defend against that. Similarly, the consumer heuristic exploits knowledge about victims to distinguish them from the general population. In contrast to the shadow heuristic, to mitigate the consumer heuristic it is not enough to demand from the receiving party to follow best practice. The consumer heuristic would be completely defeated if everybody would just create transactions with one or two outputs. Where previously the receiving party would have created a single transaction with many outputs it would now create multiple transactions. However, due to transaction fees, creating multiple transactions is economically disadvantageous and so the receiving party is incentivized to be distinguishable from a consumer wallet. In conclusion, to counteract this class of clustering strategies that try to distinguish different wallet implementations it is necessary to make wallets behave as uniform as possible.

In section 2.5 we discussed coinjoin transactions. It is obvious that the multi-input heuristic becomes useless when coinjoin is widespread. Unfortunately, there is no wallet software implementing automatic coinjoins so far – only a few toy implementations. It seems as if coinjoin is straightforward to implement and does not really have a downside. Coinjoin does not require trust because a user only signs if she is satisfied with the transaction, so she does not loose control over her coins at any point. The difficulty is to find peers in a decentralized fashion and prevent being trivially vulnerable to denial of service attacks. If, for example, one party denies to sign the coinjoin transaction, the whole coinjoin process has to be restarted, which involves finding peers, creating a transaction and broadcasting it. Another obstacle is achieving resistance against sybil attacks, which means that an attacker creates a large number of identities. Even when the coinjoin peers are selected uniformly at random there is a chance that a victim chooses many of the attacker's identities, while believing that she participates in a coinjoin with multiple different parties. The attacker can subtract his inputs and outputs from the resulting transaction and therefore has a higher chance of guessing the victim's output compared to a passive blockchain observer. Also, there is nothing in the naive protocol that prevents a peer from learning an input-output pair while the coinjoin transaction is created. This can be simply achieved by capturing the transaction at multiple stages. Then the peer can determine the input-output pairs that have been added by the parties in between. The coinshuffle protocol [RMSK14] solves this by using layered encryption when the user adds her output. A crucial problem, especially for usability and the user interface, is that every participant of a conjoin needs to use the same amount of bitcoin. Otherwise, one can try to detect the output associated to an input by finding the output with the most similar value. One proposal by Gregory Maxwell that can solve this is called *confidential transactions* and aims to extend the Bitcoin protocol to enable encryption of transaction values.

Confidential transactions also prevent the optimal change heuristic because

a blockchain observer can not find out the output values. Until confidential transactions is a reality, the coin selection process can prevent finding the optimal change most of the time. It would have to be instructed to never create a transaction with a unique output whose value is smaller than any of the input values. This might entail that more inputs have to be added and it is therefore in conflict with other optimization objectives like minimizing fees.

The clustering algorithms we covered in this project are not effective against Bitcoin *tumblers*. Tumbling refers to the process of trading outputs for the output of another party. A tumbler is usually an entity that accepts transactions from the user and send unrelated coins back to a previously communicated address of the user. However, at the moment tumblers surely have problems like being trusted centralized infrastructure. Additionally, an attacker who knows the transaction to the tumbler can observe the blockchain for some time to find an output with a similar value going back to the victim [BNM+14]. There is a proposal called *coin swap* that enables trustless peer-to-peer tumbling, but it has not yet gained traction.

In section 3.2 we came to the conclusion that the Connection Bloom Filtering vulnerability is not likely to get fixed in the near future. In principle, everybody is free to change the hard coded false positive rate constant in a BitcoinJ derivative, compile and test how much it affects the bandwidth in ones own use case. Unfortunately, the only practical option for most users who care about their sensitive transaction data being exposed the peer-to-peer network is to avoid wallets with Bloom filters. The type of SPV wallets not using Bloom filters relies on central servers to forward transaction data. The central server learns all addresses of the wallet but unlike Connection Bloom Filtering, this is just a single known entity instead of anonymous random peers in the Bitcoin network. In some cases the server address can be configured in the client and the server software is freely available. This allows to have a trusted friend that runs a server for the user. If such a friend is not available and SPV is not required then it is preferable to use a full node implementation such as Bitcoin Core.

Chapter 7

Conclusion

Our objective for this project is to assess the effect of blockchain analysis on the privacy of Bitcoin participants. We are able to do this with the help real wallet data captured from the peer-to-peer network.

This thesis started out by revisiting known and introducing two new clustering heuristics. We demonstrated how Connection Bloom Filtering used in some SPV wallets is vulnerable to leaking all public keys of a wallet including a few false positives. Using the vulnerability we captured 37,585 filters in two months and therefore showed that mass exploitation is feasible with low cost. While analysing the data we learned that we have to distinguish legacy and modern BitcoinJ wallets because their behaviours differ fundamentally. To our surprise, 7777 legacy wallets alone have received 321,825 bitcoins and there are some high value targets, one of which received 62,560 bitcoins. Without surprise, however, we have seen that the multi-input heuristic alone can retrieve on average 79.76% of pubkeys belonging to the wallet and thereby seriously impact the user's privacy.

Extracting pubkeys from filters that were generated by modern wallets always results in some false positives, so we had to develop a statistical method to estimate precision and recall that accounts for false positives. The results show that even modern wallets are not able to defend users against said clustering strategies. An attacker can learn on average 68.59% pubkeys of the wallet with the multi-input heuristic alone, up to 69.34% by combining clustering strategies and 70.94% in the theoretical case of having perfect change detection.

Bibliography

[AKR+13] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *Financial Cryptography and Data Security*. Springer, 2013.

[BKP14] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in bitcoin p2p network. In *Proceedings of the 2014 Conference on Computer and Communications Security (CCS)*. ACM, 2014.

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.

[BNM+14] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography and Data Security*. Springer, 2014.

[GCKG14] Arthur Gervais, Srdjan Capkun, Ghassan O Karame, and Damian Gruber. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Computer Security Applications Conference (CSAC)*. ACM, 2014.

[HC12] Mike Hearn and Matt Corallo. Bip 0037: Connection bloom filtering, 2012. https://github.com/bitcoin/bips. [Online; accessed July 29th, 2015].

[KKM14] Philip Koshy, Diana Koshy, and Patrick McDaniel. *An analysis of anonymity in bitcoin using p2p network traffic*. Springer, 2014.

[MPJ+13] 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 *Internet measurement conference (IMC)*. ACM, 2013.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf. [Online; accessed July 29th, 2015].

[OKH13] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the bitcoin transaction graph. *Future internet*, 2013.

[RMSK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2014.

[SMZ14] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. Bitiodine: Extracting intelligence from the bitcoin network. In *Financial Cryptography and Data Security*. Springer, 2014.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Data-driven Deanonymization in Bitcoin

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Nick	Jonas

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 08.08.2015	J. Nick

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.