Privacy Aspects and Subliminal Channels in Zcash*

Alex Biryukov
University of Luxembourg
alex.biryukov@uni.lu

Daniel Feher
University of Luxembourg
daniel.feher@uni.lu

Giuseppe Vitto
University of Luxembourg
giuseppe.vitto@uni.lu

ABSTRACT

In this paper we analyze two privacy and security issues for the privacy-oriented cryptocurrency Zcash. First we study shielded transactions and show ways to fingerprint user transactions, including active attacks. We introduce two new attacks which we call *Danaan-gift* attack and *Dust* attack. Following the recent Sapling update of Zcash protocol we study the interaction between the new and the old zk-SNARK protocols and the effects of their interaction on transaction privacy. In the second part of the paper we check for the presence of subliminal channels in the zk-SNARK protocol and in Pedersen Commitments. We show presence of efficient 70-bit channels which could be used for tagging of shielded transactions which would allow the attacker (malicious transaction verifier) to link transactions issued by a maliciously modified zk-SNARK prover, while would be indistinguishable from regular transactions for the honest verifier/user. We discuss countermeasures against both of these privacy issues.

CCS CONCEPTS

• **Security and privacy** → **Distributed systems security**; *Privacy-preserving protocols*; Cryptanalysis and other attacks; • **General and reference** → *Empirical studies.*

KEYWORDS

privacy, blockchain, Zcash, zk-SNARK, NIZK, subliminal channel

ACM Reference Format:
Alex Biryukov, Daniel Feher, and Giuseppe Vitto. 2019. Privacy Aspects and Subliminal Channels in Zcash. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3319535.3345663

1 INTRODUCTION

Cash-like privacy is one of the key properties to be implemented in modern blockchain-based cryptocurrencies. Bitcoin [16], with its pseudonymous transactions, while initially believed to offer payment privacy, was shown to suffer from transaction graph analysis and linkability issues [12, 21], mainly due to the public nature of its ledger. Aiming at addressing this aspect, there has been a rise of *privacy-preserving cryptocurrencies* such as Dash [1], Monero [2], Zcash [3]. Each of these is using different privacy enhancing technologies. The simplest one is used by the blockchain Dash, which uses so-called masternodes with built-in mixers to provide privacy. Monero's ring signatures provide another form of transaction mixing, which does not require central nodes, while also hiding the values of transactions. These ring signatures are a cryptographic primitive which provides a valid signature for a group of n users where only one user has to use his/her secret key together with the public keys of the other members. Externally it is impossible to tell which group member signed the message. For transactions, that means that multiple outputs can be spent at the same time without the ability to distinguish which signer used a secret key and which ones the public keys.

The last technology that is in use for privacy preserving blockchains are zk-SNARKs, which stands for Zero-Knowledge Succinct Non-interactive ARguments of Knowledge. This technology offers provable security, and requires a Common Reference String (CRS), which is the central trusted piece of data. In the case of Zcash blockchain the CRS was generated once at the launch of the chain, in a distributed multi-party computation (MPC) with trusted peers. Zcash launched on 28 October, 2016 as the first commercial release of zk-SNARKs. Zcash itself is based on Bitcoin style UTXO (unspent transaction output) system of tracking coins. It has a public and a private portion. The public part works exactly the same way as in Bitcoin. The private part uses zk-SNARKs and transactions that use zk-SNARKs are called *shielded* transactions. The two parts are in constant interaction with each other, which introduces some exploitable privacy leakage. One of the problems with zk-SNARK technology was that, in the first version, the creation of a proof took 40 seconds and 1.5GB of memory, so for usability reasons this blockchain kept transparent Bitcoin-style transactions as default. In October 2018 Zcash has added a new and much faster zk-SNARK protocol called Sapling in which a proof takes 3 seconds and requires 40MB of memory.

In this paper we describe attacks against privacy in the current Zcash protocol. First we show how past linkability techniques ([9, 20]) could be improved based on value matching and value fingerprinting. Then we introduce two new active attacks on user privacy. Both of these attacks require the knowledge of the target's hidden address. In the first approach, which we called the *Danaan-gift* attack, the adversary is donating small tainted amounts of Zcash to the target's shielded address in hope that the tainted value would remain when the value would be de-shielded. This attack is helped by the fact that Zcash transaction values (like Bitcoin) can be fractioned to a very high precision, so that the last four digits of a Zcash value have no economic significance but could be used as a fingerprint. The second attack is called the *Dust* attack, and is based on the observation that in a shielded transaction the number

*This work was supported by the Luxembourg National Research Fund (FNR) project FinCrypt (C17/IS/11684537).

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3345663

of inputs and outputs is visible in the new Sapling release of Zcash. The attacker can send a large set of micropayments to the target shielded address which would allow to track one extra payment hop involving this address inside the shielded transactions pool. Both of these attacks require transfer of small amount of coins to the target's hidden address. We discuss several countermeasures against such attacks.

The second category of tagging attacks that we explore in this paper is based on subliminal channels [26, 27, 30]. With a subliminal channel an attacker can *reveal b* bits of arbitrary information out of a cryptosystem, using system parameters that were not originally designed to exchange such information. This means that subliminal channels, if they exist, become a hidden part inside the cryptosystem and if the attacker controls the affected parameters, he can freely decide whether to send a secret message or not. Such hidden communication can be used to reveal secret keys or user IDs. Even worse, to maintain confidentiality of the subliminal channel, the attacker can easily use encryption to permit only some receivers that know a pre-shared secret to retrieve the plaintext message, while the others cannot even detect that any message was sent. While subliminal channels can be present in many cryptosystems, we believe that in privacy-focused setting it is very important to be aware of their existence and properties due to severe consequences for the user privacy and fungibility (for cryptocurrencies).

In Zcash we discovered three different subliminal channels, that can be used depending on what cryptosystem the attacker is able to attack: the Inner and the Outer Subliminal Channels can be used assuming that the zk-SNARK proof generation mechanism (but not the secret master key!) is under the control of the attacker, while the Pedersen Subliminal Channel requires the control of the randomness source in the commitment scheme adopted to hide values of shielded inputs and outputs. Such subliminal channels can be exploited, for example, by closed-source lightweight wallets which are used on mobile devices. Another possible attack points are computation servers (or hardware devices) which are used for zero-knowledge proof generation.

Alongside the descriptions of these subliminal channels, we provide an example of how an attacker can exploit them to permit external entities, that we will refer to as "Subliminal Verifiers", to distinguish transactions created by the same user and to decommit/reveal all shielded transaction amounts. We then discuss countermeasures to prevent the exchange of *subliminal messages*, or alternatively methods to disrupt their contents.

1.1 Related Work

There has been constant research on privacy of blockchains since the introduction of Bitcoin, which can be split into two main categories. The first category is protocols that try to improve the cryptocurrency privacy. These works consider direct extensions of Bitcoin with mixing services as in [6, 7, 11, 24, 28]. Other works used new protocols, like ring signatures adopted in the Monero blockchain [17, 29], also for mixing purposes. Yet other works introduced new protocols that used zero-knowledge proofs, such as [14, 25], and Zcash is based on these papers as well.

The second category studied privacy in existing blockchains trying to find new attacks and privacy issues. Such works on Bitcoin include [4, 5, 13, 22, 23]. There have also been works on the Monero blockchain, such as [10, 15, 31], which show the drawback of using mixing-style privacy.

Relevant for this paper, there has also been previous work on denanonymizing or linking transactions in Zcash. The first such work was by Quesnelle [20], where the direct information leakage of public values in a shielded transactions was first exposed. That work connected hiding and revealing transactions, where the two values were identical, and the number of blocks between the hide-reveal transactions was short. This approach was later examined in more detail and extended with other clustering techniques in [9], where they showed that most of the mining related shielded transactions can be linked, while also providing anonymity analysis for some shielded transactions connected to the hacker collective "The Shadow Brokers".

In this paper we further investigate linkage of hiding and revealing transactions based on their specific values which we call *value fingerprints*, and also show active attacks against user's privacy, first based on empirical measurements, then based on completely different ideas, using the cryptographic subliminal channels in the zk-SNARK protocol. Finally, it is important to note, that all the work done in this paper was done in an ethical way, never targeting or storing linkage data of individual users. For the confirmation of active attacks we did it on our own transactions.

2 INTRODUCTION TO ZCASH

Zcash is a privacy preserving blockchain based on Zerocash [25], using practical zero-knowledge proofs called zk-SNARKs. Zcash has a public open part, which exactly mimics Bitcoin's ledger based on unspent transaction outputs (UTXO), while it has a parallel hidden part. Transactions that use zero-knowledge proofs are called *shielded* transactions. The public key of a public address starts with the letter "t", while a shielded address starts with "z". Transactions between public and shielded addresses are possible, but the amount of hidden or revealed coins from or to the public addresses becomes public. In the rest of the paper we refer to a shielded transaction as a hiding transaction if it moves coins from a public to a shielded address, and to one that moves coins from a shielded to a public address as a revealing transaction. The exact amounts are referred to as hiding and revealing values.

Zcash was launched on 28 October, 2016 with the release called Sprout. This version's zk-SNARK proof mechanism took 40 seconds and required 1.5GB of memory. In the rest of the paper we will refer to these shielded transaction as *Sprout* transactions. On 29 October, 2018 the first major update to Zcash's proof system called "Sapling" was deployed. This update reduced the time to create a zk-SNARK to 3 seconds, while the memory requirement was reduced to 40MB. In the rest of the paper we will refer to these shielded transactions as *Sapling* transactions. Sprout and Sapling addresses and transactions use completely different elliptic curves and proof protocols, which makes them incompatible with each other. This also means that they are easy to distinguish while looking at the Zcash blockchain. Interaction between Sprout, Sapling and transparent transaction pools is shown in Figure 1. Our studies for this paper were performed at Zcash blockchain block height 472,285 which corresponds to 29 January, 2019.

2.1 Technical Details

Coins in Zcash are referred to as ZEC, while the smallest possible value is called Zatoshi, where 1 ZEC = 10^8 Zatoshi[1]. Zcash uses a UTXO based ledger both in its public and shielded setting. In order to create a new output with a specific value, one must consume previously unspent outputs, where the value sum of these has to be larger or equal to the value sum of the desired new outputs. The only transaction that can create new value without having to consume a previously unspent output is the so called coinbase transaction, which is always the first transaction in a block. Its base value is fixed in the protocol of the blockchain, and the miner of the block can add extra value equal to the sum of fees from all the other transactions in the block. In a general transaction if the sum of the values from the consumed outputs is larger than the new outputs, the difference can be claimed in the coinbase transaction by the miner as the transaction fee. In the rest of the paper we will refer to the consumed or spent outputs as inputs of a transactions, while to the newly created unspent outputs simply as the outputs of a transaction. Every output is connected to a public key which is also called an address. The output can be only spent with the corresponding private key of the address.

When a new coin is created and rewarded to a Zcash miner, the miner can only claim the coin by transferring it to a shielded address first. This is an attempt from the Zcash developers to have every coin shielded at least once. However most of the miners/mining pools quickly move their coins back to transparent addresses. Thus majority of the hiding and revealing transactions are of this origin.

Sprout based shielded transactions use a mechanism called joinsplit, which combines two previously unspent shielded outputs and creates two new shielded unspent outputs, while also being able to hide or reveal a clearly noted amount of ZEC. The limitation of this protocol apart from its efficiency is the fact that even if a user wants to send only 1 shielded output to another address, a dummy input and a dummy output still needs to be created to fill out the rest of the input and outputs of a joinsplit. Similarly, if a user wants to spend three unspent outputs, he will have to use two joinsplits in the same transaction.

In the rest of the paper we will refer to public transactions that do not involve any zk-SNARKs as t-to-t transactions and will refer to purely shielded transaction as z-to-z transactions. Furthermore, if a transaction is hiding coins, i.e. it is transferring coins from a public to a shielded address, we refer to it as a t-to-z transaction, and similarly to a revealing transaction as z-to-t.

2.2 Introduction of Sapling Shielded Transactions

At block height 419,200 (29 October, 2018) the Sapling hard fork of Zcash took place, which introduced a new elliptic curve and zk-SNARK protocol. It is a major improvement in terms of efficiency. On the other hand, as the new zk-SNARK protocol uses a new curve, it is not backwards compatible, meaning that if a user wants to send coins from a Sprout to a Sapling address, it has to reveal the value in-between, which is currently only possible with the involvement of public addresses, see Figure 1. This means that

[1]At the moment of this writing 1 ZEC is about 70 USD.

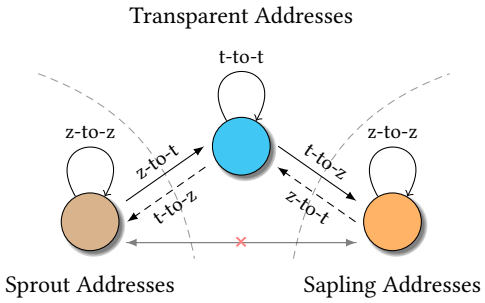

Figure 1: Sapling Turnstile

transactions between Sprout and Sapling addresses are visible, as they have to use an in-between t-address.

Sapling shielded transactions have also abandoned the joinsplit structure of the zk-SNARKs and they reveal exactly how many shielded inputs and outputs (these inputs and outputs are sometimes referred to as shielded notes as well) they have as a side-channel information. The developers did consider including mandatory dummy inputs and outputs, but they decided against it in order to reduce the average transaction size. Figure 3 explains the Sapling transaction layout.

We have checked how many values were directly linkable by already known techniques from [9, 20] (Heuristic 1), and out of the 241 outputs we have only found 4 of them that were uniquely linkable (exact and unique value matched some earlier unique Sprout input), meaning most of the users were aware of this attack and/or have accumulated coins from multiple transactions on the same address.

A natural question is whether there is also traffic the opposite way, from Sapling to Sprout. We have found 26 such transactions, where the output of a Sapling transaction is then an input to a Sprout shielded transaction. The total value of these transactions was 226 ZEC, of which 200 ZEC was in just three transactions. The reason for these transactions could be that the recipients have not updated their clients yet.

3 TRANSACTION LINKING

We have looked into ways of linking transactions, where we connect hiding and revealing values of shielded transactions. For this part, we have first removed all the relatively easily identifiable transactions concerning miners and mining pool payouts using the methods presented in [9, 20], which resulted in 92,233 hiding and 107,772 revealing transactions as a hard-core set of remaining hidden transactions.

Heuristic 1. If a hiding and a revealing value are exactly the same, their values are unique as hiding and revealing values in the observed block range and the hiding transaction is in an earlier block than the revealing transaction, then they are considered to be linked.

Using the direct match approach on these remaining transactions we find that 8,954 revealing shielded transactions out of 107,772 are uniquely matchable considering the entire Zcash blockchain.

To acquire a false positive rate for the heuristic, we deployed the following measurement. First we measure the number of unique transaction output values in the first half of the blockchain (up to block 236,142). Then we measure how many of these values stay unique in the second half of the chain. We found, that 14.7% of these values lose their uniqueness (769,071 out of 5,230,325). This means that direct value matching heuristic has false positive rate of just 14.7% over the half-blockchain duration (13 months), and less over shorter windows of time.

3.1 Direct Value Linking Including the Transaction Fees

We have also found a handful of interesting transactions, where both the hiding and revealing value is unique, but the hiding value is larger by exactly 10,000 Zatoshi, which is the default transaction fee. This leads us to a likely explanation, that the value was also moved once as a z-to-z transaction, and then the receiver is revealing it to the public. Either a user moved the coins to himself between shielded addresses wrongly assuming that this way he gets more anonymity, or more plausibly a change of ownership happened for the coin (and also not gaining anonymity compared to a change of ownership happening in the plain sight with the t-addresses).

We have investigated and extended the observation above to the case where instead of a direct value match there is a difference in value which is a multiple of the default transaction fee of 10,000 Zatoshi. This could correspond to a value making several hops inside the shielded pool before being revealed, each time losing 10,000 Zatoshis. From this the observer can also conjecture the number of hops that the value made.

Heuristic 2. If a hiding value is $n \cdot 10,000$ Zatoshi ($n < 10$) larger than a revealing value, their values are unique as a hiding and revealing value in the observed block range, the hiding transaction is in an earlier block than the revealing transaction and the transactions have not been linked for any $k < n$, then they are considered to be linked.

If we extend the possible linkable values with this technique, the original 8,954 unique links are increased to 9,919, which is a 10.8% increase considering only 1 hop. Interestingly the number of unique links does not increase by much more by increasing the number of hops (only 400 more links with up to 9 hops[2]). This seems to confirm our previous observation that many users think of shielded pool as a perfect cryptographic anonymizer and think that single z-z hop inside the shielded pool is sufficient. This expectation of gained privacy however is not true, if the value passing through the shielded pool was unique or rare in the entire chain. This also leads us to an idea of value fingerprinting described in the later section.

3.2 Subset sum

We have investigated the usefulness of subset sums for linking shielded transactions by connecting a single hiding transaction to multiple revealing transactions, or vice versa. The idea is to check if users hide their value in a single transaction, but reveal it over time in multiple payments, or similarly hide their coins in multiple phases, while revealing it in a single transaction.

First, we had to consider how many numbers can be summed up overall. If we consider the average number of remaining shielded transactions, we see that about 2,000 coin revealing and hiding transactions remain for every 10,000 blocks after removing the mining transactions. The smallest unit of account in Zcash is 1 $Zatoshi = 10^{-8}$ ZEC. Then any transaction value can be in the range of $[1, \ldots, 10^{14}]$ Zatoshis. If we consider the number of possible combinations of 2,000 inputs or outputs, then it is easy to see that even $\binom{2000}{3} = 1.3 \cdot 10^9$ which is well above the birthday bound of all possible values (which is around 10^7). This means, that some sums of just 3 values could be just due to a random match. Nevertheless this could be still of interest since even in case of collisions we can reason in terms of sizes of anonymity sets (which would still be relatively small).

The technique did find a handful of interesting matches for 2 sums, especially in cases where one member of the sum is a value with multiple non-repeating non-zero digits, while the other is a round value (e.g. consider a transaction with the input value 3.54156325 ZEC and the two outputs with the values 0.40002 ZEC and 3.14154325 ZEC[3]). This led to a further analysis idea, which is explored in the following sub-section.

3.3 Fingerprinted Values

We have found another promising technique for connecting different hiding and revealing values that had no direct connections so far. We will use an approach that we call *value fingerprints*. In our definition, the fingerprint of a transaction value is its last 7 digits in Zatoshis. In particular the last 4 digits are especially stable as a fingerprint since this value is below the typical transaction fee of 10^4 Zatoshis (which is currently below 1 US cent). Thus they usually have little economic meaning and represent just a remnant of previous transactions. The distinguishability of a fingerprint depends on its entropy, which in this case describes how rare the value is. Intuitively, round values are much more frequent than random values. It is worth noting that, in a regular economy, the digits below the fee threshold would typically stay zero. However this is not the case in the blockchain world, where mining pool payouts are in most cases computed with full precision thus creating random distribution in the least significant digits and which can be used for transaction fingerprinting. This is somewhat similar to the serial numbers on the paper cash banknotes, with a difference that the precision is not sufficient to keep them unique.

Heuristic 3. If a hiding value matches fingerprints with a revealing value, no other hiding or revealing value matches fingerprints with either of them in the observed block range and the hiding transaction is in an earlier block than the revealing transaction, then they are considered to be linked with the fingerprinting technique.

We consider two fingerprints to match if either 5 of the last 7 digits are the same, or all last 4 digits are equal. Fingerprints where the last 4 digits are round (e.g. '0000', '1000') are disregarded.

In the Section 3.1 the heuristics linked transactions where one of the values in the sum or the difference of the values is exactly

[2]Our maximum allowed since 10 hops would be 100,000 Zatoshi, which is a common value on its own.

[3]These are fictitious values to preserve the privacy of the actual transaction.

10,000 Zatoshi or a multiple of it. These transaction are linked with the fingerprint technique as well, since the last 4 digits remain the same.

3.3.1 Longevity of Fingerprint Uniqueness. One concern with this method is that the possible fingerprint set is too small, resulting in no unique links if we consider the entire chain for pairs of hiding and revealing transactions. In order to check the relative strength of the linkage, we have investigated with several block ranges how many unique pairs do we observe. Table 1 describes our results in terms of block ranges and the number of unique links.

The block range is a sliding window of blocks, where if the range is N, we consider hiding fingerprints in the first $N/2$ blocks and we check for matching revealing fingerprints in the last $N/2$ blocks. Once a matching pair is found, it is not considered in next slidings. The valid transaction set is the same hard-core set as we used for the case of exact value linking.

Sliding Window Block Range	Unique Fingerprint Links
20	5,448
100	9,436
500	9,681
1,000	9,733
2,000	9,761
4,000	9,808
8,000	10,335
12,000	10,613
16,000	10,642
24,000	10,363
32,000	9,736
64,000	6,833

Table 1: Number of unique fingerprint matches through the entire chain for a given range of blocks.

We will call this feature the *longevity of fingerprint uniqueness*. As seen in Table 1 the largest unique matching was on a sliding window of 16,000 blocks (which is approximately 4 weeks), while we received similar results between 8,000 and 24,000 blocks as well. This means that the average longevity of a fingerprint (aka "serial number") staying unique is approximately between 2-6 weeks. Note that the database is the same as with the exact value matching. If we disregard the direct unique value matches (Section 3.1), on the range of 16,000 blocks, we receive a number of 7,228 unique matches (Appendix C, Table 8) out of 107,772 total transactions.

Another approach for determining the longevity of fingerprint uniqueness is examining public transactions to determine how long the fingerprint of an output stays unique. We have implemented this approach by calculating the average number of transaction outputs it takes for a duplicate of a fingerprint to appear considering all (except mining) transactions in the chain. In this case we have recorded a fingerprint from an output, traced and saved the connected chain of transactions where this fingerprint still exists, and then checked when does the fingerprint appear again from an unrelated transaction output.

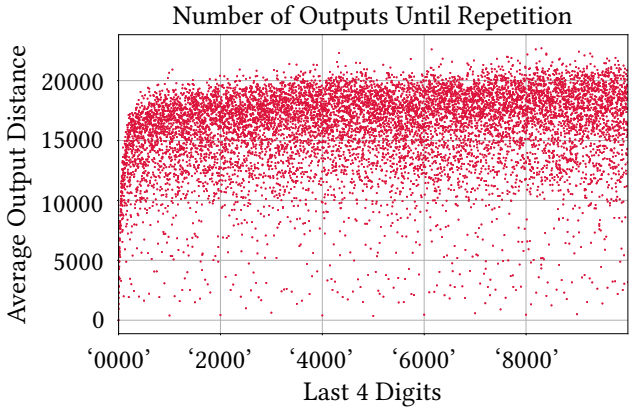

Figure 2: Average output distance until a fingerprint (last 4 digits of the transaction value) stays unique.

The result of this experiment was that the average number of outputs until a fingerprint is unique was 15,979, while the median was 16,788 outputs. Then we have approximated the average number of shielded outputs in a shielded transaction from the Sapling transactions (where this information is visible) and calculated the average number of shielded outputs per block, which is 0.95. Then by dividing these two values we get 16,820 blocks for the average and 17,671 blocks for the median value, which is in line with our measurements from Table 1. We have also created a plot, shown in Figure 2, which for every last 4 digit fingerprint represents the average number of outputs generated by the blockchain during which the fingerprint stays unique (after we removed all the mining-related transactions). From this figure we see that a "good fingerprint" stays unique during generation of about 16000 new outputs by the blockchain, which is about one month. In Table 1 the decrease in matches after increasing the window to 24,000 blocks is in line with our measurements for the longevity of fingerprint uniqueness, as the likelihood of two random revealing values having the same fingerprint significantly increases, while the table reports only the unique matches.

4 THE MODEL FOR THE PROBABILITY OF FINGERPRINT SURVIVAL

We now present a statistical model which describes the probability that a fingerprint remains unmodified through several shielded transactions. More precisely, from the public and shielded transactions data, we will model the hidden *walk* of the fingerprinted value through different hops inside the shielded pool (i.e. shielded transactions) before being revealed in a z-to-t transaction. We'll then use this model to compute the survival probability of fingerprints, i.e. the probability that a value that enters the shielded pool exits with its fingerprint unmodified. As shown in Figure 2, not all fingerprints are equally likely and hence we cannot talk about the survival probability of *any* fingerprint: we will then refer only to fingerprints -the *good fingerprints*- that have an average output distance greater than 10,000, i.e. fingerprints above the $y-$axis value of 10.000 in Figure 2, and we will assume these to be equally likely. The following model is built accordingly.

We denote with $\mathbb{P}(FP)$ the overall survival probability of any good fingerprint going through a path inside the shielded pool and then exiting with a revealing transaction and we let $\mathcal{Z}$ be a discrete random variable which counts the number of transaction-hops[4] inside the shielded pool over some path. We then denote with $\mathbb{P}(\mathcal{Z} = n)$ the probability that a walk inside the shielded pool goes through exactly n hops before exiting. We further denote with $ZPaths(n+1)$ the set of paths of length $n+1$ whose first n hops are inside the shielded pool and the last hop is an exit z-to-t hop. We can then model the survival probability of good fingerprints as

$$\begin{aligned}\mathbb{P}(FP) &= \sum_{n\geq 0} \mathbb{P}(\mathcal{Z}=n)\cdot\mathbb{P}(FP \mid \mathcal{Z}=n)\\ &= \sum_{n\geq 0} \mathbb{P}(\mathcal{Z}=n)\cdot\left(\sum_{x\in ZPaths(n+1)} \mathbb{P}(FP \mid x)\cdot\mathbb{P}(x)\right)\end{aligned}$$

In order to estimate $\mathbb{P}(FP)$ under this model, we computed the following values:

- $Zlen$: the average number of hops a path goes through inside the shielded pool before exiting. Assuming each hiding transaction is independent, we set $Zlen$ to be equal to the ratio between the number of z-to-z transactions and the number of hiding transactions. For more details see Appendix A.2.
- $\mathbb{P}(\mathcal{Z}=n)$: we modeled this probability using a Poisson distribution of parameter $Zlen$;
- $\mathbb{P}(FP \mid x)$ for any path $x \in ZPaths(n+1)$: the probability that given the path x the fingerprint survives, i.e. the product of the probabilities that the fingerprint survives in each transaction of x. The *per-transaction type* survival probabilities are obtained analyzing the fingerprint survival rate in transparent transactions that have the same number of inputs/outputs as the considered shielded and revealing ones;
- $\mathbb{P}(x)$ for any path $x \in ZPaths(n+1)$: the probability that x occurs, i.e. the product of probabilities to have each transaction-type occurring in x. The distribution of the different types of transactions in the three categories t-to-z, z-to-z and t-to-z is directly obtained from the blockchain.

More details on the underlying assumptions and how all these values can be estimated are reported in Appendix A.

Within this model we estimated that the average number of hops a path goes through inside the shielded pool is only $Zlen = 1.42$ and the survival probability of good fingerprints is $\mathbb{P}(FP) \approx 16.6\%$.

5 DANAAN-GIFT ATTACK (MALICIOUS VALUE FINGERPRINTING)

Fingerprints (Section 3.3) can be used as a tool for linking the hidden and revealed values of shielded transactions. In this section we show that in some scenarios they can be exploited for transaction tagging by an active attacker.

Suppose the attacker is trying to identify the spending of a public address, which converts all its ZECs to hidden addresses regularly. The attacker can transfer a very small but carefully chosen amount of Zatoshis to this specific address, hoping that it leaves the trail of a fingerprint when they are converted from a hidden to a public address. As the attacker sees the current public value on the address, he sends a chosen value such that the resulting sum has a detectable and possibly long-living (Section 3.3.1) fingerprint. Of course different fingerprints could be used for different addresses. Afterwards the attacker only has to monitor the revealing transactions for his set of fingerprints. He may also enhance it using the subset sum approach and reason in terms of anonymity sets rather than unique matches.

Such attack can be performed against entities which accept public donations (e.g. WikiLeaks) since for them receiving money from an unknown source would look less suspicious. Moreover the attacker may monitor the address and resend the fingerprint in case another donation erases his old tag.

Our statistical model can be applied to provide a chance for success against an average user. As we show in Sections 4, A.2 the probability for a fingerprint to survive if it is revealed in some way is around 16%. This means that the attack succeeds with a 16% chance if the target behaves as an average user. Compared to the low cost[5] of the attack, we consider it as a real danger for Zcash users. As a countermeasure one should avoid de-shielding unique or rare values and should zero the digits below the transaction fee threshold. In the long run depreciation of the public t-address pool should solve the problem.

6 DUST ATTACK

Sapling shielded transactions reveal their in/out degree, i.e. the exact number of spent and output notes. The output notes would be the unspent transaction outputs (UTXOs) in Bitcoin terminology. This also means that if a user converts all of his/her shielded funds to a public address from a previous output note, it will show in the transaction that there are no new output notes, i.e. all the value from the spent notes are public. Shielded transactions that spend more than 10 outputs are very rare (35 out of 22,249 in the three months of the study period).

Using this information, we discovered an attack that would be able to track one extra hop of shielded spending. If that spending does not have public outputs, we can only verify when did the object of the attack use its funds. The attack is the following.

First, the attacker learns the shielded address of the target user either by buying services from a user who has a shielded address or donating to an entity who accepts shielded Zcash. Then the attacker transfers funds in many small valued output notes. This can be done in two ways. The first way is by issuing many separate transactions to a target address either from separate addresses, or from the same address by slicing small values one after another. The second way is by transferring the funds in a single transaction with lots of spend notes. This is not supported directly by the official wallet (it returns an error as it does not allow the same address to appear more than once as an output even in a shielded transaction). However a custom wallet where this check is removed can easily do so. We have tested it on the Zcash testnet and the transaction was accepted.

Later, if the target user wants to spend its shielded coins, it has to create a shielded transaction where the number of spend notes will

[4] It is important to think of transactions as hops, since it is inside the transaction that a fingerprint may be spoiled.

[5] Considering that 10,000 Zatoshis are worth 0.7 US Cents, if the exchange rate is 70 USD.

be unusually high which can be monitored on the chain. At present, in the official wallets the user can not choose which outputs it wants to spend when creating a transaction. The wallet, instead, orders all available unspent shielded outputs connected to the address in a decreasing manner and picks them one-by-one until it has enough value to cover the desired total output value.

This means that even if the user notices this unusual behaviour in his/her wallet, there is currently no way to avoid these dust-spending outputs. As a *countermeasure*, the user should always do an extra hop inside the shielded pool if his z-address received transactions from external sources. This would consolidate all the outputs into a single spend note, removing the dust-tag. Such transaction itself would be noticed by the attacker, but no further tracking is possible if there were no public outputs in that transaction. This countermeasure does not require any customization from the wallet as it is a simple shielded transfer, but it is dependent on whether the user notices the attack. We have executed the attack on our own addresses on the Zcash testnet[6].

Heuristic 4. An attacker tags a target address with more than $n > 10$ dust outputs. If later he observes that the input-degree of a shielded transaction is at least n, then he links the transaction to the target address.

Let us now investigate how the attack is presented (based on our experiment) to the target in the different Sapling-supporting wallets.

6.1 Official Linux Command-line Zcash Wallet

As the official Linux RPC wallet is command line based, the user will only notice this behaviour by specifically checking for it. There are separate commands for getting the total balance for all addresses or a specific one and, similarly, a command for listing the received outputs per address (which includes the already spent ones as well) and another one to get all currently controlled unspent outputs under all addresses. Thus, it depends on the users whether they check the correct attributes of their wallet and if they will notice the strange dust values. If a user checks for the received transactions, all outputs will show as separate transactions, even though having the same ID.

On the other hand, by listing the received transactions or unspent outputs, one can see the IDs of the transactions and, if the attacker sent all the outputs in the same transaction, this might alert the user.

6.2 GUI-based Sapling-supporting Wallets

All the currently existing GUI wallets show the recent transactions on the home page of the wallet. This means that all the dust outputs are shown as separate transactions, even when they were sent in the same shielded transaction. On the other hand, these programs only show the time of the transactions. The user does not see the transaction details (not even the id) in the software, instead it has to copy the transaction to receive the id or go through a link to an online explorer to discover that they are exactly the same transaction. On the other hand from the time of the transactions it is visible that all of them were made at the same time, which could also be an indicator of malicious behaviour.

6.3 Combining Danaan and Dust Attacks

An attacker might combine these two attacks, where some of the dust values also contain a fingerprint, and their sums are fingerprinted as well. This way, even if the dust attack follows only one hop of shielded values, the fingerprint later might still reveal the values when and if the coins are revealed.

7 SUBLIMINAL CHANNELS

The Zcash Sapling protocol introduced many new features. Among these, it implements the *"Decoupled Spend Authority"* that, quoting from the official pre-release note[7], enables *"enterprises [to] perform an inexpensive signature step in a trusted environment while allowing another computer, not trusted with the spending key, to construct the proof. Additionally, hardware wallets can support shielded addresses by allowing the connected computer to construct the proof without exposing the spending key to that machine"*. To motivate why this could be a security and privacy issue, consider the following two scenarios. In the first one the zk-SNARK proof generation is delegated to a computation server (or hardware) that is able to surreptitiously embed extra tagging information in the generated proofs. Whenever it happens, we have a subliminal channel, that we will refer to as *Inner Subliminal Channel*. In the second scenario, lightweight closed source wallets are able to embed subliminal information into already generated valid zk-SNARK proofs exploiting their malleability. We will refer to such channels as *Outer Subliminal Channel*.

In the Zcash Sapling protocol we found subliminal channels that exist in both the zk-SNARK proof generation mechanism and the adopted commitment scheme. More precisely, we found an Inner Subliminal Channel and an Outer Subliminal Channel, described in the following Sections, which are related to the currently implemented zk-SNARK scheme. We further found another channel, that we called Pedersen Subliminal Channel, which refers to the Pedersen Commitment scheme used.

For each type of subliminal channels found, the techniques to embed a subliminal message are similar: for this reason we chose to describe in detail the embedding of a subliminal message and its retrieval only for the subliminal channels found during and after the zk-SNARK proof generation.

After describing the computational complexity of the proposed embedding procedures, a concrete attack scenario is discussed and some countermeasures are proposed to prevent the use of such subliminal channels. A brief discussion on the effectiveness and efficiency of these channels to tag real Zcash transactions then follows.

To clarify the context in which the subliminal channels found in this paper work and how they can be exploited to exchange subliminal messages, we'll briefly describe how transactions are created and stored on the Zcash blockchain.

The input of a Zcash Sapling transaction consists of a sequence of *Spend Descriptions* and *Transparent Inputs*, while its output consists of a sequence of *Output Descriptions* and *Transparent Outputs*. It

[6] txid: 48b364e082f90ae5860ad52a876eae37c84ed0cbb7cf4279dea2fd2a243bacb5

[7] https://z.cash/blog/whats-new-in-sapling/

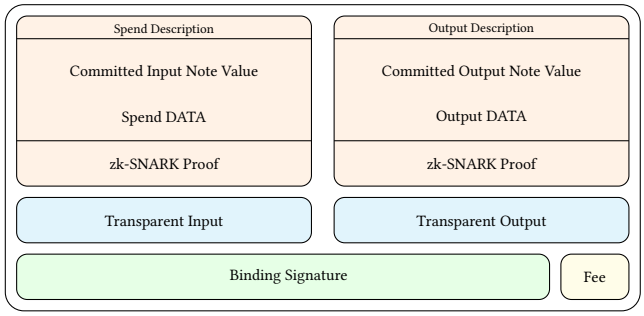

Figure 3: Sapling Transaction Layout

is up to the user to create fully shielded transactions consisting of only Spend and Output Descriptions, fully transparent ones or transactions that result a combination of them (see again Figure 1).

An example of a Sapling transaction is shown in Figure 3, where one Spend Description, one Transparent Input, one Output Description and one Transparent Output are included. This simplified layout will guide our description through the main elements of a transaction. While a Spend Description refers to a previous transaction note, which is spendable only if the corresponding *spending key* is known, an Output Description corresponds to a new one.

Each note contains a *note value* which accounts for the total amount of ZEC involved: while in Transparent Input/Outputs the note value is publicly readable, in the case of Spend and Output Descriptions, it is hidden in a form of a Pedersen Commitment [19]. Note value commitments are included, along with some other *DATA*, in the respective descriptions they refer to and allow - thanks to a Binding Signature - to publicly verify the total transaction balancing value.

Each description is finalized by appending a zk-SNARK proof that assures value commitment integrity, spend authority (as in the case for Spend Descriptions), double-spending prevention and other protocol coherence requisites.

8 THE ZK-SNARK SUBLIMINAL CHANNELS

We now show how Provers compliant with the Sapling zk-SNARK proof generation protocol can exchange extra b-bits of information with protocol-compliant Verifiers. Since these bits are exchanged by using parameters employed during the proof generation that were not intentionally designed for communications, we will refer to these communication channels as subliminal channels [26, 27, 30] and to the exchanged messages as subliminal messages.

We will describe in details two separate constructions: the *Inner Subliminal Channel* and the *Outer Subliminal Channel*. In the Inner one, the message is embedded in the zk-SNARK proof during its generation, while with the Outer Subliminal Channel a subliminal message is embedded in an already generated valid proof before it is signed.

When these subliminal channels are used, the resulting proofs will be valid and indistinguishable from any other valid proof for the same statements if some *auxiliary* information remains unknown to the Verifier.

8.1 Groth's NIZK argument

In order to set out the definitions and notations employed in the construction of the proposed subliminal channels, we briefly recall Groth's NIZK argument [8] for arithmetic circuit satisfiability currently adopted by Zcash Sapling. The setting is as follows:

- Consider a relation generator $\mathcal{R}$ that returns relations of the form

$$R = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, l, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X))$$

  where $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ is a bilinear group of order p and g, h are generators, respectively, of $\mathbb{G}_1$ and $\mathbb{G}_2$.
  The relation defines a language of statements $(a_1, ..., a_l) \in \mathbb{Z}_p^l$ and witnesses $(a_{l+1}, ..., a_m) \in \mathbb{Z}_p^{m-1}$ such that with $a_0 = 1$

$$\sum_{i=0}^{m} a_i u_i(X) \cdot \sum_{i=0}^{m} a_i v_i(X) = \sum_{i=0}^{m} a_i w_i(X) + h(X)t(X)$$

  for some degree $n-2$ polynomial $h(x)$.

A Prover that generates a proof π for statement $(a_1, ..., a_l)$ should be able to convince any Verifier that he knows the corresponding witness $(a_{l+1}, ..., a_m)$ without revealing any information related to the witness.

In Groth's scheme, both proof generation and verification are done in the Common Reference String model. The Setup, Prover and Verifier procedures are as follows:

- $\sigma = (\sigma_1, \sigma_2) \leftarrow \text{Setup}(R)$:
  - Pick $\alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_p^*$ and compute

$$\sigma_1 = \left(\begin{array}{c} g^{\alpha}, g^{\beta}, g^{\delta}, \{g^{x^i}\}_{i=0}^{n-1}, \left\{g^{\frac{x^i t(x)}{\delta}}\right\}_{i=0}^{n-2}, \\ \left\{g^{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}}\right\}_{i=0}^{l} \end{array} \right)$$

$$\sigma_2 = \left(h^{\beta}, h^{\gamma}, h^{\delta}, \{h^{x^i}\}_{i=0}^{n-1}\right)$$

    $\sigma = (\sigma_1, \sigma_2)$ is referred to as *Common Reference String*.
- $\pi \leftarrow \text{Prover}(R, \sigma, (a_1, ..., a_m))$:
  - Pick $r, s \leftarrow \mathbb{Z}_p$ and compute $\pi = (A, B, C)$ where

$$A = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta} \qquad B = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta}$$

$$\hat{C} = g^{\frac{\sum_{i=l+1}^{m} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta}}$$

$$\hat{B} = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta} \qquad C = \hat{C} \cdot A^s \cdot \hat{B}^r \cdot g^{-rs\delta}$$

- $0/1 \leftarrow \text{Verifier}(R, \sigma, (a_1, ..., a_l), \pi)$:
  - Compute

$$T = g^{\frac{\sum_{i=0}^{l} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\gamma}}$$

  - Parse $\pi = (A, B, C)$ and accept the proof if and only if

$$e(A, B) = e(g^{\alpha}, h^{\beta}) e(T, h^{\gamma}) e(C, h^{\delta})$$

8.2 The Inner Subliminal Channel

Suppose that a Prover, hereafter called *"Subliminal Prover"*, wishes to send b-bits of information to a Verifier, that from now on we will refer to as *"Subliminal Verifier"*. A Subliminal Prover cannot directly communicate to a Subliminal Verifier, because otherwise there is no need to use subliminal channels: when a message is embedded, the Prover doesn't know when it will be recovered and he cannot warn the Verifier to recover the message from a specific proof rather than another.

To overcome these limitations, in our constructions the Subliminal Prover and Verifier share some secret auxiliary information, denoted as *aux*, that permits the Verifier to distinguish a proof with a subliminal message from an honest random proof generated for the same statements.

We let $\omega : \{0,1\}^* \to \{0,1\}^b$ and $\xi : \{0,1\}^* \to \{0,1\}^{\frac{b}{2}}$ be two uniformly distributed polynomial-time computable functions. The abstract purpose of ω is to obtain the message to embed from the statements of the proof, while ξ acts as a message extractor, that is it recovers parts of the embedded message from the proof. To both bind a subliminal message to some specific information and to permit Subliminal Verifiers to recognize proofs with subliminal messages embedded, we require that both ω and ξ take as input the auxiliary information *aux*.

In short, for statement $(a_1, ..., a_l)$, we let

$$M = \omega(aux, a_1, .., a_l)$$

be the message that the Subliminal Prover would like to send to a Subliminal Verifier, assuming they both know ω, ξ and *aux*.

As opposed to the Subliminal Prover, the Subliminal Verifier takes as input a set $\Omega = \{\omega_i\}_{i \in I}$, for some index set I. Each function ω_i corresponds to a different method the Subliminal Prover can use to obtain the subliminal message, especially in the case where the Prover uses a unique $\omega = \omega_i$ associated to a user.

In order to embed and recover the subliminal message M, the Prover and Verifier procedures are therefore modified as follows:

- $\pi \leftarrow$ SubliminalProver$(R, \sigma, (a_1, ..., a_m), \omega, \xi, aux)$:
  - Compute $M = \omega(aux, a_1, .., a_l)$ and denote with m_1, m_2 the first and the last $\frac{b}{2}$ bits of M, respectively.
  - Randomly select $r \leftarrow \mathbb{Z}_p$ until
  $$\xi\left(aux, a_1, ..., a_l, \left(g^{\alpha+\sum_{i=0}^{m} a_i u_i(x)}\right) \cdot \left(g^{\delta}\right)^r\right) = m_1$$
  - Randomly select $s \leftarrow \mathbb{Z}_p$ until
  $$\xi\left(aux, a_1, ..., a_l, \left(h^{\beta+\sum_{i=0}^{m} a_i v_i(x)}\right) \cdot \left(h^{\delta}\right)^s\right) = m_2$$
  - With the r, s obtained compute $\pi = (A, B, C)$, where
  $$A = g^{\alpha+\sum_{i=0}^{m} a_i u_i(x)+r\delta} \qquad B = h^{\beta+\sum_{i=0}^{m} a_i v_i(x)+s\delta}$$
  $$\hat{C} = g^{\frac{\sum_{i=l+1}^{m} a_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))+h(x)t(x)}{\delta}}$$
  $$\hat{B} = g^{\beta+\sum_{i=0}^{m} a_i v_i(x)+s\delta} \qquad C = \hat{C} \cdot A^s \cdot \hat{B}^r \cdot g^{-rs\delta}$$
- $(0/1, M) \leftarrow$ SubliminalVerifier$(R, \sigma, \pi, (a_1, ..., a_l), \Omega, \xi, aux)$:
  - Parse $\pi = (A, B, C)$, and let
  $$m_1 = \xi(aux, a_1, ..., a_l, A) \qquad m_2 = \xi(aux, a_1, ..., a_l, B)$$
  - If exists an $\omega \in \Omega$ such that $\omega(aux, a_1, ..., a_l) = (m_1||m_2)$, recover M as
  $$M = (m_1||m_2)$$
  Otherwise, set $M = \perp$.
  - Return (Verifier$(R, \sigma, (a_1, ..., a_l), \pi)$, M)

8.3 The Outer Subliminal Channel

In Groth's original scheme and in its corresponding Zcash Sapling implementation, a user that possesses, for a certain statement, a valid proof $\pi = (A, B, C)$ is able to transform it into a different, but still valid, proof $\pi' = (A', B', C')$ for the same statement.

The proof can be transformed using the following transformations, exploiting the multiplicative nature of the proof structure:

- Select random $\tilde{r} \in \mathbb{Z}_p$ and set
$$A' = A^{\tilde{r}} \qquad B' = B^{\frac{1}{\tilde{r}}} \qquad C' = C$$
- Select random $\tilde{s} \in \mathbb{Z}_p$ and set
$$A' = A \qquad B' = B \cdot (h^{\delta})^{\tilde{s}} \qquad C' = A^{\tilde{s}} \cdot C$$
- Select random $\tilde{r}, \tilde{s} \in \mathbb{Z}_p$ and set
$$A' = A^{\tilde{r}} \qquad B' = B^{\frac{1}{\tilde{r}}} \cdot (h^{\delta})^{\frac{\tilde{s}}{\tilde{r}}} \qquad C' = A^{\tilde{s}} \cdot C$$

It is straightforward to see that if $\pi = (A, B, C)$ is accepted by a Verifier, then $\pi' = (A', B', C')$ will be accepted as well: indeed, considering the last transformation (the other two are special cases of this), we have

$$\begin{aligned}
e(A', B') &= e(g^{\alpha}, h^{\beta}) e(T, h^{\gamma}) e(C', h^{\delta}) &\Leftrightarrow \\
e(A^{\tilde{r}}, B^{\frac{1}{\tilde{r}}} \cdot (h^{\delta})^{\frac{\tilde{s}}{\tilde{r}}}) &= e(g^{\alpha}, h^{\beta}) e(T, h^{\gamma}) e(A^{\tilde{s}} \cdot C, h^{\delta}) &\Leftrightarrow \\
e(A, B) e(A^{\tilde{s}}, h^{\delta}) &= e(g^{\alpha}, h^{\beta}) e(T, h^{\gamma}) e(A^{\tilde{s}}, h^{\delta}) e(C, h^{\delta}) &\Leftrightarrow \\
e(A, B) &= e(g^{\alpha}, h^{\beta}) e(T, h^{\gamma}) e(C, h^{\delta})
\end{aligned}$$

With similar techniques as employed in the Inner Subliminal Channel, we show how these three transformations can be used to embed a message into a valid proof π, thus creating another subliminal channel that we will refer to as *"Outer Subliminal Channel"*. We will demonstrate it using the 3rd proof transformation, but the construction generalizes easily to the other two transformations. Here we can also take advantage of parallel computation since the randomized group elements $A' = A^{\tilde{r}}$ and $C' = A^{\tilde{s}} \cdot C$ can be computed independently.

We let again $\omega : \{0,1\}^* \to \{0,1\}^b$ and $\xi : \{0,1\}^* \to \{0,1\}^{\frac{b}{2}}$ be two uniformly distributed polynomial-time computable functions and for statement $(a_1, ..., a_l)$ and some auxiliary information *aux*, we let $M = \omega(aux, a_1, .., a_l)$ be the subliminal message a Subliminal Prover would like to embed in a proof π for the statement $(a_1, .., a_l)$.

The new SubliminalProver and SubliminalVerifier procedures are as follows:

- $\pi \leftarrow$ SubliminalProver$(R, \sigma, (a_1, ..., a_m), \omega, \xi, aux)$:
  - $\tilde{\pi} = (\tilde{A}, \tilde{B}, \tilde{C}) \leftarrow$ Prover$(R, \sigma, (a_1, ..., a_m))$.
  - Compute $M = \omega(aux, a_1, .., a_l)$ and denote with m_1, m_2 the first and the last $\frac{b}{2}$ bits of M, respectively.
  - Randomly select $r \leftarrow \mathbb{Z}_p$ until
  $$\xi(aux, a_1, .., a_l, \tilde{A}^r) = m_1$$
  - Randomly select $s \leftarrow \mathbb{Z}_p$ until
  $$\xi(aux, a_1, .., a_l, \tilde{A}^s \cdot C) = m_2$$
  - With the r, s obtained compute $\pi = (A, B, C)$, where
  $$A = \tilde{A}^r \qquad B = \tilde{B}^{\frac{1}{r}} \cdot (h^{\delta})^{\frac{s}{r}} \qquad C = \tilde{A}^s \cdot C$$

- $(0/1, M) \leftarrow$ SubliminalVerifier$(R, \sigma, \pi, (a_1, ..., a_l), \Omega, \xi, aux)$:
  - Parse $\pi = (A, B, C)$, and let

    $$m_1 = \xi(aux, a_1, ..., a_l, A) \qquad m_2 = \xi(aux, a_1, ..., a_l, C)$$

  - If exists an $\omega \in \Omega$ such that $\omega(aux, a_1, ..., a_l) = (m_1||m_2)$, recover M as

    $$M = (m_1||m_2)$$

    Otherwise, set $M = \perp$.
  - Return (Verifier$(R, \sigma, (a_1, ..., a_l), \pi)$, M)

8.4 Computational Complexity

We'll now discuss the average number of operations required to successfully embed a b-bits message M in a valid proof π, using the Inner or the Outer Subliminal Channel, respectively. Both estimations are done assuming that ξ is uniformly distributed on $\{0,1\}^{\frac{b}{2}}$.

For the Inner Subliminal Channel, a b-bits message can be embedded with an average cost of $O(2^{\frac{b}{2}})$ parallelizable step operations, where each step consists of a scalar-point multiplication, a point addition and one ξ-evaluation. More precisely, $O(2^{\frac{b}{2}})$ step operations are needed to find r and, similarly, $O(2^{\frac{b}{2}})$ operations are needed to find s, such that the resulting A, B successfully embed the first and the second half, respectively, of the subliminal message M.

These costs directly derive from the hypothesis of the uniform distribution of ξ and from the fact that both $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic groups of order p with $log_2(p) >> \frac{b}{2}$. For example, fix a random value $M \in \{0,1\}^{\frac{b}{2}}$ and let, as in the case for r, $f(i) = \left(g^{\alpha+\sum_{i=0}^{m} a_i u_i(x)}\right) \cdot \left(g^{\delta}\right)^i$; for $i \in [0, 2^{\frac{b}{2}+1} - 1]$, $f(i)$ will take distinct values and hence, with high probability, $\xi(aux, a_1, ..., a_l, f(i))$ will take all possible values in the interval $[0, 2^{\frac{b}{2}+1} - 1]$. This means that there is a $j \in [0, 2^{\frac{b}{2}+1} - 1]$ such that $\xi(aux, a_1, ..., a_l, f(j)) = m$ and to find it, an average number of $\frac{2^{\frac{b}{2}+1}-1}{2} \approx 2^{\frac{b}{2}}$ $\xi \circ f$-evaluations are needed, that is an average cost of $O(2^{\frac{b}{2}})$ step operations.

Each step operation cost can be further improved to consists only of a point addition and one ξ-evaluation; for example, in the case of r, the Subliminal Prover can iteratively compute the values

$$A_i = \begin{cases} g^{\alpha+\sum_{i=0}^{m} a_i u_i(x)} & \text{if } i = 0 \\ A_{i-1} \cdot g^{\delta} & \text{otherwise} \end{cases}$$

until a preimage A_r for m_1 with respect to ξ is found.

With similar arguments, it is easy to prove that, similarly as in the Inner Subliminal Channel, embedding a b-bits message using the Outer Subliminal Channel requires an average number of $O(2^{\frac{b}{2}})$ point additions and ξ-evaluations.

Note that for both proposed channels, each of the values r and s can be searched in parallel: hence, if 2^c computation units are available, M could be embedded with an average number of $O(2^{\frac{b}{2}-c})$ point additions and ξ-evaluations per unit.

8.5 Adversary Assumptions

In a lightweight wallet scenario, where proof generation is delegated to a third-party Prover, subliminal channels could be maliciously exploited to share a certain amount of information related to users, thus permitting a wide variety of different activities such as proof fingerprinting, user-tracing, leak of transaction data and so on. Thus at stake is privacy and security of the user funds as well as fungibility of the affected coin.

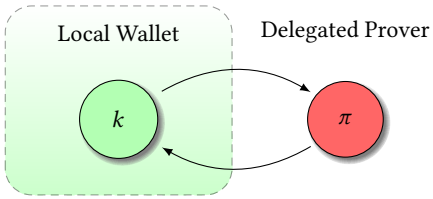

Figure 4: Delegated proof π signed with user's secret master key k.

In our use case, a Subliminal Prover could embed in all generated proofs a unique tracing information related to the user that requests the proof computation, and later share with some Subliminal Verifiers a certain auxiliary information *aux* that permits them to perform tracing analysis only to a particular subset of users they are authorized by the Prover to look at.

This is a realistic setting for an adversary in our case as well: the Zcash developers are currently working on implementing delegated proofs, where only the secret key and basic signing is done locally, while the rest of the proof generation is delegated to another system. In this setting, as shown in Figure 4, the adversary would not learn the secret master key of the target (since it is stored only in the wallet), but could still reveal, using one of the proposed subliminal channels, some transaction-tagging information.

Furthermore, to remain undetectable, the attacker wishes to hide its malicious activities and reduce the number of interactions with its target and the environment to a minimal level: for example, the attacker will not send any, even if encrypted, message to the external world indicating that he is adopting a subliminal proving mechanism on the target's machine, as this would be easily detectable by network traffic analysis. Whitin this scenario, our adversary would attack the proving system once, for example when the software is installed, and will not require any further direct interaction as long as the proving system remains malicious.

Aiming at fingerprinting proofs in order to permit later tracing activities, a Subliminal Prover could proceed as follows:

- Associate to each user U a unique random n-bit key k_U. Let $K = \{k_U\}_U$ be the set of all user keys.
- For $t \in \mathbb{N}^+$ let H_t be a $t-$bits cryptographic hash functions. For example, if $t \leq 512$, $H_t(x)$ could be defined as the last t-bits of $BLAKE2b\text{-}512(x)$.
- Let $E(M, k)$ be an easily computable algorithm that takes as input a plaintext M, a key k and returns an output of b-bits. Depending on the security property required for the subliminal channel, E can differently model a Message Authentication Code, a Block Cipher, a Public Key Encryption scheme and others. The following description indeed, easily generalizes to all these cases.
- When the user U requests a proof π for statement $(a_1, ..., a_l)$, the Subliminal Prover defines

  $$aux = \{E, k_U, b, H_t\}$$

$$\xi(aux, a_1, .., a_l, x) = H_{\frac{b}{2}}(x)$$

$$\omega(aux, a_1, ..., a_l) = E(a_1 \,||\, ... \,||\, a_l, k_U)$$

and executes

$$\pi \leftarrow \text{SubliminalProver}(R, \sigma, (a_1, ..., a_m), \omega, \xi, aux)$$

Consequently, the Subliminal Verifier is able to link proofs to users in the following way:

- The Subliminal Verifier receives from the Subliminal Prover the set
$$aux = \{E, Trace, b, H_t\}$$
where $Trace \subseteq K$ is the subset of keys associated to users he whishes to trace.
- The Subliminal Verifiers defines
$$\xi(aux, a_1, .., a_l, x) = H_{\frac{b}{2}}(x)$$
and let Ω
$$\begin{aligned}\Omega &= \{\omega_{k_U}(aux, a_1, ..., a_l)\}_{k_U \in Trace} \\ &= \{E(a_1 \,||\, ... \,||\, a_l \,, k_U)\}_{k_U \in Trace}\end{aligned}$$
be the set of functions ω_{k_U} indexed over the user keys k_U in $Trace$.
Then he executes
$$(0/1, M) \leftarrow \text{SubliminalVerifier}(R, \sigma, \pi, (a_1, ..., a_l), \Omega, \xi, aux)$$
- If the recovered message $M = (m_1 \,||\, m_2) \neq \perp$, the Subliminal Verifier associates π to the user U such that, with $\omega_{k_U} \in \Omega$, $\omega_{k_U}(aux, a_1, ..., a_l) = M$.

It is straightforward to see how this construction can be easily generalized to allow different embedding techniques and other malicious activities such as transaction data leaks and more.

8.6 Countermeasures

If ω and ξ have some cryptographic properties, even if their definitions become public, in principle it should not be possible without the auxiliary information *aux* to distinguish a random proof from one with a subliminal message embedded. Therefore, as could be the case for a lightweight wallet scenario, if a user delegates heavy cryptographic computations to a third-party entity as a Prover, he cannot know, looking at the generated proof π, if the Prover embedded a subliminal message or not, no matter what subliminal channel the Prover might have used.

To eliminate any potentially embedded subliminal message, the user should further randomize Prover's proof $\pi = (A, B, C)$ using, for example, one of the three proof transformations discussed at the beginning of the Outer Subliminal Channel Section.

Unfortunately, the most expensive transformation

$$A' = A^{\tilde{r}} \qquad B' = B^{\frac{1}{\tilde{r}}} \cdot (h^{\delta})^{\frac{\tilde{s}}{\tilde{r}}} \qquad C' = A^{\tilde{s}} \cdot C$$

is the only one that assures the user to fully disrupt all the (eventually) embedded b-bits. Clearly, the computation of this further transformation should not be delegated to a third-party, since this could, in turn, use the Outer Subliminal Channel to embed its own subliminal message.

Alternatively, if two valid proofs π_1 and π_2 for the same statement are available to the user, it is possible to combine them in a new valid proof π in a way that would disrupt a subliminal message, especially if π_1 and π_2 come from different third-party Provers.

Our construction requires, however, that the user chooses at least one of the two randomness r, s that third-party Provers should employ during the proof generation.

From this hypothesis, we therefore assume that the user possesses two proofs $\pi_1 = (A_1, B_1, C_1)$ and $\pi_2 = (A_2, B_2, C_2)$ for the same statement $(a_1, ..., a_l)$ generated accordingly to the chosen randomnesses s_1 and s_2, respectively.

Then, he computes a new proof π as

$$\pi = (A, B, C) = \left(\sqrt{A_1 \cdot A_2}, \sqrt{B_1 \cdot B_2}, \sqrt{C_1 \cdot C_2} \cdot \left(\frac{A_1}{A_2} \right)^{\frac{s_1 - s_2}{4}} \right)$$

To see that π is still valid, we explicitly write the computations, as reported in Groth's original scheme, that generate the group elements of both proofs π_1 and π_2:

$$A_1 = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + r_1 \delta} \qquad B_1 = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s_1 \delta}$$

$$A_2 = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + r_2 \delta} \qquad B_2 = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s_2 \delta}$$

$$\hat{C} = g^{\frac{\sum_{i=l+1}^{m} a_i (\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta}}$$

$$\hat{B}_1 = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s_1 \delta} \qquad C_1 = \hat{C} \cdot A_1^{s_1} \cdot \hat{B}_1^{r_1} \cdot g^{-r_1 s_1 \delta}$$

$$\hat{B}_2 = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s_2 \delta} \qquad C_2 = \hat{C} \cdot A_2^{s_2} \cdot \hat{B}_2^{r_2} \cdot g^{-r_2 s_2 \delta}$$

Hence, using the above formula, the resulting group elements A, B, C of π are

$$A = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + \frac{r_1 + r_2}{2} \delta} \qquad B = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + \frac{s_1 + s_2}{2} \delta}$$

$$\hat{B} = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + \frac{s_1 + s_2}{2} \delta} \qquad \left(\frac{A_1}{A_2} \right)^{\frac{s_1 - s_2}{4}} = g^{\frac{(r_1 - r_2)(s_1 - s_2)}{4} \delta}$$

$$\begin{aligned} C &= \hat{C} \cdot A^{\frac{s_1 + s_2}{2}} \cdot \hat{B}^{\frac{r_1 + r_2}{2}} \cdot g^{-\frac{r_1 s_1 + r_2 s_2}{2} \delta} \cdot g^{\frac{(r_1 - r_2)(s_1 - s_2)}{4} \delta} \\ &= \hat{C} \cdot A^{\frac{s_1 + s_2}{2}} \cdot \hat{B}^{\frac{r_1 + r_2}{2}} \cdot g^{-\frac{r_1 + r_2}{2} \cdot \frac{s_1 + s_2}{2} \delta} \end{aligned}$$

If we look closely, we notice that π is a proof for statement $(a_1, ..., a_l)$ whose elements A, B, C are generated using randomnesses $s = \frac{s_1 + s_2}{2}$ and $r = \frac{r_1 + r_2}{2}$ and hence it is valid by construction.

If, on the contrary, π is not accepted as valid but both π_1 and π_2 are, it means that at least one between the randomnesses s_1 and s_2 is not the one that was effectively used during the proofs generation: the user can easily spot (in a non lightweight setting) which Prover cheated, recomputing the group elements B_1 and B_2 from the Common Reference String and checking which of these is not equal to the one provided in the proofs π_1 and π_2.

All these considerations imply that, even in a lightweight scenario, a suspicious user has to perform some extra elliptic curve arithmetic on its own device to prevent embedding of subliminal messages.

At the same time, to permit users to not blindly trust third-party Provers, the implementation of a delegated proof generation mechanism should either admit further randomization of proofs, or take into consideration the combination of multiple proofs at the cost of extra $|p|-$bits of information exchanged between delegated Provers and the user.

Other than this, Trusted Execution Environments, like SGX, could help light clients to mitigate trust issues in proof delegation.

A systematic adoption of countermeasures to avoid embedding of subliminal messages is essential and not limited only to the prevention of the malicious activities we have seen so far. Unfortunately, subliminal channels pose a potential issue related to Zcash fungibility: if miners are incentivized by some entity to mine first transactions with a particular subliminal message embedded and this fact, in turn, encourage users to consciously embed such message to obtain a quicker transaction approval, then this gives rise to a certain disparity between Zcash coins (more precisely, transactions). Moreover, in this hypothetical scenario, users that refuse to embed a subliminal message in their transactions, should pay an higher fee to get the same level of priority, a win-win situation for miners.

9 THE PEDERSEN SUBLIMINAL CHANNEL

In the Pedersen Commitment scheme, given a cyclic group $\mathbb{G}$ of order p and two random generators g, h of $\mathbb{G}$ for which the discrete logarithm $\log_g(h)$ is unknown, c is said to be a commitment of $v \in \mathbb{Z}_p$ for a randomly chosen randomness $r \in \mathbb{Z}_p$, if $c = g^v h^r$.

One of the main reason why the Pedersen Commitment is employed is because it is additively homomorphic, that is, given any two commitments $c_1 = g^{v_1} h^{r_1}$ and $c_2 = g^{v_2} h^{r_2}$ for values v_1 and v_2, respectively, their product $c = c_1 c_2 = g^{v_1+v2} h^{r_1+r_2}$ is a commitment to the sum $v_1 + v_2$ with randomness $r_1 + r_2$.

The Pedersen Commitment homomorphic property combined with the adoption of a *binding signature*, that binds the note commitment values to the total balancing value, permits the signer to convince everyone that he is able to open the commitment

$$c = g^{\sum_i v_{i,IN} - \sum_j v_{j,OUT}} h^{\sum_i r_{i,IN} - \sum_j r_{j,OUT}}$$

given by the product of all input committed values $g^{v_{i,IN}} h^{r_{i,IN}}$ divided by the product of all shielded output committed values $g^{v_{j,OUT}} h^{r_{j,OUT}}$, without revealing any of the $v_{i,IN}, v_{j,OUT}$. Since, in turn, c is a commitment to the resulting transparent value change, this scheme assures at the same time both confidentially and coherence of a shielded transaction.

Note that the randomness makes any commitment for a value v indistinguishable from a random element in $\mathbb{G}$: this remark suggests that a Subliminal Signer, or an attacker that controls the Signer randomness source, using similar techniques we have seen in previous Sections, could embed a b-bits subliminal message in any commitments, carefully choosing the randomnesses until the resulting committed value satisfies some desired properties related to the message he wishes to embed. We will refer to this new subliminal channel to as *"Pedersen Subliminal Channel"*.

Other than that, if the Signer randomness source is under control of a malicious party (i.e. it runs a malicious version of the software), with the knowledge of r, a Subliminal Verifier could partially open a commitment c of v to $g^v = ch^{-r}$: in Zcash Sapling, v is the number of Zatoshi corresponding to the current shielded note, that is an integer in the interval $[0, 2.1 \cdot 10^{15}]$.

Since in our scenario v ranges in this bounded interval and in practice most transaction have note values that lies in the first part of such interval, with the partially opened committment g^v the Subliminal Verifier could easily mount a rainbow table attack [18].

Just to give an example, suppose he disposes of 4TB $\approx 8 \cdot (4 \cdot 10^{12})$ bits of storage memory. Since the Jubjub curve has a base field of 381-bits, he can store $\approx 2^{36}$ compressed elliptic curve points. Thus, he computes (eventually in parallel) and stores all the elements g^v for $v \in [0, 2^{36}]$, that is, note values up to $\approx 840 \cdot 10^8$ Zatoshi. Then, with this table and for note values lower than this bound, it is straightforward for him to find the discrete logarithm of g^v, thus revealing the shielded value. Optimizations of such rainbow table are clear and not of interest in this paper.

It remains to find a way for the attacker to make the Subliminal Verifier aware that a transaction is *weak*, in the sense that the randomness used during commitment generation is deterministic for both attacker and Verifier if some auxiliary information is known: note that, in fact, in our adversary assumptions the attacker (i.e. the malicious commitment mechanism) could not directly communicate with the external world once the target is attacked.

To get through this, the attacker could embed in some elements of the transaction a *marker* subliminal message, easily recognizable only if some auxiliary information is known, that makes the Subliminal Verifier aware that the current transaction is weak.

At first glance this seems a stronger assumption about our attacker: he has to control both the randomness source in the commitment scheme and the proof generation process if he wishes to use, for example, one among the Inner Subliminal Channel or the Outer one, to embed a subliminal message.

Actually, this extra requirement is not necessary: since all shielded input/output values are committed and the output values minus the input values has to be equal to the public balancing value, only controlling the randomness source of the commitment scheme, the attacker could embed in only one commitment a marker subliminal message using the Pedersen Subliminal Channel, hence permitting the Subliminal Verifier to become aware that the trasaction is weak and that he can proceed to the full de-committment of the shielded values; the value of the commitment that embeds the marker message can be easily computed from the balancing value, once all the others commitments are successfully de-comitted using the rainbow table attack.

10 IMPLEMENTATION RESULTS

We implemented the Inner Subliminal Channel and the Pedersen Subliminal Channel in the current Zcash official wallet (v. 2.0.5-2) and we successfully embedded 9 bytes in a fully shielded transaction with 1 shielded input and 2 shielded outputs (a typical payment transaction), i.e. 2 bytes in each proof and 1 byte in each committed value.[8] The proof generation time is proportional to the number of inputs, since each proof is generated independently. We ran our implementation on a standard Intel(R) Core(TM) i7-3770 CPU 3.40GHz desktop provided with 8.00GB of RAM and running Ubuntu 16.4 x64.

The proof generation time took on average 3.0087s, compared to the average time of 2.8412s needed for a random proof generation. This is just a 6% increase in proof time, barely noticeable even on

[8] As an example, we used our implementation to generate the following transaction 20ffc99e4e590688b465773ab7034d0055ef7d849d21320c10671253ed0db49c which can be found on Zcash testnet with a confirmed status. All proof elements A and B and values commitment have their last byte set to $0x00$. The relevant transaction data can be found in Appendix D.

closely monitored systems and totally undetectable when using hardware wallets.

Considering that we did not implement any algorithmic optimization, this shows that subliminal channels are a very powerful and effective mean for tagging transactions and it would be prudent from the user's perspective to implement countermeasures described in Sect. 8.6 (for example when using hardware wallets).

11 CONCLUSION

In this paper we have studied two different approaches to transaction tagging and linking in privacy-oriented cryptocurrency Zcash. The first approach is based on analysis of public blockchain data. It explores interplay of transparent and hidden transactions, and shows two new *active attacks* on user privacy: Danaan-gifts and Dust attacks. The second approach is based on discovery of subliminal channels in cryptographic primitives used for hidden transactions. These channels can allow malicious prover to embed tagging information about the user into each transaction, thus invalidating the purpose of the zk-SNARK information hiding. Finally we discuss countermeasures against these attacks.

REFERENCES

[1] 2013. Dash. https://www.dash.org/
[2] 2014. Monero. https://getmonero.org
[3] 2016. Zcash. https://z.cash/
[4] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. 2013. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 34–51.
[5] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. 2014. Deanonymisation of clients in Bitcoin P2P network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 15–29.
[6] George Bissias, A Pinar Ozisik, Brian N Levine, and Marc Liberatore. 2014. Sybil-resistant mixing for bitcoin. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 149–158.
[7] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. 2014. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security*. Springer, 486–504.
[8] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 305–326.
[9] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. 2018. An Empirical Analysis of Anonymity in Zcash. *ArXiv e-prints* (May 2018). arXiv:cs.CR/1805.03180
[10] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. 2017. A traceability analysis of monero's blockchain. In *European Symposium on Research in Computer Security*. Springer, 153–173.
[11] Greg Maxwell. 2013. CoinJoin: Bitcoin privacy for the real world. In *Post on Bitcoin forum*.
[12] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2013. A Fistful of Bitcoins: Characterizing Payments Among Men with No Names. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC '13)*. ACM, New York, NY, USA, 127–140. https://doi.org/10.1145/2504730.2504747
[13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. 2013. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 127–140.
[14] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. 2013. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 397–411.
[15] Andrew Miller, Malte Möser, Kevin Lee, and Arvind Narayanan. 2017. An empirical analysis of linkability in the Monero blockchain. *arXiv preprint arXiv:1704.04299* (2017).
[16] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[17] Shen Noether, Adam Mackenzie, et al. 2016. Ring confidential transactions. *Ledger* 1 (2016), 1–18.
[18] Philippe Oechslin. 2003. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *Advances in Cryptology - CRYPTO 2003*, Dan Boneh (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 617–630.
[19] Torben Pryds Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology — CRYPTO '91*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 129–140.
[20] Jeffrey Quesnelle. 2017. On the linkability of Zcash transactions. *arXiv preprint arXiv:1712.01210* (2017).
[21] Fergal Reid and Martin Harrigan. 2013. *An Analysis of Anonymity in the Bitcoin System*. Springer New York, New York, NY, 197–223. https://doi.org/10.1007/978-1-4614-4139-7_10
[22] Fergal Reid and Martin Harrigan. 2013. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*. Springer, 197–223.
[23] Dorit Ron and Adi Shamir. 2013. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*. Springer, 6–24.
[24] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2014. CoinShuffle: Practical decentralized coin mixing for Bitcoin. In *European Symposium on Research in Computer Security*. Springer, 345–364.
[25] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 459–474.
[26] Gustavus J. Simmons. 1984. *The Prisoners' Problem and the Subliminal Channel*. Springer US, Boston, MA, 51–67. https://doi.org/10.1007/978-1-4684-4730-9_5
[27] Gustavus J. Simmons. 1994. Subliminal Communication is Easy Using the DSA. In *Advances in Cryptology — EUROCRYPT '93*, Tor Helleseth (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 218–232.
[28] Luke Valenta and Brendan Rowan. 2015. Blindcoin: Blinded, accountable mixes for bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 112–126.
[29] Nicolas van Saberhagen. 2013. Cryptonote v 2.0. (2013). https://cryptonote.org/whitepaper.pdf
[30] Adam Young and Moti Yung. 2004. *Malicious Cryptography: Exposing Cryptovirology*. John Wiley & Sons, Inc., USA.
[31] Zuoxia Yu, Man Ho Au, Jiangshan Yu, Rupeng Yang, Qiuliang Xu, and Wang Fat Lau. 2019. New Empirical Traceability Analysis of CryptoNote-Style Blockchains.

A EXPERIMENTAL RESULTS

In order to provide experimental results for the presented stochastic model in Section 4, we must gather the relevant statistics from the chain itself. First, we have investigated the Sapling shielded transaction set, as that provides a view for us on the usual amount of shielded inputs and outputs for a shielded transaction. Secondly, we have observed the public blockchain data to estimate the fingerprint survival probabilities.

A.1 Notation

For the ease of exposition, we introduce a notation to describe transactions based on their number of shielded and transparent inputs and outputs. We denote a Zcash transaction as

$$(\mathtt{s}_{in} \cdot \mathtt{t}_{in} \mid \mathtt{s}_{out} \cdot \mathtt{t}_{out})$$

where $\mathtt{s}_{in}$ and $\mathtt{s}_{out}$ are the number of shielded inputs and outputs, respectively, and similarly, $\mathtt{t}_{in}$ and $\mathtt{t}_{out}$ are the number of transparent inputs and outputs.

Depending on the number of shielded and transparent inputs or outputs, transactions are then divided into hiding t-to-z, fully shielded z-to-z and revealing z-to-t transactions as follows:

- *t-to-z*: $\mathtt{t}_{in} \geq 1$ and $\mathtt{s}_{out} \geq 1$ and negative balance value (hiding)
- *z-to-z*: $\mathtt{s}_{in} \geq 1$, $\mathtt{s}_{out} \geq 1$ and $\mathtt{t}_{in} = \mathtt{t}_{out} = 0$
- *z-to-t*: $\mathtt{s}_{in} \geq 1$ and $\mathtt{t}_{out} \geq 1$ and positive balance value (revealing)

Few examples are the following transactions: $(0 \cdot 1 \mid 1 \cdot 0)$ is a hiding t-to-z transaction with 1 transparent input and 1 shielded output; $(1 \cdot 0 \mid 2 \cdot 0)$ is a fully shielded z-to-z transaction with 1 shielded input and 2 shielded outputs; $(2 \cdot 0 \mid 1 \cdot 1)$ is a z-to-t (partially) revealing transaction with 2 shielded inputs and 1 shielded and 1 transparent output.

A.2 Sapling Transactions Dataset

Our transactions dataset includes all hiding, fully shielded and revealing Sapling transactions occurred from block 419,200 (mined October 29, 2018 when the official Sapling fork happened) to block 472,285 (mined January 29, 2019).

The distribution of these collected transactions is reported in Table 2, row *All*. Since we are mainly interested in transactions that more likely occur as coin transfers between non-miner users which are not already linkable with a direct unique value match, we removed from our starting database the following transactions:

- *Mining activities*: when a block is successfully mined, the miners have to transfer their rewards to a shielded address before being able to spend them, hence creating lots of hiding transactions. Many miners transfer their shielded mining rewards directly to transparent addresses, thus creating many revealing transactions. Using the heuristics from [9] we have been able to trace back to miners' public addresses 6,827 hiding t-to-z transactions and 5,718 revealing z-to-t transactions.
- *Direct unique value matches*: where an hiding transaction is directly followed by a revealing transaction with same unique value (Heuristic 1), i.e. the paths

  $$(0{\cdot}1\,|\,1{\cdot}0)\text{-}(1{\cdot}0\,|\,0{\cdot}1)$$

  We have found 37 such transactions.
- *1-Hop fingerprint matches*: the paths

  $$(0{\cdot}1\,|\,1{\cdot}0)\text{-}(1{\cdot}0\,|\,1{\cdot}0)\text{-}(1{\cdot}0\,|\,0{\cdot}1)$$

  where a hiding transaction with a unique value is directly followed by a revealing transaction with a unique value, where the value difference is equal to 10,000 Zatoshis, the standard transaction fee (Heuristic 2). We have found 676 such transactions.
- *Small value transactions*: we noticed that while there were 2,436 t-to-z transactions with total transparent value less than 1 ZEC each (for a total hiding value of 841.74 ZEC), there were only 286 z-to-t revealing transactions with an overall value lower than 1 ZEC (for a total value of 68.76 ZEC). Checking the discrepancy between these two numbers, we have found that there were 3 transactions of the form $(>10{\cdot}0\,|\,0{\cdot}1)$ and 25 transactions of the form $(>10{\cdot}0\,|\,1{\cdot}1)$, where >10 indicates that there are more than 10 shielded inputs. The total number of shielded inputs of these 28 z-to-t transactions is 1,756 for an average revealed value of 39.97 ZEC (total revealed value 1119 ZEC). Assuming these 28 transactions were independent, the ratio between the revealed value and the number of shielded inputs is 0.45 ZEC (weighted median) and 0.64 ZEC (weighted arithmetic mean). Since the Zcash official wallet, when combining spends, takes them in decreasing value order, we speculate that these spends are mainly small value spends (less than 1 ZEC) which are collected and eventually combined with few higher value ones[9] before being transferred to a transparent address. Our speculation is enforced by the fact that in 25 transactions out of 28, the revealed values are round, e.g. 3, 5, 30, 35, 40, 50, 120 ZEC. In other words, there are many small values which are collected and then spent as soon as their total value is close to certain round amount of ZEC. Due to their unusual high number of combined shielded inputs, we suspect that these transactions are related to mining activities (e.g. transparent fractions of mining rewards that are shielded and sent to mining pool members) and can be then traced back to t-to-z transactions whose total value is less than 1 ZEC. Thus we have opted to not consider 2,436 small value t-to-z transactions[10] and 314 z-to-t transactions.

All these dataset updates are summarized in Table 2 resulting after all the removals in 1,613 t-to-z hiding, 1,570 fully shielded and 1,633 revealing transactions. The distribution of different types of transaction in the final dataset is reported in Table 3.

Assuming that all t-to-z transactions are independent, we can then approximate the average number of transaction-hops $Zlen$ for a generic path in the shielded pool as the ratio between the sum of all transactions types that have at least 1 shielded input and 1 shielded output and all the t-to-z transactions. We then obtain $Zlen \approx 1.42$.

A path in the shielded pool goes through transaction-hops that have at least one shielded input and one shielded output: in Table 4 we reported the most frequent types of these transactions ($\approx 95.8\%$) among with their distribution and the corresponding survival probability of good fingerprints observed in all public blockchain data. More precisely, given a transaction $(\mathtt{s}_{in} \cdot \mathtt{t}_{in} \mid \mathtt{s}_{out} \cdot \mathtt{t}_{out})$ happening in the shielded pool, we set the corresponding survival probability of a good fingerprint to be equal to the fingerprint survival probability observed in the transparent transaction of the same input-output degree $(0 \cdot \mathtt{s}_{in}{+}\mathtt{t}_{in} \mid 0 \cdot \mathtt{s}_{out}{+}\mathtt{t}_{out})$ multiplied by the probability that the fee ends with 4 zeroes ($\approx 96.8\%$), i.e. the fee doesn't affect the fingerprint.

Similarly, we report in Table 5 the relative distribution and survival fingerprint probabilities for the most frequent exit nodes ($\approx 96.9\%$), i.e. revealing z-to-t transactions.

Given a path $x = x_1 - \ldots - x_n - x_{n+1} \in ZPaths(n+1)$, where $x_1, \ldots, x_n$, are transactions inside the shielded pool while x_{n+1} is a revealing transaction, it is now straightforward to estimate both $\mathbb{P}(FP \mid x)$ and $\mathbb{P}(x)$ and, ultimately, $\mathbb{P}(FP)$.

Indeed, assuming each transaction independent and considering only paths x made by the most frequent types of transactions, we have that $\mathbb{P}(x) = \prod_{i=1}^{n+1} \mathbb{P}(x_i) = \left(\prod_{i=1}^{n+1} p(x_i)\right) \cdot (0.958^n \cdot 0.969)$, where $p(x_i)$ denotes the probability to have a transaction type equal to x_i, i.e. the values in column "%" of Table 4 and 5. Similarly, $\mathbb{P}(FP \mid x) = \prod_{i=1}^{n+1} p_{FP}(x_i)$, where $p_{FP}(x_i)$ denotes the fingerprint survival probability for transaction x_i, which are again reported in Table 4 and 5.

Letting $ZPaths'(n+1) \subset ZPaths(n+1)$ be the set of paths consisting of the most frequent transactions types reported in Table 4 and 5 and modeling $\mathbb{P}(\mathcal{Z} = n) = \frac{Zlen^n \cdot e^{-n}}{n!}$ by a Poisson distribution with parameter $Zlen$, we obtain an estimation for the overall fingerprint survival probability as:

$$\mathbb{P}(FP) \approx \sum_{n=0}^{k} \frac{Zlen^n \cdot e^{-n}}{n!} \cdot \left(\sum_{x \in ZPaths'(n+1)} \mathbb{P}(FP \mid x) \cdot \mathbb{P}(x) \right)$$

Letting $k = 5$ we obtained $\mathbb{P}(FP) \approx 16.6\%$

[9] This could also explain why the revealed value of 1,119 ZEC is greater than 841.74 ZEC, the overall total hidden value of small value t-to-z transactions

[10] It is not statistically relevant which ones are removed, since 99.8% of t-to-z transaction are of the single input-single output form: $(0{\cdot}1\,|\,1{\cdot}0)$.

	t-to-z	z-to-z	z-to-t
All	11,589	2,246	8,408
Remove Mining	4,762	2,246	2,690
Remove Direct Matches	4,725	2,246	2,653
Remove 1-hop Matches	4,049	1,570	1,977
Remove Small Values	1,613	1,570	1,663

Table 2: Number of transactions after each corresponding dataset update.

t-to-z		z-to-z		z-to-t	
Type	#	Type	#	Type	#
(0·1|1·0)	1,594	(1·0|1·0)	699	(1·0|0·1)	664
(0·t|s·0)	15	(1·0|2·0)	491	(1·0|1·1)	504
(s·t|s·0)	4	(2·0|2·0)	176	(2·0|0·1)	224
		(2·0|1·0)	78	(2·0|1·1)	129
		(3·0|1·0)	36	(3·0|0·1)	54
		(3·0|2·0)	21	(3·0|1·1)	36
		(s·0|s·0)	69	(s·t|0·t)	31
				(s·t|s·t)	21
Total	1,613		1,570		1,663

Table 3: The transaction distribution of our final dataset. The s and t represents all the remaining number of input/outputs possible.

Type	#	%	Fingerprint Survival Probability
(1·0|1·0)	699	0.318	0.968
(1·0|1·1)	504	0.230	0.475
(1·0|2·0)	491	0.224	0.475
(2·0|2·0)	176	0.080	0.155
(2·0|1·1)	129	0.059	0.155
(2·0|1·0)	78	0.036	0.204
(3·0|1·0)	36	0.016	0.086
(3·0|1·1)	36	0.016	0.066
(3·0|2·0)	21	0.010	0.066

Table 4: The relevant z-to-z and z-to-t transactions where at least some coins stay shielded. The last column is the survival probability of a fingerprint based on the public blockchain data.

B USAGE OF ZK-SNARKS

Let us investigate the usage and adoption rate of Sapling transactions compared to Sprout transactions.

From Table 6, we notice that even though the number of transactions decreased, the usage of shielded transactions from regular users mostly switched to Sapling transactions and the main remaining users of Sprout transactions are miners and mining pools who did not change their use practices yet.

Type	#	%	Fingerprint Survival Probability
(1·0|0·1)	664	0.412	0.968
(1·0|1·1)	504	0.313	0.337
(2·0|0·1)	224	0.139	0.155
(2·0|1·1)	129	0.080	0.204
(3·0|0·1)	54	0.034	0.086
(3·0|1·1)	36	0.022	0.066

Table 5: The relevant z-to-t transactions where some coins are revealed. The last column is the survival probability of a fingerprint based on the public blockchain data.

	15/10/18-29/10/18	15/01/19-29/01/19
Num of Transactions	52,438	41,961
Num of Sprout Txs	7,241	3,592
Hidden Value	129K ZEC	81K ZEC
Without Mining Rewards	25K ZEC	23K ZEC
Num of Sapling Txs	0	4,748
Hidden Value	0	67K ZEC
Without Mining Rewards	0	50K ZEC

Table 6: Sapling zk-SNARK usage

B.1 Interaction Between Sapling and Sprout Transactions

Another aspect of the adoption rate for Sapling transactions is how many users have transferred their values from a Sprout shielded address to a Sapling shielded address. To investigate this, we have checked how many transaction outputs of a shielded Sprout transaction were spent directly as an input to a Sapling shielded transaction.

We have found 241 such outputs, hidden in 213 transactions overall. Although this might not seem as a huge amount of transactions, in the total value they cover more than 46K ZEC, which is more than 20% of all Sapling hidden value not related to mining rewards (212K ZEC) since the time of the Sapling hard fork.

Another question is whether these revealing transactions can be tracked by the linking methods presented previously in Section 3. The recommended method of coin transfer is described on the Zcash website[11], where the presence of this linkability is mentioned, warning users and suggesting methods to avoid it by splitting the values into smaller round denominations and moving them with certain delays.

C FURTHER RESULTS

In Heuristic 1 we have used unique value matches. However even in the case where a matching is not unique, the proposed methods still provide a probabilistic linkability feature.

[11] Sapling Turnstile - https://zcash.readthedocs.io/en/latest/rtd_pages/sapling_turnstile.html

$\|h\| \cdot \|r\|$	Num of links
1	9,919
2	882
3	308
4	344
5	134
6-10	721

Table 7: Number of equal in/out *value* pairs (modulo single z-z hop fee) entering and exiting the shielded pool over entire chain history.

$\|h\| \cdot \|r\|$	Num of links	Non-Unique Complete Matches
1	10,642	7,228
2	5,212	3,513
3	2,192	1,456
4	1,150	738
5	684	418
6-10	1,913	1,227

Table 8: Number of possible *fingerprint* pairs and how many times they happen for a sliding window of 16,000 blocks. In the last column we have removed the matches that were also tagged as a unique complete match in Table 7 (the first line of the table are the unique matches). The matches are not reduced only from the unique matches (first line) because even though a pair is a unique match, value match in the chain does not mean it is a unique fingerprint match as well, there can be more hiding and revealing transaction with the same fingerprints, but with full values that are not the same.

Let us call the probability of a correct value match[12] of the unique input to the unique output going through the shielded pool by P. Then to estimate the probability of correct non-unique matches we can divide P by the number of possible pairings. If e.g. our baseline best case is 85.3% probability (assuming 14.7% false positive rate), with possible 3 hiding and 2 revealing values which are all identical, the probability of correct linkage is $\frac{1}{3 \cdot 2} \cdot 0.853 = 0.142$. Generalizing this approach, the probability of correct linkage is $\frac{1}{|h| \cdot |r|} \cdot P$, where $|h|$ and $|r|$ is the number of times the exact value in question has appeared as a hiding or a revealing value respectively.

This approach can be directly translated to fingerprints as well, where a lower value of P might be applied as these links provide less accuracy in general. Considering these metrics, Tables 7 and Table 8 summarize the number of links registered based on the possible number of pairs ($|h| \cdot |r|$). Further exploration of probabilistic matching together with anonymity set sizes derived from the subset sums approach could be a direction for future research.

[12]The values a called matching when they differ by single fixed z-z hop fee.

D EXAMPLE OF A TAGGED TRANSACTION

In this section we show a test tagged transaction using the proposed subliminal channels. The data is in the decoded hex Zcash transaction in the JSON format. The transaction has two shielded inputs and one shielded output. In the JSON listing the following features contain the subliminal message. The first is the commitment value `cv` which contains the subliminal message in the last byte as the result of applying the Pedersen subliminal channel (Section 9). The second is the zk-SNARK proof value `proof`, where the subliminal message is embedded in the 48th and 144th byte using the Inner Subliminal Channel (Section 8.2). For the ease of readability the subliminal message is simply the zero value byte which we have marked with bold font. The transaction can be verified on the Zcash testnet using the hash of the transaction, which is the `txid` value.


```
"txid": "20ffc99e4e590688b465773ab7034d0055ef7d849d2
1320c10671253ed0db49c",
"overwintered": true,
"version": 4,
"versiongroupid": "892f2085",
"locktime": 0,
"expiryheight": 501319,
"vin": [],
"vout": [],
"vjoinsplit": [],
"valueBalance": 0.00000000,
"vShieldedSpend": [
"cv": "840de77de2ccce945cf5e605b6e3b3fe34ac1210adb528
8555ac151f388c3200",
"anchor": "19f636b14e7a7983ba89a14f6c03b5cfe540b867f8
c6fe718e8e751fadbf3880"
"nullifier": "80dc569355b7eab9c101c7dba7a266982ac9b10
c64e113c09a44f0959e78bc4d",
"rk": "c21c2c6db475d2a4ef8eb3f6219112a339ffb5e3d6303e
abf017f9345da0ff84",
"proof": "a3ab169ed20718a175e421bde1609e8f94e7f6593
0ce8d5a93459e164d614285b4dbd9aebbd31492f057f23d9bbb5
a008632c4b7e9d833181f9bb14b7261e27e7eafc03aa9b6d3374b
d9d2169bdcec21f1143bf1f79ea3ec49e33765648e289010acd6a
3b9dabb5e5421a237bba50ca88ef446f877eb87e0ee2e50907023
bf9232ce72df4c4081873fa42c61188af700a6b76af9d28ddd9df
1b026996407073162e292ac301eff406a3a4aecfcb35cd801090e
c7957f95fbe3d01702e3c09417",
"spendAuthSig": "b69c1f7f0f7891775cbd1b7bf6ef9335d88e
2c31b6dc69c61624c5940db4a7720054847954fe934646ed9c1a5
b768ac12395f37b58eb308c3485fb5b6437e808"
],
"vShieldedOutput": [
"cv": "e74a704a0190b634e6eaaade90dbac40f2794f3d0821a6
7c5b90c6fc56d10500",
"cmu": "1201a0844ab631ab418f429fc3d2ec64e5bafab4afa85
8d6a29a6b0302a1a8de",
"ephemeralKey": "f0d7d39f7a748b725108004402d9dcccc095
a1c89c4522fd7f46551adec0babf",
"outCiphertext": "300a380831c98423665517289347d58edc4
6d019240e33bd824916061eff80c2745e9f22ccbec2cef1511914
20bd21f911dbb5eaa5ec4cd09adaa09658a69a337b6a28cf383ed
```

```
bfbfd01789f9911a9a1",
"proof": "b6868e9d4a8ad6677cdd8e5e0a53f5b07fe6bfa24
6833855e6f2b7139e4b3b6c46c46c7afe8163806f34f8dd1c98d
000087186b68c8a4311ec81602290bc2f2e13d4bbde46d9baf0ba8
0ca3bb3986d581c3d8cb85fa8956541ec74c32ea7f2afb0d3d839
fc8fc4ede4314ff71553307d2046a30ce3e6910f9477fb89f1353
a04d126797d1a1d2c2eda973da208eb3a200095235a7dd26608ca6
d3cd6607ec18bf37fb85aae651ad49df523ddd1165ae896a2c8dd
e70e3a8ebeb4507e4d2af5b186",
"cv": "129841260f65ced05c33ffd720ff6d98afdec8f15d1be5
c3c6be14181716d600",
"cmu": "399e239a682b1e072f04b81bf5fff37c06b57c23499bd
7394d03fae961ffddcb",
"ephemeralKey": "ef61fd9ca8b37702190ba953f259d931ab00
99087d3fe4f8b20cdbb24cb4f657",
"outCiphertext": "206491a259a6b7fe9d49dafa4eddc2e8d1b
700d0a06dd5b58e4d69227f798049c7ccddfbb1be7cb285e75560
f629277ae29eaf47410fac0a57ba705a20a4d926cbd8443ad8688
ec02a563159de96dc21",
"proof": "80716af531c0c68b555d5fef665c83cbbe3ad1134
ae876dcc188ba138af7e4553c2854c6c32412e1191c9b82f1a6b
300a50c737ffb62723a4b56c1bfaef84c8fafd0fe825c11a13b21
5ede04074f9e59460a472fbc2035e546021f5b884118900dde468
04895ce78c2b49b9d8d20b050c3ac6c7a969636855825707971e5
b63cedb0ecc144300c6eea725a7067d38f00b3357681f9d25c210
0f1989992e3ec2236f18f7e1acd32b750bb25c31eedde336a5ad6
7acfbb540f009d35d8677e2d7b"
],
"bindingSig": "ea99c5d837508db138cc2d6388d990a43454cd
4ccb444587276c523b326205a466e017a30eb0081b1b75405c863
32f2d3ee2ecc1077ac21ce413ca5e2294d306",
"blockhash": "0018a360262860c4059793b72695785a2276ed7
1192ec39b06af6aac3b28ad44",
"confirmations": 1089,
"time": 1558901286,
"blocktime": 1558901286
```