Privacy in Grin, Monero and Zcash - an overview

David Lifshitz

October 7, 2021

Abstract

Grin, Monero and Zcash are three cryptocurrencies which aim to give strong privacy guaranties to their users. Each one has a unique set of techniques to realise privacy while maintaining currencies soundness. We will review the three cryptocurrencies and discuss the prose and cons of each one.

1 Introduction

Privacy has always been important feature of any currency. Few people and organization would want to fully expose their financial data. (Why privacy is important)

In classic cryptocurrencies (like Bitcoin), we have little to no privacy. A blockchain's data is public (by its nature). Given the public address of the user, I can easily see how much money they have, who are their business partners, etc.

In this article, we will look at cryptocurrencies who offer strong privacy features. After understanding the underlying mechanics for each of them, we will discuss and compare the different privacy features.

2 MimbleWimble and Grin

2.1 Short history

The original white paper for MimbleWimble has been published on the bitcoin research channel on August 2016. The paper has been published under a pseudonym of "Tom Elvis Jedusor" [Jed16]. This paper presented an algorithm-outline for Bitcoin which allows for:

1. Improved privacy over the existing model.
2. Smaller history data (smaller blockchain-node storage requirments)

A detailed paper making the idea precise and developing the scallability improvement further has been published by Andrew Poelstra on October 2016. [Poe16]

A few days later, the Grin project has started. It's purpose was creating a new cryptocurrency using the ideas presented by both papers. Grin is an open source project, which can be found under `https://github.com/mimblewimble`. The actual launch of the cryptocurrency was in January 2019. [gria]

2.2 Transaction - Basic definitions

Grin currency is transferred via Transactions. A Transaction has one or more inputs, one or more outputs and a transaction fee. The sum of the amounts for each input is the currency sent by the "Transaction sender". The sum of the amounts for each output is the currency acquired by the "Transaction receiver". And the fee is the senders payment to a node for inserting his transaction to the blockchain.

All inputs are created from an output of the same amount received by the "current sender" earlier in history and not yet used to generate another input (when generating an input from an output, the output is "spent"). The exception for this rule is Mining.

Only outputs are recorded in the blockchain - not the transactions. Transaction is the "protocol" through which Grins exchange owners. Transaction are sent from the "money sender" to a node to be validated and accepted into the blokchain. After a block containing the transaction's exchange has been added into the blockchain, the transaction is removed from the nodes memory (a malicious node might record transactions passing through him for a longer period).

The actual amount of money in each output (and input) is hidden using a Pedersen commitments.

2.3 Transaction - Pedersen commitments

MimbleWimble Transaction inputs and outputs doesn't specify in clear-text the currency amount they contain and the owner. Instead, each one of them is a "Pedersen commitment". A Pedersen commitment is a commitment that has the property of being additively homomorphic.[kN20a] The use of Pedersen commitments to store values will be explained below.

Pedersen commitment is constructed in the following fashion:

$$C = r * G + v * H \tag{1}$$

Equation dictionary:
C - Pedersen commitment
G,H - elliptic curve generators
v - the amount of money this input/output contains.
r - a "private key" matching this commitment (ownership proof). Since r is different for each commitment and helps us in hiding v, it is also known as "blinding factor" or "commitment mask".

Let us consider the properties of Pedersen commitments.

H is an elliptic curve generator. Due to Elliptic curve cryptography properties ,v*H is "easy" to compute given v and H, while finding the value of v given H and v*H as difficult a problem as prime factorization . The addition of r*G makes the task of extracting the value of v (the amount for the given input/output) from C (the computed Pedersen commitment) even harder. Even if 2 elements transfer the same sum, r*G will serve as a "blinding factor" (r*G is different for every commitment). The amount of money associated with each of the inputs and outputs of the transaction is hidden.

Pedersen commitments also allow us to validate that the amount of money sent in the transaction outputs is equal to the amount of money received by the outputs, even without knowing the actual sum exchanged. How so? Elliptic curve generator are distributive in relation to scalars multiplication ($a * H + b * H = (a + b) * H$). Lets look at an example of a simple transaction, with 1 input and 2 outputs.

$$\text{Input 1: } C_1 = r_1 * G + v_1 * H$$
$$\text{Output 1: } C_2 = r_2 * G + v_2 * H$$
$$\text{Output 2: } C_3 = r_3 * G + v_3 * H$$

If this is a valid transaction, the amount of money entering the transaction must be equal to the amount exiting the transaction. In this example, this translates to $v_1 = v_2 + v_3$.

$$v_1 = v_2 + v_3$$
$$v_1 * H = v_2 * H + v_3 * H$$
$$v_1 * H - (v_2 * H + v_3 * H) = 0$$
$$sum(inputs) - sum(outputs) = C_1 - (C_2 + C_3) = r_1 * G - (r_2 * G + r_3 * G) = r_1 - (r_2 + r_3)$$

This property, combined with requirement for $r_1 = r_2 + r_3$, allows us to say that "a transaction is valid if and only if the sum of the inputs (represented by Pedersen commitments) is equal to the sum of the outputs (also represented by Pedersen commitments)". Generalizing the previous example, any

valid translation will satisfy:

$$sum(inputs) = sum(outputs)$$

$$(\sum_{i=1}^{|\text{inputs}|} r1_i * G + \sum_{i=1}^{|\text{inputs}|} v1_i * H) - (\sum_{i=1}^{|\text{outputs}|} r2_i * G + \sum_{i=1}^{|\text{outputs}|} v2_i * H) = 0$$

IFF

$$\sum_{i=1}^{|\text{inputs}|} r1_i = \sum_{i=1}^{|\text{outputs}|} r2_i \;\&\&\; \sum_{i=1}^{|\text{inputs}|} v1_i = \sum_{i=1}^{|\text{outputs}|} v2_i$$

The "if and only if" relation can be used here because H and G are Elliptic curve generators. Guessing the correct values for the input/output set of r's and v's is as hard as breaking an ECC encryption using the same parameters - deducing the private key (r or v) out of it's matching public key (r*G or v*H) and generator (G or H). Grin valid input/output amount has minimum and maximum range. "Range proofs" are attached to the transaction and allow us to validate this constraint.

So far, we have seen the improvement of 1 privacy property: The sum of every input/output is hidden while maintaining the ability to validate a transaction. In the following sections, we will use this as a base for moving forward.

2.4 Transaction - ownership

In the previous section, we deduced the requirements for a valid transaction:

$$\sum_{i=1}^{|\text{inputs}|} r1_i = \sum_{i=1}^{|\text{outputs}|} r2_i \;\&\&\; \sum_{i=1}^{|\text{inputs}|} v1_i = \sum_{i=1}^{|\text{outputs}|} v2_i$$

Lets look at the r, the one we named "the commitment's private key". The first part of the requirement ($\sum_{i=1}^{|\text{inputs}|} r1_i = \sum_{i=1}^{|\text{outputs}|} r2_i$) means that to "spend" an output, you must know it's matching r (or the sum of r's, in the group). Without this knowledge, we won't be able to construct $\sum_{i=1}^{|\text{inputs}|} r1_i$ for the spending transaction, and the "first Pedersen commitments requirement of a valid transaction" will fail. Knowing the correct r and v values signifies ownership over this output - only with this knowledge we can "use" (==spend) it.

Alas, this idea of "output ownership" has a loophole. Both parties involved in the transaction know the values for "r sum" and the amount of money transferred in the transaction (=="v sum"). After Alice has sent money to Bob using transaction α, Alice can now use the outputs of transaction α (outputs that only bob should be able to use) to construct a valid transaction β which sends the same amount to Eve. Alice used valid transactions to steal money from Bob!

To fix this loophole, we will create a slight alteration to the transaction structure. In this "alteration", $\sum_{i=1}^{|\text{inputs}|} r1_i! = \sum_{i=1}^{|\text{outputs}|} r2_i$ and thus the knowledge of the r keys (with the amount stored in the output) is indeed equal to "output ownership".

2.5 Transaction - kernel excess

To prevent the previously mentioned loophole in the previous section, we will modify the definition of a valid transaction while maintaining the properties we previously achieved. Now, a transaction is valid if:

$$sum(inputs) - sum(outputs) = k1 * G$$

$$(\sum_{i=1}^{|\text{inputs}|} r1_i * G + \sum_{i=1}^{|\text{inputs}|} v1_i * H) - (\sum_{i=1}^{|\text{outputs}|} r2_i * G + \sum_{i=1}^{|\text{outputs}|} v2_i * H) = k1 * G$$

And we will include a "proof" for the correct value for k1 * G inside the transaction.

k1 is called "the excess value" and k1 * G is called "kernel excess". The addition of this value translates the requirement

$$\sum_{i=1}^{|\text{inputs}|} r1_i = \sum_{i=1}^{|\text{outputs}|} r2_i$$

to

$$\sum_{i=1}^{|\text{inputs}|} r1_i = \sum_{i=1}^{|\text{outputs}|} r2_i + k1$$

while the requirement for the money amounts ($\sum_{i=1}^{|\text{inputs}|} v1_i = \sum_{i=1}^{|\text{outputs}|} v2_i$) remains the same.

k1 should be known to the receiver only. With the addition of k1 to the "private keys output sum", the sender no longer posses ownership knowledge over the properties of the output. The loophole is closed.

Now, when validating a transaction, we will validate that $sum(inputs) - sum(outputs) = k1 * G$. Part of transaction validation will be validating k1*G itself. The receiver will sign a constant know value (Grin chose this value to be "empty string") with k1 as a private key. The transaction validator will use k1*G as a private key to validate this signature.

2.6 Transaction - blockchain fee

The transaction fee is added to the transaction in clear-text. fee*G is added as part of the outputs sum.

2.7 Transaction - structure

As seen before, a transaction data required for validation will include the following values:

1. List of inputs referencing previous outputs. Each input is written as a Pedersen commitment. For generating the the inputs, ownership keys of the referenced outputs has been used (as proof of ownership).
2. List of outputs. The r values are randomly generated by the receiver, and excess value (k1) is computed.
3. A range proof that among other things shows that v is non-negative. (Not discussed here, but a range proof is required for the mathematical validity of some of the previous statements).
4. A transaction fee in clear text.
5. A signature signed with the excess value (k1) as the private key.

2.8 Blockchain data - cut-through

In this section, we will use define a technique which allow us to merge two (or more) transactions. Later, we will use this technique to obscure transactions data in the blockchain while keeping the ability to validate and add new transactions into the chain.

Cut-through is a technique which allow us to merge two (or more) transactions. A valid transactions satisfies the requirement $sum(inputs) - sum(outputs) = kernel_excess$. With two transactions we get:

$$sum(T1_inputs) - sum(T1_outputs) = T1 - kernel_excess$$
$$sum(T2_inputs) - sum(T2_outputs) = T2 - kernel_excess$$

If we add this equations to one another, we get:

$$sum(T1_{inputs}) - sum(T1_{outputs}) + sum(T2_{inputs}) - sum(T2_{outputs}) = T1_{kernel-excess} + T2_{kernel-excess}$$

$$sum(T1_{inputs} \cup T2_{inputs}) - sum(T1_{outputs} \cup T2_{outputs}) = T1_{kernel-excess} + T2_{kernel-excess}$$

This suggests a merging method. A "basic" cut-through transaction will be constructed thus:

1. The inputs for the new transaction will be a union of all inputs from the transactions to-be-merged.
2. The outputs for the new transaction will be a union of all outputs from the transactions to-be-merged.
3. The kernel-excess for the new transaction will be a sum of all kernel-excess from the transactions to-be-merged.

If we look at the list of the newly-constructed transaction, we might see that some of the inputs in the transactions are referencing (==spent-by) outputs which are also used in the transaction. These inputs and outputs will cancel each other in $sum(inputs) - sum(outputs)$ of the cut-through transaction. We can remove both these inputs and outputs without changing $sum(inputs) - sum(outputs)$ and the transaction will remain valid.

Another small improvement we can add is randomizing the union-ed lists of the inputs and outputs. This reduces the ability to guess which outputs and inputs originated from the same transaction.

2.9 Blockchain data - Block and full history data cut-through

Cut-through transaction can be generated for a whole block. Because we remove outputs spent in this block, most of the inputs and outputs cancel each other. The Block Cut-through transaction will contain:

1. A list of inputs. The data in this list has originated from 2 different sources:

   All mining profits for this block.

   All commitments spent in this block and generated before it.

   Simply put, it is the commitments for "all money entering the block".
2. A list of outputs. This list will contain all outputs generated by transactions of this block, without being spent.

   Simply put, it is the commitments for "all money exiting the block".
3. A list of kernel-excess representing all the transactions in this block.

The privacy benefits here are immense - all of the transaction structure has been removed. Deducing data form the structural metadata of the transactions (a tool widely used by bitcoin de-anonymization services) is no longer possible!

Because of the structure removal, the amount of data we need to save for any validation using this is much smaller.

The next logical step is to create a "Cut-through All The Way" transaction - cut-through transaction for the whole blockchain history. The Block Cut-through transaction will contain:

1. The All mining profits for this chain.
2. Complete set of unspent outputs, represented by Pedersen commitments.
3. A list of kernel-excess representing all the transactions ever to occur on the network.

This transaction applies the benefits of the Block cut-through transaction for the whole network. On he privacy side - all the transferred amounts are encrypted, not ownership "addresses" and most of the transaction metadata is destroyed. The blockchain a node needs to maintain is very small (on the order of a few gigabytes for a bitcoin-sized blockchain, and potentially optimizable to a few hundreds of megabytes).[gric]

2.10 Wallet communications

When 2 Grin users create a transaction, they exchange the data in a P2P fashion, not through a blockchain node. After the transaction is fully constructed, it will be posted to a node to be validated and accepted by the chain.

The default wallet implemented by Grin offers 3 P2P ways of communication: [grid]

1. HTTP : The sender and the recipient will know each other's IP.
2. TOR : The recipient has he's wallet hidden as an onion service. This method P2P communication with little to no data that one party can learn about the other.
3. Files : This allows for a truly asynchronous transaction creation. This method is also the most flexible of the three. If used correctly, this feature can allow to P2P "communication" with anonymity.

This behavior can be exploited by "evil super nodes". The attacker will deploy nodes which will receiver many of the transactions posting occurring over the network. By looking at the source IPs of incoming communications, he will be able to deduce many of the transactions originators IP's. [grib]

The user can protect itself against such attacks by hiding behind Tor, VPN or NAT. Another solution is Dandelion.

2.11 Transaction posting - Dandelion

Dandelion is a spreading protocol. This protocol allows a group of peers to mask the origin of a request (In our case, a transaction POST request). The protocol will bounce the transaction request between some random peers in the group, and then all the nodes in the group will send the request to it destination (In our case, a Grin node). Grin has implemented a simplified version of the Dandelion++ protocol, with several changes. One of them is transaction aggregation.

Because of Grin's transaction nature, we can aggregate transactions before they are being broadcasted to the entire network (using cut-through). This is a benefit to blockchains that enable non-interactive CoinJoins on the protocol level, such as Mimblewimble. Despite its good privacy features, some input and output linking is still possible in Mimblewimble and Grin. If you know which input spends to which output, it is possible to construct a (very limited) transaction graph and follow a chain of transaction outputs (TXOs) as they are being spent. Aggregating transactions make this more difficult to carry out, as it becomes less clear which input spends to which output (Figure 3). In order for this to be effective, there needs to be a large anonymity set, i.e. many transactions to aggregate a transaction with. Dandelion enables this aggregation to occur before transactions are fluffed and diffused to the entire network. This adds obfuscation to the transaction graph, as a malicious observer who is not participating in the stemming or fluffing would not only need to figure out from where a transaction originated, but also which TXOs out of a larger group should be attributed to the originating transaction [dana].

3 Monero

3.1 Short history

The cryptocurrency Monero, initially known as BitMonero, was created in April 2014 as a derivative of the proof-of-concept currency CryptoNote [bit]. In Esperanto, Monero means 'money'.

CryptoNote itself started has been build and improved using ideas from several different papers. Some of the most infuantial amongst them include CryptoNote V2.0 [vS], Borromean Ring Signatures (for amount hiding) [MP15] and Ring Confidential Transactions (for ring signatures) [SNT16] . [kN20b]

Monero's main benefit over "Standard cryptocurrencies" is improved anonymity. Monero has receiver anonymity through the use of one-time addresses, sender ambiguity by means of ring signatures and amount hiding based on Pedersen commitments.

3.2 Transaction - Basic definitions

Monero transactions are called "Ring Confidential Transactions" - or RingTC in short. Transaction represents a money exchange. Outputs from previous transactions are used as "inputs" - we take the money they contain, and redistribute it into different outputs. Outputs are owned by different entities, each one with a wallet.

Not all of the inputs are used to supply money into the transaction. For any 1 "real" input, we have v "decoy" inputs. The purpose of the the decoy inputs is to hide the origin of the money incoming into the transaction.

This transaction characteristic is an important part of the fungibility property to Monero. The term fungibility refers to assets whose units are considered indistinguishable and interchangeable [StMc18a]. Unlike Bitcoin, where the whole history of the money flow can be seen, Monero hides it.

Like Grin, The actual amount of money in each output (and input) is hidden using a Pedersen commitments.

3.3 Monero user addresses

The wallet of each user is defined by a set of 2 asymmetric keys. We will call the 2 private keys k_v, k_s and their matching public keys K_v, K_s. k_v name is "view key". Anyone who has this key can find all the outputs ever owned by this wallet, and the amount of money in each one. The view key can be sent to other trusted parties, and they will be able to see your activity. We can think of owning view key as read permissions. k_v name is "spend key". Without this key, the outputs of this wallet cannot be spend - not even if you have the view key. We can think of owning view key as write permissions.

Monero has 2 different user addresses:

1. Primary address - Each wallet has a single primary address. The primary address is K_v, K_s. This address can can be published, and payments to this address can be spent by the owner of the wallet. The best practice is to keep the primary address private, and use a subaddresses instead.

2. Subaddress - Monero allows to derive new set of keys $k_{v,i}, k_{s,i}$ and $K_{v,i}, K_{s,i}$. We can think on a subaddresses as a separate wallet with 2 differences. The subaddress will have a view key, spend key and address that enables money receiving , like a normal wallet. The 2 differences are - subaddress cannot have a subaddress, and the primary keys can be used as the view key and spend key of the subaddress. For example - If I have the view key of the primary address, I can view the amount of money owned by any subaddress generated for this wallet. Outside viewer cannot know the relationship between subaddresses. For example - 2 address for 2 subadresses generated by the same primary cannot be identified as "belonging to the same wallet" by outside viewer.

3.4 Transaction outputs - one time address

Each transaction output has a one-time address. This address represents ownership of of the output, without revealing the public address of the user owning this transaction. Every transaction output is created with a unique set of keys - (K^0, k^0). The one-time address of the transaction is K^0. The sender (who generated the one-time address) uses a random seed r and the public keys of the receiver (K_v, K_s) to generate this address. The one-time address is attached to the output as part of the transaction data sent to the blockchain.

We will look at the functions generating K^0 and k^0. After that, we will show why these keys can act as an ownership statement.

$$k_B^v, K_B^v - \text{private, public set for the recipient view keys}$$
$$k_B^s, K_B^s - \text{private, public set for the recipient spend keys}$$
$$K^0 = H_n(r * K_B^v)G + K_B^s = (H_n(r * K_B^v) + k_B^s)G$$
and this means that the appropriate private key k^0 is:
$$k^0 = H_n(r * K_B^v) + k_B^s$$

Before we can dive into the properties, there are a few details to keep in mind.

1. H_n is a hash function. It remains the same for all transactions.
2. G is an elliptic curve point generator. t remains the same for all transactions. Because G is an elliptic curve point generator, r is "hard" to extract from r*G.
3. r*G is written as part of the transaction data. It is called "transaction public key".

Lemma: Anyone who has the view key of the $B1$, k^v_{B1}, and it's public address (K^v_{B1}, K^s_{B1}) can find out if an output with a one time address K^0 is addressed to the user $B1$.

Proof : Any with the view private key k^v_{B1} can calculate $r * K^v_{B1}$.

$$k^v_{B1} * \text{transaction public key} = k^v_{B1} * (r * G) = r * K^v_{B1}$$

Using this, we can calculate $K'^s_B = K^0 - H_n(r * K^v_{B1})G$. If and only if $K'^s_B = K^s_{B1}$ the output belongs to $B1$. This is what we want from a view key - any who has it, can see which outputs in the blockchain belong to the wallet.

Without the view key, validating that $B1$ "owns" the output with K^0 is "hard". If we do not have k^v_B, we must try and use only public data to deduce the target user of the output. Even if we suspect that $B1$ is the target user, we must validate that $K^0 = H_n(r * K^v_{B1})G + K^s_{B1}$. r is kept secret - this means that we cannot calculate $H_n(r * K^v_{B1})G + K^s_{B1}$ directly. $H_n(r * K^v_{B1})G$ must be "hard" to reverse, or else, H_n is a bad hash function.

All of the above means, that users can detect incoming funds by moving on new outputs and checking for each of them if $K^0 - H_n(r * K^v_{user})G$ matches K^s_{user}. To check for all subaddresses belonging to the wallet in one iteration, the user can use his wallet primary view key. The view private key can be given to third parties (business partners, taxes department, etc.) and then these entities will be able to see the wallet funds without being able to spend them. Subaddresses view keys can be shared if we don't want to share the full data of the wallet.

Lemma: Anyone who has both keys k^v_B, k^s_B, can calculate the private key k^0.

Proof: As we have seen before, $k^0 = H_n(r * K^v_B)G + k^s_B$. We have shown before that using the transaction public key and k^v_B we can calculate $r * K^v_B$. This means that we can calculate $H_n(r * K^v_B)G$. With k^s_B known, $H_n(r * K^v_B)G + k^s_B$ can be calculated.

In order to use this output later as a new transaction input, k^0 must be known to the spender (as a part of output key image computation) [kN20c]. This means the spend key should be known to you if you want to spend the output. This key shouldn't be shared.

3.5 Ring signatures

Signatures are a well known cryptographic building block. They serve as a proof system - allowing us to validate the authenticity and integrity of data. For regular signatures only a single signing key (either symmetric or asymmetric) exists. Validating the signatures insures that the data has been signed by the particular user/person that owns this particular key.

But what if we want to say that at least one entity from a group has signed the data, without revealing which one of the group's entities has actually signed the data? we could use a Group signature scheme - a signature schemes designed as signatures with an anonymity layer, but with an administrator. Group signatures allows for single user to sign a data anonymity on behalf of the group. In these schemes, only the administrator can reveal the original signer add or remove users.

Ring signatures is an improved version of group signatures - no admin exists. The anonymity is always insured and no additional setup is required.

Monero uses ring signatures to hide which inputs are real and which outputs are fake. More details will be given later.

3.6 Double spending

Monero hides the outputs actually used in each transaction. This feature is important for keeping the transaction privacy, yet, it poses a problem - how can the network validate that no output has been spent twice by a user? If users will be able to spend output twice or more, owning 1 Monero will be the same value as owning 100 Monero. For a money system, this shouldn't be allowed. For every input used in the transaction, the sender attaches an "output key image".

$$K_{image} = k^0 * H_p * (K^0)$$

H_p is a hash function and k^0 is the private key of the spent output. This means that given K_{image}, it is "hard" to calculate K^0 and find which output has produced this key image. In every transaction, the sender attaches a list of key images. Each of this key images has been produced from an input actually spent in this transaction. The Monero network stores the key image of every spent output. Before accepting any new transaction, the node verifies that non of key images listed in the transaction appear in the "key images of spent outputs" list. k^0 of an the private key of the output. The key image cannot be generated without owning the output, and no transaction can be accepted without valid key images.

3.7 Pedersen commitments

Like Grin, Monero uses Pedersen commitments to hide the amount of money included in each output and input. The actual way Monero does it is different.

A Pedersen commitment has the following structure:

$$C = b * G + v * H$$

b is the "blinding factor" (a random value, different for each commitment) helping us to hide the actual amount. It is also known as "mask" or "commitment mask". v is the amount of money the commitment hides. The blinding factor should be kept a secret. $v * H$ is hard to deduce theoretically, but using a Rainbow table attack can helps us deduce the correct amount for common values of v. Monero transaction is fully calculated on the senders side. If so, how can the blinding factor be shared secretly? The blinding factor is calculated using $r * K_v^B$ (transaction secret seed and the receiver's public view key) with a specific equation. Inverting this equation requires the transaction public key (written in the transaction) and private view key. This insures that only entities with viewing permission can read the received amount. The actual details on the commitments balancing (allowing outside viewer to verify that the sum of funds coming from the inputs is equal to the sum exiting the transaction via outputs) are technical and less relevant to our discussion. For any interested parties, full details can be found in - [kN20d].

Monero valid input/output amount has minimum and maximum range. "Range proofs" are attached to the transaction and allow us to validate this constraint.

3.8 Transaction posting

Monero doesn't have out-of-the-box feature to protect a users against network traffic analysis or node network data logging. Malicious node who logs the source IP of incoming transaction is an example for such an attack. The Monero community has also supported the development of Kovri, a privacy approach based on the decentralized Invisible Internet Project (I2P) specifications [StMc18b]. I2P is a fully encrypted private network layer. I2P has a privacy network service, and has many similarities to TOR. At the moment the future of this I2P router is unclear and an integration with Monero is not planned. [get].

4 Zerocash and Zcash

4.1 Short history

Zerocoin is a privacy protocol proposed in 2013 by Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin [Rub13]. Zerocoin has been proposed as an extension over the Bitcoin system, allowing for

hidden payment input. Zerocoin did not hide transaction amount or destination address.

Zerocash is an evolution of Zerocoin, allowing fully anonymous payment. Unlike Zerocoin, Zerocash is designed as a separate currency. It was first proposed in the article "Zerocash: Decentralized Anonymous Payments from Bitcoin" [EBSACIMCGET14b].

Zcash is an implementation of the Decentralized Anonymous Payment scheme Zerocash, with security fixes and improvements to performance and functionality [Wil21a]. Zcash has been first released in 2016.

Note: Zcash and Zerocash use different names for the same objects. I chose to manly use the Zerocash names, except for the cases where the Zcash names made the objects easier to understand.

4.2 transparent and shielded values

Zerocash allow for both transparent and shielded payment. A transparent payment will contain all data in a cleartext, much like a standart Bitcoin transaction. A shielded payment hides the values, source inputs and created outputs allowing for improved privacy characteristic.

Transparent and shielded money amounts are treated as 2 interchangeable currencies on the same network. A Mint transaction takes a transparent amount and translates it into shielded amount. A Pour transactions main propose is transferring shielded amounts to different outputs (which may be owned by different users), but it can also be used to translate shielded amounts to transparent amounts.

4.3 zk-SNARK, commitment scheme and PRF

A zero knowledge proof is a proof system in which one party can prove to another party (the verifier) that a given statement is true, without conveying any information apart from the fact that the statement is indeed true.

Non-interactive zero-knowledge (NIZK) conveys a the the proof without direct communication. This is an example for such a scheme [Ros20]:

1. Anna wants to prove to Carl that she knows a value x such that $y = g^x$ to a base g.
2. Anna picks a random value v from a set of values Z, and computes $t = g^v$.
3. Anna computes $c = H(g, y, t)$ where $H()$ is a hash function.
4. Anna computes $r = v – c * x$.
5. Carl or anyone can then check if $t = g^r * y^c$.

A zk-SNARK is a computationally efficient NIZK.

PRF is an acronym for a pseudo-random function. With the proper build, several different PRF can be derived base on specific single function. $r = PRF_z^a(q)$.

The commitment scheme of Zerocash accepts 2 parameters - random seed and a message $c = COMM(r, m)$.

The pseudo-random function and commitment scheme (which we will call comm) of Zerocash are based on SHA256.

4.4 Address structure

Each user of the Zerocash network will have a wallet. The wallet will know the proper communication protocols (for generating and sending all kinds of transaction) and keep all of the user data.

A wallet can have several addresses. Each address will have 3 pairs of keys.

The first pair is (a_pk, a_sk). The public address is the address others will use when they are sending money to the wallet. The private address is a must-know to spend outputs created for the public address. The public key is generated using a PRF and the secret key - they are not a "normal" key pair.

The second pair is (pk_enc, sk_enc). pk_enc are used to encrypt data addressed for the wallet (and it's owner).

The third pair is (pk_sig, sk_sig). This pair is used to sign transactions sent by the wallet against any tempering.

In later Zcash versions, the key structure has been altered to add the functionality of a "viewing key". In Sapling and Orchard (Zcash versions), for each spending key there is a full viewing key that allows recognizing both incoming and outgoing notes without having spend authority. This is implemented by an additional ciphertext in each Output description or Action description [Wil21b].

4.5 Mint transaction

A Mint transaction takes a transparent amount and translates it into shielded amount.

We will explain the ideas in Mint transaction using some "devolved" Mint transaction example. (The Zerocash article does the same).

Toy example - user anonymity with fixed-value coins.

We will assume that all mint transaction will translate 1 transparent coin into 1 shielded coin.

Create the Mint transaction:

1. The user u generate 2 random value - serial number sn and regular normal number r. Knowing this 2 values will prove the ownership of the shielded coin.
2. Generate the coin commitment $cm := COMM(r, sn)$.
3. Create the new shielded coin $c := (r, sn, cm)$.
4. $tx_Mint = (c)$. send this transaction, while paying 1 transparent coin to the network. As we will show, the user u have enough information to use it later.

The network can validate tx_Mint. Because c contains r and sn, calculating $COMM(r, sn)$ will validate the value of cm.

In order to allow the spending of this c, the network will save cm in a merkel tree. We will call it CM-Tree. This will allow the Zerocash network to answer the following query - given a cm, has it ever been sent to the network?

Using a Merkel tree allows us to append and query efficiently for large amounts of received cms.

Spending the transaction:

The user u can send a spend mint transaction $tx_Spend = (sn, \pi)$ to the network to spend the c the user has minted before. π is a zk-SNARK for "I know r such that $COMM(r, sn)$ appears in the CM-tree of coin commitments as a leaf".

Knowing r, sn and the ability to proof that cm already exists in the system is a proofs of both historical deposit (that we can now "take back") and the ownership of this deposit. Because we used a zero proof, tx_Spend doesn't actually reveal which minted coin we spent.

To avoid double spending, the sn in tx_Spend will be verified against the "serial numbers of spent coins" set. If it doesn't exist there and everything checks out, allow the spending and add sn to the "sn spent set".

Now, let us improve on this foundation. We would like to be able to mint changing amounts of coins and allow for targeted user payments. To allow this feature, we will incorporate the target user address (a_pk) and amount (v) into the minting.

Create the Mint transaction:

1. The user u generate 3 random values - ρ, s and r.
2. Generate the serial number sn based on ρ and the user's secret key (a_{sk}): $sn := PRF^sn_{a_{sk}}(\rho)$.
3. Generate base commitment incorporating the public address and ρ (representing sn data) $k = COMM(r, a_{pk}||\rho)$.
4. Complete the generation the coin commitment $cm = COMM(s, v||k)$.
5. Create the new shielded coin $c := (a_{pk}, v, \rho, r, s, cm)$. (Do not send the full c to the network).

6. $tx_M int = (v, k, s, cm)$. send this transaction, while paying v transparent coin to the network. As we will show, the user u have enough information to use it later.

Like in the toy example, the coin can be verified. The public address and ρ are hidden by the first $COMM$ (not required for the cm verification) and thus the user remains anonymous and the serial number a secret. The minted amount v is known.

The transfer of shielded amounts and shielded to transparent translation is made using a pour transaction.

4.6 Pour transaction

Pour transaction are used either to consume one or more shielded coins (as inputs) or shielded amounts to transparent amounts.

Coins are spent using the pour operation, which takes a set of input coins, to be consumed, and "pours" their value into a set of fresh output coins — such that the total value of output coins equals the total value of the input coins. Suppose that u, with address key pair $(a_{pk}^{old}, a_{sk}^{old})$, wishes to consume his coin $c^{old} = (a_{pk}^{old}, v^{old}, \rho^{old}, r^{old}, s^{old}, cm^{old})$ and produce two new coins c_1^{new}, c_2^{new} with $v_1^{new} + v_2^{new} = v^{old}$, respectively targeted at address public keys $a_{pk,1}^{new}$ and $a_{pk,1}^{new}$. (The addresses $a_{pk,1}^{new}$ and $a_{pk,1}^{new}$ may belong to u or to some other user.) [EBSACIMCGET14a]

The user u, for each i in 1, 2, proceeds as follows:

1. u samples serial number randomness ρ_i^{new}
2. u computes $k_i^{new} = COMM(r_i^{new}, a_{pk,i}^{new} || \rho_i^{new})$ for random r_i^{new}.
3. u computes $cm_i^{new} = COMM(s_i^{new}, v_i^{new} || k_i^{new})$ for random s_i^{new}.

By this action, u has created 2 new shielded coins with the values v_1^{new}, v_2^{new} targeting $a_{pk,1}^{new}$ and $a_{pk,1}^{new}$.

This yields the coins:
$c_1^{new} = (a_{pk,1}^{new}, v_1^{new}, \rho_1^{new}, r_1^{new}, s_1^{new}, cm_1^{new})$
$c_2^{new} = (a_{pk,2}^{new}, v_2^{new}, \rho_2^{new}, r_2^{new}, s_2^{new}, cm_2^{new})$.

Next, u produces a zk-SNARK proof π_{POUR} aiming to prove:

1. The soundness for c^{old}, c_1^{new} and c_2^{new}
2. user u owns c^{old}.
3. c^{old} is a non-spent coin in the Zerocash network at this point.
4. $v_1^{new} + v_2^{new} = v^{old}$.

A resulting pour transaction $tx_{Pour} := (rt, sn^{old}, cm_1^{new}, cm_2^{new}, \pi_{POUR})$ is appended to the ledger. (rt is a CM-Tree root for cm^{old} existing). (As before, the transaction is rejected if the serial number sn appears in a previous transaction.)

Now suppose that u does not know, say, the address secret key $a_{sk,1}^{new}$ that is associated with the public key $a_{pk,1}^{new}$. Then, u cannot spend c_1^{new} because he cannot provide $a_{sk,1}^{new}$ as part of the witness of a subsequent pour operation. Furthermore, when a user who knows $a_{sk,1}^{new}$ does spend c_1^{new}, the user u cannot track it, because he knows no information about its revealed serial number, which is $an_1^{new} = PRF^s n_{a_{sk,1}^{new}}(\rho_1^{new})$. [EBSACIMCGET14a]

Also observe that tx_{Pour} reveals no information about how the value of the consumed coin was divided among the two new fresh coins, nor which coin commitment corresponds to the consumed coin, nor the address public keys to which the two new fresh coins are targeted. The payment was conducted in full anonymity. [EBSACIMCGET14a]

In order to allow u_1 to actually spend the new coin c^{new} produced above, u must somehow send the secret values in c_1^{new} to u_1. We encrypt them using $pk_{enc,1}^{new}$ and add them as extra data to the transaction. The user u_1 can then find and decrypt this message using his private encryption key for the address.

The construction so far allows users to mint, merge, and split coins. But how can a user redeem one of his coins, i.e., convert it back to the transparent currency. For this, we modify the pour operation to include a public output. This output can be 0, if the feature isn't required for this transaction.

4.7 Zcash changes

Over the time and versions, Zcash has added features and solved security errors. These include the addition:

1. "viewing key" - possessing this key allows to find all outputs addressed for a specific address, without the ability to spend them.
2. Instead of Mint and pour transaction, Zcash transaction can have a list of Zcash operations in one transaction. This allows for several operations (like mint and immediately use in pour) in one transaction.
3. Some extra fields in the transaction (like a "memo" field).
4. Some security fixes [Wil21c].
5. Amount hiding using Pedersen commitments, much like Monero.

A full list can be found at "Zcash Protocol Specification" section 8.

5 Privacy features analysis

5.1 User-addresses hiding

A public address is a popular way of representing a wallet to different parties across the chain. Naturally, This means that the address should not be written openly in any elements on the blockchain (outputs/transactions).

In Grin transaction, the sender and recipient communicate directly before sending a transaction data to the network. Because of this, Grin doesn't add the recipient address as a part of the transaction data sent to the network. This approach pose a different challenge - how to establish direct communication between the sender and the receiver without giving away data which may lead to the identification of either one of the parties. We will discuss this question later.

Monero hides the address parts (K_B^v, K_B^s) in the transaction under the one-time address using Hash function (H_n) and elliptic curve point generator (G).

Zerocash uses hash function to hide the function inside the coin commitment parameters (specifically k in $cm = COMM(s, v||k)$).

5.2 Amount hiding

All cryptocurrencies allow its users to send different amounts of money in a transaction. Analysing the amounts exchanged in different transaction might reveal data on the transaction participants, even if the user addresses are hidden.

Lets create an toy example:

Bitcoin (and most cryptocurrencies) value in $ varies considerably every second. Bitcoin allows payments in very precise amounts. This means that a payment of X$ worth of BTC at specific date and time can be translated to a unique value.

If we know that Alice payed Bob X$ at 12:00 01/01/3333, we can translate it to a "AB.abcdefghi" BTC (all leters representing digits). If we find a transaction producing an output with the amount "AB.abcdefghi" in a block of a matching time frame, we can be fairly certain that this is the transaction Alice used to pay Bob. Now, moving over the transaction graph (which can be deduced from the blockchain), we can find more inputs and outputs related to Alice. Crossing this information with other sources will help us extend our intelligence on Alice.

Grin, Monero and Zcash use Pedersen commitments to hide the payment amounts. In the original Zerocash article, the amounts remain unhidden.

5.3 Inputs and outputs linking - transaction history

The basic definition that is given for metadata is that it is "data about data.". One of the several types of metadata is structural metadata. This type of metadata holds containers of data and specifies how compound objects are put together. [met20].

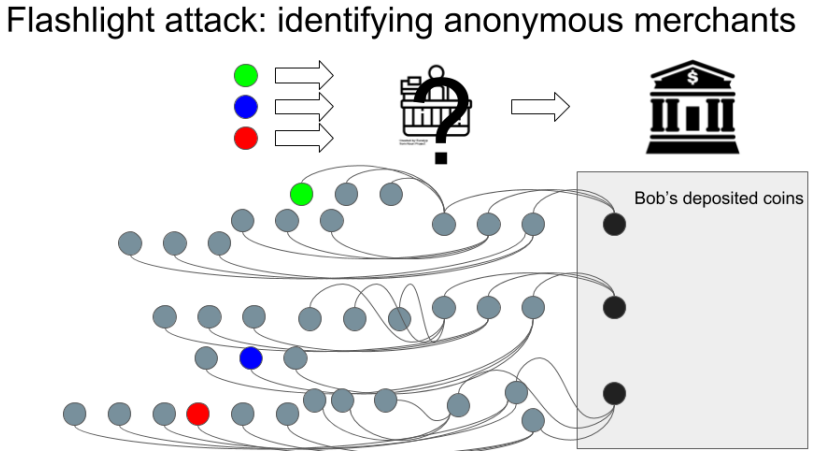
No currency should be easy to forge. If a user can add to himself as much money as he wants, there is no real difference between owning 1 coin to 1000 coins, and the currency doesn't mean anything.

One of the tools cryptocurrencies in proving that is attaching the inputs used and outputs created in the transaction. This provides a structure of money changing hands. This is structural metadata, which we can use to gather intelligence on users using analysis and attacks.

Monero uses decoys to deform this connection graph. In a transaction, for each "real" input (an input consumed by this transaction) it adds 10 additional "decoy" inputs. Using ring signatures and key images, Monero allows for sound transaction and indistinguishability of "real" inputs from "decoy" inputs.

Some attacks can still work, even if decoys are used. One such an attack is a flashlight attack.

This attack tries to find if person X owns the wallet Y. This attack assumes that X is a repeating customer at a specific business (preferably the exchange) and the attacker has a view key for that business. As a first step, the attacker sends multiple payments to Y. This means that the blockchain will contain several outputs owned by Y, and they will be connected to inputs used in the attacker payments. Lets call these outputs Z. The attacker wait, and checks the transactions chain. If a large group of Z outputs are connected to the business, then with high probability X owns Y. Why? choosing a specific output as a "decoy" input has a very low probability. The probability several Z items will be used as decoys for transactions of specific business (not too large) for not-so-long time frame are very low. If they are used as real inputs, the chance X used some of his inputs which are also in Z is much larger.



Zerocash doesn't connect hidden inputs to its transaction directly. It provides a a zero-knowledge proof that it had deposited a minted coin containing the amount v. It doesn't point to a specific one, and doesn't use decoys. In this category, Zerocash has the best solution.

Grin doesn't save transactions history. A malicious node can record all the outputs and inputs as seen at each time. This will allow the attacker to reconstruct the transactions graph. Grin forgets used outputs, so without recording the data it cannot be retrieved. Grin is the closest to "the right to be forgotten" in this aspect.

5.4 User internet characteristics

Client network properties (like IP) are another kind of metadata that reveals information on the user. A state-scale attacker can gather this information on users by large deploy several nodes. By observing the spreading dynamic of a transaction, it has been proven possible to link it (and therefore also the

sender's Bitcoin address) to the originating IP address with a high degree of accuracy, and as a result deanonymize users[danb]. A state-scale might also be able to Sigint tracking of any parties communicating with the blockchain servers. This attacker might then, for example, mark the cryptocurrency users as "suspicious" and put them under surveillance.

Using anonymization networks (like Tor) when communicating with the network is the main solution suggested by most of the articles.

Grin wallet reference implementation has the ability to publish a communication end-point as an onion service. Grin has more solutions - more on this later (subsection 4.6).

5.5 Wrong user usage

No matter how sophisticated the solution, the human factor is always a weak point. A good design limits the dangerous operations a user can do.

Grin is a bit of a mixed bag in that aspect.

The requirement of direct communications between the sender and receiver of the transaction might reveal network properties (like IP). The wallet reference implementation allows for direct HTTP communication between them. Using this method without protection like VPN or Tor exposes the 1 or 2 sides of the transaction. The files exchange method (to transfer transaction data between one another) should be used carefully. If used incorrectly, it can is dangerous on several levels.

- An attacker might send a virus file under the pretence that it is part of the transaction, and the victim will most likely download the file to his computer before validating it. The can
- If the file is transferred via hardware (flush drive, etc), the user can be tempted to insert a hardware sent for him into his own computer (for accessing the file). This is an excellent jump point for the attacker (the Stuxnet is a good example).
- The use of files allow to easily integrate the communications between the parties into secure channels, without any code on the wallets side.

Monero and Zerocash use public addresses to allow the transaction to happen without direct communication between the users. These addresses are wallet identifiers. If a user has two different activities and he doesn't want to link them, 2 different addresses should be used. Both of these currencies support several addresses per 1 wallet. Changing the address between different clients and/or transactions is good practice.

Monero wallet has a primary address and keys, from which all subaddresses are generated. The primary view key can be used as a view key for any subaddresses, and the primary spend key can be used as a spend key for any subaddresses. This key allows for efficient wallet management, but poses a risk. If the view primary key is revealed to a third-party, this wallet become transparent for this third party. Even payments made for subaddresses created after the primary view key transfer can be seen. If both of them is revealed, present and future holding of this wallet become spendable to any who have this key. The better practice is sharing the view/spend keys of the required subaddresses.

5.6 CoinJoins (Mixers)

Of the 3 cryptocurrencies, only the Grin documentations mentions CoinJoins.

The Grin CoinJoins will use the cut-through technique to merge several transaction into a single one. The Grin development team plans to integrate transaction mixing with the Dandelion system. The CoinJoins will be effective only when using large number of transactions. When using a cut-through, only outputs which serve as inputs in different transaction will disappear. Other inputs and outputs remain the same. An Improved understanding of the benefits of transaction aggregation prior to fluffing is listed under "future work" [danc]. CoinJoins might be a partial solution to the potential transaction linking problem we mentioned before.

5.7 Post-quantum privacy guaranty

In 2021, quantum computers are no longer fiction. Quantum computers today (to the best of public knowledge) have small amounts of qbits, but corporations and states are pushing the the limit

each year. It is possible that at some point not too distant in the future we might reach quantum supremacy. Several cryptosystems like RSA and elliptic curve (EEC) are based on the difficulty of prime factorization. Peter W. Shor has published in 1994 an article detailing an algorithm for prime factorization in polynomial-time using a quantum computer [Sho94]. Unlike EEC, SHA-256 is theorized to be quantum-resistant[sha]. This means that in a few years cryptocurrencies attackers might be able to break systems like Pedersen commitments, while other parts remain resistant. This will effect the privacy guaranty of these cryptocurrencies.

Each and every one of the cryptocurrencies we have seen are using EEC and similar algorithm for encryption and signing. These parts will probably have have to be changed or replaced. For example, Pedersen commitments will no longer be difficult to break by attackers with quantum computer access. Grin, Monero and Zcash will have to find another way to hide the transaction amounts.

Other parts ,like using SHA256 as a commitment scheme, can remain the same.

Monero and Zcash save the full transaction history on their networks. The attackers will be able to use this information to decode parts of the old data.

6 Conclusion

In this article, we have over-viewed three cryptocurrencies - Grin, Monero and Zerocash. These three currencies has privacy guaranties, unlike Bitcoin. In this overview we learned how these cryptocurrencies operate, while paying special attention to anything concerning the privacy features.

After the overviews, We have gone over a list of privacy features. Each one has been explained, and its existence and limitation in each of the three currencies has been presented.As we have seen multiple times, the correct handling of metadata is an important part of privacy guaranty for every system.

Grin, Monero and Zcash - Each has its own limitations and strong points. There is no one "solution" or "ultimate coin" to insure perfect privacy.

References

[bit] Bitmonero - a new coin based on cryptonote technology - launched.

[dana] Dandelion++ in grin: Privacy-preserving transaction aggregation and propagation.

[danb] Dandelion++ in grin: Privacy-preserving transaction aggregation and propagation.

[danc] Dandelion++ in grin: Privacy-preserving transaction aggregation and propagation.

[EBSACIMCGET14a] Matthew Green Madars Virza Eli Ben-Sasson Alessandro Chiesa Ian Miers Christina Garman Eran Tromer. In *Zerocash: Decentralized Anonymous Payments from Bitcoin*, pages 7–8. extended version edition, 2014.

[EBSACIMCGET14b] Matthew Green Madars Virza Eli Ben-Sasson Alessandro Chiesa Ian Miers Christina Garman Eran Tromer. Zerocash: Decentralized anonymous payments from bitcoin (extended version). 2014.

[get] Moneropedia - kovri.

[gria] A-brief-history-of-mimblewimble-white-paper.

[grib] Grin dandelion - motivation.

[gric] Grin github into - page 12.

[grid] Grin wallet user guide.

[Jed16] Tom Elvis Jedusor. Mimblewimble. *public domain*, 2016.

[kN20a] Kurt M. Alonso koe and Sarang Noether. In *Zero to Monero*, page 44. public domain, 2.0 edition, April 2020.

[kN20b] Kurt M. Alonso koe and Sarang Noether. In *Zero to Monero*, pages 3–4. public domain, 2.0 edition, April 2020.

[kN20c] Kurt M. Alonso koe and Sarang Noether. In *Zero to Monero*, page 38. public domain, 2.0 edition, April 2020.

[kN20d] Kurt M. Alonso koe and Sarang Noether. In *Zero to Monero*, pages 43–48,49. public domain, 2.0 edition, April 2020.

[met20] Metadata and signals intelligence (sigint), December 2020.

[MP15] Gregory Maxwell and Andrew Poelstra. Borromean ring signatures. 2015.

[Poe16] Andrew Poelstra. Mimblewimble. *public domain*, 2016.

[Ros20] Ameer Rosic. What are zksnarks? the comprehensive spooky moon math guide. 2020.

[Rub13] Ian Miers Christina Garman Matthew Green Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. 2013.

[sha] Dandelion++ in grin: Privacy-preserving transaction aggregation and propagation.

[Sho94] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. 1994.

[SNT16] Adam Mackenzie Shen Noether and Monero Core Team. Ring confidential transactions. 2016.

[StMc18a] SerHack and the Monero community. In *Mastering Monero*, page 28. Justin Ehrenhofer, 1 edition, December 2018.

[StMc18b] SerHack and the Monero community. In *Mastering Monero*, page 73. Justin Ehrenhofer, 1 edition, December 2018.

[vS] Nicolas van Saberhagen. Cryptonote v2.0.

[Wil21a] Daira Hopwood Sean Bowe Taylor Hornby Nathan Wilcox. In *Zcash Protocol Specification*, page 1. Version 2021.2.16 [nu5 proposal] edition, September 2021.

[Wil21b] Daira Hopwood Sean Bowe Taylor Hornby Nathan Wilcox. In *Zcash Protocol Specification*, page 9. Version 2021.2.16 [nu5 proposal] edition, September 2021.

[Wil21c] Daira Hopwood Sean Bowe Taylor Hornby Nathan Wilcox. In *Zcash Protocol Specification*, pages 137–142. Version 2021.2.16 [nu5 proposal] edition, September 2021.