Monero
Privacy in the blockchain

Kurt M. Alonso
kurt@oktav.se

Universitat Autònoma de Barcelona

Abstract

A cryptocurrency blockchain is commonly understood as a public distributed ledger containing transactions verifiable by third parties, be it the mining community or the public in general. It would seem that transactions would need to be sent and stored in clear text format in order to make them publicly verifiable.

As we will show in this report, this is an incorrect assumption. It is indeed possible to use cryptographic artifacts to conceal participants of transactions as well as the amounts involved. And yet, allow transactions to be verified and consensuated by the mining community.

Contents

CHAPTER 1

Introduction

The purpose of blockchains is to furnish trust to operations between unrelated parties, without requiring the collaboration of a trusted third party.

Trust is attained through the use of cryptographic artifacts which cater for virtual immutability and non-falsifiability of data registered in a readily accessible database — the blockchain. In other words, a blockchain is a public distributed database, containing data whose legitimacy cannot be disputed by any party.

Cryptocurrencies store transactions in the blockchain. The latter acts as a public ledger of all the consensuated currency operations. Most cryptocurrencies store transactions in clear text, to facilitate the verification of transactions by the community.

Clearly, an open blockchain defies any basic understanding of privacy, since it virtually *publicizes* complete transaction histories of its users.

To address the lack of privacy, users of cyrptocurrencies such as Bitcoin can obfuscate transactions by using temporary intermediate addresses [15]. However, in spite of such measures, with appropriate tools it is possible to analyze flows and to a large extent link true senders with receivers [20, 7, 18].

In contrast, the cryptocurrency Monero, attempts to tackle the issue of privacy by storing only stealth, single-use addresses in the blockchain. In this manner, there will be no effective way of linking senders with receivers nor tracing the origins of funds [1].

Additionally, transaction amounts in the Monero blockchain are concealed behind cryptographic constructions, so as to render more complicated to infer currency flows.

The result is a high level of privacy, possibly unmatched by other common cryptocurrencies.

1.1 Objectives

Monero is a cryptocurrency of recent creation, yet it displays a steady growth in popularity[1]. Unfortunately, there is little comprehensive documentation available describing the mechanisms it uses. Even worse, important parts of the theoretical framework in the currency have been published in non peer-reviewed papers which are incomplete and/or contain errors. For significant parts of the theoretical framework of Monero, only the source code is reliable enough as source of information.

We intend to palliate this situation by collecting in-depth information about Monero's inner workings, reviewing algorithms and cryptographic schemes, and by discussing the degree to which they might afford sufficient transaction privacy and security to its users.

We have centered our attention on release 0.11.1.0 of the Monero software suite, the most recent release at the moment this is written. All transaction related mechanisms described here belong to this version. Deprecated transaction schemes have not been explored to any extent, even if they may be partially supported for backward compatibility reasons.

1.2 Readership

We expect the reader to possess a basic understanding of discrete mathematics and algebraic structures, but possibly only fundamental insights in the field of cryptography. We also expect the user to have a basic understanding of how a cryptocurrency like Bitcoin works.

A reader with this background should be able to follow our constructive, step-by-step description of the elements of the Monero cryptocurrency.

We have omitted on purpose some mathematical technicalities, when they would be in the way of clarity. We have also omitted concrete implementation details where we thought they were not essential. Our objective has been to present the subject half-way between mathematical cryptography and computer programming, aiming at completeness and conceptual clarity.

Using a consistent notation, a succinct and single-threaded explanation, we believe that it is possible to lead a reader with this background to a deep understanding of the essential aspects of the Monero cryptocurrency.

1.3 Origins of the Monero cryptocurrency

The cryptocurrency Monero, originally known as BitMonero, was created in April, 2014, as a derivative of the proof-of-concept currency CryptoNote.

[1] As of December 28th, 2017, Monero occupies the 10th position as regards market capitalization, see https://coinmarketcap.com/

The latter is a cryptocurrency devised by an individual or team under the pseudonym of Nicolas van Saberhagen. His/their work was published in October 2013 in [21]. It offered sender and receiver anonymity through the use of one-time addresses, and untraceability of flows by means of ring signatures.

Since its inception, Monero has further strengthened its privacy aspects by implementing amount hiding as described by Greg Maxwell, among others in [14], as well as Shen Noether's improvements on ring signatures [17].

1.4 Outline

As hinted earlier, our aim has been to deliver a self-contained and step-by-step description of the Monero cryptocurrency. This report has been structured to fulfill this objective and lead the reader through all elements needed to describe the inner workings of the currency.

In our quest for comprehensiveness, we have chosen to present all the basic elements of cryptography needed to understand the complexities of Monero. In Chapter 2 we develop essential aspects of Elliptic Curve cryptography.

Chapter 3 outlines the ring signature related algorithms that will be applied to achieve confidential but linkable transactions.

In Chapter 4 we introduce the cryptographic mechanisms used to conceal amounts.

Finally, with all the components in place we will be able to expose the transaction schemes used in Monero in Chapter 5.

Appendices A and B describe the structure of sample transactions in the blockchain, providing a connection between the theoretical elements described in earlier sections with their real-life implementation.

CHAPTER 2

Basic concepts

We will use this section to introduce basic cryptographic building elements that will pervade the rest of this report.

2.1 A few words about notation

One main objective in this report has been to collect, review, correct and homogenize all existing information concerning the inner workings of the Monero cryptocurrency. And at the same time supply all the necessary details to present all this material in a constructive and single-threaded manner.

An important instrument to achieve this has been to settle for a number of notational conventions that we think contribute to make this material more readable.

Among others, we have used

- lower case letters to denote simple values, integers, strings, bit representations, etc
- upper case letters to denote curve points and complex constructs

For items with a special meaning, we have tried to use in as much as possible the same symbols throughout the whole document. For instance, a curve generator is always denoted by G, its order is N, private/public keys are denoted whenever possible by k/K respectively, etc.

Beyond that, we have aimed at being *conceptual* in our presentation of algorithms and schemes. A reader with a computer science background could consider that we have neglected questions like the bit representation of items, or in some cases, how to carry out concrete operations.

However, we don't see this as a loss. A simple object such as an integer or a string can always be represented by a bit string. So-called *endianness* is rarely relevant, and is mostly a matter of convention for our algorithms.

Elliptic curve points would normally be represented by pairs (x, y), and therefore could be represented as two integer. However, in the world of cryptography it is common to apply *point compression* techniques, which allow representing a point using only the space of one coordinate. For our conceptual approach it is quite often accessory whether point compression is used or not. But most of the times it is implicitly assumed.

We have also used freely hash functions without specifying any concrete algorithms. In the case of Monero, it will typically be *SHA3*, but if not explictly mentioned then it is not important to the theory.

These hash functions will be applied to any objects, integers, strings, curve points, or combinations of these objects. These occurrences should be interpreted as hashes of bit representations, or the contatenation of such representations.

2.2 Elliptic curve cryptography

2.2.1 What are elliptic curves

A finite field $\mathbb{F}_p$ where p is a prime number, is the field formed by the set $\{0, 1, 2, ..., p-1\}$. with arithmetic operations $(+, \cdot)$ calculated $\pmod p$.

Typically, elliptic curves are defined as the set of points (x, y) satisfying a *Weierstraß* equation:

$$y^2 = x^3 + ax + b \quad \text{where} \quad a, b, x, y \in \mathbb{F}_p$$

However, the cryptocurrency Monero uses a special curve known to offer improved security over other commonly used *NIST* curves, as well as excellent performance of cryptographic primitives. The curve used belongs to the category of s.c. *Twisted Edwards* curves, which are commonly expressed as:

$$ax^2 + y^2 = 1 + dx^2y^2 \quad \text{where} \quad a, d, x, y \in \mathbb{F}_p$$

In what follows we will prefer this second form. The advantage it offers over the previously mentioned Weierstraß form is that basic cryptographic primitives require less arithmetic operations. This results in faster cryptographic algorithms, see Bernstein et al. in [5] for details.

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be 2 points belonging to an elliptic curve. We can proceed to define the addition operation $P_1 + P_2 = P_3$ in the following manner

$$x_3 = \frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2} \pmod{p}$$
$$y_3 = \frac{y_1 y_2 - a x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \pmod{p}$$

These formulas for addition will also apply for point doubling, that is, when $P_1 = P_2$.

It turns out that elliptic curves have *abelian group* structure under the addition operation described[1].

The order N of an elliptic curve EC can be loosely defined as the number of points in the curve. Clearly, by the definition of addition above it follows that an elliptic curve over a finite field will be equally finite.

A generator G of EC is a point in the curve such that for every other point P in EC there exists n satisfying that $P = nG$.

Calculating the scalar product nP can be done without difficulties. However, finding n such that $P_1 = nP_2$ is known to be computationally hard. By analogy to modular arithmetic, this problem is often called the *discrete logarithm problem* (DLP). In other words, scalar multiplication can be seen as a *one-way function*, which paves the way for using elliptic curves for cryptography.

2.2.2 Public key cryptography with elliptic curves

Public key cryptography algorithms can be devised in an analogous way as with modular arithmetic.

Let k be a randomly selected number satisfying $1 < k < N$, and call it *private key*. Calculate the corresponding *public key* $K = kG$.

Due to the hardness of the *discrete logarithm problem* (DLP) we can not easily deduce the value k from K alone. This property allows us to use the values in (k, K) in common public key cryptography algorithms.

2.2.3 Diffie-Hellman key exchange with elliptic curves

A basic *Diffie-Hellman* exchange of a shared secret between *Alice* and *Bob* could take place in the following manner:

1. Alice and Bob generate their own private/public keys (k_A, K_A) and (k_B, K_B). Both publish or exchange their public keys, and keep the private keys for themselves.

[1] A concise definition of this notion can be found under https://brilliant.org/wiki/abelian-group/

2. Clearly, it holds that

$$S = k_A K_B = k_A k_B G = k_B k_A G = k_B K_A$$

Therefore, Alice could calculate $S = k_A K_B$ and Bob $S = k_B K_A$, and use this single value as a shared secret.

An external observer would not be able to calculate easily the shared secret due to the hardness of the DLP.

2.2.4 DSA signatures with elliptic curves (ECDSA)

Typically, a cryptographic signature is performed on a cryptographic hash of a message rather than the message itself. However, in this whole report we will loosely use the term *message* indistinctly to refer to the message properly speaking and/or its hash value.

Signature

Assume that Alice has the private/public key pair (k, K). To sign univocally an arbitrary message $\mathfrak{m}$, she could execute the following steps [10]:

1. Calculate a hash of the message using a cryptographically secure hash function, $h = \mathcal{H}(\mathfrak{m})$
2. Generate a random integer r such that $1 < r < N$ and compute $P = (x, y) = rG$.
   If $x = 0$ generate another random integer.
3. Calculate $s = r^{-1}(h + xk) \pmod N$. If $s = 0$ then go to previous step and repeat
4. The signature is (x, s)

Verification

Any third party can verify the signature by calculating

$$\begin{aligned} u_1 &= s^{-1}h \\ u_2 &= s^{-1}x \\ Q &= u_1 G + u_2 K \end{aligned}$$

The signature will be valid if and only if the first coordinate of $Q = (x_Q, y_Q)$ satisfies that

$$x_Q = x \pmod p$$

.

Why it works

This stems from the fact that

$$\begin{aligned} Q &= u_1 G + u_2 K \\ &= s^{-1} hG + s^{-1} xkG \\ &= s^{-1}(h + xk)G \end{aligned}$$

Since $s = r^{-1}(h + xk)$, it follows that $r = s^{-1}(h + xk)$, whereby it becomes proved that

$$Q = rG$$

.

2.3 Curve Ed25519

Monero uses a particular Twisted Edwards elliptic curve for cryptographic operations, *Ed25519*, *birational equivalent*[2] of the Montogomery curve *Curve25519*.

Both Curve25519 and Ed25519 were released by Bernstein *et al.* [3, 4, 6].

The curve is defined over the prime field $\mathbb{F}_{2^{255}-19}$ by means of the following equation:

$$-x^2 + y^2 = 1 - \frac{121665}{121666} x^2 y^2$$

This curve addresses many concerns raised by the cryptography community. It is well known that *NIST*[3] standard algorithms have issues. In recent times it has become clear that the random number generation algorithm *PNRG* is flawed and could contain a potential backdoor [9]. Seen from a broader perspective, curves endorsed by the NIST, are also indirectly endorsed by the NSA, something that the cryptography community sees with suspicion.

Curve Ed25519 is not subject to any patents (see [12] for a discussion on this subject), and the team behind it has developed and adapted basic cryptographic algorithms with efficiency in mind [6]. More importantly, it is currently thought to be secure.

Twisted Edwards curves have order expressable as $2^c q$, where q is a prime number and c a positive integer. In the case of curve Ed25519, its order is

$$2^3 \cdot 7237005577332262213973186563042994240857116359379907606001950938285454250989$$

[2] Without giving further details, birational equivalence can be thought of as an isomorphism that can be expressed using rational terms

[3] National Institute of Standards and Technology, https://www.nist.gov/

2.3.1 Binary representation

Elements of $\mathbb{F}_{2^{255}-19}$ are 256-bit integers. In other words, they can be represented using 32 bytes.

Consequently, any point in Ed25519 could be represented using 64 bytes. Applying *Point compression* techniques, described here below, however, it is possible to reduce this amount by half, to 32 bytes..

2.3.2 Point compression

The Ed25519 curve has the property that its points can be easily compressed, so that representing a point will consume only the space of one coordinate. We will not delve into the mathematics necessary to justify this, but we can give a brief insight into how it works.

Assume that we want to compress a point (x, y). We will employ a little-endian representation of integers.

Encoding The most significant bit of the y coordinate will always be 0. We set this bit to 0 if x is even, and 1 if it is odd. The resulting value y will represent the curve point.

Decoding Retrieve and clear the most significant bit of the stored value and that will be y.
Let $u = y^2 - 1$ and $v = dy^2 + 1$.
Compute $x = uv^3(uv^7)^{(p-5)/8}$

1. If $vx^2 = u \pmod p$ then keep x
2. Else set $x = x2^{(p-1)/4} \pmod p$
3. Use the parity bit b retrieved in the first step, if $b \neq x \pmod p$ then return $p - x$, otherwise return x

2.3.3 EdDSA signature algorithm

Bernstein and his team have developed a number of basic algorithms based on curve Ed25519. For illustration purposes we will describe here a highly optimized and secure alternative to the ECDSA signature scheme which according to the authors allows producing over 100 000 signatures per second using a commodity Intel Xeon processor [4]. The algorithm can also be found described in Internet RFC8032 [11].

Among other things, instead of generating random integers every time, it uses a hash value derived from the private key of the signer, and the message itself. This circumvents security flaws related to the implementation of random number generators. Also, another goal of the algorithm is to avoid accessing secret or unpredictable memory locations to prevent so-called *cache timing attacks*,

We provide here an outline of the steps performed by the algorithm for illustration purposes only. A complete description and sample implementation in the Python language can be found in [11].

Signature

1. Let h_k be a hash $\mathcal{H}(k)$of the signer's private key k. Compute r as a hash $r = \mathcal{H}(h_k, \mathfrak{m})$ of the hashed private key and message.
2. Calculate $R = rG$ and $s = (r + \mathcal{H}(R, K, \mathfrak{m}) \cdot k))$
3. the signature is the pair (R, s)

Verification

Verification is performed as follows

1. Compute $h = \mathcal{H}(R, K, \mathfrak{m})$
2. If the equality $(2^c s)G = 2^c R + 2^c \mathcal{H}(R, K, \mathfrak{m})K$ holds then the signature is valid

Why it works

Why the signature verification works can be derived from

$$2^c sG = 2^c((r + \mathcal{H}(R, K, \mathfrak{m}) \cdot k) \cdot G = 2^c R + 2^c \mathcal{H}(R, K, \mathfrak{m}) \cdot K)$$

Binary representation

By default, an EdDSA signature would need $64+32$ bytes to be represented. However, RFC8032 assumes that point R is compressed, whereby space requirements become reduced to $32 + 32$ bytes only.

CHAPTER 3

Ring signatures

Ring signatures are signatures generated with a single private key and a set of unrelated public keys. The whole set of public keys, including the one corresponding to the private key at hand, is usually called a *ring*. Somebody verifying the signature would not be able to tell which private key from the ring was used to produce the signature.

Ring signatures were originally called *Group Signatures* in that they were thought of as a way of proving that a signer belongs to a group, without necessarily identifying the individual at hand. In the context of Monero transactions, they will help making currency flows untraceable.

Ring signature schemes can display a number of properties that will be useful for producing confidential transactions:

Anonymity An observer should not be able to determine the identity of the true signer of the message. Only that the private key used corresponds to one of the public keys in the ring.

Linkability If a private key is used to sign two different messages, then the messages will become linked and the duplicity will be uncovered. In the case of Monero, this property will help preventing double-spending attacks.

Exculpability The linkability property does not apply to non-signing public keys. That is, a ring member whose public key has been used twice in different signatures will not be linked.

3.1 Linkable Spontaneous Anonymous Group Signatures (LSAG)

Originally (for instance in [8]), group signature schemes required trusted group members to manage the collective signatures, who had the theoretical possibility of disclosing the original signer.

Relying on a single signature manager is not at all desirable, since it causes a dependency on a single group member, something that translates into a disclosure risk

A more interesting scheme was presented by Liu *et al.* in [13]. The authors detailed an algorithm to cater for *Linkable and Spontaneous group signatures*, not requiring the collaboration of any possible co-signers. In other words, the signer could select any set of involuntary co-signers to anonymize his own signature.

Signature

Let $\mathfrak{m}$ be the message to sign, $\mathcal{R} = \{K_1, K_2, ..., K_n\}$ a set of distinct public keys (a group/ring), k_π the private key corresponding to $K_\pi \in \mathcal{R}$. Assume also the existence of two hash functions, $\mathcal{H}_n$ and $\mathcal{H}_p$, mapping to integers and curve points respectively[1].

1. Compute $\tilde{K} = k_\pi \mathcal{H}_p(\mathcal{R})$
2. Generate random numbers $\alpha \in_R \mathbb{Z}_q$ and $r_i \in_R \mathbb{Z}_q$ for $i \in \{0, 1, ..., n\}$ and $i \neq \pi$
3. Calculate
$$c_{\pi+1} = \mathcal{H}_n(\mathcal{R}, \tilde{K}, \mathfrak{m}, \alpha G, \alpha \tilde{K})$$
4. For $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$ calculate, replacing $n + 1 \to 1$
$$c_{i+1} = \mathcal{H}_n(\mathcal{R}, \tilde{K}, \mathfrak{m}, r_i G + c_i K_i, r_i \mathcal{H}_p(R) + c_i \tilde{K})$$
5. define $r_\pi = \alpha - k_\pi c_\pi \pmod N$

The signature will be $\sigma(\mathfrak{m}) = (c_1, r_1, ..., r_n, \tilde{K})$

Verification

The verification of a signature is done in the following manner

1. For $i = 1, 2, ..., n$ compute, replacing $n + 1 \to 1$
$$\begin{aligned} z_i' &= r_i G + c_i K_i \\ z_i'' &= r_i \mathcal{H}_p(\mathcal{R}) + c_i \tilde{K} \\ c_{i+1}' &= \mathcal{H}_n(\mathcal{R}, \tilde{K}, \mathfrak{m}, z_i', z_i'') \end{aligned}$$
2. if $c_1' = c_1$ then the signature is valid

[1] A simple definition for $\mathcal{H}_p$ could be $\mathcal{H}_p(x) = \mathcal{H}_n(x)G$

Why it works

We can convince ourselves that the algorithm works by observing the following:

If $i \neq \pi$ then c'_{i+1} is defined as in the signature algorithm.

If $i = \pi$ then

$$z'_i = r_i G + c_i K_i = (\alpha - k_\pi c_\pi)G + c_\pi K_\pi = \alpha G$$
$$z''_i = r_i \mathcal{H}_p(\mathcal{R}) + c_i \tilde{K} = (\alpha - k_\pi c_\pi)\mathcal{H}_p(\mathcal{R}) + c_\pi k_\pi \mathcal{H}_p(\mathcal{R}) = \alpha \tilde{K}$$

So even in this case the expression $c'_{i+1} = \mathcal{H}_n(\mathcal{R}, \tilde{K}, \mathfrak{m}, {z_n}', {z_n}'')$ will equal c_{i+1}

Linkability

Given a fixed set of public keys $\mathcal{R}$, and two valid signatures for different messages,

$$\sigma = (c_1, s_1, ..., s_n, \tilde{K})$$
$$\sigma' = (c'_1, s'_1, ..., s'_n, \tilde{K}')$$

if $\tilde{K} = \tilde{K}'$ then clearly both signatures come from the same signing ring and private key

In other words, the LSAG signature scheme yields mutually **linkable** signatures in the case a ring and a private key would be re-used.

Exculpability

At the same time, given that $\tilde{K} = k_\pi \mathcal{H}_p(\mathcal{R})$, we can readily see that linkability would only apply if private key k_π were re-used. Hence, no other group/ring member could be accused of signing twice.

3.2 Back Linkable Spontaneous Anonymous Group Signatures (bLSAG)

In the LSAG signature scheme, linkability of signatures can only be guaranteed if the ring is constant. This can be seen by looking at the definition of $\tilde{K}$.

In this section we present an enhanced version of the LSAG algorithm that provides linkability of the private key used, allowing the ring to contain different spurious keys.

The modification was unraveled in [16]. It is based on a publication by A. Back regarding the CryptoNote ring signature algorithm, see [2] for details.

Signature

1. Calculate $\tilde{K} = k_\pi \mathcal{H}_p(K_\pi)$
2. Generate random numbers $\alpha \in_R \mathbb{Z}_q$ and $r_i \in_R \mathbb{Z}_q$ for $i \in \{0, 1, ..., n\}$ and $i \neq \pi$
3. Compute
$$c_{\pi+1} = \mathcal{H}_n(\mathfrak{m}, \alpha G, \alpha \mathcal{H}_p(K_\pi))$$
4. For $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$ calculate, replacing $n + 1 \to 1$
$$c_{i+1} = \mathcal{H}_n(\mathfrak{m}, r_i G + c_i K_i, r_i \mathcal{H}_p(K_i) + c_i \tilde{K})$$
5. Define $r_\pi = \alpha - k_\pi c_\pi \pmod N$

The signature will be $\sigma(\mathfrak{m}) = (c_1, r_1, ..., r_n, \tilde{K})$.

As in the original LSAG scheme, verification takes place by recalculating the value c_1.

Correctness of the approach can be verified in a similar way.

The alert reader will no doubt notice that the value $\tilde{K}$ depends only on the keys of the true signer. In other words, two signatures will now be linkable if and only if the same private key was used for create the signature, independently of the ring used to anonymize the signer.

This notion of linkability will prove to be more useful for Monero than the one offered by the LSAG algorithm, as it will allow detecting double-spending attempts without putting constraints on the ring members used.

3.3 Multilayer Linkable Spontaneous Anonymous Group Signatures (MLSAG)

In order to be able to sign multi-input transactions, one has to be able to sign with m private keys. In [16, 17], Noether S. et al. describe a multi-layered generalization of the bLSAG signature scheme applicable when we have a set of $n \cdot m$ keys, that is, the set

$$\mathcal{R} = \{K_{i,j}\} \quad \text{for} \quad i \in \{1, 2, ..., n\} \quad \text{and} \quad j \in \{1, 2, ..., m\}$$

for which we know the private keys $\{k_{\pi,j}\}$ corresponding to the subset $\{K_{\pi,j}\}$ for some index π.

Such an algorithm would indeed address our multi-input needs, provided we generalize the notion of linkability.

Linkability: if any of private keys $k_{\pi,j}$ would be used in 2 different signatures, then these signatures would be automatically linked.

Signature

The *MLSAG* algorithm would be described by the following steps:

1. Calculate $\tilde{K_j} = k_{\pi_i}\mathcal{H}_p(K_{\pi,j})$ for all $j \in \{1, 2, ..., m\}$
2. Generate random numbers $\alpha_j \in_R \mathbb{Z}_q$, and $r_{i,j} \in_R \mathbb{Z}_q$ (excluding all $r_{\pi,j}$) for $j \in \{1, 2, ..., m\}$
3. Compute
$$c_{\pi+1} = \mathcal{H}_n(\mathfrak{m}, \alpha_1 G, \alpha_1 \mathcal{H}_p(K_{\pi,1}), ..., \alpha_n G, \alpha_n \mathcal{H}_p(K_{\pi,m}))$$
4. For $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$ calculate, replacing $n + 1 \to 1$
$$c_{i+1} = \mathcal{H}_n(\mathfrak{m}, r_{i,1}G + c_i K_{i,1}, r_{i,1}\mathcal{H}_p(K_{i,1}) + c_i\tilde{K_1}, ..., r_{i,m}G + c_i K_{i,m}, r_{i,m}\mathcal{H}_p(K_{i,m}) + c_i\tilde{K_m})$$
5. Define $r_{\pi,j} = \alpha_j - k_{\pi,j}c_\pi \pmod N$

The signature will be $\sigma(\mathfrak{m}) = (c_1, r_{1,1}, ..., r_{n,1}, ..., r_{1,m}, ..., r_{n,m}, \tilde{K_1}, ..., \tilde{K_m})$.

Verification

The verification of a signature is done in the following manner

1. For $i = 1, ..., n$ compute, replacing $n + 1 \to 1$
$$c'_{i+1} = \mathcal{H}_n(\mathfrak{m}, r_{i,1}G + c_i K_{i,1}, r_{i,1}\mathcal{H}_p(K_{i,1}) + c_i\tilde{K_1}, ..., r_{i,m}G + c_i K_{i,m}, r_{i,m}\mathcal{H}_p(K_{i,m}) + c_i\tilde{K_m})$$
2. if $c'_1 = c_1$ then the signature is valid

Why it works

Just as with the original LSAG algorithm, we can readily observe that

if $i \neq \pi$ then clearly the values c'_{i+1} are calculated as described in the signature algorithm

if $i = \pi$ then, since $r_{\pi,j} = \alpha_j - k_{\pi,j}c_\pi$

$$r_{\pi,j}G + c_\pi K_{\pi,j} = (\alpha_j - k_{\pi,j}c_\pi)G + c_\pi K_{\pi,j} = \alpha_j G$$
$$r_{\pi,j}\mathcal{H}_p(K_{\pi,j}) + c_\pi\tilde{K_j} = (\alpha_j - k_{\pi,j}c_\pi)\mathcal{H}_p(K_{\pi,j}) + c_\pi\tilde{K_j} = \alpha_j\mathcal{H}_p(K_{\pi,j})$$

In other words, it holds also that $c'_{\pi+1} = c_{\pi+1}$

Linkability

In case a private key $k_{\pi,j}$ would be re-used, then the corresponding value $\tilde{K_j}$ supplied in the signature would reveal it. This observation matches our generalized definition of linkability.

Space requirements

Assuming point compression, an MLSAG signature would clearly consume a total of

$$(1 + nm + m) \cdot 32 \quad \text{bytes}$$

3.4 Borromean Ring Signatures

We will see in later sections of this report that it will be necessary to prove that transaction amounts are within expected ranges. This can be acomplished with ring signatures. However, to this particular end it is not necessary that signatures be linkable, which allows us to select more efficient algorithms in terms of space consumed.

In this context, and for the specific purpose of proving amount ranges, Monero uses a signature scheme developed by G. Maxwell, which he described in [14]. We present here a simplified version of the scheme, in that we will assume that we have the same number of keys for any value of the first index i.

In our case, range proofs will require exactly 2 keys for each digit, so this simplification will not have any negative impact.

Assume that we have a set of public keys $\{K_{i,j}\}$ for $i \in \{1, 2, ..., n\}$ and $j \in \{1, 2, ..., m\}$.

Furthermore, we assume also that for each i there is an index π_i such that signer knows the private key k_{i,π_i} corresponding to K_{i,π_i}.

In what follows we will use $\mathfrak{m}$ for the hash of the message concatenated with keys $\{K_{i,j}\}$.

Signature

1. For each $i = 1, ..., n$:

   (a) generate a random value $\alpha_i \in_R \mathbb{Z}_q$

   (b) set $c_{i,\pi_i} = \mathcal{H}_n(\mathfrak{m}, \alpha_i G, i, \pi_i)$

   (c) for $j = \pi_i + 1, ..., m - 1$ generate random numbers $r_{i,j} \in_R \mathbb{Z}_q$ and compute,

   $$c_{i,j+1} = \mathcal{H}_n(\mathfrak{m}, r_{i,j}G - c_{i,j}K_{i,j}, i, j)$$

2. For $i = 1, ..., n$ generate random numbers $r_{i,m} \in_R \mathbb{Z}_q$ and compute

   $$c_1 = \mathcal{H}_n(r_{1,m}G - c_{1,m}K_{1,m}, ..., r_{n,m}G - c_{n,m}K_{i,m})$$

3. For $i = 1, .., n$:

(a) for $j = 1, ..., \pi_i - 1$ generate random numbers $r_{i,j} \in_R \mathbb{Z}_q$ and compute

$$c_{i,j+1} = \mathcal{H}_n(\mathfrak{m}, r_{i,j}G - c_{i,j}K_{i,j}, i, j)$$

Here we interpret references to $c_{i,1}$ as c_1, see previous step

(b) set $r_{i,\pi_i} = \alpha_i + k_{i,\pi_i}c_{i,\pi_i}$

The signature will be

$$\sigma = (c_1, r_{1,1}, r_{1,2}, ..., r_{1,m}, ..., r_{n,m})$$

Verification

As before, let $\mathfrak{m}$ be the hash of the message to sign and the set of signing keys.

The verification of a given signature is performed as follows:

1. For $i = 1, ..., n$ and $j = 1, ..., m$ compute:

$$\begin{aligned} R'_{i,j+1} &= r_{i,j}G - c'_{i,j}K_{i,j} \\ c'_{i,j+1} &= \mathcal{H}_n(\mathfrak{m}, R'_{i,j+1}, i, j) \end{aligned}$$

   Interpret any $c'_{i,1}$ as c_1

2. Compute $c'_1 = \mathcal{H}(R'_{1,m}, ..., R'_{n,m})$

The signature will be valid if $c'_1 = c_1$.

Why it works

1. For $j \neq \pi_i$ and for all i we can readily see that $c'_{i,j+1} = c_{i,j+1}$
2. When $j = \pi_i$, for all i

$$\begin{aligned} R'_{i,j+1} &= r_{i,j}G - c'_{i,j}K_{i,j} \\ &= (\alpha_i + k_{i,\pi_i}c'_{i,\pi_i})G - c'_{i,\pi_i}K_{i,\pi_i} \\ &= \alpha_i G + k_{i,\pi_i}c'_{i,\pi_i}G - c'_{i,\pi_i}k_{i,\pi_i}G \\ &= \alpha_i G \end{aligned}$$

   In other words, $c'_{i,\pi_i} = \mathcal{H}_n(\mathfrak{m}, \alpha_i G, i, \pi_i) = c_{i,\pi+1}$.

Therefore we can conclude that the verification step identifies correctly valid signatures.

CHAPTER 4

Pedersen commitments

Generally speaking, a cryptographic *commitment scheme* is a way of publishing a commitment to a value without revealing the value itself.

As an example, in a flip-coin game, Alice could commit to one outcome before Bob flips the coin, by publishing the value hashed with secret data. After flipping the coin, Alice could prove which value she committed to by publishing her secret data, so that Bob could verify that she did indeed hash the outcome she later declared.

In other words, assume that Alice has a secret string b and that the value she wants to commit to is v. She could simply hash $\mathcal{H}(b||v)$ and tell the result to Bob. Bob flips the coin and then Alice could prove that she guessed the right outcome v by telling Bob what the secret string b was. Bob would then recalculate $\mathcal{H}(b||v)$ and verify that Alice did indeed guess right.

4.1 Pedersen commitments

A *Pedersen commitment* [19] is a commitment that has the property of being *additive*. In other words, if $C(a)$ and $C(b)$ denote the commitments for amounts a and b respectively, then $C(a+b) = C(a) + C(b)$ is a commitment for $a+b$. This property is useful when committing transaction amounts, as one could prove, for instance, that inputs equal outputs, without unveiling the amounts at hand.

Fortunately, Pedersen commitments are easy to implement with elliptic curve cryptography, as the following holds trivially

$$aG + bG = (a+b)G$$

.

Clearly, with a simple definition of commitment like $C(a) = aG$, we would recognize immediately commitments to 0. We could also end up creating cheat tables of commitments to help us recognize common amounts.

To attain information-theoretical privacy, one needs to add a secret *blinding factor* and another generator H, such that it is unknown for which value of γ the following holds $H = \gamma G$. The hardness of the discrete logarithm problem ensures the unfeasibility of calculating this value.

We can then define the commitment of an amount a as $C(x, a) = xG + aH$, where x is a blinding factor.

In the case of Monero, $H = to\_point(SHA3(G))$, where *SHA3* stands for the novel *Keccak* hashing algorithm, and *to_point* is a function mapping scalars to curve points.

4.2 Monero commitments

A transaction in a cryptocurrency is a collection of inputs and outputs, which must balance. That is, if Alice spends 100 units of the currency (the inputs), then the receiver(s) should receive exactly 100 units in total. In case the payment to Bob is of 50 units only, then a second output of 50 units back to Alice should be added.

In other words, if we had a transaction with inputs $a_1, ..., a_n$ and outputs $b_1, ..., b_m$, then an observer would justifiably expect that:

$$\sum_i a_i - \sum_j b_j = 0$$

Since commitments are additive, then their sum would also equal zero:

$$\sum_i C_{i,in} - \sum_j C_{j,out} = 0$$

This fact would be used by the network to verify that the sender of the transaction is not spending more money than he has previously received.

However, to avoid identifiability of a sender, Shen Noether proposes in [16] verifying instead that the commitments sum to certain non-zero value:

$$\sum_i C_{i,in} - \sum_j C_{j,out} = zG$$
$$\sum_i (x_i G + a_i H) - \sum_j (y_j G + b_j H) = zG$$
$$\sum_i x_i - \sum_j y_j = z$$

The reasons why this is useful will become clear later, when we discuss the structure of transactions.

Nevertheless, while not really summing up to 0, we will still refer to valid transaction commitments as *Commitments to zero*, as long as the private key z is known to the committer.

4.3 Range proofs

One problem with additive commitments is that, if we have commitments for a, b and z and we intend to use them to prove that $a + b = z$, then those commitments would also apply if we replace each value in the equation by its additive inverse $\pmod q$.

For instance, if our base field were $\mathbb{Z}_7$ and $a = 3$, $b = 2$, $z = 5$, then the equation would also hold for $a = 4$, $b = 5$ and $z = 2 \pmod 7$.

Therefore, any commitments we could create for the amounts at hand would also apply to the inverses of the amounts. So we could effectively claim later that the amounts were different, whereby we would be creating money!

The solution used in Monero to address this issue is to sign the ranges of the numbers at hand using a Borromean signature scheme described in Section 3.4, in the manner described here.

For any amount a, use its binary representation $(a_0, a_1, ..., a_k)$ such that

$$a = a_0 2^0 + a_1 2^1 + ... + a_k 2^k$$

.

Generate random numbers $x_1, ..., x_k \in_R \mathbb{Z}$ to be used as blinding factors. Define also Pedersen commitments for each a_i, $C_i = x_i G + a_i 2^i H$, and derive public keys $\{C_i, C_i - 2^i H\}$.

Clearly one of those public keys will equal $x_i G$:

$$\text{if } a_i = 0 \text{ then} \quad C_i = x_i G + 0H = x_i G$$
$$\text{if } a_i = 1 \text{ then} \quad C_i - 2^i H = x_i G + 2^i H - 2^i H = x_i G$$

In other words, a blinding factor x_i will always be the private key corresponding to one of $\{C_i, C_i - 2^i H\}$. Therefore we will be able to sign an amount a in a transaction using the Borromean Ring Signature scheme of Section 3.4 with the ring:

$$\{\{C_0, C_0 - 2^0 H\}, ..., \{C_k, C_k - 2^k H\}\}$$

4.4 Range proofs in a blockchain

In the context of Monero we will use range proofs to commit to individual bit components and to prove that their sum equals the total amount committed. Therefore, it will not be necessary for the receiver nor any other party to know the blinding factors $x_i G$. In other words, it is sufficient to know that

$$\sum_{i=0}^{k} C_i = C$$

In the blockchain we will store only the commitments/keys C_i. The mining community will have to check that the equation above holds and that the private key of either C_i or $C_i - 2^i H$ has been used to sign the amount.

The Borromean signature scheme requires knowledge of x_i to produce a signature. In consequence, upon verifying this relationship between keys, any third party will be able to convince herself that amounts fall within ranges and that money is not being artificially *created*.

CHAPTER 5

Monero Transactions

5.1 User keys

Unlike Bitcoin, Monero users have two sets of private/public keys. (k_1, K_1) and (k_2, K_2), generated as described in Section 2.2.2.

The address of a user is the pair of public keys (K_1, K_2). Her private keys will be the corresponding pair (k_1, k_2).

Using two sets of keys allows segregation of functions. The rationale will become clear later in this report, but for the moment being let us advance that private key k_1 will be called *view key* whereas k_2 will be the *spend key*. The former key will be used to determine whether an output is addressed to the user at hand, and the second one will be necessary to use the output in a spend transaction.

5.2 One-time addresses

As described, every Monero user has a pair of public keys as public address. However, this address is never used directly. Instead, new addresses based on a Diffie-Hellman-like exchange are created every time an amount is to be paid to a user. In this way, external observers will not be able to identify receivers in transactions.

Imagine a very simple transaction, containing exactly one input and one output — a payment from Alice to Bob.

Bob has private/public keys (k_{B_1}, k_{B_2}) and (K_{B_1}, K_{B_2}). To create one-time keys, Alice would proceed as depicted here (see [21]):

1. Alice generates a random number r such that $1 < r < N$, and calculates the output public key $K_o = \mathcal{H}_n(rK_{B_1})G + K_{B_2}$
2. Alice sets K_o as the addressee of the payment, adds the value rG to the transaction data and submits it to the network. The value rG will be used by the receiver to calculate a Diffie-Hellman-like shared secret.
3. Bob receives the data and sees the value rG. Hence, he can calculate $k_{B_1}rG = rK_{B_1}$. Therefore, he will also be able to calculate $K_o = \mathcal{H}_n(rK_{B_1})G + K_{B_2}$. When he sees the addressee of the output he will know that it is addressed to him.
4. The one-time keys for the output are

$$K_o = \mathcal{H}_n(rK_{B_1})G + k_{B_2}G = (\mathcal{H}_n(rK_{B_1}) + k_{B_2})G$$
$$k_o = \mathcal{H}_n(rK_{B_1}) + k_{B_2}$$

While Alice can calculate the public key for the address, she can not compute the corresponding private key, since it would require either knowing Bob's second private key, or solving the discrete logarithm problem for $K_{B_2} = k_{B_2}G$, which we assume to be hard.

As mentioned earlier, the private key k_{B_1} is often called the *view key*. The reason is that it allows a third party to verify if an output is addressed to Bob, and yet, without the knowledge of the other private key, k_{B_2}, this third party would not be able to spend the amount, as he would not be able to sign with the private key of the one-time address.

Such a third party could be a trusted custodian, an auditor, a tax authority, etc. Somebody who would have read access to the user's transaction history, without any further rights. This third party would also be able to decrypt the amounts of Section 5.4.1.

The private key k_o can only be calculated with knowledge of k_{B_2}. As we will see, spending an output will require calculating k_o, which in turn entails knowing the *spend key* k_{B_2}.

5.2.1 Multi-output transactions

Most transactions will contain more than one output. If nothing else, to transfer back any change to the sender himself.

Monero senders generate only one random value r. The value rG is normally known as the *Transaction public key* and is published in the blockchain.

To ensure that all output addresses in a transaction are different even in cases where the same addressee is used twice, Monero uses the output index. Every output will have an index $l \in$

$\{1, ..., s\}$. By appending this value to the shared secret before hashing it, one can ensure that the resulting addresses will be unique:

$$K_o = \mathcal{H}_n(rK_{B_1}, l)G + k_{B_2}G = (\mathcal{H}_n(rK_{B_1}, l) + k_{B_2})G$$
$$k_o = \mathcal{H}_n(rK_{B_1}, l) + k_{B_2}$$

5.3 Transaction types

Monero is a cryptocurrency under steady development. Transaction structures, protocols and cryptographic schemes are always prone to evolve to meet new objectives, or to avoid newly discovered threats.

In this report we have focused our attention on *Ring Confidential Transactions* as they are implemented in the current version of Monero. Therefore we will not describe here any other transaction types, even if they are still partially supported,

The transaction types we will describe in this section are `RCTTypeFull` and `RCTTypeSimple`. The former category follows closely the ideas exposed by S.Noether at al. in [17]. At the time the paper was written, the intention was most likely to replace fully the original CryptoNote transaction scheme by the new scheme.

However, for multi-input transactions and with the formulation used in the paper mentioned, the signature scheme used was thought to entail a risk on traceability. This will become clear when we supply technical details, but as it stands let us advance that if one spent output would become identifiable, then the rest of the spent outputs would also become identifiable. This would have an impact on the traceability of currency flows, not only for the transaction originator affected, but also for the rest of the blockchain.

To mitigate this risk, the Monero Research Lab decided to use a related, yet different signature scheme for multi-input transactions. The transaction type `RCTTypeSimple` is the one used in these occasions. The main difference, as we will see later, is that each input will be signed independently.

5.4 Ring Confidential Transactions of type `RCTTypeFull`

By default, the current code base applies this type of signature scheme when transactions have only one input. The scheme itself allows multi-input transactions, but when it was introduced, the Monero Research Lab decided that it would be advisable to use it only on single-input transactions. For multi-input transactions, existing Monero wallets use the `RCTTypeSimple` scheme described later.

Our perception is that the decision of doing so was rather hastily taken, and that it might change in the future, perhaps if the algorithm to select additional mix-in outputs is improved and the ring sizes are increased. Also, S. Noether's original description in [17] did not envision constraints of this type. At any rate, this is not a hard constraint. An alternative wallet might indeed choose to sign transactions using either scheme, independently of the number of inputs involved.

We have therefore chosen to describe the scheme as if it were meant also for multi-input transactions.

An actual example of transactions of this type, with all its components, can be inspected in Appendix A.

5.4.1 Amount Commitments

Recall from Section 4.2 that we had defined a commitment to an amount b as:

$$C(b) = xG + bH$$

In the context of Monero, it is necessary that the receiver can verify that the amount in an output is the one promised. This means that the blinding factor xG must be somehow communicated to the receiver.

The solution adopted in Monero is to transmit this value to the receiver by using the Diffie-Hellman shared secret rK_{B_1}. For each transaction output in the blockchain there will be 2 values called *mask* and *amount* satisfying

$$\begin{aligned} mask &= x + \mathcal{H}_n(rK_{B_1}) \\ amount &= b + \mathcal{H}_n(\mathcal{H}_n(rK_{B_1})) \end{aligned}$$

The receiver will be able to calculate backwards the blinding factor x and the amount b. He will also be able to check that the commitment provided in the transaction corresponds to the amount at hand.

Furthermore, a third party with access to the view key mentioned earlier would also be able to decrypt the amount at hand.

5.4.2 Commitments to zero

Assume that the sender of the transaction has previously received amounts $a_1, ..., a_m$ from various outputs, addressed to one-time addresses $K_{\pi,1}, ..., K_{\pi,m}$ and with commitments $C^a_{\pi,1}, ..., C^a_{\pi,m}$.

This sender knows the private keys $k_{\pi,1}, ..., k_{\pi,m}$ corresponding to the one-time addresses. The sender knows also the blinding factors used in commitments $C^a_{\pi,i}$.

A transaction consists of inputs $a_1, ..., a_m$ and outputs $b_1, ..., b_s$ such that $\sum_j a_j - \sum_i b_i = 0$.

The sender re-uses the commitments from the previous outputs, $C^a_{\pi,1}, ..., C^a_{\pi,m}$, and creates commitments for $b_1, ..., b_s$. Let these commitments be $C^b_{\pi,1}, ..., C^b_{\pi,s}$

As hinted in Section 4.2, the sum of the commitments will not be truly 0, but a curve point zG:

$$\sum_j C^a_{\pi,j} - \sum_i C^b_{\pi,i} = zG$$

It is precisely the knowledge of value z by the sender what characterizes this equation as a *commitment to zero*. Indeed, it will be the case that

$$\begin{aligned}
&\sum_j C^a_{\pi,j} - \sum_i C^b_{\pi,i} \\
=&\sum_j x_j G - \sum_i y_i G + (\sum_j a_j - \sum_i b_i)H \\
=&\sum_j x_j G - \sum_i y_i G \\
=&zG
\end{aligned}$$

5.4.3 Signature

The sender selects q sets of size m, of additional unrelated addresses from the blockchain, corresponding to apparently unspent outputs. She mixes the addresses in a *ring*, adding false commitments to zero, as follows:

$$\begin{aligned}
\mathcal{R} = \{\{&K_{1,1}, ..., K_{1,m}, (\sum_j C_{1,j} - \sum_i C^b_{\pi,i})\}, \\
&... \\
&\{K_{\pi,1}, ..., K_{\pi,m}, (\sum_j C^a_{\pi,j} - \sum_i C^b_{\pi,i})\}, \\
&... \\
&\{K_{q+1,1}, ..., K_{q+1,m}, (\sum_j C_{q+1,j} - \sum_i C^b_{\pi,i})\}\}
\end{aligned}$$

Looking at the structure of the key ring, we see that if it were the case that

$$\sum_j C^a_{\pi,j} - \sum_i C^b_{\pi,i} = 0$$

then any observer would recognize the set of addresses

$$\{K_{\pi,1}, ..., K_{\pi,m}\}$$

as the ones in use as inputs, and therefore currency flows would be traceable.

With this observation made we can see why commitments to zero means in reality that they sum to a value zG.

Step 1: Signature of outputs The private keys for $\{K_{\pi,1}, ..., K_{\pi,m}, (\sum_j C_{\pi,j} - \sum_i C^b_{\pi,i})\}$ are $k_{\pi,1}, ..., k_{\pi,m}, z$, which are known to the sender. Hence, he will be able to apply the MLSAG signature scheme to sign the set of outputs/commitments $\{(K_{t,1}, C_{b,1}), ..., (K_{t,s}, C_{b,s})\}$ with the whole ring.

Step 2: Range proofs To avoid the amount ambiguity of outputs described in Section 4.3, the sender must also employ the Borromean signature scheme of Section 3.4 to sign ranges.

The corresponding inputs do not need to be signed explicitly, since the commitments stem from previously signed outputs.

In the current version of the Monero software, each amount is expressed as a fixed point number of 64 bits. Hence the ring will contain $2 \cdot 64$ keys.

5.4.4 Transaction fees

Transaction fees are stored in clear in the data transmitted to the network. Miners must be able to verify that zG includes a transaction fee, in order to add the corresponding additional output to themselves. In turn, this means that this amount must also be turned into a commitment.

The solution is to calculate the commitment of the fee f without the masking effect of any blinding factor. That is $C(f) = fH$.

To verify the correctness of zG, the network can therefore compute

$$(\sum_j C^a_{\pi,j} - \sum_i C^b_{\pi,i}) - fH = zG$$

.

5.4.5 Avoiding double-spending

An MLSAG signature contains images $\tilde{K}_{\pi,j}$ of private keys $k_{\pi,j}$. An important property in any cryptographic signature scheme is that it should not be forgeable with non-negligible probability. Therefore, to all practical effects, we can assume that the key images must have been deterministically produced from the private keys at hand.

The network needs only verify that these key images included in MLSAG signatures have not appeared before in other transactions. If they have, then we can be sure that we are witnessing an attempt to spend twice a previously received output $C^a_{\pi,j}$, addressed to $K_{\pi,j}$.

5.4.6 Space requirements

MLSAG signature

From Section 3.3 we recall that an MLSAG signature would be expressed as

$\sigma(\mathfrak{m}) = (c_1, r_{1,1}, ..., r_{q+1,1}, ..., r_{1,m+1}, ..., r_{q+1,m+1}, \tilde{K}_1, ..., \tilde{K}_m)$

As a result of the heritage from CryptoNote the values $\tilde{K}_j$ are not referred to as part of the signature, but rather as *images* of the private keys $k_{\pi,j}$ (and z). These values are normally stored separately in the transaction structure as they are used to detect double-spending attacks.

With this in mind and assuming point compression, an MLSAG signature will require $((q+1) \cdot (m+1) + 1) \cdot 32$ bytes of storage. In other words, a transaction with 1 input and a ring size of 32 would consume $(32 \cdot 2 + 1) \cdot 32 = 2080$ bytes.

To this value we would add 32 bytes to store the key image of an input, and additional space to store the ring member offsets in the blockchain. These offsets are stored as variable length integers, hence we can not quantify exactly the space needed.

The Monero Research Lab is currently developing an alternative signature algorithm to MLSAG. In its current version, space requirements for signatures with the new scheme are logarithmic, or in big-O notation, $\mathcal{O}(\log n)$. We are not able to provide references, as no information has yet been published concerning this new signature scheme.

Range proofs

From Section 3.4 and Section 4.3 we obtain that a Borromean signature takes the form of an n-tuple

$$\sigma = (c_1, r_{1,1}, r_{1,2}, r_{2,1}..., r_{64,2})$$

In the case of Borromean signatures, the ring keys are considered part of the signature. However, for verifiability it is only necessary to store the commitments C_j, as the ring key counterparts can be derived as $C_j - 2^j H$.

Respecting this convention, a range proof will require $(1 + 64 \cdot 2 + 64)32 = 6176$ bytes for each output.

5.5 Ring Confidential Transactions of type `RCTTypeSimple`

In the current Monero code base, transactions having more than one input are signed using a different scheme, referred to as `RCTTypeSimple`.

The main characteristic of this approach is that instead of signing the entire set of inputs as a whole, the sender signs each of the inputs individually.

Among other things, this has the consequence that one can not use commitments to zero in the same way as for `RCTTypeFull` transactions. The rationale is that a public key zG is a commitment to zero if and only if the sender knows the corresponding private key z. If the amounts alone do not sum to zero, then due to the hardness of determining γ such that $H = \gamma G$, it would not be possible to know z.

In more detail, assume that Alice wants to sign input j. Imagine for a moment that we could sign with an expression like the following

$$C_j^a - \sum_i C_i^b = x_j G - \sum_i y_i G + (a_j - \sum_i b_i) H$$

Since $a_j - \sum_i b_i \neq 0$, Alice would have to solve the DLP for $H = \gamma G$ in order to obtain the private key of the expression, something we have assumed to be a computationally difficult problem.

5.5.1 Amount Commitments

As explained, the sender is not able to sign against the outputs of the current transaction. On the other hand, the sender is spending previous outputs addressed to him, whose amounts are equal the current inputs. Therefore, the sender could create new commitments to the input amounts and commit to zero respect to each of the previous outputs being spent. In this way, the sender would be proving that the transaction spends exactly the outputs from previous transactions being used.

In other words, assume that the amounts being spent are $a_1, ..., a_m$. These amounts were outputs in previous transactions, in which they had commitments

$$C_j = x_j G + a_i H$$

The sender can create new commitments to the same amounts but using different blinding factors, that is

$$C'_j = x'_j G + a_i H$$

Clearly, she would know the private key of the difference between the two commitments:

$$C_j - C'_j = (x_j - x'_j) G$$

hence, she would be able to use this value as a *commitment to zero*.

Similarly to `RCTTypeFull` transactions, the sender can include the encoded blinding factor and amount in the transaction (see Section 5.4.1), which will allow the receiver to decode the corresponding values using the shared secret.

Before committing a transaction to the blockchain, the network will want to verify that the transaction balances. In the case of `RCTTypeFull` transactions, this was simple, as the signature scheme implied that the sender had signed with the private key of a commitment to zero.

For `RCTTypeSimple` transactions, the solution used by Monero is to select blinding factors for input and output commitments such that

$$\sum_i x_i - \sum_j y_j = 0$$

This will have the effect that

$$(\sum_j C_j^a - \sum_i C_i^b) - fH = 0$$

.

Fortunately, choosing such blinding factors is simple. In the current version of Monero, x_m will be simply set to

$$x_m = \sum_i y_i - \sum_{j=1}^{m-1} x_j$$

5.5.2 Signature

As we advanced earlier, in transactions of type `RCTTypeSimple` each input is signed individually. Furthermore, the signature scheme employed will be the same as for `RCTTypeFull` transactions, except that the signing keys will be different.

Assume that Alice is signing input j. This input spends a previous output with key $K_{\pi,j}$ that had commitment $C_{\pi,j}$. Let $C'_{\pi,j}$ be a new commitment for the same amount but with a different blinding factor.

Similarly to the previous scheme, the sender selects q unrelated outputs and their respective commitments from the blockchain, to mix with the real output

$$K_{1,j}, ..., K_{\pi-1,j}, K_{\pi+1,j}, ..., K_{q+1,j}$$
$$C_{1,j}, ..., C_{\pi-1,j}, C_{\pi+1,j}, ..., C_{q+1,j}$$

She can then sign using the following ring:

$$\begin{aligned}\mathcal{R}_j = \{&\{K_{1,j}, C_{1,j} - C'_{\pi,j}\},\\ &\ldots\\ &\{K_{\pi,j}, C_{\pi,j} - C'_{\pi,j}\},\\ &\ldots\\ &\{K_{q+1,j}, C_{q+1,j} - C'_{q+1,j}\}\}\end{aligned}$$

Indeed, Alice will know the private key for $K_{\pi,j}$ as well as the one for the commitment to zero $C_{\pi,j} - C'_{\pi,j}$. Therefore she will be able to sign the input with the ring at hand.

Each input in `RCTTypeSimple` transactions is signed individually, applying the scheme described in Section 5.4.3, but using rings like $\mathcal{R}_j$ as defined above.

The advantage of signing inputs individually is that the set of real inputs and commitments to zero are not placed at the same index, according to the formulation of the MLSAG algorithm. Therefore, even if the origin of one input became traceable, the rest of the inputs would not.

5.5.3 Space Requirements

MLSAG signature

Each ring $\mathcal{R}_j$ contains $(q+1)\cdot 2$ keys. From Section 3.3 we then derive that using point compression, an input signature will require $(2(q+1)+1)\cdot 32$ bytes.

A transaction with 20 inputs using rings with 32 members will need $((32\cdot 2+1)\cdot 32)20 = 41600$ bytes.

For the sake of comparison, if we were to apply the `RCTTypeFull` scheme to the same transaction, the MLSAG signature itself would require $(32\cdot 21+1)\cdot 32 = 21536$ bytes.

Range proofs

The size of range proofs is constant for each output. As we calculated for `RCTTypeFull` transactions, a single proof will require 6176 bytes of storage.

Bibliography

[1] How does monero's privacy work? https://www.monero.how/how-does-monero-privacy-work.

[2] Adam Back. Ring signature efficiency. BitcoinTalk, 2015. https://bitcointalk.org/index.php?topic=972541.msg10619684#msg10619684.

[3] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. *Twisted Edwards Curves*, pages 389–405. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[4] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, Sep 2012.

[5] Daniel J. Bernstein and Tanja Lange. *Faster Addition and Doubling on Elliptic Curves*, pages 29–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[6] Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. Cryptology ePrint Archive, Report 2007/286, 2007. http://eprint.iacr.org/2007/286.

[7] Karina Bjørnholdt. Dansk politi har knækket bitcoin-koden, May 2017. http://www.dansk-politi.dk/artikler/2017/maj/dansk-politi-har-knaekket-bitcoin-koden.

[8] David Chaum and Eugène Van Heyst. Group signatures. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'91, pages 257–265, Berlin, Heidelberg, 1991. Springer-Verlag.

[9] Thomas C Hales. The NSA back door to NIST. *Notices of the AMS*, 61(2):190–192.

[10] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[11] Simon Josefsson and Ilari Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, January 2017.

[12] Alexander Klimov. ECC patents?, October 2005. http://article.gmane.org/gmane.comp.encryption.general/7522.

[13] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. *Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups*, pages 325–335. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

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

[15] Arvind Narayanan and Malte Möser. Obfuscation in bitcoin: Techniques and politics. *CoRR*, abs/1706.05432, 2017.

[16] Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. http://eprint.iacr.org/2015/1098.

[17] Shen Noether, Adam Mackenzie, and the Monero Research Lab. Ring confidential transactions. *Ledger*, 1(0):1–18, 2016.

[18] Michael Padilla. Beating bitcoin bad guys, August 2016. http://www.sandia.gov/news/publications/labnews/articles/2016/19-08/bitcoin.html.

[19] Torben Pryds Pedersen. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, pages 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[20] QingChun ShenTu and Jianping Yu. Research on anonymization and de-anonymization in the bitcoin system. *CoRR*, abs/1510.07782, 2015.

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

Appendices

APPENDIX A

`RCTTypeFull` Transaction structure

We present in this chapter a dump from a real Monero transaction of type `RCTTypeFull`, together with explanatory notes for relevant fields.

The dump was obtained executing command `print_tx <TransactionID>` in the `monerod` daemon run in non-detached mode. The first line printed shows the actual command run, which the interested reader can use to replicate our results.

For editorial reasons we have shortened long hexadecimal chains, presenting only the beginning and end as in `0200010c7f[...]409`.

Component `rctsig_prunable`, as indicated by its name, is in theory *prunable* from the blockchain. That is, once a block has been consensuated and the current chain length rules out all possibilities of double-spending attacks, this whole field could be pruned. This is something that has not yet been done in the Monero blockchain, but it is nevertheless a possibility. This would yield considerable space savings.

Key images and ring keys are stored separately, in the non-prunable area of transactions. Indeed, these components are essential for detecting double-spend attacks and can not be pruned away.

Our sample transaction has 1 input and 2 outputs.

```
1 print_tx  b43a7ac21e1b60ad748ec905d6e03cf3165e5d8c9e1c61c263d328118c42eaa6
2 Found in blockchain at height 1467685
```

```
0200010c7f[...]409
{
   "version": 2,
   "unlock_time": 0,
   "vin": [ {
      "key": {
         "amount": 0,
         "key_offsets": [ 799048, 782511, 1197717, 216704, 841722
         ],
         "k_image": "595a612d0df27181c46a8af70a9bd682f2a000124b873ba5d2b9f4b4e4efd672"
      }
   }
   ],
   "vout": [ {
      "amount": 0,
      "target": {
         "key": "aa9595f55f2cfaed3bd2a67453bb064dc7fd454a09c2418d7338782790185fe3"
      }
   }, {
      "amount": 0,
      "target": {
         "key": "0ccb48ed2ebbcaa8e8831111029f3300069cff0d1408acffbfc3810b362ea217"
      }
   }
   ],
   "extra": [ 2, 33, 0, 129, 70, 77, 194, 248, 93, 24, 94, 15, 107, 233, 0, 229, 82,
   175, 243, 123, 58, 204, 135, 171, 100, 101, 192, 42, 187, 157, 168, 222, 98, 192,
   110, 1, 1, 185, 87, 22, 38, 116, 81, 124, 85, 68, 36, 44, 229, 235, 46, 159, 139,
   114, 234, 211, 50, 41, 28, 92, 26, 249, 184, 228, 197, 64, 139, 5
   ],
   "rct_signatures": {
      "type": 1,
      "txnFee": 26000000000,
      "ecdhInfo": [ {
         "mask": "68f508c5515694ce5a33b316b990e8b67a944725c93d806767e61b2e0b13d300",
         "amount": "913372a2424b22bd9712183f5a7c8027c8d9af89b52d1e7d06fd1f87a1e5d20d"
      }, {
         "mask": "fbc3e5bdb36fc58e5800ffc549ab7bd533fadb7e6b64898c82ea620d749fc80e",
         "amount": "b9335c3dc0afb774f812f9f58a412c849f3c828d873f1c16ab102963799d9809"
      }],
      "outPk": [ "cf141f5dfe04df14afad6b451d600aa5826a9be44a76a1630850c1d5951d482e",
                 "e10bb69b66af5dabec765c7f5f7528926088877fa36746833828a0575896ae57"]
   },
   "rctsig_prunable": {
```

```
47        "rangeSigs": [ {
48           "asig": "b9b544a7[...]d4c5726e81c4c4b6205dacc05208",
49           "Ci": "bc7ae457[...]fe490458"
50        }, {
51           "asig": "9c457b41[...]545b60c",
52           "Ci": "ce9b4d8e[...]03a6752"
53        }],
54        "MGs": [ {
55           "ss": [[ "a8120b96f5f2ac5bceab37f7d6bf8d86554d87c4af3441007cad92f54a24d908",
56            "2e6bc016297a5d398936c9f45e7a80215138f69e55179b337922e2d51c1a9f00"],
57            ["1e1052a68c38bb88b6e8f257d999c13f1d5f4fa219cc23479ccbfa6b14b5960a",
58             "e914d35eed0d27344fbc3a89b91bd445d433b561efc844c9f466a61ebb5f6d09"],
59            ["e04d011f515461fdbd8d13536c23143dc365d87dd323defb1af834e540a8fc0e",
60             "f9b41a117a1415fec54f1cc16aeef859b2cab1494b9e26a95fc9eaf4f571fa00"],
61            ["de7a7b30795cab310b632f708c6c2546847a5cbcc27ff48e75c1556c3f6f180c",
62             "6218695558359d115e308b008d9aa368c38672732d2fc21c6317ad7d15918c05"],
63            ["0ca70bbdea0e391b1e24e2540f33b48dd9dc554c61ebf23bb3691aab5094e40f",
64             "dafecd436b2448504c0a3a1997b356c141f1d4b5977cc66e5f55592f13731501"]],
65           "cc": "5059757cf06216215955aaa108e8dd40be157856749a9d883bcac611e395a409"
66        }]
67     }
68  }
69
```


Transaction components

- `vin` (line 7-14) - List of inputs
- `amount` (line 9) - Deprecated amount field for type 1 transactions
- `key_offset` (line 10) - Relative offsets respect to previous block of ring components, spent output and mixin outputs. As an illustration, 799048 is to be interpreted as the 799048^{th} transaction counting backwards from the current transaction and starting from the previous block
- `k_image` (line 12) - Key image $\tilde{K_j}$ from Section 3.3
- `vout` (lines 16-27) - List of outputs
- `amount` (line 17) - Deprecated amount field for type 1 transactions
- `key` (line 19) - One-time destination key as described in Section 5.2
- `extra` (lines 28-32) - Miscellaneous data, including the Transaction key, or share secret rG of Section 5.2
- `rct_signatures` (lines 33-45) - First part of signature data

- `type` (line 34) - Signature type, in this case `RCTTypeFull`
- `txnFee` (line 35) - Transaction fee in clear, in this case 0.026 XMR
- `ecdhInfo` (lines 36-42) - Encrypted mask and amount of each of the outputs
- `mask` (line 37) - Field *mask* as described in Section 5.4.1
- `amount` (line 38) - Field *amount* as described in Section 5.4.1
- `outPk` (lines 43-44) - Output commitments
- `rctsig_prunable` (lines 46-67) - Second part of signatures
- `rangeSigs` (lines 47-53) - Range proofs for output commitments
- `asig` (line 48) - Borromean signature of the amount, see Section 5.4.6
- `Ci` (line 49) - Borromean commitments (ring keys), as described in Section 5.4.3. As hinted in Section 5.4.6 only the K_j need to be stored, as the values $K_j - 2^j H$ can be easily derived by the network.
- `MGs` (lines 54-66) - Remaining elements of the MLSAG signature
- `ss` (lines 55-64) - Components $r_{i,j}$ from the MLSAG signature

$$\sigma(\mathfrak{m}) = (c_1, r_{1,1}, ..., r_{n,1}, ..., r_{1,m}, ..., r_{n,m}, \tilde{K}_1, ..., \tilde{K}_m)$$

- `cc` (line 65) - Component c_1 from aforementioned MLSAG signature

APPENDIX B

RCTTypeSimple Transaction structure

In this section we show the structure of a sample transaction of type `RCTTypeSimple`. The transaction has 4 inputs and 2 outputs.

```
print_tx 3ebf45fc5f8fd6830378073841228 17d5debfa762c7a7845cb7ccfe9ee20940b
Found in blockchain at height 1469563
020004[...]923b3d70d
{
  "version": 2,
  "unlock_time": 0,
  "vin": [ {
      "key": {
        "amount": 0,
        "key_offsets": [ 1567249, 1991110, 349235, 15551, 3620
        ],
        "k_image": "9661119b4b54529e1be14ef97fbdc0504d17a6c8dfedd55d2455b93a6336bb41"
      }
    }, {
      "key": {
        "amount": 0,
        "key_offsets": [ 2502375, 650851, 337433, 396459, 39529
        ],
        "k_image": "2102414d8edfa229f9ebf32ab90acd9cf23963a8c3b6ba0e181fc1d5782c046c"
      }
    }, {
      "key": {
```

```
        "amount": 0,
        "key_offsets": [ 1907097, 696508, 806254, 510195, 6709
        ],
        "k_image": "de14ec8958b311bd38a05aa3fb08fdd360001f1b9c060264eecdd8c08c9e83c4"
      }
    }, {
      "key": {
        "amount": 0,
        "key_offsets": [ 1150236, 1943388, 788506, 37175, 7462
        ],
        "k_image": "e470f77dd5a4149210cb61ee107e73caea1ef9f61d05384e3bd4372fdc85bf17"
      }
    }
  ],
  "vout": [ {
      "amount": 0,
      "target": {
        "key": "787cad1ebb181e1fc04b24d4d06c3d2882c38b262a7635de8ad487c536e40a12"
      }
    }, {
      "amount": 0,
      "target": {
        "key": "faf4137928392b39ccf0a830c0261573009959787697f9d4fb769c25781fb911"
      }
    }
  ],
  "extra": [ 1, 20, 56, 120, 111, 89, 89, 64, 10, 98, 96, 255, 202, 235, 203,
             255, 2, 197, 176, 147, 61, 60, 41, 145, 207, 178, 212, 71, 37, 69, 19,
             147, 205],
  "rct_signatures": {
    "type": 2,
    "txnFee": 558805800000,
    "pseudoOuts": [ "64dea29ac5560f93773240d58ca5768b879fd3c95e0b3b50a80ec36a6ff3a6da",
                    "a60e7a00e65ff2a6299b92b166a629e9b0d62f6df50e40535140716757efe4c0",
                    "4c67403adbc9dc0ca5a1a6abc846ab6d232dc3fa295099b3c7a9d005bac60eba",
                    "635b26d78117d77899859ecb61e10125c3956a5c113b932f33c92c561acddaa3"],
    "ecdhInfo": [ {
        "mask": "ccffac42a86bec7b36ce9957cbdfe481d419bc5353335d0c236c347aea758d0c",
        "amount": "c0d6cf3e1db55dd459b73faf34d7339c3fa1b3d3356cfb2adc3faf798264b00e"
      }, {
        "mask": "62cc846003d9c5425c6cf33b30754a4c044f5d9d02621460e45664b886673109",
        "amount": "726dbacad62022bf0f5af05c72482b3f040d631d3f576b5e2615ea72f84c5f06"
      }],
    "outPk": [ "cb3b729b4fca6e66736666201633e3f905c367a2f3d18e31fe3d3c18d2be93fd",
```

```
                "1e8c86b7f211a99e1762bf62254efe65ea5c5328b62b0ea8d679b2e52800f633"]
  },
  "rctsig_prunable": {
    "rangeSigs": [ {
        "asig": "9bb7cce09[...]61de7ce0a",
        "Ci": "50dfd2e8[...]b3a7c8fca1b1"
      }, {
        "asig": "e3905fa5c[...]b5213444908",
        "Ci": "595c2cec5f2[...]72a628ab5c"
      }],
    "MGs": [ {
        "ss": [ [ "3b2d26ea7628015fd8317e4e298ceda6b534ac894b83f7b6190a353cee6ec702",
                  "2d772ad7b7ff2ba8a1a66c8d69c0a0d49d72808eaf803c59f13c3d78b653440c"],
                [ "c188891fb37d76305f0209222f52d22ede43018facfe91f949ecb8dcf709b30a",
                  "303896ca67ea7969544641d5bb94a436558bcf6522bb9bc77bd1abb5f2146c08"],
                [ "e01c88b7308403a9dd023d9eacf1ade17ab0fa54250148431b5a33c98e636100",
                  "ba36e34e5245e89c7c21af845b949cf3e82188df639390f094e31c9ba773060c"],
                [ "37175d72d2bef3f8bd9e65fb2861f7bce91e3b1e30278b2dcf26112831ac9405",
                  "8e77a5dd641a89e86dbe0708e8f59d0e2dc9fe4ddfd9b367c3a93522198a4706"],
                [ "fbb4f94f9ce0f081421e63677a63d5914f0536a481d57b6e5fc5379c84dfcb05",
                  "1ec40aa3c8a94c6b1915b7796423b0d7d6011aa2d6af636aff309b832f193408"]],
        "cc": "a03119e4257cca37f89ac3e97f0598b712c79162c73932d58ab4ce08c4ad6709"
      }, {
        "ss": [ [ "f7aedeec462d7588330c71589fde5f0f234a627a6e5ed72cff34825a04d41707",
                  "31d7a5ba4e782db5c0704ab751a2ef8c4732f3cf699bc8f9994e79a97cd3190e"],
                [ "00e1d1ecdf31fdf7d57661f2234bfc859cfdc4dbfdfd0f5eec0576ef22592203",
                  "fdf08b803fa6de18bf0e0dc6855e877877bda0101eceb81e2223fe0175606300"],
                [ "3397ba3f9e8db066e3c4911b896debefbc73efbac4988e6aff5731ff8db15405",
                  "43d2b03d5263de99f56c256e646be503edd63dd03d377a469379fbf487e8600e"],
                [ "31afd1d5c3b07170ac127605fc35cbcc19cf963a35b2ff8f804e17e3b804000d",
                  "3a3bc124a10b0cf416656f8f682a427445140895440cca644c6aa38966399f0c"],
                [ "f499f0e4922d5cba35e3cc033489b60ec7ea26ff19cc9dd29357670f4bf8790b",
                  "1a4732b31f0f1a7d3322be5c4baca098f0a032c192bf9f8a6b5fd83cbdd9d401"]],
        "cc": "095fcab7ebf64c2ffecdacf11b70f97f0e709de0a84b3a13abca627f9df2c901"
      }, {
        "ss": [ [ "1808924b4154118c48f0b305562b6ffba86f38c64d4d8a087823f3383cddd006",
                  "4b18544be50aee8c4594b568d6be741c155a132cb83392d9b1a4cf35c3d5760c"],
                [ "9a95eeeecf3c3a48c43873a372c263357ff5f258a7bf8ed29a767237b0b0f202",
                  "4f1ea4d9d4b56db780dd078a3e8219d0f54eaccc197901671002a206f063cb0e"],
                [ "2c800017cab2b8388f58fed0d61a46570f64cacd8fabc4e84ddee735b3135f0a",
                  "458016f6fdf58b329fae0f929226ee2b8e410a14db8c6ede9b74fb718de71507"],
                [ "ea0fcaf793602dce25c8b2c4d17163f4298933b3fb09874307d8cde9a63c2c0c",
                  "df76fbcd336c07f37e90e1a0d0db1ba49519ba4325062228bc9242af2c525703"],
                [ "e3a7a0477eeb602a9a8203a6a496cca90c4769d57410246c4c8d665df34df900",
```

```
111                   "44d206154f0ca85e12a92eefdbc3784e17e701a32ff93b550467679f67500c0d"]],
112           "cc": "6f80de1c1d566776d2831f15c9a85fb1d8e8cecfd0d2753b318f0e84d89d3b08"
113         }, {
114           "ss": [ [ "5b82b4644b57e3d623de7c72c6ebd52959815c12c80b479e4cbe5437cf67640c",
115                     "b70e0b69c75faba6a1630429f9f497db351347c210467f69e1b1c5f1a72afe02"],
116                   [ "f25a93a98f980cf489eb8f69369f4ec63eaea91fd677decab9b6ca0fe2feb606",
117                     "124536c374cb6023a6aa6f22ae6e115a1ba12cb36c48f5f5ad43ce90f471da02"],
118                   [ "151ddc82322456d7f31b8b4b2290098c3bf2428370c7ef325660b5463ff26404",
119                     "2fb9d2979e16c2b1131686bb85068ec559f9c6c64581e609b451bb2cd9d5740d"],
120                   [ "c03bed01d6ad60b3da5d2c88cf2e5023b51133c37e4917511715a11f09d8740d",
121                     "432e01c2075ab6361af8636cc1c9254e12db98f5c323088792dfb42a1c894401"],
122                   [ "52206d801214e70d20ee7ca53823c143aa06c3d1b22b118cc8a15c9f861f0102",
123                     "563c134f56f7a290e0980877e93bc4b08651e53dade079b1e6c066b70fb81406"]],
124           "cc": "4102cd245db3e0d7c0e2280cfdba38b9b7a7ad8715b8fe68c1170cf923b3d70d"
125         }]
126     }
127 }
128
129
130
```

Transaction components

What follows is a short explanation of most important elements in transactions. We do only mention components that are specific or differ from the previous `RCTTypeFull` transaction type.

- `type` (line 53) - Signature type, in this case the value 2, corresponding to `RCTTypeSimple` transactions
- `pseudoOuts` (lines 55-58) - Pseudo-outputs used for commitments to zero against the real previous outputs, as described in Section 5.5.1. Please recall that the sum of these commitments will equal the sum of the 2 output commitments of the transaction.