Incognito Chain

Project Documentation

Privacy Version 2

(Draft V.0.1)

November 16, 2021

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Privacy for all Blockchains

Blockchains have introduced an entirely new asset class: cryptoassets. These cryptoassets use peer-to-peer technology to operate without central authority or banks. The management of transactions and the issuance of cryptoassets are carried out collectively by the users of these networks.

Bitcoin was the first cryptoasset; today, there are over 1,600. People have started buying Bitcoin, instead of gold, as their long-term store of value. Under the mattresses of volatile economies, the world's most desirable fiat currencies are replaced by stable coins that can be sent and received with borderless freedom. Waves of startups now sell their native cryptoassets to investors, not equity.

Today, anyone can send BTC, ETH, and thousands of other cryptocurrencies to another party without going through a centralized financial institution [1, 2]. For those who value privacy, these cryptocurrencies come with a big tradeoff. Transactions are recorded on public ledgers, displaying amounts involved, inscribing virtual identities of their senders and receivers. Given the choice, we strongly believe that very few people will willingly disclose their crypto financials to the entire world. The inherent lack of privacy to crypto networks today is a real threat to the entire crypto space. Existing solutions like Monero[1], Zcash[2], and Grin[3] introduced their own version of cryptocurrencies that focus on privacy, based on CryptoNote [3], Zerocash [4], and Mimblewimble [5] respectively.
Incognito takes a different approach, based on the premise that people don't want a new cryptocurrency with privacy. What they really want is privacy for their existing cryptocurrencies: incognito mode for any cryptocurrency. Incognito is designed so users do not have to choose between their favorite cryptocurrencies and privacy coins. They can have both. They can hold any cryptocurrency and still be able to use it confidentially whenever they want. Privacy needs to be ubiquitous, inclusive, and accessible.

[1]https://www.getmonero.org/
[2]https://z.cash
[3]https://grin.mw

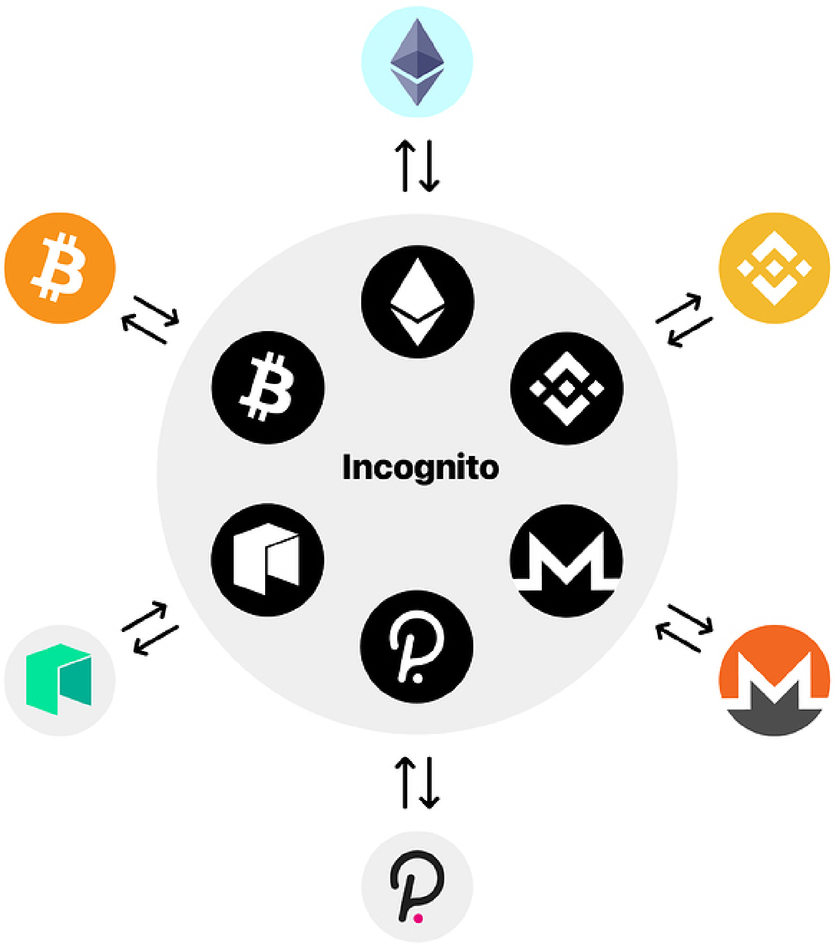

Figure 1.1: Incognito as a privacy hub. It is interoperable with other cryptonetworks via shielding and unshielding processes , which allow cryptocurrencies like BTC and ETH to go incognito and back.

First, we propose a solution to shield any cryptocurrency such as BTC, ETH, and USDT. In effect, any cryptocurrency can now be a privacy coin. Both shielding and unshielding processes are carried out via a decentralized group of trustless custodians . Once shielded, transactions are confidential and untraceable. To provide privacy, we employed: *linkable ring signatures* [6], *homomorphic commitments* [7], *zero-knowledge range proofs* [8], *confidential assets* [9].

Second, we present a solution to scale out a privacy-focused cryptonetwork by implementing sharding on privacy transactions and a new consensus based on proof-of-stake [10], pBFT [11], and BLS [12]. Transaction throughput scales out linearly with the number of shards. Currently, with 8 shards active, Incognito can handle 100 transactions per second (TPS). And with a full deployment of 64 shards, Incognito can handle 800 TPS – a significantly higher number than that of other privacy blockchains, which usually can only handle less than 10 TPS.

This document describes Incognito's privacy features and implementations. For the sake of brevity, some lesser details have been discarded.

1.2 Improvements

Privacy version 2 is an upgrade, designed to afford the user the maximum amount of privacy possible. We summarize these upgrades in Table 1.1.

Table 1.1: A comparison between privacy version 1 and version 2

Property	**Privacy V1**	**Privacy V2**	**Note**
Hide amount	YES	YES	Pedersen commitments and Bulletproofs
Hide sender	YES	YES	One-of-many proofs (V1) vs MLSAG (V2)
Hide receiver	NO	YES	One-time addresses
Hide asset type	NO	YES	Confidential assets
Fullnode dependency	High	Low	New key pairs
Transaction Fee	PRV + Token	PRV	Tx V2 only supports PRV fees

1.3 Notations

Throughout this document, the following notations are used.

- $\mathcal{E}$ denotes the elliptic curve used, with G being its generator, of order q.
- Regular lower case letters (e.g. x, y), denote scalars within the elliptic curve (i.e. $x, y \in \{0, 1, \ldots, q\}$), or simple values, strings, etc.
- Regular upper case letters (e.g. A, B), denote curve points (i.e. $A, B \in \mathcal{E}$), and complicated constructs.
- $\mathcal{H}_s$ denotes a hash function that digests an arbitrary string into a scalar in the set $\{0, 1, \ldots, q\}$.
- $\mathcal{H}_p$ denotes a hash function that digests an arbitrary string into a point on the elliptic curve $\mathcal{E}$.
- Textsf lower case letters, e.g. k, are used to denote private keys while public keys are denoted by textsf upper case letters, like K.
- All listing codes are written in Golang.
- When needed (e.g. operands with different fonts), the operator $\cdot$ will be used to denote the multiplication, either of two scalars or a scalar with an elliptic curve point, to enhance readability. For example, $c \cdot \mathsf{k}$ is the multiplication of two scalars c and k; $c \cdot \mathsf{SN}$ is the multiplication of the scalar c with the elliptic curve point SN.

Chapter 2

Backgrounds

For more information on the mathematics, elliptic curves and other primitives used in this document, we encourage readers to refer to this section (Chapter 2 of [13]).

Here, we present only the construction of these primitives. When needed, we will explain why they work, and how they help bring privacy to the blockchain. Inessential concrete implementation details that are outside the scope of this document have been omitted.

2.1 Ed25519

Incognito makes use of the Ed25519 elliptic curve, introduced by Bernstein *et al.* [14], offering 128 bits of security. It is one of the fastest ECC curves and is not patented. Moreover, a point on the Ed25519 curve can easily be compressed using only one coordinate.

The Ed25519 curve is defined by

$$\mathcal{E} : -x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2,$$

over the finite field $\mathbb{F}_p$ with $p = 2^{255} - 19$ (this is why it is called Ed25519), and it uses the base point with $y = 4/5$ and a unique positive x. This point generates a cyclic subgroup, whose order is the prime

$$q = 7237005577332262213973186563042994240857116359379907606001950938285454250989.$$

Remark 1. The Ed25519 curve is birationally equivalent to the Montgomery curve Curve25519 [14].

2.2 Pedersen Commitments

Inputs and outputs of Incognito's transactions are expressed by Pedersen commitments [7]

$$C = \mathsf{Com}_G(x; r) = xG + rH$$

where C denotes the commitment, x the amount, r the blinding factor, G and H the two points chosen from $\mathcal{E}$ (the discrete logarithm relation between G and H is unknown).

Pedersen commitments satisfy the properties of **perfectly hiding** and **computationally binding**.

- **Perfectly hiding**. An attacker with infinite computing power cannot tell what amount has been committed. In other words, a Pedersen commitment is information-theoretically private since there are many combinations of (x, r) that produce the same commitment C.
- **Computationally binding**. No efficient algorithm running in a computationally-bounded amount of time can produce fake commitments, except with a negligible probability.

Besides, Pedersen commitments are *additively homomorphic*, which is a powerful feature allowing Incognito to keep transaction amounts private. If $C(x_1), C(x_2)$ are two commitments for values x_1 and x_2, then

$$C(x_1) + C(x_2) = \mathsf{Com}_G(x_1; r_{x_1}) + \mathsf{Com}_G(x_2; r_{x_2}) = \mathsf{Com}_G(x_1 + x_2; r_{x_1} + r_{x_2}) = C(x_1 + x_2).$$

This property enables us to prove that the sum of inputs equals the sum of outputs, without revealing the actual amounts transacted.

In subsequent sections, we will see that different tokens use different bases G for commitments. However, when we talk about general commitments, the point G will be used to denote the base value of a commitment.

2.3 Range Proofs

One problem with additive commitments is that if we had commitments $C(x_1), C(x_2)$ (inputs), $C(y_1), C(y_2)$ (outputs) and we intended to use them to prove that $(x_1 + x_2) - (y_1 + y_2) = 0$, then those commitments would still apply even when a value in the equation is negative.

For example, we could have $(x_1, x_2, y_1, y_2) = (10, 5, 16, -1)$, and

$$(10 + 5) - (16 + (-1) = 0,$$

where

$$16G - G = 16 + (q - 1)G = (q + 15)G.$$

In the context of a payment system, we could use two small inputs and produce a very large output. We would thus have spent more than what we possess. In the example above, with a total input value of 15, we have created a transaction that produces a total output value of $q + 15$. Therefore, what is needed is to prove that each committed amount is non-negative.

One approach is to use zero-knowledge range proofs (ZKRP), which is a cryptographic technique that proves that a secret value belongs to a certain interval. ZKRPs do not leak any information about the secret value, other than the fact that they lie in the interval. For example, if we define this interval to be all integers between 18 and 200, a person can use the ZKRP scheme to prove that she is over 18.

There are many constructions for ZKRPs in the literature. The first constructions of ZKRP protocols were proposed in 1993 by Damgard [15] and in 1997 by Fujisaki and Okamoto [16]. Unfortunately, these proposals cannot be used efficiently in practice. The first practical construction was proposed by Boudot in 2000 [17]. Since then, more and more constructions have been proposed. They can be primarily classified into four categories: *square decomposition-based* (such as [17, 18, 19]); *signature-based* (such as [20]); *multi-base decomposition-based* (such as [21, 22, 23]); and Bulletproofs [24]. While all the schemes of the first three categories depend upon a trusted setup, Bulletproofs does not. Moreover, Bulletproofs produces very small proof sizes along with an efficient implementation.

2.4 Schnorr Signatures

Digital signatures are at the heart of not just the Incognito chain, but also any other blockchain. Digital signatures help you prove the ownership of coins and give you the ability to send coins to other people in a transaction. Without these signatures, you cannot spend any coins on the network.

Digital signatures have been around since Diffie and Hellman first introduced them in 1976 [25]. The first digital signatures were based on RSA [26], which were popular in the '90s. In recent years, discrete logarithm and elliptic curve approaches have become more popular because they allow for improved computational efficiency and smaller key size without diminishing security.

In the discrete logarithm space, ElGamal-based signature schemes have been the most frequently deployed, due to the National Institute of Standards and Technology (NIST) releasing their patented DSA under a royalty-free license when implementing the FIPS 186 standard in '93. Even Bitcoin has adopted a variant of DSA: the elliptic-curve based ECDSA, which uses the secp256k1 elliptic curve.

However, DSA is not the only answer for modern digital signatures. Schnorr signatures [27] offer numerous benefits over traditional methods[1].

- **Simplicity**. Schnorr signatures are considered the simplest form of digital signature.
- **Multisig**. Schnorr signatures can be added together in a very simple manner, which creates a number of multi-signatures (also called *multisig*). These signatures look exactly like a single signature.
- **Group Signatures**. In Section 2.5, we will see how Schnorr signatures are used to construct group signatures, which allow us to prove that a signer belongs to a group, without needing to know his identity.

To sign a message, the signer uses the Algorithm 1 to sign a message. The Algorithm 2 is used to verify the correctness of a signature.

Algorithm 1 Schnorr signing

```
1: procedure SchnorrSign($\mathsf{k}, \mathsf{msg}$)
2:     $\alpha \overset{\$}{\leftarrow} \{0, 1, \ldots, q\}$
3:     $c \leftarrow \mathcal{H}_s(\mathsf{msg}, \alpha G)$
4:     $r \leftarrow \alpha - c \cdot \mathsf{k}$
5:     return $\sigma = (c, r)$
```

Algorithm 2 Schnorr verification

```
1: procedure SchnorrVerify($\mathsf{msg}, \sigma, pub$)
2:     $(c, r) \leftarrow \sigma$
3:     $c' \leftarrow \mathcal{H}_s(\mathsf{msg}, rG + c \cdot \mathsf{K})$
4:     return $c \overset{?}{=} c'$
```

[1] https://github.com/WebOfTrustInfo/rwot1-sf/blob/master/topics-and-advance-readings/Schnorr-Signatures–An-Overview.md

Why verification works

It is easy to see that $c' = c$ if the signed message equals the verified message

$$\begin{aligned} rG + c \cdot \mathsf{K} &= (\alpha - c \cdot \mathsf{k})G + c(\mathsf{k} \cdot G) \\ &= \alpha G \end{aligned}$$

and hence $c' = \mathcal{H}_s(\mathsf{msg}, rG + c \cdot \mathsf{K}) = \mathcal{H}_s(\mathsf{msg}, \alpha G) = c$.

2.5 Ring Signatures

2.5.1 Overview

A ring signature scheme allows a member of a group to sign a message on behalf of the group without revealing his identity [8]. The Schnorr signature schmeme in Section 2.4 can be considered a one-key ring signature.

The first proposal [8] of ring signatures requires a trusted setup and is managed by a trusted party. This party has the power to break the anonymity of a signature. Liu *et al.* [6] presented a more interesting scheme called 'LSAG' that provides: *anonimity*, *linkability* and *spontaneity*.

Figure 2.1: The identity of the signer is obscured. For example, if you encounter a ring signature with the public keys of Annie, Bob, John, and Peter, you will be able to claim that one of these users is the signer, but not be able to pinpoint him or her.

In the LSAG signature scheme, group formation is spontaneous. There is no group manager to reveal the identity of the true signer. Due to this property, we call the group an *ad hoc* group or a ring. The signer can form a group by simply collecting the public keys of other group members. These diversion group members, often called *decoys* or *mixins*, are pulled from historical transactions. The unified signature provides anonymity to the signer.

In Incognito, a ring signature is used to authorize the spending of an Unspent Transaction Output [1], or "UTXO" without revealing the spender's identity. The ring consists of the actual UTXO being spent as well as its decoys, which are various random outputs from historical transactions. The actual UTXO and its decoys collectively make up the inputs of the transaction. To the public, any of these inputs could equally be the actual output being spent (see Figure 2.2).

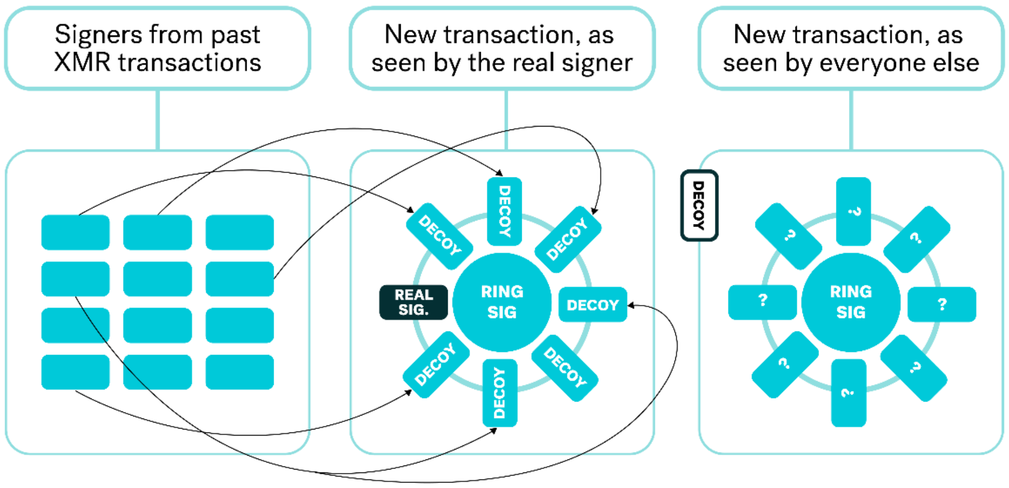
Signers from past XMR transactions
New transaction, as seen by the real signer
New transaction, as seen by everyone else
DECOY
DECOY
DECOY
REAL SIG.
RING SIG
DECOY
DECOY
DECOY
DECOY
DECOY
RING SIG
?

Figure 2.2: Visualization of ring signature.

To prevent double-spending attacks, we require ring signatures to be linkable. That is, it is able to link two signatures issued under the same private key. This property is powered by introducing a *so-called* key-image, which can only be calculated using the knowledge of the private key, and is unique for each private/public key. With key images in place, anyone can verify whether two signatures have been issued by the same group member without learning who the signer is. A key image is derived from each UTXO being spent and is part of every ring signature. A list of all used key images is stored permanently as part of the transaction data so that any new ring signature that attempts to reuse an existing key image is considered double-spent.

In general, apart from other native properties of a digital signature scheme, the ring signature scheme in Incognito must also satisfy the following properties.

- **Signer Ambiguity**. The signer of a message can only be identified as a member of a ring, not the member himself. We use this to provide untraceability of transactions.
- **Linkability**. An observer should be able to link two different messages if they are signed by the same private key. This property helps prevent double-spending attacks.

2.5.2 Linkable Spontaneous Anonymous Group Signatures

We first consider the case of one-layered rings and then show how to extend this one-layered ring to the case of multi-layered rings in the next section. Let msg be the message to be signed, $\mathcal{R} = \{\mathsf{K}_1, \mathsf{K}_2, \ldots, \mathsf{K}_n\}$ be a set of distinct public keys, k_π be the signer's private key (corresponding to his public key $\mathsf{K}_\pi = \mathsf{k}_\pi G \in \mathcal{R}$), where π is a secret index.
To sign a message, the signer proceeds as in Algorithm 3. A verifer uses the Algorithm 4 to verify the correctness of a signature.

Why verification works

If the signed message is the same as the verified message, we have the following.

- $c'_1 = c_1$ (as in Line 5).

Algorithm 3 LSAG signing algorithm

```
1: procedure LSAGSign($\mathsf{k}_\pi, \mathsf{msg}, \mathcal{R}$)
2:     $\tilde{\mathsf{K}} \leftarrow \mathsf{k}_\pi \mathcal{H}_p(\mathsf{K}_\pi)$
3:     $\alpha \xleftarrow{\$} \mathbb{Z}_q$
4:     $c_{\pi+1} \leftarrow \mathcal{H}_s(\mathsf{msg}, \alpha G, \alpha \mathcal{H}_p(\mathsf{K}_\pi))$
5:     for $i \leftarrow \pi + 1$ to $\pi - 1$ do                    ▷ replace $1 \leftarrow n + 1$
6:         $r_i \xleftarrow{\$} \{0, 1, \dots, q\}$
7:         $c_{i+1} \leftarrow \mathcal{H}_s(\mathsf{msg}, r_i G + c_i \mathsf{K}_i, r_i \mathcal{H}_p(\mathsf{K}_i) + c_i \tilde{\mathsf{K}})$
8:     $r_\pi \leftarrow \alpha - c_\pi \mathsf{k}_\pi \mod q$
9:     return $\sigma = (c_1, r_1, \dots, r_n)$
```

Algorithm 4 LSAG verification algorithm

```
1: procedure LSAGVerify($\mathsf{msg}, \mathcal{R}, \sigma, \tilde{\mathsf{K}}$)
2:     if $q\tilde{\mathsf{K}} \neq 0$ then
3:         return 0
4:     $(c_1, r_1, \dots, r_n) \leftarrow \sigma$
5:     $c'_1 \leftarrow c_1$
6:     for $i = 1$ to $n$ do                    ▷ replace $1 \leftarrow n + 1$
7:         $c'_{i+1} \leftarrow \mathcal{H}_s(\mathsf{msg}, r_i G + c'_i \mathsf{K}_i, r_i \mathcal{H}_p(\mathsf{K}_i) + c'_i \tilde{\mathsf{K}})$
8:     return $c_1 \overset{?}{=} c'_1$
```

- For all index i,
  - if $i = \pi$,
  $$\begin{aligned} c'_{i+1} = c'_{\pi+1} &= \mathcal{H}_s(\mathsf{msg}, r_\pi G + c'_\pi \mathsf{K}_\pi, r_\pi \mathcal{H}_p(\mathsf{K}_\pi) + c'_\pi \tilde{\mathsf{K}}) \\ &= \mathcal{H}_s(\mathsf{msg}, r_\pi G + c_\pi(\mathsf{k}_\pi G), r_\pi \mathcal{H}_p(\mathsf{K}_\pi) + c_\pi(\mathsf{k}_\pi \mathcal{H}_p(\mathsf{K}_\pi)) \\ &= \mathcal{H}_s(\mathsf{msg}, \alpha G), \alpha \mathcal{H}_p(\mathsf{K}_\pi)) \\ &= c_{\pi+1} \qquad \text{(the same as in Line 4 of Algorithm 3);} \end{aligned}$$
  - if $i \neq \pi$ then c'_{i+1} of the verification (Line 7 of Algorithm 4) is calculated in the same way as c_{i+1} in the siging algorithm (Line 7 of Algorithm 3), and thus we can easily see $c'_{i+1} = c_{i+1}$.
- So for all cases, we have $c'_i = c_i$. The final check will therefore be valid.

Linkability. From the way key-images are generated,

$$\tilde{\mathsf{K}} = \mathsf{k}_\pi \mathcal{H}_p(\mathsf{K}_\pi),$$

it is easy to see that $\tilde{\mathsf{K}}$ is unique for each pair $(\mathsf{k}_\pi, \mathsf{K}_\pi)$. Hence, two signatures with the same key-image imply two signatures are signed by the same private key.

Remark 2. Linkability applies to two signatures generated from two rings $\mathcal{R}_1$, $\mathcal{R}_2$ which are not necessarily identical.

Remark 3. Theoretically, each public key in the ring has $\frac{1}{n}$ probability of being the true signer. Therefore, the larger the ring is, the more anonymous the signer gets. However, because the signature size is linear to the size of the ring, the larger the ring is, the longer the signature is.

2.5.3 Multilayered linkable spontaneous anonymous group signatures

For a transaction with multiple inputs, one has to sign with multiple private keys. Shen Noether of Monero [28] described a multi-layered generalization of the LSAG scheme. The signing and verification procedures are shown in Algorithm 5, 6, respectively.

Algorithm 5 MLSAG signing algorithm

```
1: procedure MLSAGSign($\{\mathsf{k}_{\pi,j}\}, \mathsf{msg}, \mathcal{R}$)
2:     for $j \leftarrow 1$ to $m$ do
3:         $\tilde{\mathsf{K}}_j \leftarrow \mathsf{k}_{\pi,j}\mathcal{H}_p(\mathsf{K}_{\pi,j})$
4:         $\alpha_j \xleftarrow{\$} \{0, 1, \ldots, q\}$
5:         for $i \leftarrow 1$ to $n$ do                                   ▷ except for $i = \pi$
6:             $r_{i,j} \xleftarrow{\$} \{0, 1, \ldots, q\}$
7:     $c_{\pi+1} \leftarrow \mathcal{H}_s(\mathsf{msg}, \alpha_1 G, \alpha_1 \mathcal{H}_p(\mathsf{K}_{\pi,1}), \ldots, \alpha_m G, \alpha_m \mathcal{H}_p(\mathsf{K}_{\pi,m}))$
8:     for $i \leftarrow \pi + 1$ to $\pi - 1$ do                              ▷ replace $1 \leftarrow n + 1$
9:         $c_{i+1} \leftarrow \mathcal{H}_s(\mathsf{msg}, r_{i,1}G + c_i\mathsf{K}_{i,1}, r_{i,1}\mathcal{H}_p(\mathsf{K}_{i,1}) + c_i\tilde{\mathsf{K}}_1, \ldots, r_{i,m}G + c_i\mathsf{K}_{i,m}, r_{i,m}\mathcal{H}_p(\mathsf{K}_{i,m}) + c_i\tilde{\mathsf{K}}_m)$
10:    for $j \leftarrow 1$ to $m$ do
11:        $r_{\pi,j} \leftarrow \alpha_j - c_\pi \mathsf{k}_{\pi,j} \mod q$
12:    return $\sigma = (c_1, r_{1,1}, \ldots, r_{1,m} \ldots, r_{n,1}, \ldots, r_{n,m})$
```

Algorithm 6 MLSAG verification algorithm

```
1: procedure MLSAGVerify($\mathsf{msg}, \mathcal{R}, \sigma, \{\tilde{\mathsf{K}}_1, \ldots, \tilde{\mathsf{K}}_m\}$)
2:     for $j \leftarrow 1$ to $m$ do
3:         if $q\tilde{\mathsf{K}}_j \neq 0$ then
4:             return 0
5:     $(c_1, r_{1,1}, \ldots, r_{1,m} \ldots, r_{n,1}, \ldots, r_{n,m}) \leftarrow \sigma$, $c'_1 \leftarrow c_1$
6:     for $i \leftarrow 1$ to $n$ do                                       ▷ replace $1 \leftarrow n + 1$
7:         $c'_{i+1} \leftarrow \mathcal{H}_s(\mathsf{msg}, r_{i,1}G + c'_i\mathsf{K}_{i,1}, r_{i,1}\mathcal{H}_p(\mathsf{K}_{i,1}) + c'_i\tilde{\mathsf{K}}_1, \ldots, r_{i,m}G + c'_i\mathsf{K}_{i,m}, r_{i,m}\mathcal{H}_p(\mathsf{K}_{i,m}) + c'_i\tilde{\mathsf{K}}_m)$
8:     return $c_1 \overset{?}{=} c'_1$
```

Why verification works. Verification works as it is for the case of the one-layered ring signature scheme.

Linkability. Just as before, if a private key $\mathsf{k}_{\pi,j}$ is used to sign two different messages, an observer is able to spot this since the same $\tilde{\mathsf{K}}_j$ will be used.

Remark 4. Note that when considering the ring as an $n \times m$ matrix, all the public keys of the signer must be in the same column. Thus, the probability that the true signer is spotted is the same as in the LSAG scheme, which is $\frac{1}{n}$.

Remark 5. If one public key of the real signer is identified, other public keys will also be identified.

Chapter 3

Confidential Transactions

Confidential transactions (CT) is a cryptographic protocol which results in the amount value of a transaction being ‘encrypted’. The encryption process is special because it makes it possible to verify that no coins have been created or destroyed within a transaction, while still obscuring the exact transaction amounts[1].

3.1 Ingredients of a Confidential Transaction

In Incognito, each privacy token transaction is confidential and untraceable. Incognito uses several cryptographic primitives to build confidential transactions. These primitives can be found in Table 3.1 along with how they help make Incognito’s transactions confidential.

Table 3.1: Cryptographic primitives and their purposes

Primitive	**Purposes**
Pedersen commitments	Hide transaction amounts
Ring signatures	Hide senders; prove asset validity; prove amount validity
One-time addresses	Hide recipients
Blinded asset tags	Hide transacted tokens
Bulletproofs	Prove that amounts transacted are not inflationary

3.2 User Keys

In the Incognito Chain, each user possesses the following key pairs.

- $\mathsf{private/public}$ key $(\mathsf{k}, \mathsf{K} = \mathsf{k}G)$: the master private/public key.
- $\mathsf{privateOTA/publicOTA}$ key $(\mathsf{k}_{\mathsf{ota}}, \mathsf{K}_{\mathsf{ota}} = \mathsf{k}_{\mathsf{ota}}G)$: used to generate one-time addresses and conceal the $\mathsf{tokenID}$ of transactions.
- $\mathsf{privateView/publicView}$ key $(\mathsf{k}_{\mathsf{view}}, \mathsf{K}_{\mathsf{view}} = \mathsf{k}_{\mathsf{view}}G)$: used to encrypt/decrypt (or seal/unseal) the amount of output coins.

[1]https://en.bitcoin.it/wiki/Confidential$_{t}ransactions$

3.3 Hiding amounts

3.3.1 Pedersen commitments for privacy

In privacy version 2, each output amount x is stored as a Pedersen commitment in the following form

$$c = \mathsf{Com}_G(x; r) = xG + rH, \tag{3.1}$$

where G is the commitment base point, and G will be different for different assets.

For the recipient to know much money he receives, the blinding factor r must be communicated to him. If the sender encloses this blinding factor with the transaction in cleartext, anyone will be able to see the amount being transacted. Therefore, the sender must “encrypt” the amount and the blinding factor in such a way that the recipient can “decrypt” and spend.

Here is how the encryption works for the sender. Given the receiver’s key pairs (k, K), $(\mathsf{k}_{\mathsf{view}}, \mathsf{K}_{\mathsf{view}})$, the amount x, the blinding factor r, the sender proceeds as follows.

1. The sender chooses a random scalar r_{amt} and computes $\mathsf{SS} = r_{\mathsf{amt}}\mathsf{K}_{\mathsf{view}}$, $R_{\mathsf{amt}} = r_{\mathsf{amt}}G$.
2. The sender computes $\hat{r}_{\mathsf{amt}} = r_{\mathsf{amt}} + \mathcal{H}_s(\mathcal{H}_s(\mathsf{SS}))$.
3. The sender computes $\hat{x} = x + \mathcal{H}_s(\mathcal{H}_s(\mathcal{H}_s(\mathsf{SS})))$.
4. $(\hat{x}, \hat{r}_{\mathsf{amt}})$ and R_{amt} are communicated to the recipient.

Upon receving $(\hat{x}, \hat{r}_{\mathsf{amt}})$ and R_{amt}, the recipient decrypts the output by

1. re-calculating the shared secret $\mathsf{SS}' = \mathsf{k}_{\mathsf{view}}R_{\mathsf{amt}}$;
2. re-computing the blinding factor $r'_{\mathsf{amt}} = \hat{r}_{\mathsf{amt}} - \mathcal{H}_s(\mathcal{H}_s(\mathsf{SS}'))$;
3. re-computing the amount $x' = \hat{x} - \mathcal{H}_s(\mathcal{H}_s(\mathcal{H}_s(\mathsf{SS}')))$;
4. checking if $c \overset{?}{=} \mathsf{Com}_G(x'; r'_{\mathsf{amt}})$.

Due to the binding property of a Pedersen commitment, if $c = \mathsf{Com}_G(x'; r')$, the recipient can be sure that the transacted amount is x'.

Remark 6. In the case of limited computing resources, the recipient can ask other third parties (e.g. full nodes) to decrypt the outputs sent to him by sending his $\mathsf{k}_{\mathsf{view}}$ to these parties.

3.3.2 Transactions in play

In order for a transaction to be valid, its inputs must be referenced with some ouputs of previous transactions. Each output consists of a one-time address, an asset tag, and an output commitment hiding the amount. Incognito employs the same technique as in Monero [28] to help an observer verify that the sum of input amounts equals the sum of output amounts, without ever seeing the actual amount.

Suppose we have a transaction of m inputs with amounts $x_1, \dots, x_m$, and p outputs with amounts $y_1, \dots, y_p$. What we need to verify is

$$\sum_j x_j - \sum_i y_i = 0. \tag{3.2}$$

As we use commitments to hide the actual amounts, an observer only sees $C(x_j)$ and $C(y_i)$. Therefore, we can prove that input sum equals output sum by making

$$\sum_j C(x_j) - \sum_i C(y_i) = 0. \tag{3.3}$$

However, if we use the original commitments for inputs (as produced by previous transactions), an observer can easily identify which UTXOs are being spent. Instead, replace the input commitments by $C'(x_j) = C(x_j) + r_j H$. Note that $C'(x_j)$ and $C(x_j)$ are commitments to the same value. The Equation (3.3) becomes

$$\sum_{j=1}^{m} C'(x_j) - \sum_{i=1}^{p} C(y_i) = \sum_{j=1}^{m} r_j H, \tag{3.4}$$

Hence, the sender would be able to use this value as a *commitment to zero*, since she can make a signature using the private key $\sum_j r_j$ and prove there is no G component to the sum.

3.3.3 Adding fees

In the presence of a fee f, the Equation (3.2) becomes

$$\sum_j x_j - \sum_i y_i - f = 0. \tag{3.5}$$

Since the fee must be transparent, we can calculate its commitment as $C(f) = fG$ (there is no blinding factor). Now the proof can prove that the input amounts equal output amounts plus fee as

$$\sum_{j=1}^{m} C'(x_j) - \sum_{i=1}^{p} C(y_i) - fG = \sum_{j=1}^{m} r_j H. \tag{3.6}$$

3.3.4 Bulletproofs

One more thing we have to prove with transaction amounts is that each committed value is not inflationary. As we discussed in Section 2.3, a ZKRP can be used to address this problem.

Bulletproofs [24] is adopted in the Incognito chain because of its succint and trustless nature. Unlike other ZKRPs, Bulletproofs do not require a trusted setup. Specifically, Bulletproofs provide the following characteristics which are desired in the Incognito network. Due to the lengthiness and high complexity of Bulletproofs, please refer to the original paper [24] for a formal description of these characteristics.

- **Public verifiability**. Anyone can verify the validity of the proof.
- **Zero-knowledgeness**. A verifier learns nothing about the fact that the prover knows the opening of the commitment and the committed value lies in the predefined range.
- **Non-interactiveness**. The proof generation process is one-shot, meaning that no interaction is needed between the prover and the verifier.
- **Succintness**. The proof size is only logarithmic in the witness size. For a range of length n, the proof only contains $2\log_2 n + 9$ group and field elements.

- **No trusted setup**. Unlike zk-SNARKs, Bulletproofs require no trusted setup, which helps preserve the decentralized nature of the blockchain.
- **Aggregatability**. Bulletproofs supports aggregation of range proofs, so that a party can prove that m committed values lie in a given range by providing only an additive $\mathcal{O}(\log m)$ group elements over the length of a *single* proof.
- **Efficient verification**. Proof generation and verification times are linear in n.

3.4 Hiding receivers

In a typical cryptonetwork like Bitcoin or Ethereum, a public address is all that is needed for anyone to view incoming and outgoing transactions associated with that address [1, 2]. These transactions are public and can be easily linked together to infer total balances and spending patterns.

As privacy is Incognito's main focus, two crucial properties that each transaction in the network must satisfy include:

- **Untraceability**. For each incoming transaction all possible senders are equiprobable.
- **Unlinkability**. For any two outgoing transactions it is impossible for an observer to tell if they were sent to the same person.

Privacy version 1 satisfies the untraceability property, where the sender of a transaction is part of a group of decoys. However, the payment address is still used directly in each transaction to receive assets. Therefore, recipients of a transaction can be detected. In privacy version 2, we introduce stealth addresses in the form of one-time addresses (OTA), which help hide the identities of the recipients of a transaction. These stealth addresses can be thought of as one-time deposit boxes. Only the recipient can open the box to see what is inside and spend it.

3.4.1 What is a one-time address (OTA)?

OTAs are based on the Diffie-Hellman key exchange protocol [29], a cryptographic method that allows two users to create a shared secret even in the presence of an adversary who can observe all communications between them. An OTA is created for each transaction output and is never repeated in the network, i.e, two distinct outputs cannot have the same OTA. In this way, even if two outputs are sent to the same recipient, no one (other than the sender and the recipient) can link them together.

Let's say that Alice makes a simple transaction transferring two output coins to Carol, and suppose that Alice knows Carol's payment address. Alice proceeds as follows.

- Alice generates two OTAs for two output coins using Carol's payment address.
- Alice creates a transaction, transferring these coins to Carol.
- Carol uses her private key to recognize the UTXOs being sent to her by scanning all incoming transactions. Once the UTXOs are found, Carol is able to compute the one-time private keys that correspond to the one-time public keys. Carol can spend these UTXOs with her private spend key.

Because an OTA is unique for each output, even when these two coins are sent to the same person, an observer cannot tell whether or not these coins belong to the same person.

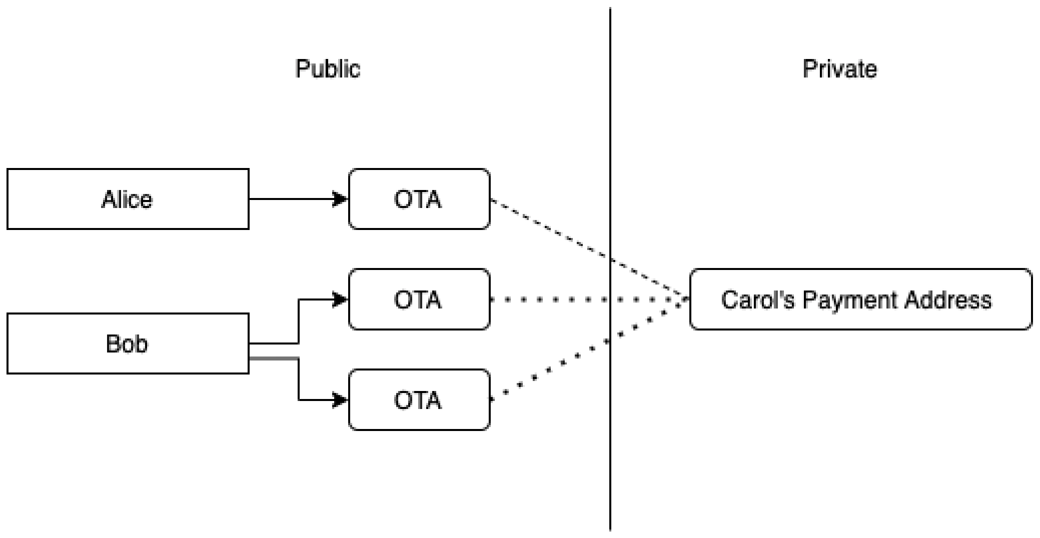

Figure 3.1: OTA Addresses

3.4.2 How to generate an OTA?

Suppose that Bob has key pairs (k, K) and $(\mathsf{k}_{\mathsf{ota}}, \mathsf{K}_{\mathsf{ota}})$ and Alice knows $(\mathsf{K}, \mathsf{K}_{\mathsf{ota}})$. To transfer a coin to Bob, Alice proceeds as follows.

1. Alice chooses a random scalar r_{ota} and computes $\mathsf{SS} = r_{\mathsf{ota}}\mathsf{K}_{\mathsf{ota}}$, $R_{\mathsf{amt}} = r_{\mathsf{amt}}G$.
2. Alice computes the one-time address $\mathsf{K}^{\mathsf{coin}} = \mathcal{H}_s(\mathsf{SS})G + \mathsf{K}$. $\mathsf{K}^{\mathsf{coin}}$ is the public key of the transacted coin.
3. Alice communicates $\mathsf{K}^{\mathsf{coin}}$ and R_{ota} along with the transaction to Bob.

At Bob side, using the privateOTA key, he can detect if an UTXO belongs to him by

1. re-computing the shared secret $\mathsf{SS}' = \mathsf{k}_{\mathsf{ota}}R_{\mathsf{ota}} = r_{\mathsf{ota}}\mathsf{K}_{\mathsf{ota}}$;
2. re-computing the one-time address $\mathsf{K}' = \mathsf{K}^{\mathsf{coin}} - \mathcal{H}_s(\mathsf{SS}')G$;
3. and checking if $\mathsf{K} \overset{?}{=} \mathsf{K}'$; if so, Bob knows that this output coin belongs to him.

From the way $\mathsf{K}^{\mathsf{coin}} =$ is calculated, we have

$$\mathsf{K}^{\mathsf{coin}} = \mathcal{H}_s(\mathsf{SS})G + \mathsf{K} = (\mathcal{H}_s(r_{\mathsf{ota}}\mathsf{K}_{\mathsf{ota}}) + \mathsf{k})G,$$

and thus the corresponding private key of this coin is $\mathsf{k}^{\mathsf{coin}} = \mathcal{H}_s(r_{\mathsf{ota}}\mathsf{K}_{\mathsf{ota}}) + \mathsf{k}$. Only Bob is able to recover the private key $\mathsf{k}_{\mathsf{ota}}$ because he knows the shared secret SS and the master private key k. As we saw in Section 2.5, without the knowledge of the master private key, Alice cannot compute the key image of the output coin she sent to Bob. Therefore, Alice is unable to spend this output coin, and does not know for sure if Bob has spent this output coin.

On the other hand, anyone with privateOTA key can tell if a coin belongs to Bob. Therefore, Bob may give his privateOTA to a third party to listen to new output coins sent to him.

Remark 7. Monero uses the same key pair (named *view key*) for generating OTAs and encrypting transaction amounts. This means that anyone with access to this view key can see the transactions sent to a specific person, including the amounts transacted. Therefore, to obtain full privacy, Monero users are encouraged to run their own full node. This can be fairly expensive for regular users, who often give their view key to a (trusted) full node instead to listen to new transactions sent to them. As a result, this trusted full node also knows all their UTXO amounts. At Incognito, we use separate key pairs for generating one-time addresses and encrypting transaction amounts. A user can give a full node both the $\mathsf{privateOTA}$ and $\mathsf{privateView}$ keys, or just the $\mathsf{privateOTA}$ key.

Remark 8. Since the Incognito network is split into shards, we need to ensure that the public key $\mathsf{K}^{\mathsf{coin}}$ is in the same shard as the public key K. In the implementation, $\mathsf{K}^{\mathsf{coin}}$ is calculated as

$$\mathsf{K}^{\mathsf{coin}} = \mathcal{H}_s(\mathsf{SS}, \mathsf{idx})G + \mathsf{K} = (\mathcal{H}_s(r_{\mathsf{ota}}\mathsf{K}_{\mathsf{ota}}, \mathsf{idx}) + \mathsf{k})G,$$

for some index idx, and idx is also communicated to the recipient. As a result, the corresponding private key is $\mathsf{k}^{\mathsf{coin}} = \mathcal{H}_s(r_{\mathsf{ota}}\mathsf{K}_{\mathsf{ota}}, \mathsf{idx}) + \mathsf{k}$.

3.5 Hiding senders

In Section 2.5, we introduced the notion of MLSAG signatures to hide the true sender of a transaction in a group of unrelated people. Suppose Alice makes a transaction spending m input coins which have OTAs $\mathsf{K}^{\mathsf{coin}}_{\pi,1} \ldots, \mathsf{K}^{\mathsf{coin}}_{\pi,m}$, respectively. Alice of course knows the corresponding private keys $\mathsf{k}^{\mathsf{coin}}_{\pi,i}$. For each input coin, she chooses $n-1$ other random output coins as mixins and applies the Algorithm 5 with private keys $\{\mathsf{k}^{\mathsf{coin}}_{\pi,1}, \ldots, \mathsf{k}^{\mathsf{coin}}_{\pi,m}\}$ to sign the transaction.

Remark 9. Note that the algorithm for choosing random mixins will affect the anonymity of a transaction. Recent studies (e.g. [30, 31]) show that the majority of Monero inputs are traceable when MLSAG signatures are employed in their network, due to the observation that (1) $n-1$ mixins are chosen randomly among all output coins from genesis blocks to current blocks; and (2) input coins are most likely the recent ones. Therefore, by calculating the age of each coin in the ring, we can reduce the anonymity of a Monero transaction. Different strategies can be employed to address this problem. Perhaps we can used fixed ring for each input, or when creating a transaction, users are encouraged to form the ring with care.

Preventing double-spending attacks. As we discussed in Section 2.5, MLSAG signatures satisfy the *linkability* property. Namely, if two signatures have a common key image, we know that these signatures are produced by the same private key. Therefore, to prevent double-spending from happening, the network only needs to verify that each key image of a transaction has not been seen before.

Chapter 4

Confidential Assets

In Privacy version 1, each asset type in the Incognito Chain has a unique identifier ($\mathsf{tokenID}$). For example,

pUSDT: "716fd1009e2a1669caacc36891e707bfdf02590f96ebd897548e8963c95ebac0"

pBTC: "b832e5d3b1f01a4f0623f7fe91d6673461e1f5d37d91fe78c5c2e6183ff39696".

When transferring a token, its $\mathsf{tokenID}$ must be specified. This makes it easy to tell which token is associated with a transaction. In version 2, we employ confidential assets [9], which uses randomness to blind the $\mathsf{tokenID}$ transacted.

4.1 Asset commitments

For each $\mathsf{tokenID}$ A, its non-blinded (clear) asset tag is calculated as $G_A = \mathcal{H}_p(A)$. Consider the Petersen commitment of a transaction with two inputs and two outputs involving two distinct $\mathsf{tokenID}$s A and B.

$$\begin{aligned} c_{inA} = \mathsf{Com}_{G_A}(x_1; r_{A_1}) = x_1 G_A + r_{A_1} H,\ c_{outA} = \mathsf{Com}_{G_A}(x_2; r_{A_2}) = x_2 G_A + r_{A_2} H, \\ c_{inB} = \mathsf{Com}_{G_B}(y_1; r_{B_1}) = y_1 G_B + r_{B_1} H,\ c_{outB} = \mathsf{Com}_{G_B}(y_2; r_{B_2}) = y_2 G_B + r_{B_2} H. \end{aligned} \tag{4.1}$$

For a transaction to be valid, the sum of output coins must be equal to the sum of input coins. In this case, we must have

$$(c_{outA} + c_{outB}) - (c_{inA} + c_{inB}) = 0G_A + 0G_B + rH \tag{4.2}$$

$$\Leftrightarrow \quad (x_2 - x_1)G_A + (y_2 - y_1)G_B + (r_{A_2} + r_{B_2} - r_{A_1} - r_{B_1})H = 0G_A + 0G_B + rH. \tag{4.3}$$

As G_A and G_B are independent (i.e. we do not know their discrete logarithm), the only way the above relation holds is when the total input and output amounts of A are equal, similarly for B $(x_1 = x_2, y_1 = y_2)$.

Remark 10. Although the Equation (4.3) shows that a transaction can contain multiple assets, the Incognito network only allows a single asset to be transfered within a transaction.

If the sender uses G_A to commit his tokens, an observer can easily figure out which token he is transacting by simply mapping all $\mathsf{tokenID}$s and their hashed values. To prevent this from happening, the sender can replace G_A with a blinded asset tag calculated as follows

$$\hat{G}_A = G_A + r_A H, \tag{4.4}$$

for some blinder $r_A \in \{0, 1, \ldots, q\}$.

Now, the commitment for value x of the asset A becomes

$$\begin{aligned}\mathsf{Com}_{\hat{G}_A}(x; r) = x\hat{G}_A + rH &= x(G_A + r_A H) + rH \\ &= xG_A + (r + xr_A)H \\ &= xG_A + \hat{r}H \\ &= \mathsf{Com}_{G_A}(x; \hat{r}),\end{aligned} \tag{4.5}$$

where $\hat{r} = r + xr_A \in \{0, 1, \ldots, q\}$ is also a random scalar.

It is easy to observe that the commitment for value x with the blinded asset tag $\hat{G}_A$ is also a commitment for the same value with the asset tag G_A.

4.2 Generating blinded asset tags

Suppose that Alice wants to send some of token A to Bob, and uses the blinded asset tag to prevent anyone other than herself and Bob from knowing which token is transacted. If she generates the blinded asset tag as in Equation (4.4) using a randomly chosen r_A, Bob cannot recover the original tokenID, and thus does not know which token Alice has transferred to him. To address this problem, we adopt the same solution as in generating one-time addresses (Section 3.4). To blind the asset tag, Alice proceeds as follows.

1. Alice computes the blinder $r_A = \mathcal{H}_s(\mathsf{SS}, \text{“}\mathsf{assettag}\text{”})$, where SS is generated as in Section 3.4.
2. Alice computes the blinded asset tag as $\hat{G}_A = G_A + r_A G$.

On Bob's side, after receiving $\hat{G}_A$ and R_{ota}, he computes the original tokenID as follows.

1. He computes the blinding factor $r'_A = \mathcal{H}_s(\mathsf{SS}, \text{“}\mathsf{assettag}\text{”})$.
2. He computes the clear asset tag as $G_A = \hat{G}_A - r'_A G$.

Remark 11. In general, one can use separate key pairs for OTAs and asset tags. However, this will increase the size of payment addresses, as well as make it more difficult to manage these keys. Therefore, an observer, given the $\mathsf{k_{ota}}$ of a user, can detect which output coins belong to him as well as identify their $\mathsf{tokenIDs}$.

Remark 12. In non-privacy transactions, the clear asset tag G_A will be used to commit token outputs.

4.3 Asset surjection proofs (ASPs)

It is easy to observe that the commitment for value x with the blinded asset tag $\hat{G}_A$ is also a commitment for the same value with the asset tag G_A.

However, it is possible to introduce a negative amount of asset type by computing $\hat{G}_A = -G_A + r_A H$. In this case, $\mathsf{Com}_{\hat{G}_A}(x; r) = \mathsf{Com}_{G_A}(-x; \hat{r})$, for some blinding factors $r, \hat{r}' \in \{0, 1, \ldots, q\}$. What this means is that $\mathsf{Com}_{\hat{G}_A}(x; r)$ becomes a commitment to a negative amount for the token A.
To prevent this scenario from happening, an asset surjection proof (ASP) is introduced. ASPs are derived from the observation that if $\hat{G}_A$ and $\hat{G}'_A$ are two blinded asset tags generated from G_A, the following should hold

$$\hat{G}_A - \hat{G}'_A = (G_A + r_A H) - (G_A + r'_A H) = 0G_A + (r_A - r'_A)H.$$

This says that $\hat{G}_A - \hat{G}'_A$ is also a commitment to $\mathsf{0}$ (with respect to G_A). An ASP proves that the difference between the asset tags in the inputs and outputs of a transaction is a commitment of $\mathsf{0}$, concerning the token being transacted. Based on this observation, we can use the ring signature (Section 3.5) to prove that for each transaction output, there exists at least a blinded asset tag from the transaction inputs so that both of them commit to the same asset tag.

Chapter 5

Transactions in Incognito

See Appendix for the description of each type of transactions.

5.1 Output Coins in Privacy Version 1

To avoid double-spending in version 1, each coin is associated with a unique serial number (SN). Whenever a coin is spent, its SN will be checked with the blockchain to make sure it has not been seen before. If it has been, then the blockchain will reject this transaction.

Each output coin of a transaction is associated with a so-called *serial number derivator* (snd). This snd is created by the sender of a transaction, and will be committed together with the amount of the outcoin. Equation 5.1 shows how a serial number is derived from an snd

$$\mathsf{SN} = \frac{1}{\mathsf{k} + \mathsf{snd}} G \in \mathcal{E}, \tag{5.1}$$

where k is the master private key. Here are a few things that we can infer from Equation 5.1.

- The calculation of SN is restricted to anyone who has the private key k. Hence, only the owner of a coin can check whether or not this coin has been spent.
- The knowledge of snd and SN is not sufficient to recover the private key k.
- For an snd and a private key k, the corresponding serial number is unique.
- For two different pairs of $(\mathsf{k}_1, \mathsf{snd}_1)$ and $(\mathsf{k}_2, \mathsf{snd}_2)$, their serial numbers may be the same (when $\mathsf{k}_1 + \mathsf{snd}_1 = \mathsf{k}_2 + \mathsf{snd}_2$. However, the probability is negligible.

Observe that Equation 5.1 can be rewritten as $(\mathsf{k} + \mathsf{snd})\mathsf{SN} = G$. Hence, we can employ the sigma protocol [32] to prove that SN has been correctly generated, by proving that we know the discrete log of G with respect to SN (details are shown in Algorithm 5.1). An observer uses Algorithm 8 to check the validity of this proof.

Why it works

- The first check is to ensure that the public key is correctly generated from the private key

$$\begin{aligned} zG &= (c \cdot \mathsf{k} + r)G \\ &= c \cdot (\mathsf{k} \cdot G) + rG \\ &= c \cdot \mathsf{K} + T_1. \end{aligned}$$

Algorithm 7 SNNoPrivacy proving algorithm

```
1: procedure SNNoPrivacyProve($\mathsf{k}, \mathsf{K}, \mathsf{snd}, \mathsf{SN}$)
2:     $r \xleftarrow{\$} \{0, 1, \ldots, q\}$
3:     $T_1 \leftarrow rG$
4:     $T_2 \leftarrow r \cdot \mathsf{SN}$
5:     $c \leftarrow \mathcal{H}_s(G, \mathsf{K}, \mathsf{SN}, T_1, T_2)$        ▷ generate the challenge
6:     $z \leftarrow c \cdot \mathsf{k} + r$
7:     $\mathsf{stmt} \leftarrow (\mathsf{K}, \mathsf{snd}, \mathsf{SN})$
8:     return $\Pi_{\mathsf{sn}} = (T_1, T_2, z)$
```

Algorithm 8 SNNoPrivacy verification algorithm

```
1: procedure SNNoPrivacyVerify($\mathsf{stmt}, \Pi_{\mathsf{sn}}$)
2:     $(T_1, T_2, z) \leftarrow \Pi_{\mathsf{sn}}$
3:     $(\mathsf{K}, \mathsf{snd}, \mathsf{SN}) \leftarrow \mathsf{stmt}$
4:     $c \leftarrow \mathcal{H}_s(G, \mathsf{K}, \mathsf{SN}, T_1, T_2)$        ▷ generate the challenge
5:     return $zG \overset{?}{=} c \cdot \mathsf{K} + T_1$ and $(z + c \cdot \mathsf{snd}) \cdot \mathsf{SN} \overset{?}{=} cG + T_2$
```

- The second is to ensure that the serial number is generated following the Equation 5.1

$$\begin{aligned}(z + c \cdot \mathsf{snd}) \cdot \mathsf{SN} &= (c\mathsf{k} + r + c \cdot \mathsf{snd}) \cdot \mathsf{SN} \\ &= c(\mathsf{k} + \mathsf{snd}) \cdot \mathsf{SN} + r \cdot \mathsf{SN} \\ &= c \cdot G + T_2.\end{aligned}$$

Remark 13. Algorithms 7 and 8 only show how the proving and verification processes work when the public key is known. This only happens in non-privacy transactions. In privacy transactions, only a commitment of the private key is public, therefore, the proving and verification algorithms will be slightly different.

5.2 Conversion Transactions

To be able to use UTXOs from privacy version 1, we need a mechanism to convert them to version 2. This mechanism is implemented as a conversion transaction. A conversion transaction contains the following components (Figure 5.1).

- sig. The signature signed by the Schnorr signature scheme as described in Section 2.4.
- publicKey. Public key of the transaction.
- fee. Transaction fee.
- info. Memo for the transaction (optional).
- paymentProof. The proof for the validity of the transaction.
  - inputCoins. List of input coins of the transaction. Input coins will not be encrypted or mixed with any decoys.

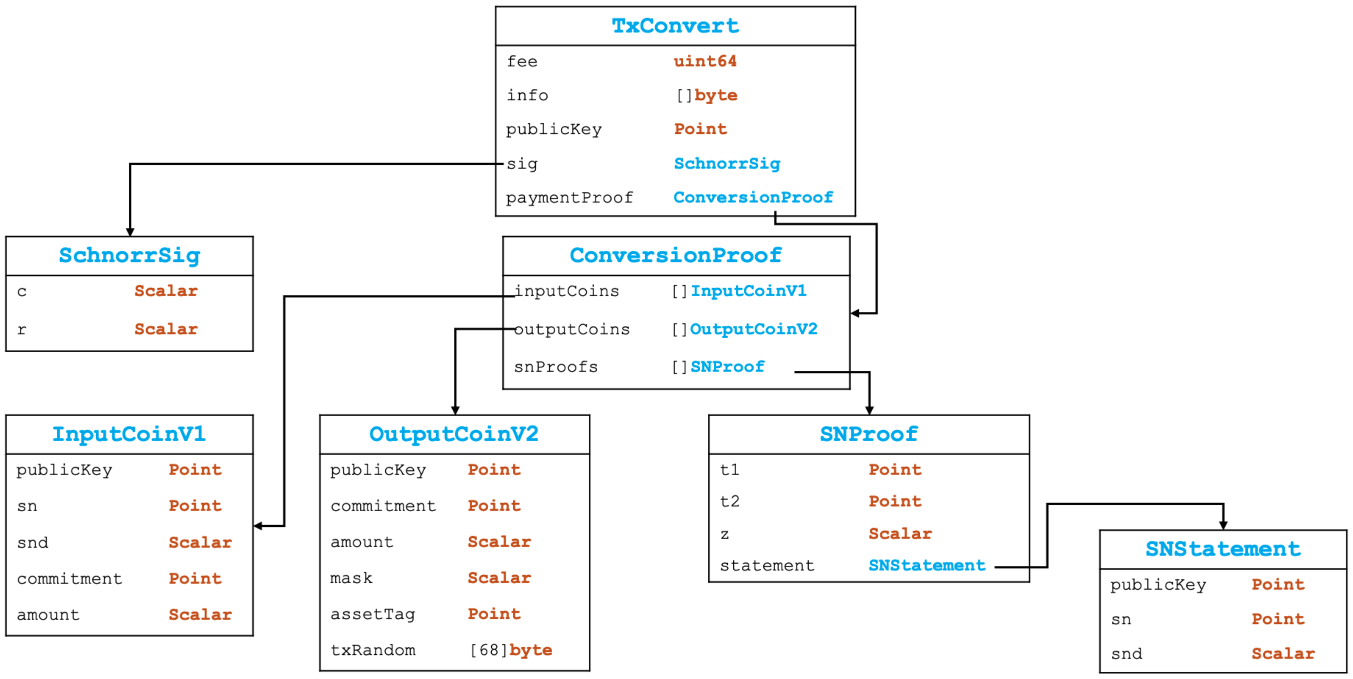

Figure 5.1: Conversion transaction

   - outputCoins. List of output coins of the transaction. In this transaction, output coin amounts and asset tags will not be encrypted. However, one-time addresses are still applied.
   - snProofs. Each element of snProofs is a proof of the correct calculation of the serial number in each input coin (Section 5.1).

5.3 Transactions Version 2

Transactions in version 2 can be privacy or non-privacy. Privacy is the default mode of a transaction version 2.

- sig. The signature signed by the MLSAG signature scheme as described in Section 2.5.
- sigPubKey. The public key for MLSAG signatures.
   - The sigPubKey is the ring $\mathcal{R}$ in the MLSAG scheme. As each output coin in Incognito is indexed, the sigPubKey of a transaction only consists of a ring of indices.
- fee. Transaction fee.
- info. Memo for the transaction (optional).
- paymentProof. The proof for the validity of the transaction. In a transaction version 2, ProofV2 is used.
   - inputCoins. List of input coins of the transaction. The input coin of every transaction in version 2 will only contain the key image required to check for double-spending.

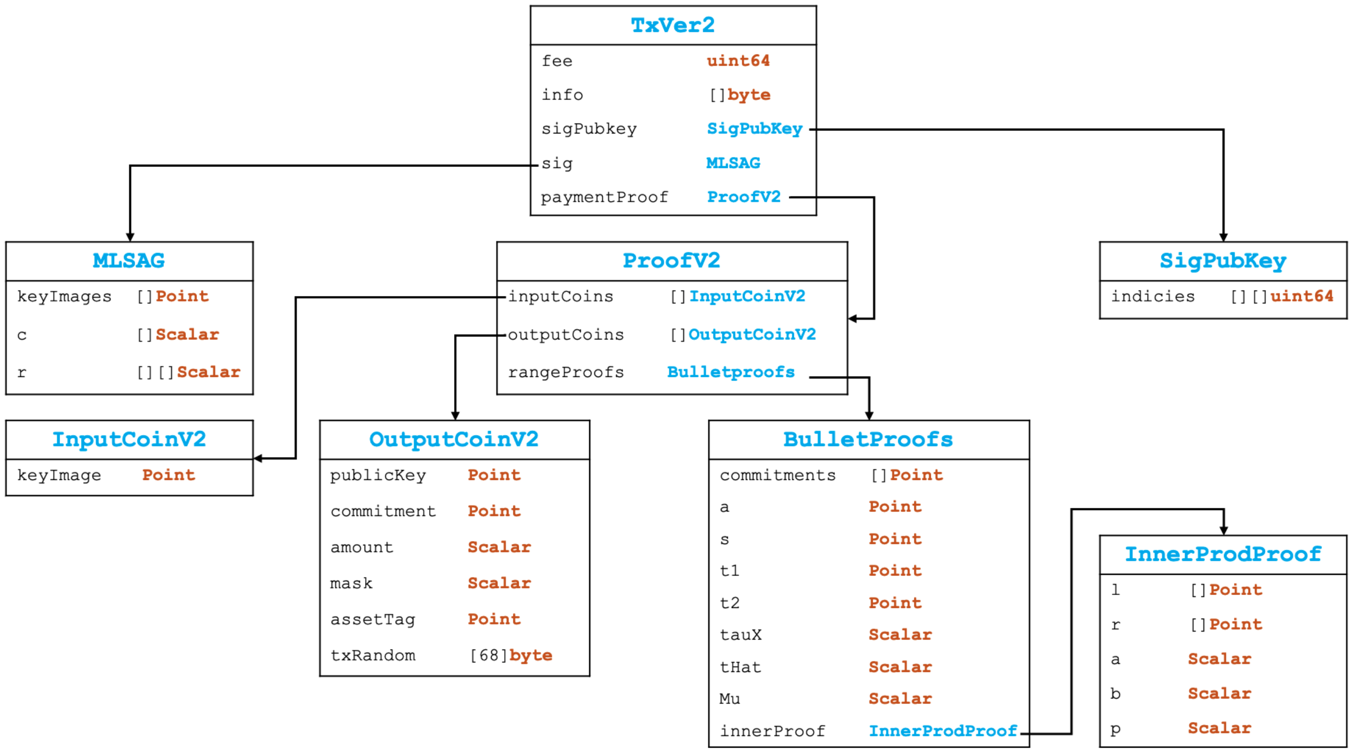

Figure 5.2: Transaction version 2

- outputCoins. List of output coins of the transaction. In this transaction, if privacy mode is on, output coin amounts and asset tags will be concealed. Otherwise, they are not. One-time addresses are still applied.
- rangeProofs. The ZKRP proof for proving that output amounts are not inflationary (Section 3.3.4).

5.4 Token Transactions Version 2

As shown in Figure 5.3, a token transaction basically consists of two smaller transactions (or *sub-transactions*): one for paying the transaction fee (*fee sub-transaction*), and one for transferring tokens (*token sub-transaction*). Each sub-transaction acts as a transaction version 2 (Section 5.3). Unlike the previous version, privacy verison 2 only supports transaction fees in PRV.

The signature of the fee sub-transaction is considered the signature of the whole transaction. The constraint of these two sub-transactions is guaranteed by taking the hash of the sub-transaction transferring tokens as a part of the message being signed of the fee sub-transaction.
Following are some of the attributes of a token transaction.

- TxTokenDataVer2. The detail of the token being transferred.
  - tokenID. ID of the transacted token.
  - tokenName. Name of the transacted token.
  - sig. The MLSAG signature of the token sub-transaction.
  - sigPubKey. The public key for verifying the signature of this sub-transaction.

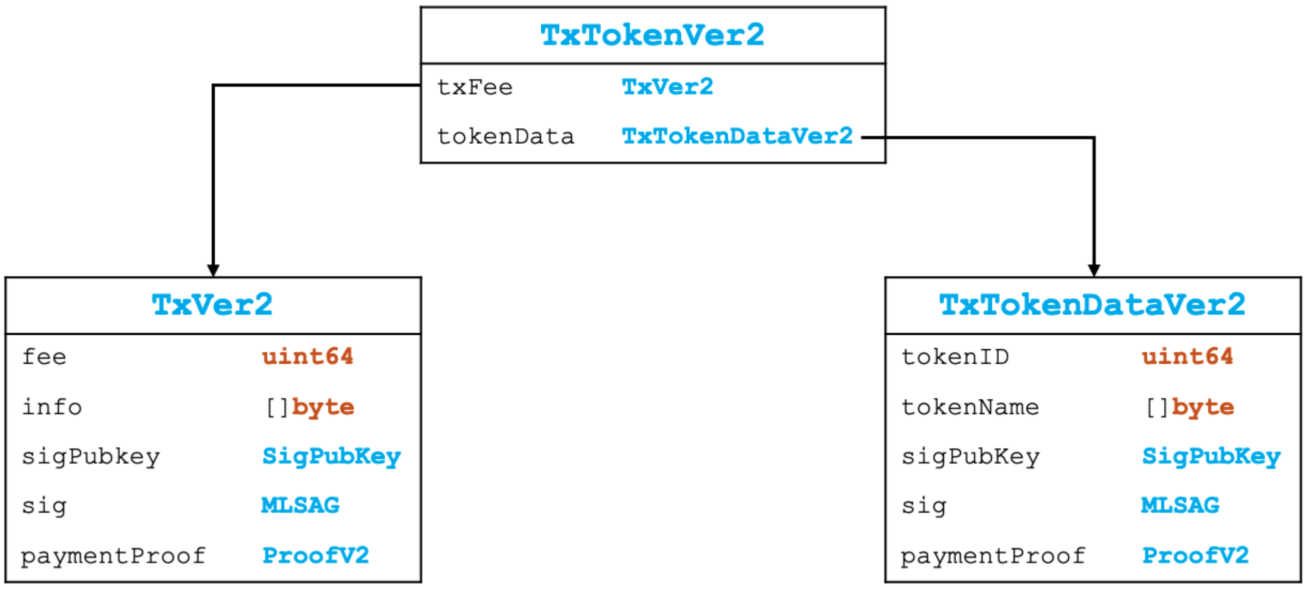

Figure 5.3: Token transaction version 2

  - paymentProof. The proof for the validity of transferring tokens.

- TxFee. The sub-transaction for paying the transaction fee.
  - sig. The MLSAG signature for txFee. It is also the signature of the whole signature. Notice that the hash of the token sub-transaction is a part of the message corresponding to this signature.
  - sigPubKey. The public key for verifying the signature of this sub-transaction.
  - fee. Transaction fee in PRV.
  - paymentProof. The proof for the validity of paying the transaction fee.

5.5 Token Conversion Transactions

A token conversion transaction is a token transaction (Section 5.4) with the token sub-transaction being a conversion transaction (Section 5.2).

Bibliography

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

[2] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[3] Nicolas Van Saberhagen. Cryptonote v 2.0, 2013.

[4] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[5] Andrew Poelstra. Mimblewimble. 2016.

[6] Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Australasian Conference on Information Security and Privacy*, pages 325–335. Springer, 2004.

[7] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[8] David Chaum and Eugène Van Heyst. Group signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 257–265. Springer, 1991.

[9] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *International Conference on Financial Cryptography and Data Security*, pages 43–63. Springer, 2018.

[10] Robby Houben and Alexander Snyers. Cryptocurrencies and blockchain. *Bruxelles: European Parliament*, 2018.

[11] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[12] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 56–73. Springer, 2004.

[13] Kurt M Alonso. Zero to monero, 2020.

[14] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted edwards curves. In *International Conference on Cryptology in Africa*, pages 389–405. Springer, 2008.

[15] Ivan Bjerre Damgård. Practical and provably secure release of a secret and exchange of signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 200–217. Springer, 1993.

[16] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Annual International Cryptology Conference*, pages 16–30. Springer, 1997.

[17] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 431–444. Springer, 2000.

[18] Helger Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 398–415. Springer, 2003.

[19] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *International Conference on Applied Cryptography and Network Security*, pages 467–482. Springer, 2005.

[20] Jan Camenisch, Rafik Chaabouni, et al. Efficient protocols for set membership and range proofs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 234–252. Springer, 2008.

[21] Berry Schoenmakers. Some efficient zeroknowledge proof techniques. In *Workshop on cryptographic protocols*, 2001.

[22] Sébastien Canard, Iwen Coisel, Amandine Jambert, and Jacques Traoré. New results for the practical use of range proofs. In *European Public Key Infrastructure Workshop*, pages 47–64. Springer, 2013.

[23] Helger Lipmaa, N Asokan, and Valtteri Niemi. Secure vickrey auctions without threshold trust. In *International Conference on Financial Cryptography*, pages 87–101. Springer, 2002.

[24] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[25] Whitfield Diffie. New direction in cryptography. *IEEE Trans. Inform. Theory*, 22:472–492, 1976.

[26] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[27] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[28] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[29] Ralph C Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[30] Andrew Miller, Malte Möser, Kevin Lee, and Arvind Narayanan. An empirical analysis of linkability in the monero blockchain. *arXiv preprint arXiv:1704.04299*, 2017.

[31] D. A. Wijaya, J. Liu, R. Steinfeld, and D. Liu. Monero ring attack: Recreating zero mixin transaction effect. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1196–1201, 2018.

[32] Ivan Damgård. On σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, page 84, 2002.

Appendix

A Conversion Transaction Details

```
{
  "BlockHash": "dda45ce3202de228b1e776b98848a8e5ec77702fd14af0ae3084fa331eb742ef",
  "BlockHeight": 17,
  "TxSize": 2,
  "Index": 0,
  "ShardID": 0,
  "Hash": "1db943665947ecf0ebb7675607bb21b7f370dcee6ada650aa99f7721ef355e33",
  "Version": 2,
  "Type": "cv",
  "LockTime": "2020-12-25T14:42:02",
  "Fee": 4,
  "Image": "",
  "IsPrivacy": false,
  "Proof":
   ↪  "/wGuIOcVUs3GaQN7h5gTuXoxDZNd4msnsSC6KckgAnYni7oAIP/pG18cvHrBukFBBwURfewdER
   ↪  gGp5L27mPPXg+Wa+hvICgWaAKnsz1WSdSICOsLruO0ReQkbDA7F89vYeAjYq8OIKxZxkDZJM+94
   ↪  y2u93XeRexRWHc2DZMDYlmT6BbMAPf/IHoLApDa+yphVIfh+9VvHPm1mppxjbwcG4lOXjwDJjgM
   ↪  B1jRXhdg+QcAAQEPAgAguBrjmdgp6tB3eR4HVLISWh1q0JqzzOmjNosQmVbaqigg9wg9eBgO38W
   ↪  3e5be/zZpf6jhyBblUzQ2Gus1MgoFylwAIAgariYzD+xMI6yKun6+O5T4GyErrL7W7ZQDNOoAjX
   ↪  EBIM8AGNwnZEJWJq4Y01H5/ptE+b80FLXC3oCrhUHl+2EERLArD8F0fOKLxD2+Sgi7kqrChVPj8
   ↪  L4hmulcIOnL9U5SAAAABY2K6hoBQ250GskNda/kj2zDMABM9oBIekyy6TK10omSIKv4Ui6s52p+
   ↪  f8SFmG3AybfxpF+juxDAn2HzNlhJsisIIAP5YBde0VgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ↪  AAAHArFnGQNkkz73jLa73dd5F7FFYdzYNkwNiWZPoFswA9//nFVLNxmkDe4eYE7l6MQ2TXeJrJ7
   ↪  EguinJIAJ2J4u6ACgWaAKnsz1WSdSICOsLruO0ReQkbDA7F89vYeAjYq8Oop5xSiVmOS+bdouOE
   ↪  NUMtPhgq/oNdtuQgSXYKkgaFXhacjb4VHpFXF5eTzARGHg/FWC4ZarZHHicU5Fuj0tbILyffo+Y
   ↪  rzZ4bMrmZmSiuew143fBMMGVogOuKEU+lOMA",
  "ProofDetail": {
    "InputCoins": [
      {
        "Version": 1,
        "PublicKey": "12kmiLFDSxaezVQe32Ze7D9TFm9TLs1Rx5cgh4ix5Eh4hZrvtXB",
        "Commitment": "12whtkAdmWVdHzsciUZ3dZRoMvnNLEQi1vCPRJfLQGuip5jQpVt",
        "SNDerivator": "1JeyxVLTDLACjFeEboXJpmYsXqfyNzgSMjStnwZJt47BD5oESA",
        "KeyImage": "12JuU5SLPBhdixUP32TGda4U5sMqcTzaHytTUiBvhYaL4t8HRgT",
        "Randomness": "1vkRe9xLcdieYgk4uJPbSQ6EriFXm4XszHJNPttKaxegHx9Tdm",
        "Value": 24999999999899911,
        "Info": "",
        "CoinDetailsEncrypted": ""
      }
    ],
    "OutputCoins": [
      {
        "Version": 2,
        "Index": 0,
        "Info": "",
        "PublicKey": "12Q5iXjwWAVURCpr5ckwtBBuovPJCxvRjWyKUhXtJFSxX8H9Aym",
        "Commitment": "12so78Y9bM71Ufdww4TNkBGb8HLaTj9fDxnhLAvjJUjFMPUt85x",
        "KeyImage": "",
        "TxRandom":
         ↪  "13bpNh5asJqZLzdxvXeMPGJqZmczmP9mZgkX7o5yGrQA8WhJdvT2zEK9kEXCiW99Caf
         ↪  nbzo8egknxVp86gPWpW8Q2SZ8njk6EAVu",
        "Value": "24999999999899907",
        "Randomness": "12Jjk6GJsWYnWX3Q6Yd4s5sA5D5L74oggPGERLEbLsh3H3uqLav",
        "AssetTag": ""
      }
    ]
  },
  "InputCoinPubKey": "12kmiLFDSxaezVQe32Ze7D9TFm9TLs1Rx5cgh4ix5Eh4hZrvtXB",
  "SigPubKey": "[]",
  "Sig": "1WRVgdt96ZgJhJnYJDDPeMoMkEc1wHwfPRez6sr4BprzctvPWztMrAYPB95zKTfAXxKcLaGsh
   ↪  UszutMbcPW1TXZhyZyi3Z",
  "Metadata": "",
  "CustomTokenData": "",
  "PrivacyCustomTokenID": "",
  "PrivacyCustomTokenName": "",
  "PrivacyCustomTokenSymbol": "",
  "PrivacyCustomTokenData": "",
  "PrivacyCustomTokenProofDetail": {
    "InputCoins": null,
    "OutputCoins": null
  },
  "PrivacyCustomTokenIsPrivacy": false,
  "PrivacyCustomTokenFee": 0,
  "IsInMempool": false,
  "IsInBlock": true,
  "Info": ""
}
```

Listing 1: An example of a conversion transaction

- BlockHash. The blockID that contains this transaction.

- BlockHeight. The block height at which the transaction was generated.
- TxSize. Size of the transaction in KB.
- Index. The index of the transaction in the block.
- ShardID. The original shard where the transaction comes from.
- Hash. Transaction hash.
- Version. The version of this transaction (the current version is 2).
- Type. Transaction type: n - normal transaction; tp - token transaction; s - salary transaction; rs - return staking transaction; cv - conversion transaction; tcv - token conversion transaction.
- LockTime. The time at which this transaction was created.
- Fee. Transaction fee in PRV
- IsPrivacy. Indicator of whether this transaction is privacy or non-privacy. Conversion transactions are non-privacy transactions.
- Proof. The payment proof of this transaction, encoded in base64.
- ProofDetail. The detail of input coins and output coins of the transaction.
- InputCoinPubKey. The public key of all input coins.
- SigPubKey. The public key for verifying the signature of the transaction. In this transaction (non-privacy), the SigPubKey is the same as the InputCoinPubKey.
- Sig. The signature of this transaction. In this case, the signature is a Schnorr signature.
- Metadata. Indicator of whether this transaction contains any metadata. Metadata transactions are special transactions in Incognito. They might be used to trade tokens, process minting or burning tokens, etc. The scope of this document will not consider metadata transactions.
- Lines 48-58. Detail of transaction tokens.
- IsInMempool. Indicator of whether this transaction is in the mempool.
- IsInBlock. Indicator of whether this transaction has been successfully added to the blockchain.
- Info. Memo of the transaction.

A.1 Input Coins

```
{
  "Version": 1,
  "PublicKey": "12kmiLFDSxaezVQe32Ze7D9TFm9TLs1Rx5cgh4ix5Eh4hZrvtXB",
  "Commitment": "12whtkAdmWVdHzsciUZ3dZRoMvnNLEQi1vCPRJfLQGuip5jQpVt",
  "SNDerivator": "1JeyxVLTDLACjFeEboXJpmYsXqfyNzgSMjStnwZJt47BD5oESA",
  "KeyImage": "12JuU5SLPBhdixUP32TGda4U5sMqcTzaHytTUiBvhYaL4t8HRgT",
  "Randomness": "1vkRe9xLcdieYgk4uJPbSQ6EriFXm4XszHJNPttKaxegHx9Tdm",
  "Value": 24999999999899911,
  "Info": "",
  "CoinDetailsEncrypted": ""
}
```

Listing 2: An input coin of a conversion transaction

- Version. The version of the input coin. In this transaction, the version is 1 since we are converting it into version 2.
- PublicKey. The public key of the input coin.
- Commitmet. The input coin commitment.
- SNDerivator. The input coin serial number derivator.
- KeyImage. The key image of the input coin, for checking double-spending. Because this coin is of version 1, this field is the serial number SN.
- Randomness. The blinding factor in the commitment.
- Value. The input coin amount.
- Info. Memo of the input coin.
- CoinDetailsEncrypted. The encrypted content of coins, used in privacy transaction. This transaction is a non-privacy transaction, therefore, this field is empty.

A.2 Output Coins

```
{
  "Version": 2,
  "Index": 0,
  "Info": "",
  "PublicKey": "12Q5iXjwWAVURCpr5ckwtBBuovPJCxvRjWyKUhXtJFSxX8H9Aym",
  "Commitment": "12so78Y9bM71Ufdww4TNkBGb8HLaTj9fDxnhLAvjJUjFMPUt85x",
  "KeyImage": "",
  "TxRandom": "13bpNh5asJqZLzdxvXeMPGJqZmczmP9mZgkX7o5yGrQA8WhJdvT2zEK9kEXCiW99Caf
   ↪ nbzo8egknxVp86gPWpW8Q2SZ8njk6EAVu",
  "Value": "24999999999899907",
  "Randomness": "12Jjk6GJsWYnWX3Q6Yd4s5sA5D5L74oggPGERLEbLsh3H3uqLav",
  "AssetTag": ""
}
```

Listing 3: An output coin of a conversion transaction

- Version. The version of the output coin. Since we are converting coins of version 1 into version 2, the output coin version is 2.
- PublicKey. The one-time address of the output coin.
- Commitmet. The output coin commitment.
- KeyImage. The key image of the output coin, for checking double-spending. Because this is an output coin, the KeyImage has not been calculated.
- TxRandom. Containing $R_{\mathsf{ota}}, \mathsf{idx}$ (Section 3.4) and R_{amt} (Section 3.3).
- Value. The output coin amount. In this transaction, this value is not encrypted.
- Randomness. The blinding factor in the commitment. In this transaction, this value is not encrypted.

A.3 Signatures

The signature of this transaction contains two scalars (c, r) generated by the Schnorr signature scheme (Section 2.4).

B Transaction Version 2 Details

```
{
  "BlockHash": "cf0de9efaaf9e1e29eea84c25e351470a18dac273f26c751d826ffefebad57fb",
  "BlockHeight": 20,
  "TxSize": 3,
  "Index": 0,
  "ShardID": 0,
  "Hash": "0b4f2768a2072b31dd3c9665d4b06c710168fac36f2193b114d221a63c5aa9b7",
  "Version": 2,
  "Type": "n",
  "LockTime": "2020-12-25T14:42:39",
  "Fee": 5,
  "Image": "",
  "IsPrivacy": true,
  "Proof":
    ↪ "AgAAA0ICQidW4luFEh/sfgRnFMonfJoT9cgSy54C3bSkAJfDumgJXMV3PanwmHiv2adrzLcbSC0
    ↪ vAuT+ONG3rRLmVfnoxXpluaNt3DCwgvP6F6hRglY5KKqPzhH187N8bXLggdE7jgG17HrPbsCbzXa
    ↪ 2wqRRLNS0XnC4a2hMOE2UZp16vAGJkbvBIaGKQQ+jVUQi7h/I51lNeA/EJBYQNli9YmObqi03FAM
    ↪ ih1rHqA5zA8WWK6satN9NzcLt8YKjMpOz/1Z1uQHsXaEM099ZxnUS6VOY2ljHbeW8Fqt471Pf62m
    ↪ UVw6MQT+U+eFGdSyyxhGkMcn7W9nMQWG15OjdRT0e7tEgB3ZBtR06h+WHtWfO9jFusUheapFfZ4L
    ↪ tuRRlC7t/KYsJB7OIE96YmlyJoa8ZBnB6Vul2aI4yKLilSonG62nXdtGZqfUoW+IPIGNATstfAOy
    ↪ EoQqyxFY+xLAis0STZhvQkBLQPQts7pzRTQFsQrHuVlAwt+pXqo1AJZLe3DKGzssvSjghppJC0ZG
    ↪ EnHR5EZ8o/HZKIj1o7K0+/94Fyu7b+7k1e8mGBA0gdot6YUkKNAuB7b30ZL1VRYjDpsfL1vTi4cF
    ↪ Uae1boWtcHs4l1zom/HzsSJbLYzA1XkgylHw38H5mmKmcc0RChtAqIHcRvhRQMNSW6r8qPRA61On
    ↪ A2W11hby8cnKLx2iY0b14cOUic0tthKYiONtfFDdFAK0yhWRzHBstEKDz/BE7hQGgkfACTkq6a35
    ↪ sHNb2JZ/XbGfECWVY+wK6LGaNFtDf8sNBeaiPg7E0o5Jgqz+rtHSQUbHNGpTtNp1w4z9PySLYk9D
    ↪ zsBUrZCbkjMgGo4xPpiqO/yaCU5zwvmgfnU/MGj9n9PIFIHDKHHe0ot4cPCWtdOYpDpXfxH4aNE5
    ↪ ZD8rPlYkeEmJC48e6m1gZH0VUAGlXlHbWCIOcvyKOU5K1gpCGy0k03cah7Pg9dBBHagl8tU4USGp
    ↪ TlI409hhhZO/tWU1J7rBbHac9lFU/fWW+2fcLxrW3PBbhDpTadKSFhhh0AkQcXpj40vLxsItC4cz
    ↪ haz1I1P2UBN8C8OC67WuC3qciFdlKkxKWw16pr7I7uTLE89MVQ/1e0VUBjwIAAAAg2B8HxMnfrTX
    ↪ j23YVBM3/6D/jXwL4hwh3tC1x2uDlhykAAEQBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    ↪ AAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAAAA
    ↪ AAAAAAAAAAAAAAAAAAAAAAs8CACCAaJWPWLyx0yWmNekcl+kf59jMGZnQKvgasYjqPLSa0SBCJ1b
    ↪ iW4USH+x+BGcUyid8mhP1yBLLngLdtKQAl8O6aAAAAETPZNV4X8d6+qdg+aQaQ/6Lvj1Bl+HCbh0
    ↪ QBwmsrDYl/wAAAAFOMMB+995HTlR/MxsBbpAeCU75005jsX/WQAtvgEqJhiDiW+J0G5vr++W+s+L
    ↪ smmDadtr08lG4vtEpIh8ebxc5BiCPYYNfVanA9LiB8JctHlNmiTouG50BQ8bk5IF74yOAAADPAgA
    ↪ gJQDHcA3Q7/wRLXW/sLabNb1nEESklwhqFt5NmAGLLJAgCVzFdz2p8Jh4r9mna8y3G0gtLwLk/jj
    ↪ Rt60S5lX56MUAAABEQlPIJy22Na/2ktkIsyMc62h7D0xB8RAQjSuXo+9Rh+cAAAACYTkfkVrYAbf
    ↪ O2ZLCdtZINHwh+c+mIYm6qnDmUXye0DggeQI+ZaBhZaFBKsvmTK4tpXTUlCMA75LpR2NypW5r6ws
    ↪ g+eHXjvTONLkw+quHIDH0bINnldfPPIxLl8bA/vh3ZwMA",
  "ProofDetail": {
    "InputCoins": [
      {
        "Version": 2,
        "Index": 0,
        "Info": "",
        "PublicKey": "",
        "Commitment": "",
        "KeyImage": "12eBXXUxbdCwZ23E8GARJRPkp3wr4SA4kAGSbxHDzd7CyWUYWTE",
        "TxRandom":
          ↪ "1reuV8KE5Z6G6fqumea2gkgCuBK8kYcishLv5qfdmjBYMvbp7jUxknSxUinjM93mWJQCF
          ↪ jpsaHLa6dLQUoroJiNZ1YGB1NcvVe",
        "Value": "0",
        "Randomness": "",
        "AssetTag": ""
      }
    ],
    "OutputCoins": [
      {
        "Version": 2,
        "Index": 0,
          "Info": "",
          "PublicKey": "1yZ2NRcjFDkkTNrNnxYTYQbBMwZy14CuTPKELeGFmqb4vihAjQ",
          "Commitment": "1W8onjo1G6MEp9DueuQFQkcCWfGTvtgVC5RnuMuPjzazJmL7fX",
          "KeyImage": "",
          "TxRandom":
           ↪  "144YqfJr6oskEhpqZHKx9zQg2TrEDUAimQg7Hs1ER7ukwy35SW8sLe8ooYHCtX1Sicz7z
           ↪  4LQE5XbUYvtoGSUCGYWBVBFLhxKa6TT",
          "Value": "0",
          "Randomness": "12ih32Si21r3CYzWDpBN5F8iYz7Ds7KgsVCMDoTBnCdjqSVdYQ7",
          "AssetTag": ""
        },
        {
          "Version": 2,
          "Index": 0,
          "Info": "",
          "PublicKey": "1HJCCWBSciQE7EvRGXz5ULpFERazWHbrXsewKMRknLZdRz9H8S",
          "Commitment": "1589gPBfFjsdptjSEB8viCuAmAYv4KaN7T1DLCNNUWjmxZ4jt6",
          "KeyImage": "",
          "TxRandom":
           ↪  "1ynPErfjA2RvaNPmhwgp5rJvbGwc8G2h3KjZ6JLmqJ57ASL7qscm4o74yhwyUWRWDdRAS
           ↪  VvRhCgyVsoFkimgccgfeGr9ntRv2E9",
          "Value": "0",
          "Randomness": "1vJ1NoVFp3USvVq67mzekLDez4q7cNXL6KS8Yes4FPCYiJsXKe",
          "AssetTag": ""
        }
      ]
    },
    "InputCoinPubKey": "",
    "SigPubKey": "{\"Indexes\":[[0],[0],[1],[1],[1],[1],[0],[1]]}",
    "Sig":
     ↪  "128Htg7beMGAquhuP24HX57CxYFmhnXPTd1fCiJ3SujbLJG1qyMjMJkxZFDLBM6WEdfRpdEGvB1
     ↪  2m4qJBXgEGpATutUq2MNDCjXQ2fbJ29Fp5jUuCRrt5hfSdyrDGUGTBtt395EoYecc3VfjP2xtDz7
     ↪  4idLaaW4pV2xEaHF857wrT7ySoaNUAhiwAXuegwdEEmcwhFjVbkectX9gy6cazqNMY6Sv4KSbCT8
     ↪  8prC3RaTrkFyRKL9p5M9BMxpNyKK2PHjuj7uCgyryNx9v99rHbg1LhmSQ9P29kf22pNbG1ccPdvR
     ↪  FDwjVt3GgFQhfzAEYmq6ePAAoXx4dmKzxjj7bo68X42hdEZSBqev88kcmRyCsTWgi15iPLtb7TSZ
     ↪  Neao3FfpvBWyAN6hKFGrutgSkJSzch57j5Rtj6Edf8Fqj57XJCaDm9SSLnJy8ctWtFzachTqMei8
     ↪  RRsCSTaE83SZhkrYNKQkuCyyGm8evsi3Dh4MJkgjXPybFqHcWeEsYwnAhq6JCyhcPG47HhDXkEa6
     ↪  SQHsm4deWnmJfvipGxRip85TBwbHSnFwFwvU13RZbADghfMWriJbDq3wAu95At2UvCUqAxZQSit5
     ↪  CUsb1WaydUysFSiATTazbeAungo5soiu5Z4Pu7L3kn98pgyJrkLbQuapdJPTPw4DP5z7qsZczMPD
     ↪  AArbUoJei5B9ikEWVKwYoo6Q4WJ13yB8hDhcU9qesGir7EK1ci2JTy8wXpjtG6w6xNSwpUWVt",
    "Metadata": "",
    "CustomTokenData": "",
    "PrivacyCustomTokenID": "",
    "PrivacyCustomTokenName": "",
    "PrivacyCustomTokenSymbol": "",
    "PrivacyCustomTokenData": "",
    "PrivacyCustomTokenProofDetail": {
```

```
    "InputCoins": null,
    "OutputCoins": null
  },
  "PrivacyCustomTokenIsPrivacy": false,
  "PrivacyCustomTokenFee": 0,
  "IsInMempool": false,
  "IsInBlock": true,
  "Info": ""
}
```

Listing 4: An example of a transaction version 2

There are a lot in common between a conversion transaction and a transaction version 2. Here, we only spot some of the differences.

- IsPrivacy. The default mode of a transaction version 2 is privacy.
- Proof. The payment proof of this transaction, encoded in base64.
- ProofDetail. The detail of input coins and output coins of the transaction.
- InputCoinPubKey. The public key of all input coins. Since each input coin in a txver2 transaction has its own public key. This field is empty.
- SigPubKey. The ring $\mathcal{R}$ of indices in Section 2.5.
- Sig. The MLSAG signature encoded in base58.

B.1 Input Coins

```
{
  "Version": 2,
  "Index": 0,
  "Info": "",
  "PublicKey": "",
  "Commitment": "",
  "KeyImage": "12YCw1M99bK5gCGUHq21nXcrbnHoTeHXsb8ApDeB8iCmkgTx7UY",
  "TxRandom": "",
  "Value": "0",
  "Randomness": ""
}
```

Listing 5: An input coin of a txver2 transaction. Only the KeyImage of the input coin is shown.

B.2 Output Coins

```
{
  "Version": 2,
  "Index": 0,
  "Info": "",
  "PublicKey": "1SpNyoDGutwNMFhGVRNNGm9UNXkaXmQ1aJNzndTVNZQJe3CCse",
  "Commitment": "12tmg9ytDtiNCzkkQZKizuMzgC39ymyLdzkAvmon1NjbtYJCGc3",
  "KeyImage": "",
  "TxRandom":
↪   "12NNwJwMrxwHZTQSuNUZGZ6DRQR7EFKUARDLM5hHcijiCsiXiVt2rkiU9mzUXXkKjF4CwPb
↪   ipFAyECMWFLLFYtwagSTGACuTNm1d",
  "Value": "0",
  "Randomness": "1FAPuAEMA9ZaNTDfLMyTE1qLyrAJjtjBUWdWCCLrddvbvYQpp8"
}
```

Listing 6: An output coin of a txver2 transaction

- Version. The version of the output coin. Since we are converting coins of version 1 into version 2, the output coin version is 2.
- PublicKey. The one-time address of the output coin.
- Commitmet. The output coin commitment.
- KeyImage. The key image of the output coin, for checking double-spending. Because this is an output coin, the KeyImage has not been calculated.
- TxRandom. Containing $R_{\mathsf{ota}}, \mathsf{idx}$ (Section 3.4) and R_{amt} (Section 3.3).
- Value. The output coin amount. In this transaction, this value is encrypted.
- Randomness. The blinding factor in the commitment. In this transaction, this value is encrypted.

B.3 Signatures

A txver2 transaction uses the MLSAG signature scheme to sign the transaction.

- The SigPubKey contains a ring of indices indicating where the real input coin and the mixins are stored.
- The Sig is an MLSAG signatures containing $\sigma = (c_1, r_{1,1}, \ldots, r_{1,m} \ldots, r_{n,1}, \ldots, r_{n,m})$ as described in Section 2.5.