Beldex

Private and Decentralized Ecosystem

Version 3.1.1

May 13, 2022

Legal Disclaimer

This document does not constitute or imply a prospectus of any sort. No wording contained herein should be construed as a solicitation for investment. Accordingly, this whitepaper does not pertain in any way to an offering of securities in any jurisdiction worldwide whatsoever. Rather, this whitepaper constitutes a technical description of the functionality of the Beldex Coin and the deployment of a user-friendly platform that will allow users to trade in cryptocurrencies.

The purpose of the document is to present the Beldex exchange project and details about BDX only. There should not be a redistribution of this without further written permission of the BELDEX team or this document and the information contained herein may not be sent or addressed wholly or in part directly or indirectly to any person in countries restricted. Nothing in this white paper shall be deemed to offer a document of any sort of a solicitation for investment or not in any way to offer any securities in any jurisdiction. The project plan was considered carefully before preparing this document. Beldex makes no representations and gives no warranties or guarantees of whatever nature in respect of this document including but not limited to the completeness of any information, facts or opinions contained therein. The founders, advisors, partners, directors, employees, and agents of Beldex and itself cannot be held liable for the use of and reliance on the content in this document.

The information set forth in this White Paper is subject to change. The information set forth below may not be exhaustive of all information that Beldex represents and does not imply any elements of a contractual relationship. This White Paper may be modified to provide more detailed information at any time.

Abstract

A complete privacy-based ecosystem that anonymizes transactions, messages, and your online footprint by incentivizing network validators and creating an economic model that ensures sustainable development of the ecosystem. The private ecosystem consists of the Beldex blockchain, decentralized and anonymous messenger, browser, wallet, a privacy protocol, and the Beldex Bridge connecting to other ecosystems.

This whitepaper provides an overview of the Beldex ecosystem. Any future additions or changes to the ecosystem will be released in subsequent versions of this paper.

1.0 Why Beldex?

Privacy in communication and transactions were often left out as lofty necessities in the past, however, there is a great degree of awareness among individuals about their privacy as global digitization continues to expand. Blockchain is thought to solve this problem. The early adopters of crypto assumed that a blockchain transaction could not be traced back to their origins. However, this false perception was soon debunked when the lot of blockchain analysis firms emerged with solutions for tracing blockchain transactions. This also applies to Bitcoin transactions [1]. Most crypto transactions do not hide the amount of transactions, the sender or receiver addresses and their balances. Coins such as Monero and other private crypto attempted to solve the problem of privacy. Monero, especially, remains an anonymous cryptocurrency with a strong foundation in privacy [2]. But even privacy blockchains were met with issues of scalability in proof-of-work consensus [3]. Moreover, even if a coin does support privacy, it may not support a wallet or an ecosystem. This adds to the deanonymization of transactions. For example, if there are no decentralized wallets, the user of a privacy coin may have to rely on a centralized wallet to store their coins. An institution with sufficient information about the inputs and outputs of a wallet can perform chainalysis to find information pertaining to a transaction, also known as an EABE attack [4]. Beldex proposes a unique, viable solution to the problems of privacy, scalability, and ecosystem.

2.0 Introducing Beldex Blockchain

Beldex is a mined proof-of-work coin with masternodes as a mining opportunity to the community. Originally a fork of Monero, Beldex has integrated PrivateSend privacy protocol from DASH and a few more key privacy features like ViewKey from ZEC to improve the original privacy, as well as its own configurable privacy technology.

BELDEX's core advantages,

● Untraceable Roots - Like Monero which uses RingCT, Beldex too uses RingCT network type but with a higher size of RingCT.

● Confidential Transactions - Transaction are completely anonymous and private.

● Airdrop - If an individual stakes a certain amount of crypto in the wallet they get a specific number of rewards in terms of the same coins.

2.1 Cryptonote Features

Full node architecture can be implemented with any cryptocurrency, however, Beldex utilizes Monero's source code as its base, since Monero offers greater privacy to transactions using the cryptonote protocol. The CryptoNote protocol proposed a combination of ring signatures, ringCT (confidential transactions), and stealth addresses [5]. Monero's implementation of the protocol offers greater privacy to CryptoNote transactions [6]. When interactions occur across the Beldex independent layers, for example, the second and the first, to reduce the risk due to time-based analysis, an exchange medium that underpins the ecosystem is provided. This means that, when

initiating a Flash transaction, users will retain the privacy guarantees offered by the Beldex chain and vice versa.

2.1.1 Untraceable Transactions

Untraceability and unlinkability are the two main features discussed in this section. The sender should not be required to trust the receiver or any other intermediary. Autonomy in transactions is a core feature of untraceable transactions in Beldex. Thus, each transactor independently produces cover traffic to mask their identity.

2.1.2 Definitions

Elliptic curve parameters

The fast scheme EdDSA is chosen as our base signature algorithm, which is developed and implemented by D.J. Bernstein et al. [19]. It is based on the elliptic curve discrete logarithm problem, which is similar to Bitcoin's ECDSA, so this model could also be applied to Bitcoin in future.

Common parameters are:

q: a prime number; $q = 2^{255} - 19$;

d: an element of F_q; $d = -121665/121666$;

E: an elliptic curve equation; $-x^2 + y^2 = 1 + dx^2y^2$;

G: a base point; $G = (x, -4/5)$;

l: a prime order of the base point; $l = 2^{252} + 27742317777372353535851937790883648493$;

Hs: a cryptographic hash function $\{0, 1\} * \rightarrow \mathbb{F}_q$;

Hp: a deterministic hash function $E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_q)$.

Terminology

Enhanced privacy requires a new terminology which should not be confused with Bitcoin entities.

private ec-key is a standard elliptic curve private key: a number $a \in [1, l - 1]$;

public ec-key is a standard elliptic curve public key: a point $A = aG$;

one-time keypair is a pair of private and public ec-keys;

private user key is a pair (a, b) of two different private ec-keys;

tracking key is a pair (a, B) of private and public ec-key (where $B = bG$ and $a \neq b$);

public user key is a pair (A, B) of two public ec-keys derived from (a, b);

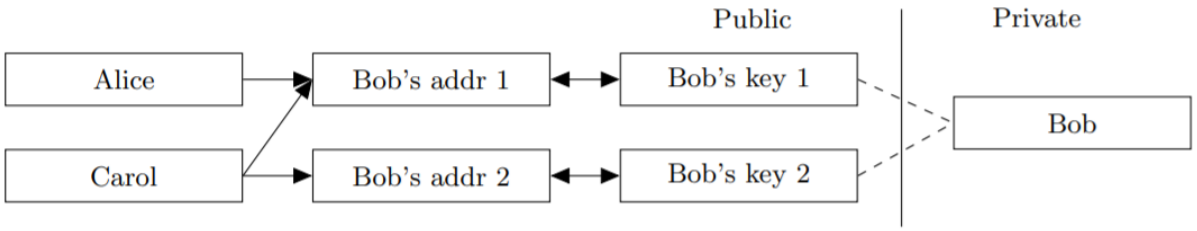
standard address is a representation of a public user key given into human friendly string with error correction;

truncated address is a representation of the second half (point *B*) of a public user key given into human friendly string with error correction.

The transaction structure remains similar to the structure in Bitcoin: every user can choose several independent incoming payments (transactions outputs), sign them with the corresponding private keys and send them to different destinations. Contrary to Bitcoin's model, where a user possesses unique private and public key, in the proposed model a sender generates a one-time public key based on the recipient's address and some random data. In this sense, an incoming transaction for the same recipient is sent to a one-time public key (not directly to a unique address) and only the recipient can recover the corresponding private part to redeem his funds (using his unique private key). The recipient can spend the funds using a ring signature, keeping his ownership and actual spending anonymous. The details of the protocol are explained in the next subsections.

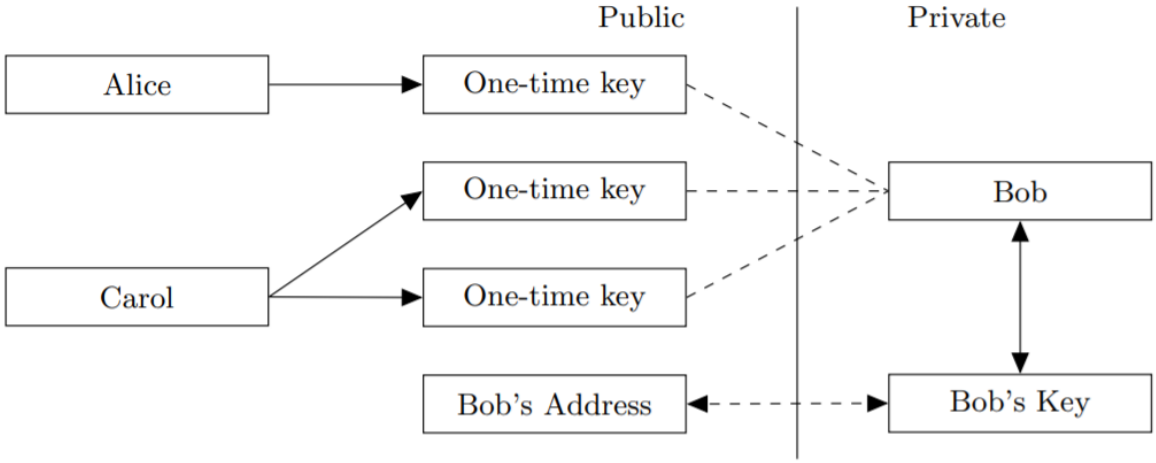
2.1.3 Unlinkable payments

Classic Bitcoin addresses, once being published, become unambiguous identifier for incoming payments, linking them together and tying to the recipient's pseudonyms. If someone wants to receive an "untied" transaction, he should convey his address to the sender by a private channel. If he wants to receive different transactions which cannot be proven to belong to the same owner he should generate all the different addresses and never publish them in his own pseudonym.



Traditional Bitcoin Keys/Transaction Model

We propose a solution allowing a user to publish a single address and receive unconditional unlinkable payments. The destination of each CryptoNote output (by default) is a public key, derived from recipient's address and sender's random data. The main advantage against Bitcoin is that every destination key is unique by default (unless the sender uses the same data for each of his transactions to the same recipient). Hence, there is no such issue as "address reuse" by design and no observer can determine if any transactions were sent to a specific address or link two addresses together.

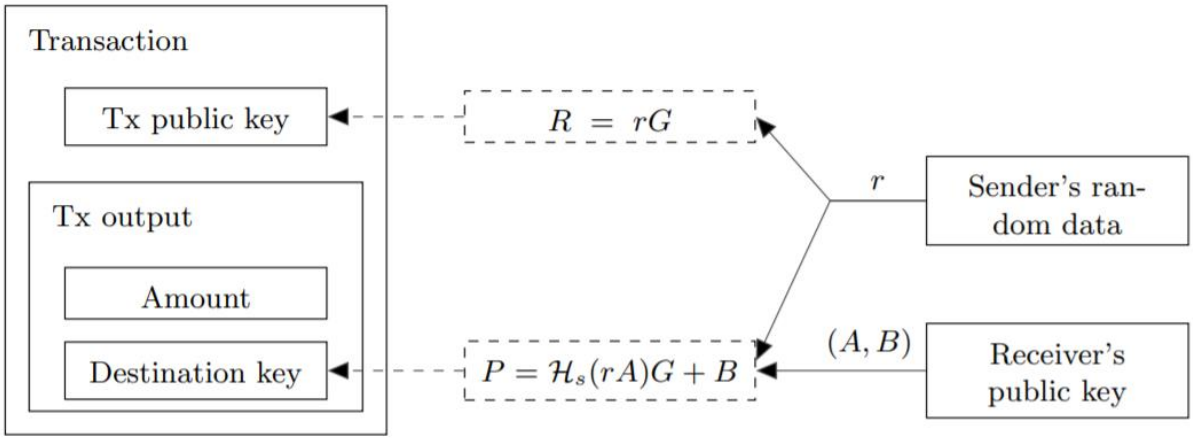


CryptoNote Keys/Transaction Model

First, the sender performs a Diffie-Hellman exchange to get a shared secret from his data and half of the recipient's address. Then he computes a one-time destination key, using the shared secret and the second half of the address. Two different ec-keys are required from the recipient for these two steps, so a standard CryptoNote address is nearly twice as large as a Bitcoin wallet address. The receiver also performs a Diffie-Hellman exchange to recover the corresponding secret key.

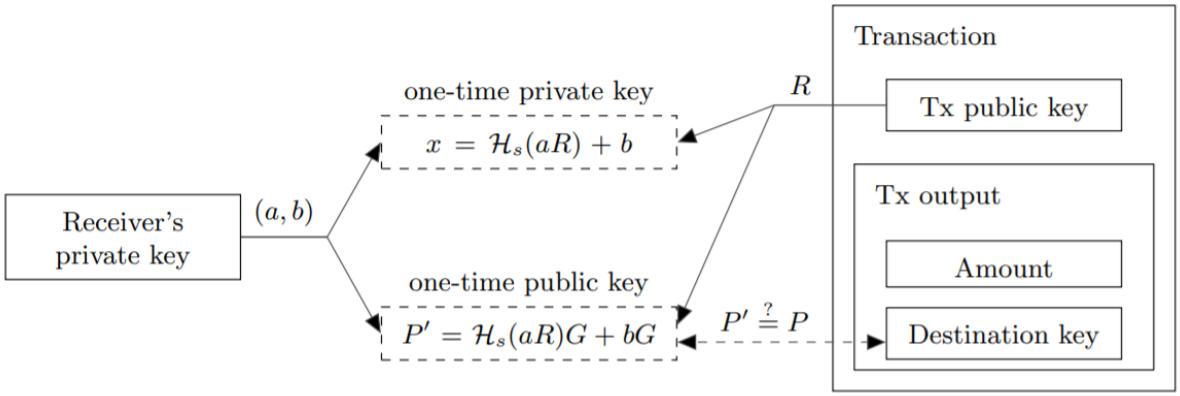
A standard transaction sequence goes as follows:

1. Alice wants to send a payment to Bob, who has published his standard address. She unpacks the address and gets Bob's public key (*A*, *B*).
2. Alice generates a random $r \in [1, l-1]$ and computes a one-time public key $P = H_s(rA)G + B$.
3. Alice uses P as a destination key for the output and also packs value $R = rG$ (as a part of the Diffie-Hellman exchange) somewhere into the transaction. Note that she can create other outputs with unique public keys: different recipients' keys (*Ai*, *Bi*) imply different *Pi* even with the same *r*.



Standard Transaction Structure

4. Alice sends the transaction.
5. Bob checks every passing transaction with his private key (*a*, *b*), and computes $P' = H_S(aR)G + B$. If Alice's transaction for with Bob as the recipient was among them, then $aR = arG = rA$ and $P' = P$.
6. Bob can recover the corresponding one-time private key: $x = H_S(aR) + b$, so as $P = xG$. He can spend this output at any time by signing a transaction with *x*.



Incoming Transaction Check

7. As a result Bob gets incoming payments, associated with one-time public keys which are unlinkable for a spectator. Some additional notes:

   • When Bob "recognizes" his transactions he practically uses only half of his private information: (*a, B*). This pair, also known as the tracking key, can be passed to a third party (Carol). Bob can delegate her processing of new transactions. Bob doesn't need to explicitly trust Carol, because she can't recover the one-time secret key *p* without Bob's full private key (*a, b*). This approach is useful when Bob lacks bandwidth or computation power (smartphones, hardware wallets etc.).

   • In case Alice wants to prove she sent a transaction to Bob's address she can either disclose *r* or use any kind of zero-knowledge protocol to prove she knows *r* (for example by signing the transaction with *r*).

   • If Bob wants to have an audit compatible address where all incoming transaction are linkable, he can either publish his tracking key or use a **truncated address**. That address represent only one public ec-key *B*, and the remaining part required by the protocol is derived from it as follows: $a = \mathcal{H}_s(B)$ and $A = \mathcal{H}_s(B)G$. In both cases every person is able to "recognize" all of Bob's incoming transaction, but, of course, none can spend the funds enclosed within them without the secret key *b*.

2.2 Ring Signatures

Ring signatures take many inputs from multiple sources and combine them with the original input from the sender. The original sender's information is concealed with a number of inputs. This masks the true history of outputs from a transaction. Beldex provides mandatory privacy, thus all Beldex transactions will employ Ring signatures and will have a fixed ring size 10 per transaction. For every unique transaction, there are nine other inputs in a ring transaction. Thus, the sender may not have to sign the transaction using their private key, but instead only the ring signature made of the private keys of ring participants.

Ring signatures are composed of a ring and a signature. Each signature is generated with a single private key and a set of unrelated public keys. Each ring is the set of public keys comprising the private key's public key, and the set of unrelated public keys. Somebody verifying a signature would not be able to tell which ring member corresponds to the private key that created it.

Ring signatures were originally called Group Signatures because they were thought of as a way to prove a signer belongs to a group, without necessarily identifying him. In the context of Beldex they will allow for unforgeable, signer-ambiguous transactions that leave currency flows largely untraceable.

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

Signer Ambiguity: An observer should be able to determine the signer must be a member of the ring (except with negligible probability), but not which member. Beldex uses this to obfuscate the origin of funds in each transaction.

Linkability If a private key is used to sign two different messages then the messages will become linked. As we will show, this property is used to prevent double-spending attacks in Beldex (except with negligible probability).

Unforgeability No attacker can forge a signature except with negligible probability. This is used to prevent theft of Beldex funds by those not in possession of the appropriate private keys.

2.2.1 One-time ring signatures

A protocol based on one-time ring signatures allows users to achieve unconditional unlinkability. Unfortunately, ordinary types of cryptographic signatures permit to trace transactions to their respective senders and receivers. Our solution to this deficiency lies in using a different signature type than those currently used in electronic cash systems.

We will first provide a general description of our algorithm with no explicit reference to electronic cash.

A one-time ring signature contains four algorithms: (**GEN, SIG, VER, LNK**):

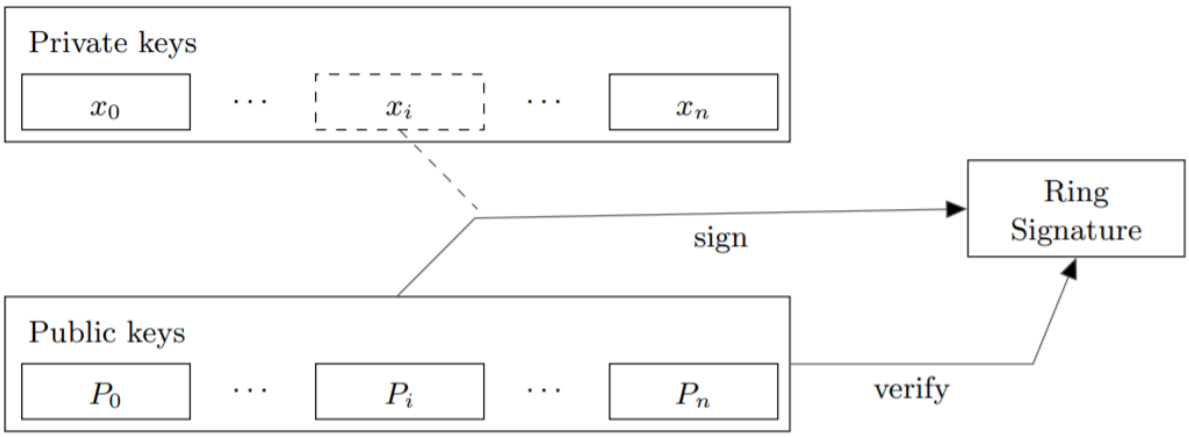
GEN: takes public parameters and outputs an ec-pair (P, x) and a public key I.

SIG: takes a message m, a set S' of public keys $\{P_i\}_{i \neq s}$, a pair (P_s, x_s) and outputs a signature σ and a set $S = S' \cup \{P_s\}$.

VER: takes a message m, a set S, a signature σ and outputs "true" or "false".

LNK: takes a set $I = \{I_i\}$, a signature σ and outputs "linked" or "indep".

The idea behind the protocol is fairly simple: a user produces a signature which can be checked by a set of public keys rather than a unique public key. The identity of the signer is indistinguishable from the other users whose public keys are in the set until the owner produces a second signature using the same keypair.



Ring Signature Anonymity

GEN: The signer picks a random secret key $x \in [1, l-1]$ and computes the corresponding public key $P = xG$. Additionally he computes another public key $I = xH_p(P)$ which we will call the "key image".

SIG: The signer generates a one-time ring signature with a non-interactive zero-knowledge proof using the techniques from [20]. He selects a random subset S' of n from the other users' public keys P_i, his own keypair (x, P) and key image I. Let $0 \leq s \leq n$ be signer's secret index in S (so that his public key is P_s).

He picks a random $\{q_i \mid i = 0 \ldots n\}$ and $\{\omega_i \mid i = 0 \ldots n, i \neq s\}$ from $(1 \ldots l)$ and applies the following *transformations*:

$$L_i = \begin{cases} q_i G, & \text{if } i = s \\ q_i G + w_i P_i, & \text{if } i \neq s \end{cases}$$
$$R_i = \begin{cases} q_i \mathcal{H}_p(P_i), & \text{if } i = s \\ q_i \mathcal{H}_p(P_i) + w_i I, & \text{if } i \neq s \end{cases}$$

The next step is getting the non-interactive challenge:

$$c = \mathcal{H}_s(m, L_1, \ldots, L_n, R_1, \ldots, R_n)$$

Finally the signer computes the response:

$$c_i = \begin{cases} w_i, & \text{if } i \neq s \\ c - \sum_{i=0}^{n} c_i \quad modl, & \text{if } i = s \end{cases}$$

$$r_i = \begin{cases} q_i, & \text{if } i \neq s \\ q_s - c_s x \quad modl, & \text{if } i = s \end{cases}$$

The resulting signature is $\sigma = (I, c_1, \ldots, c_n, r_1, \ldots, r_n)$.

VER: The verifier checks the signature by applying the inverse transformations:

$$\begin{cases} L'_i = r_i G + c_i P_i \\ R'_i = r_i \mathcal{H}_p(P_i) + c_i I \end{cases}$$

Finally, the verifier checks if $\sum_{i=0}^{n} c_i \stackrel{?}{=} H_s(m, L'_0, \ldots, L'_n, R'_0, \ldots, R'_n) \bmod l$
If this equality is correct, the verifier runs the algorithm **LNK**. Otherwise the verifier rejects the signature.

LNK: The verifier checks if I has been used in past signatures (these values are stored in the set I). Multiple uses imply that two signatures were produced under the same secret key.

The meaning of the protocol: by applying L-transformations the signer proves that he knows such x that at least one $Pi = xG$. To make this proof non-repeatable we introduce the key image as $I = x\mathcal{H}_p(P)$. The signer uses the same coefficients (r_i , c_i) to prove almost the same statement: he knows such x that at least one $\mathcal{H}_p(P_i) = I \cdot x^{-1}$.

If the mapping $x \rightarrow I$ is an injection:
1. Nobody can recover the public key from the key image and identify the signer;
2. The signer cannot make two signatures with different I's and the same x.

2.2.2 Linkable Spontaneous Anonymous Group (LSAG) signatures

Originally, group signature schemes required the system be set up, and in some cases managed, by a trusted person in order to prevent illegitimate signatures, and, in a few schemes, adjudicate disputes. Relying on a group secret is not desirable since it creates a disclosure risk that could undermine anonymity. Moreover, requiring coordination between group members (i.e. for setup and management) is not scalable beyond small groups or within companies.

Liu *et al.* presented a more interesting scheme in building on the work of Rivest *et al*. The authors detailed a group signature algorithm characterized by three properties: *anonymity, linkability, and spontaneity*. In other words, the owner of a private key could produce one anonymous signature by selecting any set of co-signers from a list of candidate public keys, without needing to collaborate with anyone.

Signature

Ring signature schemes are very Schnorr-like. In fact, our signature scheme can be considered a one-key ring signature. We get to two keys by, instead of defining r right away, generating a decoy r' and creating a new challenge to define r with.

Let $\boldsymbol{m}$ be the message to sign, $\mathcal{R} = \{K_1, K_2, ..., K_n\}$ a set of distinct public keys (a group/ring), and k_π the signer's private key corresponding to his public key $K_\pi \in \mathcal{R}$, where π is a secret index. Assume the existence of two hash functions, $\mathcal{H}_n$ and $\mathcal{H}_p$, mapping to integers from 1 to l,[5] and curve points in EC,[6,7] respectively.

1. Compute key image $\tilde{K} = k_\pi \mathcal{H}_p(\mathcal{R})$.

2. Generate random number $\alpha \in_R \mathbb{Z}_l$ and fake responses $r_i \in_R \mathbb{Z}_l$ for $i \in \{1, 2, ..., n\}$ but excluding $i = \pi$.

3. Calculate

$$c_{\pi+1} = \mathcal{H}_n(\mathcal{R}, \tilde{K}, \boldsymbol{m}, [\alpha G], [\alpha \mathcal{H}_p(\mathcal{R})])$$

4. For $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$ calculate, replacing $n + 1 \rightarrow 1$,

$$c_{i+1} = \mathcal{H}_n(\mathcal{R}, \tilde{K}, \boldsymbol{m}, [r_i G + c_i K_i], [r_i \mathcal{H}_p(\mathcal{R}) + c_i \tilde{K}])$$

5. Define the real response r_π such that $\alpha = r_\pi + c_\pi k_\pi \pmod{l}$.

The ring signature contains the signature $\sigma(\boldsymbol{m}) = (c_1, r_1, r_2, ..., r_n)$, the key image $\tilde{K}$, and the ring $\mathcal{R}$.

Verification

Verification means proving $\sigma(m)$ is a valid signature created by a private key corresponding to a public key in $\mathcal{R}$, and is done in the following manner:

1. Check $l\tilde{K} \stackrel{?}{=} 0$.

2. For $i = 1, 2, ..., n$ iteratively compute, replacing $n + 1 \rightarrow 1$,

$$c'_{i+1} = \mathcal{H}_n(\mathcal{R}, \tilde{K}, \boldsymbol{m}, [r_i G + c_i K_i], [r_i \mathcal{H}_p(\mathcal{R}) + c_i \tilde{K}])$$

3. If $c'_1 = c_1$ then the signature is valid. Note that c'_1 is the last term calculated.

We must check $l\tilde{K} \stackrel{?}{=} 0$ because it is possible to add an EC point from the subgroup of size h (the cofactor) to $\tilde{K}$ and, if all c_i are multiples of h (which we could achieve with automated trial and error using different α and r_i values), make h unlinked valid signatures using the same ring and signing key.[8] This is because an EC point multiplied by its subgroup's order is zero.[9]

To be clear, given some point K in the subgroup of order l, some point K^h with order h, and an integer c divisible by h:

$$c * (K + K^h) = cK + cK^h$$
$$= cK + 0$$

In this scheme we store (1+n) integers, have one EC key image, use n public keys, and verify with:
VBSM Variable-base scalar multiplications for some integer a, and point P: aP [1]
DVBA Double-variable-base addition for some integers a, b, and point B: $aG + bB$ [n]
VBA Variable-base additions for some integers a, b, and points A, B: $aA + bB$ [n]

Why it works

We can informally convince ourselves the algorithm works by going through an example. Consider ring $R = \{K_1, K_2, K_3\}$ with $k_\pi = k_2$. First the signature:

1. Create key image: $\tilde{K} = k_\pi \mathcal{H}_p(\mathcal{R})$

2. Generate random numbers: α, r_1, r_3

3. Seed the signature loop: $c_3 = \mathcal{H}_n$ (..., [αG], [α $\mathcal{H}_p(\mathcal{R})$])

4. Iterate: $c_1 = \mathcal{H}_n$ (..., [$r_3G + c_3K_3$], [$r_3 \mathcal{H}_p(\mathcal{R}) + c_3\tilde{K}$]) $c_2 = \mathcal{H}_n$ (..., [$r_1G + c_1K_1$], [$r_1 \mathcal{H}_p(\mathcal{R}) + c_1\tilde{K}$])

5. Close the loop by responding: $r_2 = \alpha - c_2k_2$ (mod l)

We can substitute α into c_3 to see where the word 'ring' comes from:

$c_3 = \mathcal{H}_n$ (..., [$(r_2 + c_2k_2)G$], [(r2 + c2k2)Hp(R)])

$c_3 = \mathcal{H}_n$ (..., [$r_2G + c_2K_2$], [$r_2 \mathcal{H}_p(\mathcal{R}) + c_2\tilde{K}$])

Then verification using $\mathcal{R}$, $\tilde{K}$, and $\sigma(\boldsymbol{m}) = (c_1, r_1, r_2, r_3)$:

1. We use r_1 and c_1 to compute

$$c_2' = \mathcal{H}_n (..., [r_1G + c_1K_1], [r_1 \mathcal{H}_p(\mathcal{R}) + c_1\tilde{K}])$$

2. From when we made the signature, we see $c_2' = c_2$. With r_2 and c_2' we compute

$$c_3' = \mathcal{H}_n (..., [r_2G + c_2'K_2], [r_2 \mathcal{H}_p(\mathcal{R}) + c_2'\tilde{K}])$$

3. We can easily see that $c_3' = c_3$ by substituting c_2 for c_2'. Using r_3 and c_3' we get

$$c_1' = \mathcal{H}_n (..., [r_3G + c_3'K_3], [r_3 \mathcal{H}_p(\mathcal{R}) + c_3'\tilde{K}])$$

No surprises here: $c_1' = c_1$ if we substitute c_3 for c_3'.

Linkability

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

$$\sigma(\boldsymbol{m}) = (c_1, r_1, ..., r_n) \text{ with } \tilde{K}, \text{ and}$$

$$\sigma'(\boldsymbol{m}') = (c_1', r_1', ..., r_n') \text{ with } \tilde{K}', \text{ if } \tilde{K} = \tilde{K}'$$

then clearly both signatures come from the same signing ring and private key. While an observer could link σ and σ', he wouldn't know which K_i in $\mathcal{R}$ was the culprit without solving the DLP or auditing $\mathcal{R}$ in some way (such as learning all k_i with $i \neq \pi$, or learning k_π).[10]

2.2.3 Back's Linkable Spontaneous Anonymous Group (bLSAG) signatures

In the LSAG signature scheme, linkability of signatures using the same private key can only be guaranteed if the ring is constant. This is obvious from the definition $\tilde{K} = k_\pi \mathcal{H}_p(\mathcal{R})$.

In this section we present an enhanced version of the LSAG algorithm where linkability is independent of the ring's co-signers.

The modification was unraveled based on a publication by A. Back regarding the CryptoNote ring signature algorithm (previously used in Monero, and now deprecated), which was inspired by Fujisaki and Suzuki's work.

Signature

1. Calculate key image $\tilde{K} = k_\pi \mathcal{H}_p(K_\pi)$.

2. Generate random number $\alpha \in_R \mathbb{Z}_l$ and random numbers $r_i \in_R \mathbb{Z}_l$ for $i \in \{1, 2, ..., n\}$ but excluding $i = \pi$.

3. Compute

$$c_{\pi+1} = \mathcal{H}_n(\boldsymbol{m}, [\alpha G], [\alpha \mathcal{H}_p(K_\pi)])$$

4. For $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$ calculate, replacing $n + 1 \rightarrow 1$,

$$c_{i+1} = \mathcal{H}_n(\boldsymbol{m}, [r_i G + c_i K_i], [r_i \mathcal{H}_p(K_i) + c_i \tilde{K}])$$

5. Define $r_\pi = \alpha - c_\pi k_\pi \pmod{l}$.

The signature will be $\sigma(\boldsymbol{m}) = (c_1, r_1, ..., r_n)$, with key image $\tilde{K}$ and ring $\mathcal{R}$.

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

Just like LSAG, in this scheme we store (1+n) integers have one EC key image, use n public keys, and verify with:

VBSM Variable-base scalar multiplications for some integer a, and point P: aP [1]

DVBA Double-variable-base addition for some integers a, b, and point B: $aG + bB$ [n]

VBA Variable-base additions for some integers a, b, and points A, B: $aA + bB$ [n]

Correctness can also be demonstrated (i.e. 'how it works') in a way similar to the LSAG scheme.

The alert reader will no doubt notice the key image $\tilde{K}$ only depends on the true signer's keys. In other words, two signatures will now be linkable if and only if the same private key was used to create them. In bLSAG, rings are just involved in obfuscating each signer's identity. bLSAG also simplifies the scheme by removing $\mathcal{R}$ and $\tilde{K}$ from the hash that calculates c_i.

This approach to 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.

2.2.4 Multilayer Linkable Spontaneous Anonymous Group (MLSAG) signatures

In order to sign multi-input transactions, one has to sign with ***m*** private keys. In [61], Shen Noether *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

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

where we know the private keys $\{k_{\pi,j}\}$ corresponding to the subset $\{K_{\pi,j}\}$ for some index $i = \pi$. Such an algorithm would address our multi-input needs if we generalize the notion of linkability.

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

Signature

1. Calculate key images $\tilde{K}_j = k_{\pi j}\, \mathcal{H}_p\,(K_{\pi j})$ for all $j \in \{1, 2, ..., m\}$.

2. Generate random numbers $\alpha j \in_R \mathbb{Z}_l$, and $r_{i,j} \in_R \mathbb{Z}_l$ for $i \in \{1, 2, ..., n\}$ (except $i = \pi$) and $j \in \{1, 2, ..., m\}$.

3. Compute

$$c_{\pi+1} = \mathcal{H}_n\,(m, [\alpha_1 G], [\alpha_1\, \mathcal{H}_p\,(K_{\pi,1})], ..., [\alpha_m G], [\alpha_m\, \mathcal{H}_p\,(K_{\pi,m})])$$

4. For $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$ calculate, replacing $n + 1 \rightarrow 1$,

$c_{i+1} = \mathcal{H}_n(\boldsymbol{m}, [r_{i,1}G + c_iK_{i,1}], [r_{i,1}\mathcal{H}_p(K_{i,1}) + c_i\tilde{K}_1], ..., [r_{i,m}G + c_iK_{i,m}], [r_{i,m}\mathcal{H}_p(K_{i,m}) + c_i\tilde{K}m])$

5. Define $r_{\pi,j} = \alpha_j - c_\pi k_{\pi,j} \pmod{l}$.

The signature will be $\sigma(m) = (c_1, r_{1,1}, ..., r_{1,m}, ..., r_{n,1}, ..., r_{n,m})$, with key images $(\tilde{K}_1, ..., \tilde{K}_m)$.

Verification

The verification of a signature is done in the following manner:

1. For all $j \in \{1, ..., m\}$ check $l\tilde{K}j \stackrel{?}{=} 0$.

2. For $i = 1, ..., n$ compute, replacing $n + 1 \rightarrow 1$,

$c'_{i+1} = \mathcal{H}_n(m, [r_{i,1}G + c_iK_{i,1}], [r_{i,1}\mathcal{H}_p(K_{i,1}) + c_i\tilde{K}_1], ..., [r_{i,m}G + c_iK_{i,m}], [r_{i,m}\mathcal{H}_p(K_{i,m}) + c_i\tilde{K}_m])$

3. 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'_{1+1} are calculated as described in the signature algorithm.

• If i = π then, since $r_{\pi,j} = \alpha_j - c_\pi k_{\pi,j}$ closes the loop,

$$r_{\pi,j}G + c_\pi K_{\pi,j} = (\alpha_j - c_\pi k_{\pi,j})G + c_\pi K_{\pi,j} = \alpha_j G$$

and

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

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

Linkability

If a private key $k_{\pi,j}$ is re-used to make any signature, the corresponding key image $\tilde{K}_j$ supplied in the signature will reveal it. This observation matches our generalized definition of linkability.

Space and verification requirements

In this scheme we store $(1+m*n)$ integers, have m EC key images, use $m*n$ public keys, and verify with:

VBSM Variable-base scalar multiplications for some integer a, and point P: aP [m]

DVBA Double-variable-base addition for some integers a, b, and point B: $aG + bB$ [m * n]

VBA Variable-base additions for some integers a, b, and points A, B: $aA + bB$ [m * n]

2.3 Stealth Addresses

For any crypto transaction, the receiver would have to share their address, either directly to the sender, or on a public forum, thus revealing the details of their transaction. Beldex uses a form of intermediary addresses called stealth addresses to obfuscate the identity of the receiver. Whenever a sender initiates a transaction, the receiver generates a one-time intermediary address and funds are sent to this address. Funds sent to the intermediary address always reflect on the public address. The Diffie-Hellman key exchange enables the receiver to generate a private spend key for their intermediary address. So, they can claim ownership of the funds transferred to this intermediary address [7]. Thus, the receivers in a Beldex transaction are protected by a one-time expendable stealth address.

2.4 RingCT

Protecting transactional privacy doesn't stop with obfuscating the sender and receiver identity. To prevent any time-based analysis of transactions using the amount transferred, Monero Research Labs first proposed Ring Confidential Transactions (RingCT) as a means to obfuscate the amount in a transaction. RingCT uses range proofs which employ Pedersen commitments to ensure that the amount transferred can be verified without knowing the actual value, and this range is between 0 and 2^{64}. If Alice sends an amount x to Bob, she commits a value y and hashes y to the secret x such as $H(x||y)$. Once Bob receives the amount, she can then reveal her secret and Bob can verify that Alice indeed committed the correct outcome y. Thus, with Pedersen commitments, one can ensure that only non-negative values are sent and the actual value is masked. The size of crypto transactions that use traditional range proofs are large, thus, as an alternative, bulletproofs is used to greatly reduce the size of the transaction [8]. Therefore, to improve transaction scalability, Beldex will utilise a form of bulletproofs to reduce the payload that nodes and masternodes are required to store and transmit.

2.4.1 Pedersen Commitments

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

For example, in a coin-flipping game Alice could privately commit to one outcome (i.e. 'call it') before Bob flips the coin by publishing her committed value hashed with secret data. After Bob flips the coin, Alice could declare which value she committed to and prove it by publishing her secret data. Bob could then verify her claim.

In other words, assume that Alice has secret string *blah* and the value she wants to commit to is *heads*. She could simply hash $h = \mathcal{H}(blah, heads)$ and give h to Bob. Bob flips a coin, Alice tells Bob about *blah* and informs him she committed to *heads*, and then Bob calculates $h' = \mathcal{H}(blah, heads)$. If $h' = h$, then he knows Alice called *heads* before the coin flip.

Alice uses the so-called 'salt' *blah* so Bob can't just guess $\mathcal{H}$ (*heads*) and $\mathcal{H}$(*tails*) before his coin flip, and figure out she committed to *heads*.

2.4.1.1 Pedersen commitments

A *Pedersen commitment* is a commitment that has the property of being *additively homomorphic*. If $C(a)$ and $C(b)$ denote the commitments for values a and b respectively, then $C(a + b) = C(a) + C(b)$. This property is useful when committing transaction amounts, as one could prove, for instance, that inputs equal outputs, without revealing 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, by defining a commitment as simply $C(a) = aG$, we could easily create cheat tables of commitments to help us recognize common amounts a.

To attain information-theoretic 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 calculating γ from H is infeasible. In the case of Beldex, $H = \mathcal{H}_p(G)$.

We can then define the commitment to an amount a as $C(x, a) = xG + aH$, where x is a blinding factor that prevents observers from guessing a (for example: if you just commit $C(a = 1)$, it is trivial to guess and check).

Commitment $C(x, a)$ is information-theoretically private because there are many possible combinations of x and a that would output the same C. If x is truly random, an attacker would have literally no way to figure out a.

2.4.2 Amount commitments

Owning cryptocurrency is not like a bank account, where a person's balance exists as a single value in a database. Rather, a person owns a bunch of transaction *outputs*. Each output has an 'amount', and the sum of amounts in all *unspent* outputs owned is considered a person's balance.

To send cryptocurrency to someone else, we create a transaction. A transaction references old outputs as *inputs* and addresses new outputs to recipients. Once an output has been referenced, i.e. 'spent', it can never be referenced/spent again.

Since it is rare for input amounts to equal intended output amounts, most transactions include 'change', an output that sends excess back to the sender. We will elaborate on these topics in Chapter.

In Beldex, transaction amounts are hidden using a technique called RingCT, first implemented in September, 2018. While transaction verifiers don't know how much Beldex is contained in each input and output, they still need to prove the sum of input amounts equals the sum of output amounts.

In other words, if we had a transaction with inputs containing amounts a_1, ..., a_m and outputs with amounts b_1, ..., b_p, then an observer would justifiably expect that:

$$\sum_j a_j - \sum_t b_t = 0$$

Since commitments are additive and we don't know γ, we could easily prove our inputs equal outputs to observers by making the sum of commitments to input and output amounts equal zero:4

$$\sum_j C_{j,in} - \sum_t C_{t,\,\text{out}} = 0$$

To avoid sender identifiability, Shen Noether proposed verifying that commitments sum to a certain non-zero value:

$$\sum_j C_{j,in} - \sum_t C_{t,\,\text{out}} = zG$$

$$\sum_j \left(x_j G + a_j H\right) - \sum_t \left(y_t G + b_t H\right) = zG$$

$$\sum_j x_j - \sum_t y_t = z$$

The reasons why this is useful will become clear in next chapter when we discuss the structure of transactions.

Note that $C = zG$ is called a *commitment to zero* because we can make a signature with z, which proves that there is no H component to C (assuming γ is unknown). In other words $C = zG+0H$.

2.4.3 Range proofs

One problem with additive commitments is that, if we have commitments $C(a_1)$, $C(a_2)$, $C(b_1)$, and $C(b_2)$ and we intend to use them to prove that $(a_1 + a_2) - (b_1 + b_2) = 0$, then those commitments would still apply if one value in the equation were 'negative'.

For instance, we could have $a_1 = 6$, $a_2 = 5$, $b_1 = 21$, and $b_2 = -10$.

$$(6 + 5) - (21 + -10) = 0$$

Where,

$$21G + -10G = 21G + (l - 10)G = (l + 11)G = 11G$$

Since $-10 = l - 10$, we have effectively created l more Moneroj (over 7.2×10^{74}!) than we put in.

The solution addressing this issue in Monero is to prove each output amount is in a certain range using the Borromean signature scheme. This method treats each bit in the binary representation of the amount as a separate amount which we can commit to zero.

Given a commitment $C(b)$ with blinding factor y_b for amount b, use the binary representation (b_0, b_1, ..., b_{k-1}) such that

$$b = b_0 2^0 + b_1 2^1 + \cdots + b_{k-1} 2^{k-1}$$

Generate random numbers $y_0, \ldots, y_{k-1} \in_R \mathbb{Z}_l$ to be used as blinding factors. Define also Pedersen commitments for each $b_i, C_i = y_i G + b_i 2^i H$, and derive public keys $\{C_i, C_i - 2^i H\}$.

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

$$\text{if } b_i = 0 \text{ then } C_i = y_i G + 0H = y_i G$$

$$\text{if } b_i = 1 \text{ then } C_i - 2^i H = y_i G + 2^i H - 2^i H = y_i G$$

In other words, a blinding factor y_i will always be the private key corresponding to one of the points $\{C_i, C_i - 2^i H\}$. The point that equals $y_i G$ is a *commitment to zero*, because either $b_i 2^i = 0$ or $b_i 2^i - 2^i = 0$. We can prove a transaction output's amount b is in the range $[0, ..., 2^k - 1]$ by signing it using the Borromean Ring Signature scheme with the ring of public keys:

$$\left\{\{C_0, C_0 - 2^0 H\}, \ldots, \{C_{k-1}, C_{k-1} - 2^{k-1} H\}\right\}$$

Where, we know the private keys $\{y_0, ..., y_{k-1}\}$ corresponding to each pair.

Resulting in a signature $\sigma = \left(c_1, r_{0,1}, r_{0,2}, r_{1,1} \ldots, r_{k-1,2}\right)$

2.4.4 Range proofs in a blockchain

In the context of Beldex we will use range proofs to prove there are valid amounts in the outputs of each transaction.

Transaction verifiers will have to check that the sum of each output's range proof commitments C_i equals its amount commitment C_b. For this to work we need to modify our definition of the blinding factor y_b: set $y_0, \dots, y_{k-1} \in_R \mathbb{Z}_l$ and define $y_b = \sum_{i=0}^{k-1} y_i$. The following equation now holds

$$\sum_{i=0}^{k-1} C_i = C_b$$

We will store only the range proof commitments/keys C_i, the output amount's commitment C_b, and the signature σ's terms in the blockchain. The mining community can easily calculate $C_i - 2^i H$ and verify the Borromean ring signature for each output.

It will not be necessary for the receiver nor any other party to know the blinding factors y_i, as their sole purpose is proving a new output's amount is in range.

Since the Borromean signature scheme requires knowledge of y_i to produce a signature, any third party who verifies one can convince himself that each loop contains a commitment to zero, so total amounts must fall within range and money is not being artificially created.

In this scheme we store $(1+2 * k)$ integers, use k public keys (commitments), and verify with:

DVBA Double-variable-base addition for some integers *a, b,* and point *B: aG + bB* $[2 * k]$

Verifiers need to calculate $C_i - 2^i H$ and $\sum_i C_i$, so verification times include:

PA Point addition for some points *A, B: A + B* $[k - 1]$

PS Point subtraction for some points *A, B : A − B* $[k]$

KBSM Known-base scalar multiplications for some integer *a: aG* $[k]$

Monero uses a special function to compute $C_i - 2^i H$ and $\sum_i C_i = C$ that is supposed to be faster:

VRSF Verify range proof special function $[k]$

For a total verification requirement:

VRSF Verify range proof special function $[k]$

DVBA Double-variable-base addition for some integers *a, b,* and point *B: aG + bB* $[2 * k]$

2.5 Beldex Transactions

2.5.1 User keys

Unlike Bitcoin, Monero users have two sets of private/public keys, (k^v, K^v) and (k^s, K^s), generated as described in public key cryptography.

The *address* of a user is the pair of public keys (K^v, K^s). Her private keys will be the corresponding pair (k^v, k^s).

Using two sets of keys allows function segregation. The rationale will become clear later in this chapter, but for the moment let us call private key k^v the *view key*, and k^s the *spend key*. A person can use their view key to determine if an output is addressed to them, and their spend key will allow them to send that output in a transaction (and retroactively figure out it has been spent).

2.5.2 One-time addresses

To verify a transfer of money, observers need to know the money starts off being owned by the spender. How do they own it in the first place? Every Monero user has a public address which they may distribute to other users, who can then send money to that address via transaction outputs.

The public address is never used directly. Instead, a Diffie-Hellman-like exchange is applied to it, creating a unique *one-time address* for each transaction output to be paid to the user. In this way, even external observers who know all users' public addresses will not be able to identify which user received any given transaction output.

A recipient can spend their received outputs by signing a message with the one-time addresses, thereby proving they know the private keys and must therefore own what they are spending. We will gradually flesh out this concept.

Let's start with a very simple transaction, containing exactly one input and one output — a payment from Alice to Bob.

Bob has private/public keys (k_B^v, k_B^s) and (K_B^v, K_B^s), and Alice knows his public keys. A transaction could proceed as follows:

1. Alice generates a random number $r \in_R \mathbb{Z}_l$, and calculates the one-time public key

$$K^o = \mathcal{H}_n(rK_B^v)G + K_B^s.$$

Alice sets K^o as the addressee of the payment, adds the value rG to the transaction data, and submits it to the network.

3. Bob receives the data and sees the values rG and K^o. He can calculate $k_B^v rG = rK_B^v$. He can then calculate $K_B'^s = K^o - \mathcal{H}_n(rK_B^v)G$. When he sees that $K_B'^s = K_B^s$, he knows the transaction is addressed to him.

The private key k_B^v is called the view key because anyone who has it (and Bob's public spend key K_B^s) can calculate $K_B'^s$ for every transaction output in the blockchain (record of transactions), and 'view' which ones are addressed to Bob.

4. The one-time keys for the output are:

$$K^o = \mathcal{H}_n(rK_B^v)G + K_B^s = (\mathcal{H}_n(rK_B^v) + k_B^s)G$$
$$k^o = \mathcal{H}_n(rK_B^v) + k_B^s$$

While Alice can calculate the public key K^o for the one-time address, she cannot compute the corresponding private key k^o, since it would require either knowing Bob's spend key k_B^s, or solving the discrete logarithm problem for $K_B^s = k_B^s G$, which we assume to be hard. As will become clear later in this chapter, without k o Alice can't compute the output's key image, so she can never know for sure if Bob spends the output she sent him.

A third party with Bob's view key can verify an output is addressed to Bob, yet without knowledge of the spend key this third party would not be able to spend that output nor know when it has been spent. Such a third party could be a trusted custodian, an auditor, a tax authority, etc. Somebody who could be allowed read access to the user's transaction history, without any further rights. This third party would also be able to decrypt the output's amount (to be explained in Section 2.5.9.1.1).

2.5.3 Multi-output transactions

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

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 with p outputs are different even in cases where the same addressee is used twice, Beldex uses an output index. Every output from a transaction has an index $t \in \{1, \dots, p\}$. By appending this value to the shared secret before hashing it, one can ensure the resulting one-time addresses are unique:

$$K_t^o = \mathcal{H}_n(rK_t^v, t)G + K_t^s = (\mathcal{H}_n(rK_t^v, t) + k_t^s)G$$
$$k_t^o = \mathcal{H}_n(rK_t^v, t) + k_t^s$$

2.5.4 Subaddresses

Monero users can generate subaddresses from each address. Funds sent to a subaddress can be viewed and spent using its main address' view and spend keys. By analogy: an online bank account may have

multiple balances corresponding to credit cards and deposits, yet they are all accessible and spendable from the same point of view – the account holder.

Subaddresses are convenient for receiving funds to the same place when a user doesn't want to link his activities together by publishing/using the same address. As we will see, an observer would have to solve the DLP in order to determine a given subaddress is derived from any particular address.

They are also useful for differentiating between received outputs. For example, if Alice wants to buy an apple from Bob on a Tuesday, Bob could write a receipt describing the purchase and make a subaddress for that receipt, then ask Alice to use that subaddress when she sends him the money. This way Bob can associate the money he receives with the apple he sold. We explore another way to distinguish between received outputs in the next section.

Bob generates his i^{th} subaddress (i = 1, 2, ...) from his address as a pair of public keys $\left(K^{v,i}, K^{s,i}\right)$:

$$K^{s,i} = K^s + \mathcal{H}_n(k^v, i)G$$
$$K^{v,i} = k^v K^{s,i}$$

So,

$$K^{v,i} = k^v\left(k^s + \mathcal{H}_n(k^v, i)\right)G$$
$$K^{s,i} = \left(k^s + \mathcal{H}_n(k^v, i)\right)G$$

Sending to a subaddress

Let's say Alice is going to send Bob money via his subaddress $\left(K_B^{v,1}, K_B^{s,1}\right)$ with a simple one-input, one-output transaction.

1. Alice generates a random number $r \in_R \mathbb{Z}_l$, and calculates the one-time public key

$$K^o = \mathcal{H}_n\left(rK_B^{v,1}\right)G + K_B^{s,1}$$

2. Alice sets K^o as the addressee of the payment, **adds the value $rK_B^{s,1}$ to the transaction data**, and submits it to the network.

3. Bob receives the data and sees the values $rK_B^{s,1}$ and K^o. **He can calculate** $k_B^v rK_B^{s,1} = rK_B^{v,1}$. He can then calculate $K_B'^{s,1} = K^o - \mathcal{H}_n\left(rK_B^{v,1}\right)G$. When he sees that $K_B''^{,1} = K_B^{s,1}$, he knows the transaction is addressed to him.

Bob only needs his private view key k_B^v and subaddress public spend key $K_B^{s,1}$ to find transaction outputs sent to his subaddress.

4. The one-time keys for the output are:

$$K^o = \mathcal{H}_n\left(rK_B^{v,1}\right)G + K_B^{s,1} = \left(\mathcal{H}_n\left(rK_B^{v,1}\right) + k_B^{s,1}\right)G$$
$$k^o = \mathcal{H}_n\left(rK_B^{v,1}\right) + k_B^{s,1}$$

Now, Alice' transaction public key is particular to Bob ($rK_B^{s,1}$ instead of rG). If she creates a p-output transaction with at least one output intended for a subaddress, Alice needs to make a unique transaction public key for each output $t \in \{1, \ldots, p\}$. In other words, if Alice is sending to Bob's

subaddress ($K_B^{v,1}$, $K_B^{s,1}$) and Carol's address (K_C^v, K_C^s), she will put two transaction public keys $\{r_1 K_B^{s,1}, r_2 G\}$ in the transaction data.

2.5.5 Integrated addresses

In order to differentiate between the outputs they receive, a recipient can request senders include a *payment* ID in transaction data. For example, if Alice wants to buy an apple from Bob on a Tuesday, Bob could write a receipt describing the purchase and ask Alice to include the receipt's ID number when she sends him the money. This way Bob can associate the money he receives with the apple he sold.

Senders can communicate payment IDs in clear text, but manually including the IDs in transactions is inconvenient, and a privacy hazard for recipients, who might inadvertently expose their activities. In Monero, recipients can integrate payment IDs into their addresses, and provide those *integrated addresses*, containing (K^v, K^s, payment ID), to senders. Payment IDs can technically be integrated into any kind of address, including normal addresses, subaddresses, and multisignature addresses.

Senders addressing outputs to integrated addresses can encode payment IDs using the shared secret rK_t^v and a XOR operation, which recipients can then decode with the appropriate transaction public key and another XOR procedure. Encoding payment IDs in this way allows senders to prove they made particular transaction outputs (i.e. for audits, refunds, etc.).

2.5.6 Binary operator XOR

The binary operator XOR evaluates two arguments and returns true if one, but not both, of the arguments is true. Here is its truth table:

A	B	A XOR B
T	T	F
T	F	T
F	T	T
F	F	F

In the context of computer science, XOR is equivalent to bit addition modulo 2. For example, the XOR of two bit pairs:

$$\text{XOR}(\{1, 1\}, \{1, 0\}) = \{1 + 1, 1 + 0\} \pmod 2$$

$$= \{0, 1\}$$

Examining the previous example, each of these produce the same output: XOR({1, 1}, {1, 0}), XOR({0, 0}, {0, 1}), XOR({1, 0}, {1, 1}), or XOR({0, 1}, {0, 0}). There are 2^b combinations of XOR inputs (with *b* bits each) for the same output, so if input A $\in_R \{1, ..., 2^b\}$, an observer who learned *C* = XOR(*A*, *B*) could not gain any information about *B*.

At the same time, anyone who knows two of the elements *A*, *B*, *C*, where *C* = XOR(*A*, *B*), can calculate the third element, such as *A* = XOR(*B, C*). XOR indicates if two elements are different or the same, so knowing *C* and *B* is enough to expose *A*. A careful examination of the truth table reveals this to be true.

2.5.7 Encoding

The sender encodes the payment ID for inclusion in transaction data

$k_{\text{mask}} = \mathcal{H}_n\ (rK_t^v,\ \text{pid\_tag})$

$k_{\text{payment ID}} = k_{\text{mask}} \rightarrow$ reduced to bit length of payment ID

encoded payment ID = XOR($k_{\text{payment ID}}$, payment ID)

The output of a cryptographic hash function $\mathcal{H}$ is uniformly distributed across the range of possible outputs. In other words, for some input *A*, $\mathcal{H}(A) \in_R^D \mathbb{S}_H$ where $\mathbb{S}_H$ is the set of possible outputs from $\mathcal{H}$. We use $\in_R^D$ to indicate the function is deterministically random. $\mathcal{H}(A)$ produces the same thing every time, but its output is equivalent to a random number.

We include pid_tag to ensure k_{mask} is different from the component $\mathcal{H}_n\ (rK_t^v,\ t)$ in one-time output addresses.

2.5.8 Decoding

Whichever recipient *t* the payment ID was created for can find it using his view key and the transaction public key *rG*:

$k_{\text{mask}} = \mathcal{H}_n\ (K_t^v rG,\ \text{pid\_tag})$

$k_{\text{payment ID}} = k_{\text{mask}} \rightarrow$ reduced to bit length of payment ID

payment ID = XOR($k_{\text{payment ID}}$, encoded payment ID)

Similarly, senders can decode payment IDs they had previously encoded by recalculating the shared secret $k_{\text{mask}} = \mathcal{H}_n\ (rK_t^v)$.

2.5.9 Transaction types

Monero is a cryptocurrency under steady development. Transaction structures, protocols, and cryptographic schemes are always prone to evolving as new objectives or threats are found.

In this report we have focused our attention on *Ring Confidential Transactions*, a.k.a. *RingCT*, as they are implemented in the current version of Monero. RingCT is mandatory for all new Monero

transactions, so we will not describe any deprecated transaction types, even if they are still partially supported.

The transaction types we will describe in this chapter are **RCTTypeFull** and **RCTTypeSimple**. The former category (Section 2.5.9.1) closely follows the ideas explained by Shen Noether at al. in Ring Confidential Transactions, February 2016. At the time that paper was written, the authors most likely intended to fully replace the original CryptoNote transaction scheme.

However, for multi-input transactions, the signature scheme formulated in that paper was thought to entail a risk on traceability. This will become clear when we supply technical details, but in short: if one spent output became identifiable, 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** (Section 2.5.9.2) is used in those situations. The main difference, as we will see later, is that each input is signed independently.

2.5.9.1 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 to limit **RCTTypeFull** transactions to one input 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 ring sizes are increased. Also, Shen Noether's original description in Ring Confidential Transactions did not envision constraints of this type. At any rate, it is not a hard constraint. An alternative wallet might 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 for multi-input transactions.

An actual example of a **RCTTypeFull** transaction, with all its components, can be inspected in Appendix *A*.

2.5.9.1.1 Amount commitments

Recall from Section 2.4.2 that we had defined a commitment to an output's amount b as:

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

In the context of Monero, output recipients should be able to reconstruct the amount commitments. This means the blinding factor y and amount b must be communicated to the receiver.

The solution adopted in Monero is a Diffie-Hellman shared secret rK_B^v. For any given transaction in the blockchain, each of its outputs $t \in \{1, ..., p\}$ has two associated values called *mask* and *amount* satisfying

$$\text{mask}_{\ t} = y_t + \mathcal{H}_n\big(\mathcal{H}_n(rK_B^v, t)\big)$$
$$\text{amount}_{\ t} = b_t + \mathcal{H}_n\Big(\mathcal{H}_n\big(\mathcal{H}_n(rK_B^v, t)\big)\Big)$$

The receiver Bob will be able to calculate the blinding factor y_t and the amount b_t using the *transaction public key rG* and his *view key* K_B^v. He can also check that the commitment $C(y_t, b_t)$ provided in the transaction data, henceforth denoted C_b^t, corresponds to the amount at hand.

More generally, any third party with access to Bob's *view key* could decrypt his output amounts, and make sure they agree with their associated commitments.

2.5.9.1.2 Commitments to zero

Assume a transaction sender has previously received various outputs with amounts a_1, ..., a_m addressed to one-time addresses $K_{\pi,1}^o$, ..., $K_{\pi,m}^o$, and with amount commitments $C_{\pi,1}^a$,, ..., $C_{\pi,m}^a$,.

This sender knows the private keys $k_{\pi,1}^o$, ..., $k_{\pi,\text{m}}^o$ corresponding to the one-time addresses (Section 2.5.2). The sender also knows the blinding factors x_j used in commitments $C_{\pi,\text{j}}^a$ (Section 2.5.9.1.1).

A transaction consists of inputs a_1, ...,a_m and outputs b_1, ...,b_p such that $\sum_{j=1}^{m} a_j - \sum_{t=1}^{p} b_t = 0$.

The sender re-uses the commitments from the previous outputs, $C_{\pi,\text{j}}^a$, ..., $C_{\pi,m}^a$, and creates commitments for b_1, ..., b_p. Let these new commitments be C_1^b, ..., C_1^p.

As hinted in Section 2.4.2, the sum of the commitments will not be 0, but a curve point zG:

$$\sum_j C_{\pi,j}^a - \sum_t C_{\pi,t}^b = zG$$

The sender will know z, allowing him to create a signature on this *commitment to zero*. Indeed, z follows from the blinding factors if and only if input amounts equal output amounts (recalling Section 2.4.1.1, we don't know γ in $H = \gamma G$).

$$\begin{aligned}
&\sum_{j=1}^{m} C_{\pi,j}^a - \sum_{t=1}^{p} C_{\pi,t}^b \\
&= \sum_j x_j G - \sum_t y_t G + \left(\sum_j a_j - \sum_t b_t\right) H \\
&= \sum_j x_j G - \sum_t y_t G \\
&= zG
\end{aligned}$$

2.5.9.1.3 Signature

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

$$\mathcal{R} = \left\{ \left\{ K^o_{1,1}, \dots, K^o_{1,m}, \left(\sum_j C_{1,j} - \sum_t C^b_t \right) \right\} \right.$$
$$\dots$$
$$\left\{ K^o_{\pi,1}, \dots, K^o_{\pi,m}, \left(\sum_j C^a_{\pi,j} - \sum_t C^b_t \right) \right\}$$
$$\dots$$
$$\left. \left\{ K^o_{v+1,1}, \dots, K^o_{v+1,m}, \left(\sum_j C_{v+1,j} - \sum_t C^b_t \right) \right\} \right\}$$

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

$$\sum_j C^a_{\pi,j} - \sum_t C^b_t = 0$$

Then, any observer would recognize the set of addresses {Ko π,1 , ..., Ko π,m} as the ones in use as inputs, and therefore currency flows would be traceable.

With this observation made we can see the utility of *zG*. All commitment terms in R return some EC point, and the π^{th} such term is *zG*. This allows us to create an MLSAG signature (Section 2.2.4) on $\mathcal{R}$.

MLSAG signature for inputs The private keys for $\{K^o_{\pi,1}, ..., K^o_{\pi,m} (\sum_j C^a_{\pi,j} - \sum_t C^b_t)\}$ are

$K^o_{\pi,1}$, ..., $K^o_{\pi,m}$, *z*, which are known to the sender. MLSAG in this scenario does not use a key image for the commitment to zero *zG*. This means building and verifying the signature excludes the term $r_{i,m+1}\mathcal{H}_p(K_{i,m+1}) + c_i\tilde{K}_Z$.

The message ***m*** signed in the input MLSAG is essentially a hash of all transaction information *except* for the MLSAG signature itself.

This ensures transactions are tamper-proof from the perspective of both transaction authors and verifiers. k^o is the essence of Beldex's transaction model. Signing ***m*** with k^o proves you are the owner/recipient of the amount committed to in C^a. Verifiers can be confident that transaction authors are spending their own funds.

Range proofs for outputs To avoid the amount ambiguity of outputs described in Section 2.4.3, the sender must also employ the Borromean signature scheme to sign amount ranges for each output $t \in \{1, ..., p\}$. No message *m* is signed by the Borromean signatures.

Range proofs are not needed for input amounts because they are either expressed clearly (as with transaction fees and block rewards), or were proven in range when first created as outputs. In the current version of the Monero software, each amount is expressed as a fixed point number of 64 bits.

This means the data for each range proof will contain 64 bit commitments and $2 \cdot 64 + 1$ signature terms.

2.5.9.1.4 Transaction fees

Typically transaction outputs are lower in total than transaction inputs, in order to provide a fee that will incentivize miners to include the transaction in the blockchain.16 Transaction fee amounts are stored in clear text in the transaction data transmitted to the network. Miners can create an additional output for themselves with the fee. This fee amount must be converted into a commitment so verifiers can confirm transactions sum to zero.

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

The network verifies the MLSAG signature on $\mathcal{R}$ by including fH as follows:

$$\left(\sum_j C_{i,j} - \sum_t C_t^b\right) - fH$$

This works because this is a commitment to zero:

$$\left(\sum_j C_{\pi,j} - \sum_t C_t^b\right) - fH = zG$$

2.5.9.1.5 Avoiding double-spending

An MLSAG signature (Section 2.2.4) contains images $\tilde{K}_j$ of private keys $k_{\pi,j}$. An important property in any cryptographic signature scheme is that it should be unforgeable with non-negligible probability. Therefore, to all practical effects, we can assume a signature's key images must have been deterministically produced from legitimate private keys.

The network need only verify that key images included in MLSAG signatures (corresponding to inputs and calculated as $\tilde{K}_j^o = k_{\pi,j}^o \mathcal{H}_p\left(K_{\pi,j}^o\right)\Big)$ have not appeared before in other transactions. If they have, then we can be sure we are witnessing an attempt to re-spend an output $C_{\pi,j}^a$ addressed to $K_{\pi,j}^o$.

If someone tries to spend $C_{\pi,j}^a$ twice, they will reveal the index π for both transactions where it appears. This has two effects: 1) all outputs at index π in the first transaction are revealed as its real inputs, and 2) all outputs at index π in the second transaction are revealed as not having been spent before. The second is a problem even considering miners would reject the double-spend transaction.

These effects could weaken the network benefits of ring signatures, and are part of the reason **RCTTypeFull** is only used for single-input transactions. The other main reason is that a cryptanalyst would know that, in general, all real inputs share an index.

2.5.9.1.6 Space and verification requirements

MLSAG signature (inputs)

From Section 2.2.4 we recall that an MLSAG signature in this context would be expressed as

$$\sigma(\mathfrak{m}) = \left(c_1, r_{1,1}, \dots, r_{1,m+1}, \dots, r_{v+1,1}, \dots, r_{v+1,m+1}\right) \text{ with } \left(\widetilde{K}_1^o, \dots, \widetilde{K}_m^o\right)$$

As a legacy of CryptoNote, the values $\widetilde{K}_j^o$ are not referred to as part of the signature, but rather as *images* of the private keys $\widetilde{K}_{\pi,j}^o$. These key images 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 $((v + 1) \cdot (m+ 1)+ 1)\cdot 32$ bytes of storage, where v is the mixin level and m is the number of inputs. In other words, a transaction with 1 input and a total 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 each input, for $m \cdot 32$ bytes of storage, and additional space to store the ring member offsets in the blockchain. These offsets are used by verifiers to find each MLSAG signature's ring members' output keys and commitments in the blockchain, and are stored as variable length integers, hence we cannot exactly quantify the space needed.18

Including the computation of $\left(\sum_j C_{i,j} - \sum_t C_t^b\right) - fH$, and verifying key images are in G's subgroup with $l\widetilde{K} \stackrel{?}{=} 0$, we verify a **RCTTypeFull** transaction's MLSAG (Section 2.2.4) with:

PA Point addition for some points *A, B: A + B* $[m * (v + 1)]$

PS Point subtraction for some points *A, B : A − B* $[(v + 1) * (p + 1)]$

VBSM Variable-base scalar multiplications for some integer *a*, and point *P: aP* $[m]$

KBSM Known-base scalar multiplications for some integer *a: aG* [1]

DVBA Double-variable-base addition for some integers *a, b,* and point *B: aG+bB* $[(m+1)*(v+1)]$

VBA Variable-base additions for some integers *a, b,* and points *A, B: aA + bB* $[m * (v + 1)]$

Range proofs (outputs)

We know that a Monero Borromean signature for range proofs takes the form of an n-tuple

$$\sigma = (c_1, r_{0,1}, r_{0,2}, r_{1,1}..., r_{63,2})$$

Ring keys are considered part of ring signatures. However, in this case it is only necessary to store the commitments C_i , as the ring key counterparts $C_i - 2^iH$ can be easily derived (for verification purposes).

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

Including $C_i - 2^iH$ and checking $\sum_i C_i = C$, range proofs for p outputs will require this to verify.

VRSF Verify range proof special function $[p * 64]$

DVBA Double-variable-base addition for some integers *a, b,* and point *B: aG + bB* $[p * 2 * 64]$

2.5.9.2 Ring Confidential Transactions of type RCTTypeSimple

In the current Beldex code base, transactions with more than one input are signed using a scheme known as **RCTTypeSimple**.

The main characteristic of this approach is that instead of signing the entire set of inputs at once, the sender signs each input separately.

Among other things, this means we can't use commitments to zero in the same way as for RCTTypeFull transactions. It's really unlikely that we could match an input amount to an output amount, and in most cases each individual input will be less than the sum of outputs. So, we cannot directly commit inputs − outputs = 0.

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

$$C_j^a - \sum_t C_t^b = \left(x_j - \sum_t y_t\right)G + \left(a_j - \sum_t b_t\right)H$$

Since $a_j - \sum b_t \neq 0$, 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 computationally difficult.

2.5.9.2.1 Amount commitments

As explained, if inputs are decoupled from each other than the sender can't sign an aggregate commitment to zero. On the other hand, signing each input individually implies an intermediate approach. The sender could create new commitments to the input amounts and commit to zero with respect to each of the previous outputs being spent. In this way, the sender could prove the transaction takes as input only the outputs of previous transactions.

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

$$C_j^a = x_j G + a_j H$$

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

$$C_j'^a = x_j' G + a_j H$$

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

$$C_j^a - C_j'^a = (x_j - x_j')G$$

Hence, she would be able to use this value as a commitment to zero for each input. Let us say $(x_j - x_j') = z_j$, and call each $C_j'^a$ a *pseudo output commitment*.

Similarly to **RCTTypeFull** transactions, the sender can include each output's encoded blinding factor (mask) for y_t and amount for bt in the transaction (see Section 2.5.9.1.1), which will allow each

receiver t to decode y_t and b_t using the shared secret rK_t^v. 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 MLSAG signature scheme implies each sender has signed with the private key of a commitment to zero.

For **RCTTypeSimple** transactions, blinding factors for input and output commitments are selected such that

$$\sum_j x'_j - \sum_t y_t = 0$$

This allows us to prove input amounts equal output amounts:

$$\left(\sum_j C'^a_j - \sum_t C^b_t\right) - fH = 0$$

Fortunately, choosing such blinding factors is simple. In the current version of Beldex, all blinding factors are random except x'_m, which is simply set to

$$x'_m = \sum_t y_t - \sum_{j=1}^{m-1} x'_j$$

2.5.9.2.2 Signature

As we mentioned, in transactions of type **RCTTypeSimple** each input is signed individually. We use the same MLSAG signature scheme as for **RCTTypeFull** transactions, except with different signing keys.

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

Similar to the previous scheme, the sender selects v unrelated outputs and their respective commitments from the blockchain to mix with the real, jth, input

$$K^o_{1,j}, \dots, K^o_{\pi-1,j}, K^o_{\pi+1,j}, \dots, K^o_{v+1,j}$$
$$C_{1,j}, \dots, C_{\pi-1,j}, C_{\pi+1,j}, \dots, C_{v+1,j}$$

She can sign input j using the following ring:

$$\begin{aligned}\mathcal{R}_j = \{&\left\{K^o_{1,j}, \left(C_{1,j} - C'^a_{\pi,j}\right)\right\} \\ &\dots \\ &\left\{K^o_{\pi,j}, \left(C^a_{\pi,j} - C'^a_{\pi,j}\right)\right\} \\ &\dots \\ &\left\{K^o_{v+1,j}, \left(C_{v+1,j} - C'^a_{\pi,j}\right)\right\}\}\end{aligned}$$

Alice will know the private keys $k^o_{\pi,j}$ for $K^o_{\pi,j}$, and $\mathbf{z}_j$ for the commitment to zero $(C^a_{\pi,j} - C'^a_{\pi,j})$. Recalling Section 2.5.9.1.3, there is no key image for the commitments to zero $\mathbf{z}_j G$, and consequently no corresponding key image component in each input's signature's construction.

Each input in an RCTTypeSimple transaction is signed individually, applying the scheme described in Section 2.5.9.1.3, 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 need not be placed at the same index π, as they are in the aggregated case. This means even if one input's origin became identifiable; the other inputs' origins would not. The message m signed by each input is essentially the same as for **RCTTypeFull** transactions, except it includes pseudo output commitments for the inputs. Only one message is produced, and each input MLSAG signs it.

2.5.9.2.3 Space and verification requirements

MLSAG signature (inputs)

Each ring $\mathcal{R}_j$ contains $(v + 1) \cdot 2$ keys. Using the point compression technique, an input signature σ will require $(2(v + 1) + 1) \cdot 32$ bytes. On top of this is, the key image $\widetilde{K}^o_{\pi,j}$ and the pseudo output commitment $C'^a_{\pi,j}$ leave a total of $(2(v + 1) + 3) \cdot 32$ bytes per input.

A transaction with 20 inputs using rings with 32 total members will need $((32 \cdot 2+3) \cdot 32)20 = 42880$ bytes.

For the sake of comparison, if we were to apply the **RCTTypeFull** scheme to the same transaction, then the MLSAG signature and key images would require $(32 \cdot 21 + 1) \cdot 32 + 20 \cdot 32 = 22176$ bytes.

Including the computation of $(C_{i,j} - C'^a_{\pi,j})$ and $(\sum_j C'^a_j \stackrel{?}{=} \sum_t C^b_t + fH)$, and verifying key images are in G's subgroup with lK~, we verify all of a **RCTTypeSimple** transaction's MLSAGs with:

PA Point addition for some points *A, B:* $A + B$ $[m + p + 1]$

PS Point subtraction for some points *A, B :* $A - B$ $[m * (v + 1)]$

VBSM Variable-base scalar multiplications for some integer *a*, and point *P:* aP $[m]$

KBSM Known-base scalar multiplications for some integer *a:* aG [1]

DVBA Double-variable-base addition for some integers *a, b,* and point *B:* $aG+bB$ $[m* 2 * (v + 1)]$

VBA Variable-base additions for some integers *a, b,* and points *A, B:* $aA + bB$ $[m * (v + 1)]$

Range proofs (outputs)

The size of range proofs remains the same for **RCTTypeSimple** transactions. As we calculated for **RCTTypeFull** transactions, each output will require 6176 bytes of storage. As before, each range proof will require this to verify (Section 2.4.4):

VRSF Verify range proof special function $[p * 64]$

DVBA Double-variable-base addition for some integers *a, b,* and point *B:* $aG + bB$ $[p * 2 * 64]$

2.6 Masternodes

Beldex improvises on the CryptoNote protocol to provide greater privacy; however, a group of incentivized masternodes determine most of the functionality and scalability in Beldex. These masternodes are full nodes that store the entire copy of the Beldex blockchain. To set up a Beldex masternode, a node operator must lock-in aminimum collateral and possess the required minimum bandwidth and storage capabilities. The collateral ensures that the masternode operators refrain from acting dishonestly, as they have a significant stake in the network. As compensation for their services, masternode operators receive a portion of the block reward. A masternode is continually reviewed and if it is found to act dishonestly or doesn't provide the required bandwidth at any time, then it is penalized and moved to the end of the reward queue. A maximum of three warnings is given to a masternode operator before they are removed from the network.

The network thus formed is resistant to Sybil attacks, in which a single or a group of bad actors work in coalition to undermine the network. The initial setup collateral and the periodical review of masternodes ensure that such attacks are prevented. The maximum amount of stake a single validator can have is limited by supply and demand, preventing one validator to have a disproportionately large stake so as to sabotage the second-layer ecosystem services. Resistance to Sybil attacks can also be acquired through Cryptoeconomics [9]. When a bad actor begins to accumulate Beldex (BDX) in order to obtain a majority stake within the network, the supply in circulation decreases and inturn drives the price up. Acquiring additional Beldex becomes increasingly expensive, thus making the attack economically unfeasible. This disincentivizes the attacker. To ensure that masternode validators obtain sustainable returns for their services, the circulating supply is actively reduced and the emission curves and minimum collateral requirements are designed such that enough circulating supply is locked within the masternodes.

2.7 Block Reward

Block rewards in the Beldex network are currently distributed through proof-of-work consensus. A node or a masternode creates a new block every 10 minutes by collecting the transactions, validating them, including them within the block, and adding the block to the end of the network. To perform this action, they maintain a certain bandwidth and expend power to do so. In return, they collect block rewards as new BDX coins are emitted. The Beldex block reward is set to a constant value of 2 BDX.

Node Reward: In addition to the transaction fees, 10% of the block reward is awarded to the node that constructs the block.

Master Node Reward: 90% of total Beldex rewards go to a Masternode, or two Masternodes if both the masternodes construct the block at the same time. Once the Masternode receives a reward, it assumes the last position in the reward queue. Thus, nodes that recently received rewards are moved to the end of the reward queue while nodes that have been waiting for a longer period are moved to the start of the queue. The very first time a Masternode validator is indexed on the network's distributed decentralized directory, it assumes the last position in the reward queue. The masternodes that are found to provide optimum services to the network continually move up the queue, if they are not otherwise penalized.

2.8 Authenticated Collateralization

A masternode operator may stake this amount entirely by themselves or stake part of the amount and let other users stake the remaining. In this case, they share the rewards based on their stake within the masternode. To be added to the Beldex distributed directory, each Masternode must prove to the network that they hold the necessary stake, which is 10000 BDX. The innate privacy features of Beldex prevent the masternodes from doing so as public addresses do not reveal the amount of transaction. They also cannot use viewkeys to see transactions going out of the network.

To solve this dilemma of Masternodes, Beldex uses a novel method of time-locking the collateral. The initial collateral by a masternode is locked for a specific block-height of 86,400 blocks. Thus, from the time of registration until 86,400 blocks are created, the masternodes will be unable to spend this collateral. Beldex leverages this method to ensure that a particular masternode holds the required collateral amount.

In the additional field of transaction, the Masternode operator provides the Beldex address to receive rewards. This address is the public key of the Masternode operator. The transactions for the purpose of registration are open transactions, thus they may not be included in a ring transaction as they do not serve to provide anonymity.

The existing Masternodes in the network must verify that the first time Masternode registrants possess the required collateral before they are added to the distributed decentralized directory.

2.9 Flash

Flash instant transactions are a second layer atop the Beldex blockchain that allows for confirmation of transactions before they are included in a block. Blockchain transactions typically take the time required for the transaction to be included in a block and confirmed by the network. A Beldex transaction requires 10 confirmations and the block creation time is 2 minutes. Thus, it may take anywhere between 20-40 minutes for a transaction to be confirmed. But with flash, it only takes a few seconds.

The Beldex second layer architecture (Flash) is similar to Bitcoin's lightning network in that a shortest possible route is selected between the Flash channels [10]. A channel path reducing algorithm is used to send the outputs to the receiver, where channels from users who've made transactions with each other before become relays. If B has sent or received inputs from A and C, then B can act as a relay in transactions between A and C.

Flash instant transactions are feasible through the use of Masternodes that confirm a transaction's authenticity by securing the key images associated with the UTXOs. In a ring signature transaction, the UTXOs are linked to a distinctive key image. Select Masternodes store this key image and hold onto it until the transaction is added to a block on the Beldex network. If the said key image is produced more than once, then that signals a double spending attempt. The corresponding duplicate transaction is rejected.

Similar to Ethereum, Flash will enable a mechanism for competing transactions to pay a higher fee to be completed first. But this is several times faster and will be in the order of a few seconds since it

takes place outside the blockchain. Confirmation time on the blockchain is also significantly reduced due to the network transition to a Proof-of-Stake (POS) consensus.

2.10 Coin Burning Mechanism

The Beldex coin burning mechanism is a way to curb inflation and deliver sustainable price discovery. To achieve this, the network fee obtained through Flash transactions will be burned. The Flash second layer serves to provide instant transactions and thus will have a higher fee. This fee is burned as soon as the transaction is completed.

2.11 Proof-of-Stake (POS) in Beldex Chain

POS is an alternative to POW that is designed to be economical and energy-efficient. At present, most cryptocurrencies run on a POW algorithm that consumes a lot of energy through mining hardware. Newer and powerful hardware is developed to plow through the hashrate. This hardware is often not affordable to everyone, making them less decentralized and they consume several KW of power every day. Bitcoin's annual energy consumption index is an estimated 95.87 TWh, according to Digiconomist [11].

Beldex initially proposes the implementation of the RandomX algorithm to prevent ASIC mining and provide equal opportunity to CPU and GPU miners. However, Beldex will be implementing Proof-of-Stake in its network. Currently, the Beldex POS is in the testnet stage and is being tested for attacks against privacy and security. Later, this will be upgraded to a full POS network. A full POS Beldex network will enable users to stake BDX on nodes. Those with a stake can participate in validating blocks on the network. The higher the stake, the higher the chances for the node to be selected to validate a block, and the lower is the difficulty. POS on Beldex is based on delimited competition of nodes.

Anyone is free to contribute to the Beldex network, given the required collateral staking amount is locked in the node. Dishonest nodes are disincentivized and their collateral is devalued. This prevents the nodes from acting against the network whereas honest nodes are rewarded and appraised. Beldex also provides pool staking via masternodes where the hardware is shared between a few individuals who share the stake amount (10,000 BDX).

2.12 How Does Proof of Stake Work?

Proof of Stake works on the mechanism that for a node to participate in the network it must possess the minimum required stake. The minimum required stake to run a POS node is 10000 BDX.

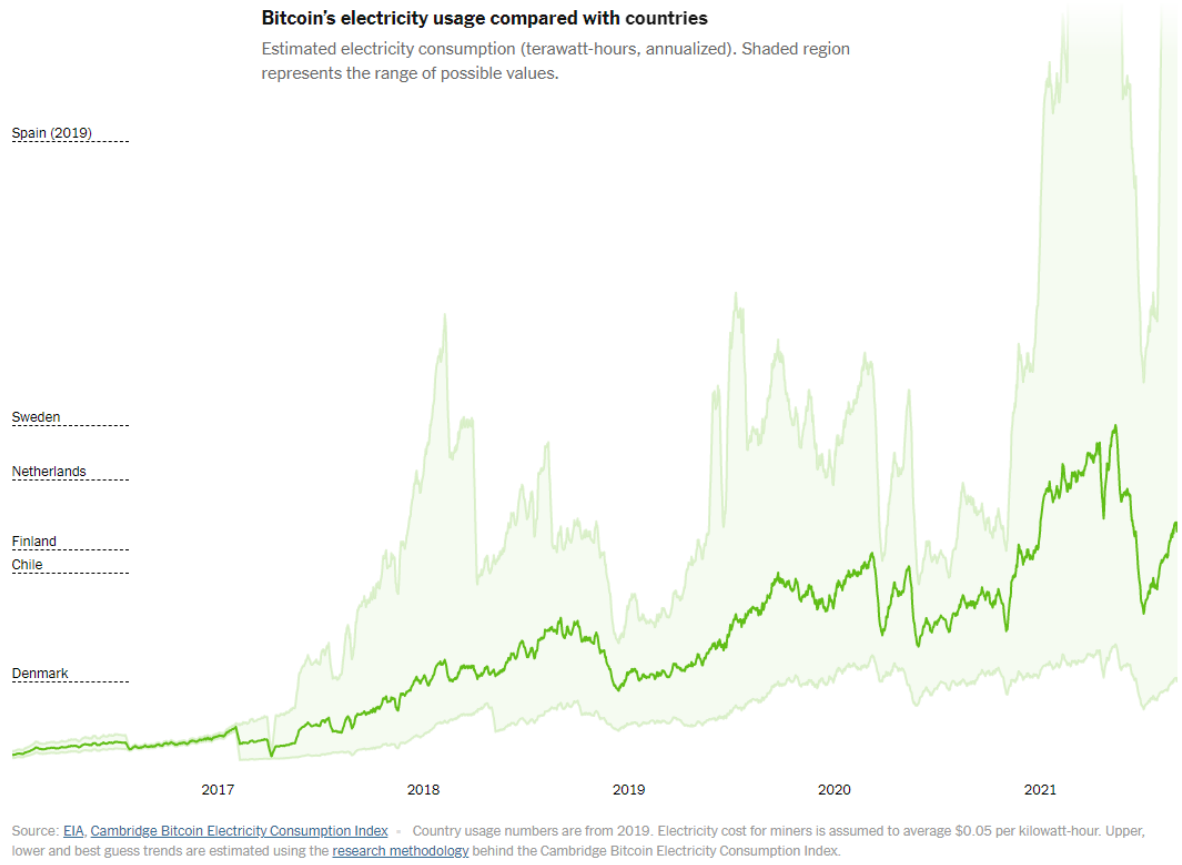
Bucephalus is Beldex's Proof of Stake implementation. POS makes the Beldex network fast, sustainable, secure, and scalable.

In POS consensus, every 30 seconds, a node is selected at random to produce a block which is then sent to a quorum vote consisting of 11 other participating validator nodes. The validators in a quorum verify the authenticity of the block and if any 7 validator nodes of the 11 attest to the block's

authenticity, it is broadcast to the network as a valid block. For the creation of the next block, a new quorum or consensus group is formed and the process repeats.

2.13 Power Consumption In A POS Blockchain

One area of concern for blockchains is sustainability. 50 or 100 years into the future, they need to be able to validate transactions the same way they do now without consuming 50X more power. The Bitcoin blockchain consumes over 200 TWh per year, that's as much power consumed by entire nation-states.



POS solves the blockchain energy problem by decoupling the hash power from the computing power of the nodes. The minimum stake provides one the hash power required to participate in the network. By transferring authority from computing power to one's stake within the network, POS essentially reduces the entry barrier to participation. POS networks could therefore have a greater degree of decentralization, scalability, and flexibility.

2.14 Security in Proof of Stake Consensus Model

In Proof of Work, an attacker would have to gain 51% of the hash power within the network. To do this, they set up a majority of the nodes, gain computational power which translates to hash power. For this reason, blockchain networks such as Bitcoin increase the cost of hardware required to attack a network. They use ASICs, FPGAs, and GPUs which are cost-intensive. However, this leads to additional complications of centralization. If the network is minable only with a specific kind of

ASIC, then the manufacturers of this equipment can play a role in who gets to participate in the network.

On the other hand, Proof of Stake provides node operators with the authority to validate a block based on their stake within the network. An attacker, in this case, would have to possess more than 50% of the staked $BDX to take over the network, which is cost-intensive and economically not feasible. The threat of a 51% attack can be negated by sufficient decentralization of the network. If more than 50% of the network's tokens are locked, then it becomes virtually impossible to take over the network. Moreover, any malicious node risks slashing, in which they stand to lose their collateral and rewards. Thus, node operators are disincentivized from acting against the network.

2.14.1 Checkpoints

Checkpoint is a mechanism employed as an additional layer of security within the network. Checkpoint works by taking a snapshot of the blockchain history every 4 blocks. Thus, even if the Beldex network were to be compromised by a group of bad actors banding together, they will not be able to alter more than 4 blocks, since everything prior to it is recorded.

2.15 Bucephalus Hardfork

The Beldex network transitioned to Proof of Stake on Dec 10, 2021, via the Bucephalus Hardfork. The proof of stake consensus introduced several changes to the network, a few of them being reduced block production time. The previous 2 minute block time is reduced four times and now sits at an average of 30 seconds. The transaction size is significantly reduced while block rewards have increased 5X from 2 BDX to 10 BDX. 62.5% of the new block rewards are allocated to masternode block producers while 37.5% is allocated to governance. Flash instant transactions were launched with a network confirmation of 2 nodes.

2.15.1 Implementation of Bucephalus POS Consensus

The Bucephalus consensus works on a number of quorum states that run simultaneously and have different functionalities.

For example, the POS quorum determines the block producer and the 11 validator nodes. The Flash quorum is based on two separate consensus groups that verify the transaction and then include them in a block. The Masternode quorum checks the nodes' uptime proof before being selected to participate in block validation while the checkpoint quorum takes a snapshot of the chain history every 4 blocks.

2.15.1.1 POS Quorum

A POS quorum is made up of 12 nodes in which 1 node produces the block and the remaining 11 nodes validate it. The producer selects valid transactions from the mempool and creates the nominee block. The producer then signs the nominee block and broadcasts it to the validator nodes in the consensus group. Each validator node broadcasts it to 3 other nodes in the network to ensure that all the nodes receive the block.

Each validator now creates a 128 bit integer and sends it to other validators. Validators reveal the pre-image of the hash, so that each validator now has a subgroup values. Validator combine the pre-images to produce a new random value. The new random value is added to the nominee block and then signed. The signature is then sent to all other validators.

The validators then check each other's signatures, ensuring they are all signing both the same nominee block and the same combined random integer. Once signatures have been validated, any validator in the quorum can submit the block to the network. For the block to be confirmed, 7/11 validator nodes should approve it.

Quorum state is checked every 1 minute. This could be set to less than 1 minute, however, a handshake between 12 nodes in a quorum requires at least 30 seconds. Thus, to ensure a quorum is formed, the maximum time is set to 1 minute, while in reality, it could be lesser. On the other hand, the blocktime is adjusted such that the average block creation time is 30 seconds.

A masternode that produces the block is rewarded 6.25 BDX and assumes the last position in the reward queue. The successive block is produced once the next set of 12 quorum nodes is selected from the 50 nodes that were set aside in Masternode Quorum.

2.5.1.2 Flash Quorum

Flash transactions are instant and have a slightly higher fee than regular transactions (which is still orders of magnitude lesser than most other blockchain networks). Flash transactions differ from other transactions in the following way.

Regular $BDX transactions are sent to the mempool where they wait to be included in a block. However, flash transactions are sent to two different quorums that instantly verifies them. The sender and receiver see that the locked balances have been updated on their wallets. Transactions are then sent to the mempool to be included in the block.

2.15.1.3 Masternode Quorum

The masternode quorum checks if any given masternode is active and eligible for the next round of POS quorum. 10 validator nodes in POS quorum check other masternodes for proof of uptime. 50 nodes are set aside for this process. The nodes that do not submit uptime proof are decommissioned in order to prevent them from obstructing the network.

2.15.1.4 Checkpoint Quorum

As mentioned in section 5.1, checkpoints are an additional layer of security. Checkpoint quorums take a snapshot of the blockchain and record the hash of every 4th block. Thus, an attacker can virtually only be able to change 4 blocks. Beyond that, the network will self-correct using the snapshot history from the Checkpoints.

2.15.2 Byzantine Fault Tolerance

When a valid block is not produced for more than 60 seconds, it results in a pause in block production. When this happens, the next block producer can produce the next nominee block and send

it to the validators by including a switch tag. This tag indicates to new handshaking nodes that there has been a failure in creating a quorum.

Fault cases: (1) the block producer doesn't create the block, (2) less than 11 validators reveal the pre-image of the hash, (3) validators signing two different blocks (4) validators reveal different values, thus they're unable to submit the block.

2.15.2.1 Block producer doesn't create the block

If a block producer fails to construct and send a nominee block, the validators will be triggered to submit intent to demerit message. If this message is signed by a simple majority of the quorum, a demerit point is to be applied to the producer and the next producer creates a switch block.

2.15.2.2 Less than 11 validators reveal the pre-image of the hash

Although all Beldex Master Nodes are incentivised to cooperate in order to avoid demerits, some participants may either act maliciously or simply fail. The protocol should be able to tolerate the failure of 6 of 11 validator nodes during the reveal pre-image of hash stage. As above, any validator that doesn't participate in this round, or reveals a different value from their commitment can be flagged by other validators with an intent to demerit message. If this message is signed by a majority of nodes, this message is submitted to the network as evidence of a node's non-performance in a block creation quorum.

If more than 6 of 11 nodes don't participate in the commit or reveal round, the quorum will not be able to create a block — which means the next quorum can take over block creation once a timeout occurs.

2.15.2.3 Validators reveal different values, thus they're unable to submit the block

There may be a number of cases where not all of the masternodes correctly receive all of the committed or revealed values. This will result in differing final random integers being calculated by the validators.

As with other errors, the scheme will progress normally as long as the revealed values for 7 of the 11 validators match. Nodes with invalid revealed values can query the other validators in their quorum again to receive any values they missed during the two rounds.

If more than 7 nodes disagree on the final revealed value, then the scheme will fail once a timeout occurs and the next producer produces a switch block.

2.15.2.4 Validator signs two different blocks

Any masternode node, including a validator or leader, may submit a deregistration transaction if they can produce signatures from any other masternode that signs multiple blocks which compete for the same height or signs a competing chain once more than two checkpoints have been applied.

2.15.3 Decommission and Deregistration

Validators can send intent to decommission message if they deem that any peer hasn't followed the rules (e.g. the block producer failing to create a nominee block) or if a validator doesn't submit proof of uptime for any of the services such as Beldex Storage Server and BelNet.

2.15.3.1 Block Credits

The decommissioned nodes have a specific block credit before which they need to submit an uptime proof. Block credits are received by the nodes based on the time elapsed since the masternode joined the network. Each masternode is awarded 96 block credits in a day approx. The maximum block credit is 5760 blocks. Thus, the longer the masternode has been active in the network, the greater will be its block credits.

3.0 Beldex Services

3.1 Beldex Decentralized Wallets

3.1.1 Android & iOS Wallet

The Android and iOS wallets come with the same features as the Beldex electron wallet. The mobile wallets provide instant access to your Beldex (BDX). Store, send, receive, and even stake BDX (feature to be launched in the upcoming release). You can download the Beldex wallet for your Android device and iOS device here.

3.1.2 Beldex Electron Desktop Wallet

The Beldex Electron desktop wallet is a DApp built for Windows, Linux, and Mac. The electron desktop wallet is fast, secure, and decentralized. It can be used to hold, send, and receive BDX. In addition to this, the electron wallet provides an extensive and easy to use GUI to setup and host your Beldex masternode. You can download it here.

3.2 Beldex Bridge

3.2.1 Binance Smart Chain Bridge

Binance Smart Chain is one of the fastest-growing ecosystems. The blockchain allows for interoperability with the Binance Chain and other platforms being built on it. Beldex will be bridging the BDX coin to the Binance Smart Chain without trading off its inherent privacy features. The BDX

coin can be swapped for the Wrapped-BDX (wBDX) token on the Binance Smart Chain. Each wBDX token is backed by the BDX coin in a 1:1 ratio. The wBDX privacy token can be utilized on the Binance Smart Chain platforms. Users can also swap back wBDX to BDX using the Beldex-Binance Smart Chain Bridge.

3.2.2 ETH Bridge

The Beldex network is a native chain, and thus, it has to be bridged to the Ethereum chain in order to utilize BDX on the Ethereum platform. The Beldex to Ethereum Bridge will allow the BDX token to exist as an ERC-20 token on the Ethereum chain in a 1:1 ratio. This will allow Beldex tokens to be utilized on various Ethereum based platforms and wallets. The corresponding ERC-20 tokens can be swapped back to BDX coins on the Ethereum bridge.

3.2.3 DOT Bridge

DOT is an emerging Ethereum competitor. Thus, Beldex will be bridged to the Polkadot chain to improve interoperability between the Beldex-Polka chains. The Beldex to Polkadot Bridge will allow the BDX token to exist as a DOT-based token on the Polkadot chain in a 1:1 ratio. This will allow Beldex tokens to be utilized on various Polkadot based platforms and wallets. The corresponding DOT-based ERC-20 tokens can be swapped back to BDX coins on the Ethereum bridge.

3.3 Beldex Privacy Protocol

The Beldex privacy protocol is cross chain payment privacy and anonymity protocol that anonymizes transactions on the Bitcoin, Ethereum, Polkadot, BSC, and Cosmos chains. To achieve this, Beldex uses a form of the zk-SNARKS algorithm first proposed by ZCash [12] along with the Zether protocol [13] and El Gamal encryption. Bulletproof range proofs are used to reduce the size of the transaction.

The Beldex privacy protocol is deployed on the substrate of the Binance Smart Chain network, essentially a second layer. Let's consider that user A wants to anonymously send Ethereum (ETH) to user B. Beldex provides the perfect platform for this operation.

User A creates an account on the Beldex protocol. They then proceed to burn their Ethereum tokens on the Ethereum chain for equivalent wrapped privacy tokens on a 1:1 ratio on the Beldex chain, say b-ETH. User 'A' then sends an X amount of b-ETH to user 'B' by initiating a transaction via the Beldex privacy protocol. Once the token is received, the user 'B' can then reclaim ETH from b-ETH. User 'B' will however, not know the amount that was locked in the contract by user 'A'.

With the aid of zk-SNARKS, the network can verify that user A has the ownership of the privacy tokens that were transferred. The El Gamal homomorphic encryption ensures the verification of the inputs and outputs, even if the actual values of transactions are not known; whereas to obfuscate sender and receiver identity, a "one-out-of-many" protocol is used. Beldex uses RingCT for this purpose.

3.4 Decentralized Messaging App (B-Chat)

B-Chat is an anonymous messaging application that routes messages using BelNet. With modern day applications such as WhatsApp and Signal, messaging has become end-to-end encrypted. The Signal protocol [14], especially, has proven to offer a greater degree of privacy than the former. However, end-to-end encryption does not offer complete privacy as the data is ultimately stored on a centralized data centre. Centralized data centres are susceptible to attacks due to their centralized nature. They are also sparsely protected against government policies and intervention. There's no preventing a government or an authority from mandating centralized messaging applications to share user data. This takes away the privacy that is offered by these applications.

B-Chat introduces routing of messages through the BelNet, a decentralized Virtual Private Network (dVPN). The decentralized VPN, the BelNet, ensures that messages are routed to their destination through the shortest possible node distance while encrypting them at each hop. The minimum hop length is set as 3, not counting the origin and destination nodes. This prevents a dishonest node from sabotaging the message as it only receives an encrypted form of the message to begin with.

Messages can be sent by the sender even if the receiver is online or offline, with the help of Beldex Secure Masternodes. When the receiver is online, the message packets get routed through the masternodes. The secure Masternodes act as the routing channel when user A wants to send a message to user B. The Masternode determines the shortest possible distance over which the message can be routed on the routing channel. To do this, the origin node (node where the message originates) gets the destination nodes' information from the distributed decentralized directory stored by the Masternodes. Then, it establishes a safe route while checking for available bandwidth and traffic congestion. It then transmits the message, which gets encrypted at each node hop and a minimum of three node hops are required for each message packed. The message is received by the destination node which in turn relays it to the user B. When the user B is offline, the closest node, or the destination node, stores the message and relays it when they return online.

3.5 BelNet

Virtual Private Networks (VPNs) have enabled people to remain anonymous with their online activity. But today's VPNs aren't completely private since most of them are managed by centralized systems. Existing onion routing protocols such as Tor offer minimum decentralization while their Distributed Hash Tables (DHTs) are managed by a handful of nodes called Directory Authorities [15]. Thus, they offer only rudimentary privacy.

On the other hand, an onion routing protocol on a blockchain network will remain completely decentralized, and therefore offers greater privacy. BelNet is a private decentralized VPN that works based on the Beldex Routing Protocol (BRP). The BRP optimizes the communication between nodes such that the traffic is traversed through the network with minimum load to the secure Masternodes.

The BRP uses the packet switching routing model and supports a wide range of internet protocols. It's Distributed Hash Table that lists the nodes on the network is maintained by Masternodes, based on a node limiting and authorizing mechanism. The governing Masternodes are a set of nodes that get randomly assigned to verify an incoming new node.

A new node must meet the bandwidth and staking requirements to obtain the status of a relay or exit node on the BelNet. The governing nodes are reassigned to each node and at fixed time intervals, they keep verifying if a node has sufficient bandwidth. If any node fails to meet the bandwidth requirements, then they are penalized by the governing nodes and their rewards get cut until they provide the required bandwidth. The significant staking capital also prevents the nodes from acting dishonestly in the network.

3.6 Beldex Browser

The online advertising industry thrives out of unwarranted advertising by tracking user activity. User privacy is disregarded day in and day out. In addition, certain applications have the practice of monitoring a users' offline data. In today's advertising industry, however, data processors or middlemen take away a large sum of this ad revenue from content authors. Large players have also monopolized the advertising market [16]. To combat this, Beldex proposes the Beldex browser, a secure and privacy-first browser that adds value to advertisers and content creators while preserving user privacy.

The Beldex browser routes user traffic using BelNet and the Beldex Masternodes, which anonymizes the source and destination of the traffic. Users choose the content they want to pay their attention while content creators and advertisers can target a more specific audience. Donning the peer-to-peer agenda, content authors can publish ad-free content to their audience who may choose to pay creators for their content with BDX.

4.0 Marketing

4.1 Community

Beldex has had a very strong community base since inception. Beldex has various channels for engaging with the community. You can find the full list of announcement boards and social media below.

Telegram Announcements: https://t.me/official_beldex
Telegram Chat: https://t.me/beldexcoin
Twitter: https://twitter.com/BeldexCoin
Discord: https://discord.gg/Hj4MAmA5gs
Facebook: https://www.facebook.com/beldexofficial
Instagram: https://www.instagram.com/beldexcoin/?hl=en
LinkedIn: https://www.linkedin.com/company/beldex-coin
Medium: https://beldexcoin.medium.com/

4.2 Bounty and Airdrop

Tokens reserved for the purpose of bounty and airdrop are 80,000 BDX tokens valued at $10,000 USD. Airdrops will be conducted at regular intervals at the discretion of Beldex.

5.0 Contributions

Beldex contributors are primarily masternode validators who strengthen the network by setting up masternodes and validating blocks. To contribute to the Beldex network, you can either set up a dedicated masternode or stake on platforms that support Beldex masternode staking. Validators can also share the minimum collateral of 10000 BDX to set up a masternode and for this purpose, Beldex will partner with shared masternode pools where they can easily stake a BDX amount of their choosing and receive rewards proportional to their stake.

Independent developers who are interested in contributing may contribute to Beldex on their own volition. Developers can find the open source Beldex source code here. To contribute, leave a message on our discord channel and our team will get back to you.

References

[1] *Bitcoin, https://bitcoin.org/en/protect-your-privacy#*

[2] *Monero, https://getmonero.org*

[3] *P. Monero-Sanchez et al., "DLSAG: Non-Interactive Refund Transactions For Interoperable Payment Channels in Monero," https://eprint.iacr.org/2019/595.pdf*

[4] *Github Comment - EABE/Knacc Attack, https://github.com/monero-project/monero/issues/1673#issuecomment-312968452*

[5] *N. van Saberhagen,"CryptoNote v 2.0," https://web.archive.org/web/20201028121818/https://cryptonote.org/whitepaper.pdf*

[6] *K. M. Alonso, J. H. Joancomarti, "Monero Privacy In The Blockchain," https://eprint.iacr.org/2018/535.pdf*

[7] *W. Diffie, M. E. Hellman, "New Directions in Cryptography," https://ee.stanford.edu/~hellman/publications/24.pdf*

[8] *B. Bünz, J. Bootle, et al.,* "*Bulletproofs: Short Proofs for Confidential Transactions and More," https://eprint.iacr.org/2017/1066.pdf*

[9] *M. Quintyne-Collins,* "*Short Paper: Towards Characterizing Sybil Attacks in Cryptocurrency Mixers," https://eprint.iacr.org/2019/1111.pdf*

[10] *J. Poon, T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," https://lightning.network/lightning-network-paper.pdf*

[11] *Bitcoin Energy Consumption, https://digiconomist.net/bitcoin-energy-consumption/*

[12] *E. Ben Sasson et al., "Zerocash: Decentralized Anonymous Payments from Bitcoin," 2014 IEEE Symposium on Security and Privacy, https://ieeexplore.ieee.org/document/6956581 2014, pp. 459-474, doi: 10.1109/SP.2014.36.*

[13] *Zether protocol, https://eprint.iacr.org/2019/191.pdf*

[14] *K. Cohn-Gordon, C. Cremers, et al., "A Formal Security Analysis of the Signal Messaging Protocol," https://eprint.iacr.org/2016/1013.pdf*

[15] *How China Blocks the Tor Anonymity Network, https://www.technologyreview.com/2012/04/04/186902/how-china-blocks-the-tor-anonymity-network/*

[16] *How Google and Facebook Have Taken Over the Digital Ad Industry, Fortune, http://fortune.com/2017/01/04/google-facebook-ad-industry/*

[17] *Monero whitepaper https://www.getmonero.org/library/Zero-to-Monero-1-0-0.pdf*

[18] *Kee Jefferys, Simon Harman, Johnathan Ross, Paul McLean,* "*Private transactions, decentralised communication," https://loki.network/wp-content/uploads/2018/10/LokiWhitepaperV3_1.pdf*

[19] *Cryptonote Whitepaper https://whitepaperdatabase.com/wp-content/uploads/2017/09/Monero-whitepaper.pdf*

[20] *Ronald Cramer, Ivan Damg˚ard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In CRYPTO, pages 174–187, 1994.*