P-DART: Bulletproofs-based Anonymous Regulated Transactions

Technical Specification

Polymesh Association

October 28, 2025

Contents

1 P-DART Math Documentation **4**
1.1 Table of Contents . . . 4

2 Notation, Prerequisites and System Model **5**
2.1 Notation . . . 5
2.2 Prerequisites . . . 5
2.2.1 Sigma protocol . . . 5
2.2.2 Chaum-Pedersen Proof of Equality for Shared Witnesses . . . 6
2.2.3 Elgamal and its variations . . . 7
2.2.4 Curve Trees . . . 7
2.2.5 Bulletproofs . . . 8
2.3 System model . . . 8
2.3.1 Assets . . . 8
2.3.2 Accounts and Settlements . . . 9
2.3.3 Accumualtors . . . 10
2.3.4 Asset-account mapping . . . 11
2.4 Key registration . . . 11

3 Account Registration **13**
3.1 Account registration . . . 13
3.1.1 Protocol . . . 13

4 Asset Minting **17**
4.1 Asset minting . . . 17
4.1.1 Protocol . . . 17

5 Settlement **20**
5.1 Settlement . . . 20
5.1.1 Leg . . . 20
5.1.2 Leg creation proof . . . 21

6 Sender/Receiver Affirmations **26**
6.1 Affirmations . . . 26
6.1.1 Protocol . . . 26
6.2 Fee . . . 31
6.2.1 Account registration . . . 31
6.2.2 Account top-up . . . 32
6.2.3 Fee payment . . . 35

7 Appendix **38**
7.1 Appendix 38
7.1.1 Constraints for refreshing ρ and s 38
7.1.2 Constraints for range proof 38

Contents

1 P-DART Math Documentation

This documentation contains the mathematical description for the zero knowledge proof of P-DART, implemented with Bulletproofs.

1.1 Table of Contents

1. **Notation, Prerequisites and System Model**
   - Notation
   - Sigma protocols
   - ElGamal encryption variants
   - Curve Trees as accumulators
   - Bulletproofs
   - System entities
2. **Account Registration**
   - Account creation
   - Zero-knowledge proof for registration
3. **Asset Minting**
   - Issuer minting assets
   - Account state transitions
4. **Settlements**
   - Leg creation and encryption
   - Settlement proof system
5. **Sender/Receiver Affirmations**
   - Account state transition proofs
   - Balance and counter updates
6. **Fee System**
   - Fee account registration
   - Fee account top-up
   - Fee payment

Appendix

2 Notation, Prerequisites and System Model

2.1 Notation

We are working in 2-cycle elliptic curve groups $\mathbb{G}_p$ and $\mathbb{G}_q$ with orders p and q respectively.

$|\mathbb{G}_p| = p, |\mathbb{G}_q| = q$

The scalar field of $\mathbb{G}_p$ is $\mathbb{Z}_p$ and its base field is $\mathbb{Z}_q$. Conversely, the scalar field of $\mathbb{G}_q$ is $\mathbb{Z}_q$ and its base field is $\mathbb{Z}_p$. Thus coordinates of points in $\mathbb{G}_p$ are in $\mathbb{Z}_q$ and coordinates of points in $\mathbb{G}_q$ are in $\mathbb{Z}_p$.

The following are the group generators (curve points):

$G, G_i, H, H_i, J \in \mathbb{G}_p$

$\widetilde{G}, \widetilde{G}_i, \widetilde{H}, \widetilde{H}_i \in \mathbb{G}_q$

The generators G_i and $\widetilde{G}_i$ can be derived using a hash-to-curve method.

For a point P, the notations $P.x$ and $P.y$ refer to its x-coordinate and y-coordinate, respectively.

We use Pallas and Vesta as the curves. The group $\mathbb{G}_p$ refers to the group of points on the Pallas curve, and the group $\mathbb{G}_q$ refers to the group of points on the Vesta curve. Thus G, G_i are generators of the Pallas curve and $\widetilde{G}, \widetilde{G}_i$ are generators of the Vesta curve. But any 2 curves with a 2-cycle can be used.

For any sequence S, $S[i]$ represents the item at the i-th index. $S[0]$ is the first item, $S[1]$ is the second item, and so on.

2.2 Prerequisites

2.2.1 Sigma protocol

For proving knowledge and equality of committed values in Pedersen commitments

Consider the following commitment to witnesses $\{w_1, ..., w_n\}$ and randomness r

$$C = r.G + \sum_{i=1}^{n} w_i.G_i$$

To prove knowledge of witnesses, following protocol is executed

Protocol 1. Prover selects random values $r_T, w_{i,T} \in \mathbb{Z}_p$ and computes T as:

$$T = r_T.G + \sum_{i=1}^{n} w_{i,T}.G_i$$

2. The prover hashes C and T to get a challenge $e \in \mathbb{Z}_p$ 3. Prover creates responses $s = \{s_r, s_1, ..., s_n\}$ as

$$s_r = r_T + e.r, \quad s_i = w_{i,T} + e.w_i$$

4. Verifier checks the proof by checking the response (s_i) relations in the exponent as

$$e.C + T \stackrel{?}{=} s_r.G + \sum_{i=1}^{n} s_i.G_i$$

2.2.2 Chaum-Pedersen Proof of Equality for Shared Witnesses

This protocol proves that a subset of k witnesses are shared between two generalized commitments, C_A and C_B ($w_i = w'_i$ for $i = 1, ..., k$).

The Prover knows the following for C_A and C_B

$$\{r_A, \{w_i\}_{i=1}^{k}, \{w_j\}_{j=k+1}^{m}\} \text{ for } C_A$$

$$\{r_B, \{w'_i\}_{i=1}^{k}, \{w'_l\}_{l=k+1}^{p}\} \text{ for } C_B$$

where the witnesses $\{w_i\}_{i=1}^{k}$ and $\{w'_i\}_{i=1}^{k}$ are the same.

$$C_A = r_A.G_A + \sum_{i=1}^{k} w_i.G_i + \sum_{j=k+1}^{m} w_j.G_j$$

$$C_B = r_B.G_B + \sum_{i=1}^{k} w'_i.G'_i + \sum_{l=k+1}^{p} w'_l.G'_l$$

Protocol 1. Prover picks common random $w_{i,T}$ for the common witnesses w_i. It selects other randoms for the other components $(r_{A,T}, r_{B,T}, w_{j,T}, w'_{l,T})$. It computes T_A and T_B:

$$T_A = r_{A,T}.G_A + \sum_{i=1}^{k} w_{i,T}.G_i + \sum_{j=k+1}^{m} w_{j,T}.G_j$$

$$T_B = r_{B,T}.G_B + \sum_{i=1}^{k} w_{i,T}.G'_i + \sum_{l=k+1}^{p} w'_{l,T}.G'_l$$

2. The prover hashes C_A, C_B, T_A, T_B to get a challenge $e \in \mathbb{Z}_p$ 3. Prover computes the common response set $\mathbf{s}_{common} = \{s_i\}_{i=1}^{k}$ for the common witnesses, and the unique responses for the remaining witnesses.

Common responses

$$s_i = w_{i,T} + e.w_i \quad \text{for } i = 1, ..., k$$

Remaining responses for C_A:

$$s_{A,r} = r_{A,T} + e.r_A, s_j = w_{j,T} + e.w_j \quad \text{for unique } j = k+1, ..., m$$

Remaining responses for C_B:

$$s_{B,r} = r_{B,T} + e.r_B, s'_l = w'_{l,T} + e.w'_l \quad \text{for unique } l = k+1, ..., p$$

4. The Verifier accepts the proof if and only if both verification equations hold. The Verifier must use the same submitted response s_i for the common witnesses components in both checks:

Verification for C_A:

$$e.C_A + T_A \stackrel{?}{=} s_{A,r}.G_A + \sum_{i=1}^{k} s_i.G_i + \sum_{j=k+1}^{m} s_j.G_j$$

Verification for C_B:

$$e.C_B + T_B \stackrel{?}{=} s_{B,r}.G_B + \sum_{i=1}^{k} s_i.G'_i + \sum_{l=k+1}^{p} s'_l.G'_l$$

2.2.3 Elgamal and its variations

Elgamal For ElGamal on G_p, a message $M \in \mathbb{G}_P$ is encrypted under a Public Key $EK(= ek.G)$ as ciphertext $C = (C_1, C_2)$ where $C_1, C_2 \in G_p$. To encrypt, select random ephemeral key $r \in \mathbb{Z}_p$ and then

$$C = (C_1, C_2) \quad \text{where} \quad C_1 = r.G \quad \text{and} \quad C_2 = M + r.EK$$

To decrypt, using secret key ek, get $M = C_2 - ek.C_1$. This variation is used to encrypt public keys.

Exponent Elgamal For ElGamal on G_p, a message $m \in \mathbb{Z}_p$ is encrypted under a Public Key $EK(= ek.G)$ as ciphertext $C = (C_1, C_2)$ where $C_1, C_2 \in G_p$. To encrypt, select random ephemeral key $r \in \mathbb{Z}_p$ and then

$$C = (C_1, C_2) \quad \text{where} \quad C_1 = r.G \quad \text{and} \quad C_2 = m.G + r.EK$$

To decrypt, using secret key ek, get $m.G = C_2 - ek.C_1$. Now solve for discrete log m in $m.G$ This variation is used to encrypt small values like account balance, transaction amount, asset id, etc.

Twisted Elgamal This variant is more useful (efficient and guarantees) when the same message is supposed to be encrypted for multiple public keys. Say the same message $M \in \mathbb{G}_P$ is to be encrypted for 2 public keys EK_1 and EK_2 with twisted Elgamal. To encrypt, select random ephemeral key $r \in \mathbb{Z}_p$ and then

$$C = (C_1, C_2, C_3) \quad \text{where} \quad C_1 = r.EK_1, \quad C_2 = r.EK_2 \quad \text{and} \quad C_3 = M + r.G$$

To decrypt, using secret key ek_1, get $M = C_3 - ek_1^{-1}.C_1$. To decrypt, using secret key ek_2, get $M = C_3 - ek_2^{-1}.C_2$.

Note that ciphertext C has 3 elements and not 4 like regular Elgamal. In general, with twisted Elgamal for n parties, ciphertext has $n + 1$ elements but each party works only 2 elements. Twisted Elgamal applies to exponent Elgamal similarly.

2.2.4 Curve Trees

We use Curve Trees 1 as accumulator that support zero knowledge proofs of membership. Think of them like merkle trees where the hash function is the non-hiding Pedersen commitment. Conceptually, the tree is created as follows:

Assume its a binary tree. The leaf of the tree are always group element, elliptic curve points in our case. Say P_0, P_1 are 2 leaves, then their parent node Q is a non-hiding (no randomness) commitment to x-coordinates of P_0, P_1 as $Q = G_0 * P_0.x + G_1 * P_1.x$. Then to form the parent of node R from Q and its siblings, the same process is followed with x-coordinates of Q and its siblings but this time the generators used are $\widetilde{G_0}, \widetilde{G_1}$ as $R = \widetilde{G_0} * Q_0.x + \widetilde{G_1} * Q_1.x$. This is because $P_0.x, P_1.x$ and $Q_0.x, Q_1.x$ are in different groups and for this reason, we need a cycle of curves. Because of the homomorphic property of the curve tree hash function (commitment), updates to the curve tree are very efficient.

The curve tree membership proof contains nodes (commitments) on the path from root to the leaf but these nodes are *randomized*, i.e. a blinding value is added to the commitment to make it a hiding commitment. So in the above example, the proof for say leaf P_0 would contain nodes

$[P'_0, Q'_0, ...]$ where $P'_0 = P_0 + G * r_0$, $Q'_0 = Q_0 + \widetilde{G} * r_1$ and so on where $[P_0, Q_0, ...]$ are the leaf P_0's path from leaf (P_0) to root.

In implementation, a public element Δ is added to a point before taking its x-coordinate and the curve tree paper explains why then taking x-coodinate is sufficient. Also, we don't use a binary tree but much larger arity like 512, 1000 or 2000

2.2.5 Bulletproofs

We use Bulletproofs as a zk-SNARK and the curve tree membership proof is expressed using R1CS circuit. We also express range proofs and some other relations as R1CS circuit. One of the benefit of using Bulletproofs is that it allows Commit-and-Prove (CP) SNARKs meaning in addition to the proof, we also get Pedersen commitments to the desired witnesses. This helps us bridge (with the Pedersen commitment) Sigma protocols and zk-SNARK proof, meaning we can ensure that certain Sigma protocol is executing over the same witness(es) as the zk-SNARK.

2.3 System model

The system has few kinds of entities: 1. Investors: These have accounts per asset and send transactions to transfer assets from their accounts or receive assets in their accounts. Each investor maintains two key pairs: an encryption key and an affirmation key. Both public keys are elements of the Pallas curve G_p and are derived from their respective secret keys. We use distinct generators G_{Enc} and G_{Aff} from $\mathbb{G}_p$. - Encryption keys: secret key $ek \in \mathbb{Z}_p$ and public key $EK = ek.G_{Enc}$ with $EK \in \mathbb{G}_p$ - Affirmation keys: secret key $sk \in \mathbb{Z}_p$ and public key $AK = sk.G_{Aff}$ with $AK \in \mathbb{G}_p$

For each investor, the chain also knows its identity ID where an ID can have any number of encryption and affirmation keys associated to it. 2. Auditors: These parties are tied to specific assets and they are supposed to be able to view all transactions for those assets. They only have an encryption key as $(ek, EK = ek.G_{Enc})$. An asset may have 0 or more auditors. 3. Mediators: These parties are also tied to specific assets and they are supposed to be able to view and accept or reject all transactions for those assets. They have both encryption and affirmation keys. An asset may have 0 or more mediators.

The encryption key is only used to decrypt the transactions while affimation keys are used to act on transactions, like approving/disapproving deposit or withdraw. Note that all public keys are in the Pallas curve.

2.3.1 Assets

Each asset has a unique numeric id and an asset can have 32 bits max and thus its maximum value is $2^{32} - 1$. These are known in code as $ASSET\_ID\_BITS$ and MAX_ASSET_ID. Each asset has an issuer which can mint that asset into its account and the chain tracks the asset issuer by its public key. The maximum total supply any asset can have is 48 bits, i.e. $2^{48} - 1$. These are known in code as $BALANCE\_BITS$ and $MAX\_BALANCE$. There are 2 types of assets, ones which can be used to pay transaction fee and the rest which cant be used for fee. Each asset can have 0 or more auditors or mediators. Each asset can optionally have 1 trustee whose public key is $pk_T \in \mathbb{G}_p$. This entity is responsible for helping with key recorvery, clawback of funds, reversal of txns etc. An account while registering for an asset that has a trustee has to encrypt certain information from his account for pk_T. The trustee does need the help of auditor/mediator to do these actions.

2.3.2 Accounts and Settlements

Contrary to many other anonymous cryptocurrencies, DART works on account model rather than UTXO. This is because we want users to prove their exact account balance and not be able hide any of it

Also, in DART assets can't be transferred unilaterally as the receiver of the asset has to agree to receive as well. An exception is portfolio move, where assets are moved between accounts with the same identity.

Each investor can have only 1 account per asset id and it can be seen as a pair $(pk, asset\_id)$. As accounts participate in transactions, their state changes. An account state is a Pedersen commitment in the Pallas curve and the i-th account state is:

$$State_i = sk.G_{Aff} + balance.G_1 + counter.G_2 + asset\_id.G_3 + \rho.G_4 + \rho^{i+2}.G_5 + s^{2^i}.G_6 + id.G_7, \quad State_i \in \mathbb{G}_p$$

where sk - the secret key for their affirmation public key id - their identity ID, to which all their public keys are associated $asset\_id$ - id of the asset for which the account is $balance$ - the current account balance $counter$ - This is number >= 0 and it can change during various transactions ρ - This is the initial nullifier secret key created during registration. ρ^{i+2} - This is the nullifier secret key for this state and used when revealing the nullifier for this state. The rationale for exponent $i + 2$ will be given in a later section. s - the randomness of the commitment. The rationale for s^{2^i} will be given in a later section.

2.3.2.1 Account state transitions For updating account state from $State_i$ to $State_{i+1}$, $State_i$ must be invalidated by revealing a **nullifier** $N = \rho^{i+2}.G_5$. The chain records N in a set and does not allow to reveal the same nullifier twice, thus preventing multiple new state transitions from the same old state. The new state will be:

$$State_{i+1} = sk.G_{Aff} + balance'.G_1 + counter'.G_2 + asset\_id.G_3 + \rho.G_4 + \rho^{i+3}.G_5 + s^{2^{i+1}}.G_6 + id.G_7, \quad State_i \in \mathbb{G}_p$$

Note that in the new state $State_{i+1}$, fields $sk, asset\_id, \rho, id$ don't change. For invalidating state $State_{i+1}$ (to transition to $State_{i+2}$), nullifier will be $\rho^{i+3}.G_5$.

During state transition from $State_i$ to $State_{i+1}$, $State_i$ must not be revealed as it allows tracking the investor. $State_{i+1}$ however is revealed.

To allow for certain operations like key recovery, clawback of funds etc, the nullifier secret key ρ^i and randomness s are created for each state in a particular way. Each new account state is supposed to have the exponent of ρ 1 more than the previous state's exponent of ρ and the randomness of new state is supposed to square of the randomness of previous state.

The starting state of an account is (balance and counter are 0 in initial state)

$$State_0 = sk.G_{Aff} + 0.G_1 + 0.G_2 + asset\_id.G_3 + \rho.G_4 + \rho^2.G_5 + s.G_6 + id.G_7$$

Then the next state is

$$State_1 = sk.G_{Aff} + balance_1.G_1 + counter_1.G_2 + asset\_id.G_3 + \rho.G_4 + \rho^3.G_5 + s^2.G_6 + id.G_7$$

The next state

$$State_2 = sk.G_{Aff} + balance_2.G_1 + counter_2.G_2 + asset\_id.G_3 + \rho.G_4 + \rho^4.G_5 + s^4.G_6 + id.G_7$$

The nullifier secret keys form these powers $\rho^2, \rho^3, \rho^4, ...$ and randomness forms these powers $s, s^2, s^4, s^8, ...$ and anyone knowing rho or s can predict subsequent values. The reason for

not constucting nullifier secret key ρ^i as ρ^2, ρ^4, ρ^8 is because $\rho^i.G_5$ is revealed which reduces security if squares are used (TODO: Link paper and add details). During account registration, i.e. creating $State_0$, if asset has a trustee, then the value s is encrypted for pk_T and published on chain.

2.3.2.2 Settlements Asset transfers in DART are done through settlements. A settlement is a transfer of 1 or more assets between 2 or more accounts. A settlement lifecyle for transfer of a single asset from party (account) A to B is typically like this: 1. An entity creates a settlement stating: transfer 10 units of asset-id at from A to B. 2. If party A agrees, it will send an affirmation txn to the chain. This involves A to change its account state such that its *balance* is decreased by 10 units, *counter* is increased by 1, and few other changes as well. A proof that the state has been correctly updated is also sent to the chain. This step will nullify (invalidate) the previous state of account A and set this new state as the current active state. 3. If party B also agrees, it will also send an affirmation txn to the chain along its new state where the *counter* is increased by 1, and some more changes and the proof that the state change is correct. Similar to above, this step will nullify (invalidate) the previous state of account B and set this new state as the current active state.

Note that the chain does not learn asset-id at or amount of 10 units or any details of accounts A and B like their public keys, identity, counter, balance, etc. But the integrity of these are enforced through zero-knowledge proofs.

Above example is just 1 variation of a settlement but settlements can be more involved. Also detailed explanation of state changes and proofs follow in subsequent sections.

2.3.2.3 Fee account The above description of account states is for assets which are not used in paying fee. For fee payment asset-ids, the account state is simpler as:

$$State_i = sk.G_{Aff} + balance.G_1 + asset\_id.G_3 + \rho.G_5 + s.G_6, \quad State_i \in \mathbb{G}_p$$

Reason for this simplicity will be given later.

The state is a Pedersen commitment in a cyclic group. Similar to above, the account state's witnesses don't have counter and "age" components.

For updating account state from $State_i$ to $State_{i+1}$, $State_i$ must be invalidated by revealing a **nullifier** $N = \rho.G_5$. Similar to above, the chain records N in a set and does not allow to reveal the same nullifier twice. The new state will be:

$$State_{i+1} = sk.G_{Aff} + balance'.G_1 + asset\_id.G_3 + \rho'.G_5 + s'.G_6, \quad State_i \in \mathbb{G}_p$$

Note that in the new state $State_{i+1}$, fields $sk, asset\_id$ don't change but new ρ', s' are sampled randomly.

2.3.3 Accumualtors

The system uses curve trees as accumulator and has 3 curve trees. 1. Asset curve tree: Each leaf of this tree corresponds to an asset and stores the $asset\_id$ and the public keys of its auditors and mediators. When issuers register a new asset, a new leaf is added to this tree. When they update existing asset, their leaf is updated accordingly. A leaf of the asset tree is a group element in $\mathbb{G}_q$ and thus a point on Vesta curve. It is Pedersen commitment to the "asset-id" and x-coordinates of the public keys of auditors and mediators as:

$$Leaf_{asset\_id} = (AT.x).\widetilde{G} + (EK_1.x).\widetilde{G_1} + (EK_2.x).\widetilde{G_2} + ... + (EK_n.x).\widetilde{G_n}, \quad Leaf_{asset\_id} \in \mathbb{G}_q$$

Here - $(AT.x)$ refers to taking the x-coordinate of point $AT = asset\_id.J \quad A \in \mathbb{G}_p$ - $(EK_i.x)$ refers to taking the x-coordinate of public key EK_i of the i-th auditor/mediator. The x-coordinates are taken using the same approach as done by curve tree paper: adding a public element Δ to EK_i and taking the x-coordinate of the result 2. Fee account curve tree: Each leaf of this tree corresponds to an account state for a fee paying asset. As fee payments are done, existing states of those accounts are invalidated by revealing the nullifier (more on that later) and new states are added. A leaf of this tree is an account state as:

$$Leaf_i = sk.G_{Aff} + balance.G_1 + asset\_id.G_2 + \rho.G_3 + s.G_4, \quad Leaf_i \in \mathbb{G}_p$$

This is an append only tree and leaves once added are never removed. And states of all fee paying accounts regardless of the asset-id are captured in this tree. 3. Main account curve tree: Each leaf of this tree corresponds to an account state for a non-fee paying asset. As corresponding transcations are done, existing states of those accounts are invalidated by revealing the nullifier (more on that later) and new states are added. A leaf of this tree is an account state as:

$$Leaf_i = sk.G_{Aff} + balance.G_1 + counter.G_2 + asset\_id.G_3 + \rho.G_4 + \rho^{i+2}.G_5 + s^{2^i}.G_6 + id.G_7, \quad Leaf_i \in \mathbb{G}_p$$

This is also an append only tree and leaves once added are never removed. And states of all non-fee paying accounts regardless of the asset-id are captured in this tree.

Reason for having 2 separate trees of different accounts is given later.

2.3.4 Asset-account mapping

The chain allows 1 public key to have only 1 account per asset-id. This is done by a Asset-account registration step where the account's public key and asset-id are revealed and the chain keeps them in a mapping disallowing that public key to register for the same asset-id again. This is to ensure that a public key can do an accurate proof-of-balance for that asset without hiding any amount of that asset.

2.4 Key registration

This is the first step where all entities- investors, auditors, mediators, register their keys. This is not registration for an asset but a proof of knowledge of secret key for the public key. A simple implementation of this is a proof of knowledge of the discrete log in these 2 relations which can be done with a folklore sigma protocol (Schnorr):

$$EK = ek.G_{Enc}, \quad AK = sk.G_{Aff}$$

But since we allow users (think institutions) to onboard a large number of keys in a single txn, we use the batch Schnorr protocol from Fig. 2 of 2 (Fig. 2 has a typo in the last equation where g^y should be g^s) and implement the key registration as: 1. Prover (user) wants to list the public keys $[(EK_1, AK_1), (EK_2, AK_2), ...(EK_n, AK_n)]$ and has the corresponding secret keys $[(ek_1, sk_1), (ek_2, sk_2), ..., (ek_n, sk_n)]$ 2. Prover picks random:

$$r_{enc} \xleftarrow{\$} \mathbb{Z}_p$$

$$r_{aff} \xleftarrow{\$} \mathbb{Z}_p$$

3. Prover creates $T_{enc} = r_{enc}.G_{Enc}, \quad T_{aff} = r_{aff}.G_{Aff}$
4. Prover hashes $[(EK_1, AK_1), (EK_2, AK_2), ...(EK_n, AK_n)]$, T_{enc}, T_{aff} to get a challenge $c \in \mathbb{Z}_p$
5. Prover creates responses as:

$$s_{enc} = r_{enc} + \sum_{i=1}^{n} c^i.ek_i$$

$$s_{aff} = r_{aff} + \sum_{i=1}^{n} c^i.sk_i$$

6. Verifier now checks:

$$s_{enc}.G_{Enc} \stackrel{?}{=} T_{enc} + \sum_{i=1}^{n} c^i.EK_i$$

$$s_{aff}.G_{Aff} \stackrel{?}{=} T_{aff} + \sum_{i=1}^{n} c^i.AK_i$$

Similarly, multiple encryption keys of auditors/mediators can be registered in one key registration proof by following the protocol above and ignoring AK_i and corresponding relations

3 Account Registration

3.1 Account registration

This process is used by an investor when it wants to start trading in an asset, i.e. be either a sender or receiver of that asset. It can only be done once by any public key for a particular asset and is done only for non-fee assets. The public key PK_{Aff} must already be registered as per the above protocol A successful completion of registration results in the following:

1. A state $State_0$ being inserted as a leaf in the accounts curve tree where

   $State_0 = sk.G_{Aff} + asset\_id.G_3 + \rho.G_4 + \rho^2.G_5 + s.G_6 + id.G_7$, balance and counter are 0 during registra

2. Pair asset-id, public key $(asset\_id, PK_{Aff}(= sk.G_{Aff}))$ is added to the Asset-account mapping
3. Nullifier $N = \rho.G_4$ is revealed by user and chain adds it to the nullifier set.
4. If asset has a trustee with public key pk_T, encryption of s for pk_T is published on chain.

Account creation

1. User picks a nonce c, appends to the asset-id at to get the combined value $at||c$. It then passes it secret key sk and $at||c$ to Poseidon2 hash to get the nullifier secret key $\rho = Poseidon2(sk, at||c)$. The nonce c is used in case user wants to re-register (account recovery, etc), it can generate a new ρ

2. User picks a random:

$$s \xleftarrow{\$} \mathbb{Z}_p$$

3. It now creates its initial account state $State_0 = sk.G_{Aff} + asset\_id.G_3 + \rho.G_4 + \rho^2.G_5 + s.G_6 + id.G_7$

3.1.1 Protocol

Here the prover (investor) wants to prove following relations:

1. $State_0 = sk.G_{Aff} + at.G_3 + \rho.G_4 + \rho^2.G_5 + s.G_6 + id.G_7$. This is equivalent to proving $State_0 = AK + at.G_3 + N + \rho^2.G_5 + s.G_6 + id.G_7$ since N and public key AK is revealed to the verifier.
2. $N = \rho.G_4$
3. $\rho = Poseidon2(sk, at||c)$
4. $\rho^2 = \rho.\rho$
5. $AK = sk.G_{Aff}$
6. If asset has associated pk_T, then ciphertext C correctly encrypts s. Since $s \in \mathbb{Z}_p$ (belongs to scalar field of Pallas curve) and is large, 255-bit in our case, if its just encrypted in exponent-Elgamal, the descryption will be impractical since the decryptor (pk_T) has to solve discrete log of a 255-bit value. Thus prover breaks it into small chunks and encrypts each chunk using exponent-Elgamal encryption. These chunks are small enough so that discrete log over them can be completeled in reasonable time (few minutes). $CHUNK\_BITS$ denotes the bit size of each chunk and NUM_CHUNKS denotes the total number of chunks. In our implementation, $CHUNK\_BITS = 48$ and $NUM\_CHUNKS = 6$ since $\lceil \frac{255}{48} \rceil = 6$ and this means 6 chunks are sufficent to encode the 255-bit value Since Elgamal is homomorphic, these encryptions of chunks can be combined together to get an encryption of the whole s which can then be used in a Sigma protocol to show its equality with the s used in account state $State_0$

Following describes the protocol assuming a pk_T exists but its simple to avoid corrsponding parts of the proof if asset does not have pk_T

Instance: $State_0, AK, at, c, N, id, C_i, pk_T, G_{Aff}, G_{Enc}, G_i, H_i$

Witness: $sk, \rho, s, s_{\text{chunks}_i}, r_{\text{enc}_i}$.

Above are the high level instance and witness values known before the protocol starts and don't include instance and witness created during protocol execution like the various blindings and T values.

3.1.1.1 Prover

1. Prover first wants to prove knowledge of ρ^2, s in the relation $\rho^2.G_5 + s.G_6 = State_0 - AK - at.G_3 - N - id.G_7$. This is just relation 1 from above rearranged where values on RHS are public. Lets call these D as $D = State_0 - AK - at.G_3 - N - id.G_7$ So prover wants to prove knowledge of ρ^2, s in $\rho.G_4 + \rho^2.G_5 + s.G_6 = D$ and will do with a Sigma protocol

2. Prover picks random $\rho_{blinding}, \rho^2_{blinding}, s_{blinding} \in \mathbb{Z}_p$ and creates $T_{state} = \rho_{blinding}.G_4 + \rho^2_{blinding}.G_5 + s_{blinding}.G_6$ $T_{null} = \rho.G_4$

3. If pk_T exists, break randomness s in NUM_CHUNKS chunks and encrypt each chunk using exponent Elgamal as: $C_i = (C_{i,0}, C_{i,1}) = (r_{\text{enc}_i}.G_{\text{Enc}}, r_{\text{enc}_i}.pk_T + s_{\text{chunks}_i}.G), \quad C_i \in \mathbb{G}_p^2$ where C_i is ciphertext of i-th chunk, s_{chunks} is the base-2^{CHUNK_BITS} representation of s and it has NUM_CHUNKS digits and $r_{enc_i} \in \mathbb{Z}_p$ is the randomness for the encryption.

4. Note that $Com_s = \sum_i 2^{i \cdot \text{CHUNK\_BITS}} \cdot C_{i_1} = \sum_i 2^{i \cdot \text{CHUNK\_BITS}} \cdot r_{\text{enc}_i} \cdot pk_T + s \cdot G$ $Com_r = \sum_i 2^{i \cdot \text{CHUNK\_BITS}} \cdot C_{i_0} = \sum_i 2^{i \cdot \text{CHUNK\_BITS}} \cdot r_{\text{enc}_i} \cdot G_{\text{Enc}}$ and the scalar multiple of pk_T in Com_s is same as the multiple of G_{Enc} in Com_r and that multiple of G is s. Prover uses Sigma protocol (Chaum Pedersen) to prove equality of both multiples: Pick random $r_{combined} \in \mathbb{Z}_p$ and reuse for s and create $T_{r-combined} = r_{combined}.G_{Enc}, \quad T_{s-combined} = r_{combined}.pk_T + s_{blinding}.G$, where $T_{r-combined}, T_{s-combined} \in \mathbb{G}_p$

5. The prover also needs to prove that each s_{chunks_i} is a $CHUNK\_BITS$-bit value else it can create large values for s_{chunks_i} which are valid base-$2^{\text{CHUNK\_BITS}}$ representation digits but are so large that can't be practically retrieved using discrete log solving algorithms. For this the prover uses Bulletproofs and enforces range proof constraints for each s_{chunk_i}

6. For proving knowledge of $r_{\text{enc}_i}, s_{\text{chunks}_i}$ the prover inializes Sigma protocols, 2 for each chunk. It first creates blindings and the commits to them as For i-th chunk, pick random $r_{\text{enc-blinding}_i}, s_{\text{chunks-blinding}_i} \in \mathbb{Z}_p$ and create $T_{\text{enc}_i} = r_{\text{enc-blinding}_i}.G_{\text{Enc}}, \quad T_{\text{chunks}_i} = r_{\text{enc-blinding}_i}.pk_T + s_{\text{chunks-blinding}_i}.G$, where $T_{\text{enc}_i}, T_{\text{chunks}_i} \in \mathbb{G}_p$

7. For proving $\rho = Poseidon2(sk, at||c)$ and $\rho^2 = \rho * \rho$, prover uses Bulletproofs constraints for Poseidon2 and a multiplication

8. Prover needs to prove that Bulletproofs constraints used in steps 5, and 7 do indeed enforce on $sk, s_{chunks_i}, \rho, \text{and}, \rho^2$. Prover uses Sigma protocol (Chaum Pedersen) to prove equality of committed values in Bulletproof's commitment (corresponding to its witnesses) and in $State_0, \text{and}, C_i$. Let C_{BP_ρ} and C_s be Bulletproof's commitments to these values:

$$C_{BP_\rho} = b.H_0 + sk.H_1 + \rho.H_2 + \rho^2.H_3, \in \mathbb{G}_p$$

$$C_{BP_s} = b'.H_0 + s_{chunks_0}.H_1 + s_{chunks_1}.H_2 + ... + s_{chunks_{NUM\_CHUNKS-1}}.H_{NUM\_CHUNKS}, \in \mathbb{G}_p$$

Here b, b' are blindings to the commitment added by Bulletproof.

9. Prover proves knowledge of above committed values and equality of these with other commitments as: Pick random $r_{BP_\rho} \in \mathbb{Z}_p$ and $r_{BP_s} \in \mathbb{Z}_p$ and create

$$T_{BP_\rho} = r_{BP_\rho}.H_0 + sk_{blinding}.H_1 + \rho_{blinding}.H_2 + \rho^2_{blinding}.H_3, \in \mathbb{G}_p$$

$$T_{BP_s} = r_{BP_s}.H_0 + s_{chunks-blinding_0}.H_1 + s_{chunks-blinding_1}.H_2 + ... + s_{chunks-blinding_{NUM\_CHUNKS-1}}.H_{NUM\_CH}$$

10. And to prove knowledge of secret key in its affirmation key AK, it picks random $sk_{blinding}$ and creates $T_{pk} = sk_{blinding}.G_{Aff}$
11. Prover hashes the following to create challenge c as:

$$c = Hash(State_0, AK, at, c, N, id, C_i, pk_T, Com_s, Com_r, C_{BP_\rho}, C_{BP_s}, T_{state}, T_{null}, T_{r-combined}, T_{s-combine}$$

12. Prover now creates responses for each sigma protocol as: $Resp_{state} = [\rho^2_{blinding} + \rho^2.c, \quad s_{blinding} + s.c]$ $Resp_{pk} = sk_{blinding} + sk.c$ $Resp_{null} = \rho_{blinding} + \rho.c$ $Resp_r = r_{\text{combined}} + \sum_i 2^{i \cdot \text{CHUNK\_BITS}}.$ $r_{\text{enc}_i} \cdot c$ Note that since prover already created response for witness s as $s_{blinding} + s.c$ in $Resp_{state}$, it has both responses for Com_s.
13. Creates responses for C_i: $Resp_{i,0} = r_{enc-blinding_i} + r_{enc_i}.c$ $Resp_{s_{chunks_{si}}} = s_{chunks-blinding_i} + s_{chunks_i}.c$ Note that $Resp_{i,0}$ is used as well for checking relation $C_{i,1}$
14. For relations, C_{BP_ρ}, C_{BP_s}, only the responses for witnesses b, b' need to be created as the responses of other witnesses are already created. $Resp_b = r_{BP_\rho} + b.c$ $Resp_{b'} = r_{BP_s} + b'.c$
15. The proof is

$$(C_{BP_\rho}, C_{BP_s}, T_{state}, T_{null}, T_{r-combined}, T_{s-combined}, T_{enc_i}, T_{chunks_i}, T_{BP_\rho}, T_{BP_s}, T_{pk}, Resp_{state}, Resp_{pk}, Res$$

3.1.1.2 Verifier

1. Verifier hashes the following to create challenge c as:

$$c = Hash(State_0, AK, at, c, N, id, C_i, pk_T, Com_s, Com_r, C_{BP_\rho}, C_{BP_s}, T_{state}, T_{null}, T_{r-combined}, T_{s-combine}$$

2. Enforces constraints in Bulletproof for Poseidon2 and a multiplication for ρ generation and $\rho^2 = \rho.\rho$ respectively
3. Enforces constraints in Bulletproof for NUM_CHUNKS range proofs, one for each chunk.
4. Verifies correctness of $State_0$ by checking: $Resp_{state}[0].G_5 + Resp_{state}[1].G_6 \stackrel{?}{=} T_{state} + State_0.c \implies (\rho^2_{blinding} + \rho^2.c).G_5 + (s_{blinding} + s.c).G_6 \stackrel{?}{=} T_{state} + State_0.c$
5. Verifies correctness of pk by checking: $Resp_{pk}.G_{Aff} \stackrel{?}{=} T_{pk} + AK.c \implies (sk_{blinding} + sk.c).G_{Aff} \stackrel{?}{=} T_{pk} + AK.c$
6. Verifies correctness of N by checking: $Resp_{null}.G_4 \stackrel{?}{=} T_{null} + N.c \implies (\rho_{blinding} + \rho.c).G_4 \stackrel{?}{=} T_{null} + N.c$

7. Verifier creates Com_r and Com_s as: $Com_r = \sum_i 2^{i \cdot \text{CHUNK\_BITS}} \cdot C_{i_0}$ $Com_s = \sum_i 2^{i \cdot \text{CHUNK\_BITS}} \cdot C_{i_1}$
8. Verifies correctness of Com_r by checking: $Resp_r.G_{enc} \stackrel{?}{=} T_{r-combined} + Com_r.c \implies (r_{\text{combined}} + \sum_i 2^{i \cdot \text{CHUNK\_BITS}} \cdot r_{\text{enc}_i} \cdot c) \cdot G_{\text{enc}} \stackrel{?}{=} T_{r-\text{combined}} + Com_r \cdot c$
9. Verifies correctness of Com_s by checking: $Resp_r.pk_T + Resp_{state}[1].G \stackrel{?}{=} T_{s-combined} + Com_s.c \implies (r_{\text{combined}} + \sum_i 2^{i \cdot \text{CHUNK\_BITS}} \cdot r_{\text{enc}_i} \cdot c) \cdot pk_T + (s_{\text{blinding}} + s \cdot c) \cdot G \stackrel{?}{=} T_{s-\text{combined}} + Com_s \cdot c$
10. Verifier checks response for $C_{i,0}$ as: $Resp_{i,0}.G_{Enc} \stackrel{?}{=} T_{enc_i} + C_{i,0}.c \implies (r_{enc-blinding_i} + r_{enc_i}.c).G_{Enc} \stackrel{?}{=} T_{enc_i} + C_{i,0}.c$
11. Verifier checks response for $C_{i,1}$ as: $Resp_{i,0}.pk_T + Resp_{s_{chunks_i}}.G \stackrel{?}{=} T_{chunks_i} + C_{i,1}.c$ $\implies (r_{enc-blinding_i} + r_{enc_i}.c).pk_T + (s_{chunks-blinding_i} + s_{chunks_i}.c).G \stackrel{?}{=} T_{chunks_i} + C_{i,1}.c$
12. Verifier checks response for C_{BP_ρ} as: $Resp_b.H_0 + Resp_{pk}.H_1 + Resp_{null}.H_2 + Resp_{state}[0].H_3 \stackrel{?}{=} T_{BP_\rho} + C_{BP_\rho}.c \implies (r_{BP_\rho} + b.c).H_0 + (sk_{blinding} + sk.c).H_1 + (\rho_{blinding} + \rho.c).H_2 + (\rho^2_{blinding} + \rho^2.c).H_3 \stackrel{?}{=} T_{BP_\rho} + C_{BP_\rho}.c$
13. Verifier checks response for C_{BP_s} as: $Resp_{b'}.H_0 + Resp_{s_{chunks_0}}.H_1 + Resp_{s_{chunks_1}}.H_2 + ... + Resp_{s_{chunks_{NUM_C HUNKS-1}}}.H_{NUM_C HUNKS} \stackrel{?}{=} T_{BP_s} + C_{BP_s}.c \implies (r_{BP_s} + b'.c).H_0 + (s_{chunks-blinding_0} + s_{chunks_0}).H_1 + (s_{chunks-blinding_1} + s_{chunks_1}).H_2 + ... + (s_{chunks-blinding_{NUM_C HUNKS-1}} + s_{chunks_{NUM_C HUNKS-1}}).H_{NUM_C HUNKS} \stackrel{?}{=} T_{BP_s} + C_{BP_s}.c$
14. Verifier finally checks the Bulletproofs proof that verifies all constraints, i.e for Poseidon2, multiplication and range proofs

4 Asset Minting

4.1 Asset minting

This process is done by an issuer who wants to mint some quantity of an asset that he created into his own account. The asset-id is public and so is the minted amount. Issuer's key AK and it identity id are also public. The chain ensures that the total minted amount can never exceed $MAX\_BALANCE$ and thus there is no range proof required on the balance in the new state.

A successful completion of minting results in the following: 1. The issuer's old account state $State_{old}$ is invalidated and new state $State_{new}$ is created. However, $State_{old}$ cannot be revealed as it identifies which account state is being invalidated. So a randomized version of it $State_{old_r}$ is used in relations. 2. Nullifier $N = \rho_i.G_5$ is revealed by issuer and chain adds it to the nullifier set. Here ρ_i refers to the nullifier secret key of $State_{old}$ 3. The balance in $State_{new}$ is more than balance in $State_{old}$ by the minted amount 4. $State_{new}$ is inserted as a new leaf in the curve tree.

4.1.1 Protocol

Here the prover (issuer) wants to prove following relations when minting amount v:

1. $State_{old} = sk.G_{Aff} + bal_0.G_1 + cnt.G_2 + at.G_3 + \rho.G_4 + \rho^i.G_5 + s^j.G_6 + id.G_7$. This is equivalent to proving $State_{old} = AK + bal_0.G_1 + cnt.G_2 + at.G_3 + \rho.G_4 + \rho^i.G_5 + s^j.G_6 + id.G_7$ since public key AK is revealed to the verifier.
2. Similarly $State_{new} = AK + bal_1.G_1 + cnt.G_2 + at.G_3 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 + id.G_7$. Since $bal_0 = v + bal_1$, this is equivalent to $State_{new} - v.G_1 = AK + bal_0.G_1 + cnt.G_2 + at.G_3 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 + id.G_7$
3. $State_{old}$ exists in the accumulator (account curve tree)
4. $N = \rho^i.G_5$
5. $AK = sk.G_{Aff}$
6. $\rho^{i+1} = \rho.\rho^i$
7. $s^{2.j} = s^j.s^j$
8. $bal_1 = bal_0 + v$

Instance: $State_{old_r}, State_{new}, AK, at, N, id, v, Path_r, Root, G_{Aff}, G_i, H_i, \widetilde{G}, \widetilde{G_i}, \widetilde{H}, \widetilde{H_i} \in \mathbb{G}_q$

Witness: $sk, bal_0, bal_1, cnt, \rho, \rho^i, \rho^{i+1}, s^j, s^{2.j}, Path$.

Above are the high level instance and witness values known before the protocol starts and don't include instance and witness created during protocol execution like the various blindings and T values.

4.1.1.1 Prover

1. Prover first wants to prove knowledge of $bal_0, cnt, \rho, \rho^i, s^j$ in the relation $bal_0.G_1 + cnt.G_2 + \rho.G_4 + s^j.G_6 = State_{old} - AK - at.G_3 - N - id.G_7$. This is just relation 1 from above rearranged where values on RHS are public except $State_{old}$. Now prover cannot reveal $State_{old}$. Point 5 below explains how.
2. Similarly, taking relation 2 from above, prover wants to prove knowledge of $bal_0, c, \rho, \rho^{i+1}, s^{2.j}$ in $bal_0.G_1 + cnt.G_2 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 = D$ where $D = State_{new} - v.G_1 - AK - at.G_3 - id.G_7$.
3. Prover creates nullifier $N = \rho^i.G_5$
4. Prover gets the path of the leaf $State_{old}$ as $Path$, an array of nodes (curve points), and randomizes it, i.e all nodes in the path have a blinding added to them like $b_i * H_0$ or $b_i * \widetilde{H}_0$.

This will result in the leaf $State_{old}$ being transformed into $State_{old_r} = State_{old} + b_0.H_0$ and $Path$ transforms into $Path_r$. Prover enforces the constraints for curve tree membership using $Path, Path_r, b_i$

5. Since prover knows the opening of $State_{old}$ and b_0, it can prove the knowledge of opening of $State_{old_r}$ as well. So relation 1 can be transformed as

$$bal_0.G_1 + cnt.G_2 + \rho.G_4 + \rho^i.G_5 + s^j.G_6 + b_0.H_0 = State_{old_r} - AK - at.G_3 - N - id.G_7$$

where the RHS is public. Now it starts proving the knowledge of these.

6. Prover picks random $sk_{blinding}, cnt_{blinding}, bal_{0_{blinding}}, \rho_{blinding}, \rho_{i_{blinding}}, \rho_{i+1_{blinding}}, s^j_{blinding}, s^{2.j}_{blinding}, b_{0_{bl}}$ $\mathbb{Z}_p$ and creates:

$$T_{State_{old_r}} = bal_{0_{blinding}}.G_1 + cnt_{blinding}.G_2 + \rho_{blinding}.G_4 + \rho_{i_{blinding}}.G_5 + s^j_{blinding}.G_6 + b_{0_{blinding}}.H_0, \in \mathbb{G}_p$$

$$T_{State_{new}} = bal_{0_{blinding}}.G_1 + cnt_{blinding}.G_2 + \rho_{blinding}.G_4 + \rho_{i+1_{blinding}}.G_5 + s^{2.j}_{blinding}.G_6, \in \mathbb{G}_p$$

Same blinding $bal_{0_{blinding}}$ is used for both old and new balance since the change in balance is public and can thus be used in verification check accordingly.

7. For the correctness of nullifier and knowledge of secret key, prover creates $T_{null} = \rho_{i_{blinding}}.G_5, \in \mathbb{G}_p$ $T_{pk} = sk_{blinding}.G_{Aff}, \in \mathbb{G}_p$

8. For enforcing, $\rho^{i+1} = \rho.\rho^i$, $s^{2.j} = s^j.s^j$, prover setups the constraints in Bulletproof as both of these are just multiplications. Prover uses Sigma protocol (Chaum Pedersen) to prove equality of committed values in Bulletproof's commitment (corresponding to its witnesses) and in $State_{old}$, and, $State_{new}$. Let $C_{BP_{\rho,s}}$ be the commitment to these values:

$$C_{BP_{\rho,s}} = b'.H_0 + \rho.H_1 + \rho^i.H_2 + \rho^{i+1}.H_3 + s^j.H_4 + s^{2.j}.H_5, \in \mathbb{G}_p$$

Here b' is the blinding to the commitment added by Bulletproof.

9. Prover proves knowledge of above committed values and equality of these with other commitments as: Pick random $r_{BP_{\rho,s}} \in \mathbb{Z}_p$ and create

$$T_{BP_{\rho,s}} = r_{BP_{\rho,s}}.H_0 + \rho_{blinding}.H_1 + \rho_{i_{blinding}}.H_2 + \rho_{i+1_{blinding}}.H_3 + s^j_{blinding}.H_4 + s^{2.j}_{blinding}.H_5, \in \mathbb{G}_p$$

10. Prover hashes the following to create challenge c as:

$$c = Hash(State_{old_r}, State_{new}, AK, at, N, id, v, Path_r, Root, C_{BP_{\rho,s}}, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{BP_{\rho,s}}, T_{pk}$$

11. Prover now creates responses for sigma protocol for $Resp_{state_{old_r}}$:

$$Resp_{state_{old_r}} = [bal_{0_{blinding}} + bal_0.c, \quad cnt_{blinding} + cnt.c, \quad \rho_{blinding} + \rho.c, \quad \rho_{i_{blinding}} + \rho_i.c, \quad s_{j_{blinding}} + s$$

12. Responses for sigma protocol for $Resp_{state_{new}}$:

$$Resp_{state_{new}} = [\perp, \quad \perp, \quad \perp, \quad \rho_{i+1_{blinding}} + \rho_{i+1}.c, \quad s_{2.j_{blinding}} + s_{2.j}.c]$$

The symbol $\perp$ indicates that no response is created for that witness since a response has been created for the same witness above. Because responses for bal_1, cnt and ρ are already created in $Resp_{state_{old_r}}$.

13. The response for nullifier secret key ρ^i is already created in $Resp_{state_{old_r}}$. Response for witness sk $Resp_{pk} = sk_{blinding} + sk.c$
14. For relation $C_{BP_{\rho,s}}$, only the response for witnesses b' needs to be created as the responses of other witnesses are already created in $Resp_{state_{old_r}}$ and $Resp_{state_{new}}$. $Resp_{b'} = r_{BP_{\rho,s}} + b'.c$
15. The proof is

    $(C_{BP_{\rho,s}}, State_{old_r}, State_{new}, Path_r, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{BP_{\rho,s}}, T_{pk}, Resp_{state_{old_r}}, Resp_{state_{new}}, Resp$

4.1.1.2 Verifier

1. Verifier hashes the following to create challenge c as:

   $c = Hash(State_{old_r}, State_{new}, AK, at, N, id, v, Path_r, Root, C_{BP_{\rho,s}}, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{BP_{\rho,s}}, T_{pk}$

2. Verifier enforces the Bulletproof constraints for the 2 multiplications: $\rho^{i+1} = \rho.\rho^i$ and $s^{2.j} = s^j.s^j$.
3. Verifier enforces the Bulletproof constraints for curve tree membership to verify that $Path_r$ leads to $Root$.
4. Verifies correctness of $State_{old_r}$ by checking:

   $Resp_{state_{old_r}}[0].G_1 + Resp_{state_{old_r}}[1].G_2 + Resp_{state_{old_r}}[2].G_4 + Resp_{state_{old_r}}[3].G_5 + Resp_{state_{old_r}}[4].G_6 + Res$

$\implies (bal_{0_{blinding}} + bal_0.c).G_1 + (cnt_{blinding} + cnt.c).G_2 + (\rho_{blinding} + \rho.c).G_4 + (\rho_{i_{blinding}} + \rho_i.c).G_5 + (s^j_{blinding} + s^j.c).$

5. Verifies correctness of $State_{new}$ by checking:

   $Resp_{state_{old_r}}[0].G_1 + Resp_{state_{old_r}}[1].G_2 + Resp_{state_{old_r}}[2].G_4 + Resp_{state_{new}}[3].G_5 + Resp_{state_{new}}[4].G_6 \stackrel{?}{=} T_S$

   $\implies (bal_{0_{blinding}} + bal_0.c).G_1 + (cnt_{blinding} + cnt.c).G_2 + (\rho_{blinding} + \rho.c).G_4 + (\rho_{i+1_{blinding}} + \rho_{i+1}.c).G_5 + (s^{2.j}_{blind}$

   Note that for the bal_0, cnt, and ρ witnesses, verifier reuses responses from $Resp_{state_{old_r}}$.
6. Verifies correctness of nullifier by checking:

$$Resp_{state_{old_r}}[3].G_5 \stackrel{?}{=} T_{null} + N.c$$

$$\implies (\rho_{i_{blinding}} + \rho_i.c).G_5 \stackrel{?}{=} T_{null} + N.c$$

7. Verifies knowledge of secret key by checking:

$$Resp_{pk}.G_{Aff} \stackrel{?}{=} T_{pk} + AK.c$$

$$\implies (sk_{blinding} + sk.c).G_{Aff} \stackrel{?}{=} T_{pk} + AK.c$$

8. Verifier checks response for $C_{BP_{\rho,s}}$ as:

   $Resp_{b'}.H_0 + Resp_{state_{old_r}}[2].H_1 + Resp_{state_{old_r}}[3].H_2 + Resp_{state_{new}}[3].H_3 + Resp_{state_{old_r}}[4].H_4 + Resp_{state_{ne}}$

   $\implies (r_{BP_{\rho,s}} + b'.c).H_0 + (\rho_{blinding} + \rho.c).H_1 + (\rho_{i_{blinding}} + \rho_i.c).H_2 + (\rho_{i+1_{blinding}} + \rho_{i+1}.c).H_3 + (s^j_{blinding} + s^j.c)$

9. Verifier finally checks the Bulletproofs proof for all constraints i.e. curve tree membership and multiplications

5 Settlement

5.1 Settlement

A settlement can have 1 or more **Leg**s as each Leg represents one asset transfer. Say Alice wants to send 10 units of asset id at to Bob, then such a settlement will have 1 leg with Alice as sender and Bob as receiver. If Alice wants to swap asset 10 units of at with Bob for 20 units of at', then such a settlement will have 2 legs with Alice acting as sender in one leg and recevier in other and vice-versa for Bob Settlements might be created by sender, receiver or a third party. To keep the amount, and identities of parties confidential, the settlement creator encrypts these details including asset-id so that the chain does not even learn which asset is being traded. However, the chain knows how many legs the settlement has. The encryption is done for all parties to that leg which include sender, receiver and all auditors and mediators for that asset. An assumption here is that the verifier does not need to ensure that the encryption is correctly done for the sender, receiver or mediator since they need to respond to a settlement for it to succeed so if the decryption can't be done, it will eventually fail. This isn't true for auditors and they just need to observe and can't influence the settlement anyway. Following describes the creation of a leg, its encryption, decryption and the proof that the leg is correctly generated.

5.1.1 Leg

A leg has:

1. $AK_s = sk_s.G_{Aff}$ - Affirmation key of sender
2. $AK_r = sk_r.G_{Aff}$ - Affirmation key of receiver
3. $EK_s = ek_s.G_{Enc}$ - Encryption key of sender
4. $EK_r = ek_r.G_{Enc}$ - Encryption key of receiver
5. at - asset-id being transferred
6. v - Amount of units being transferred
7. $keys$ - A list of keys of asset's auditors and mediators along with their role specified such that $keys_i = (role, EK_i)$ where $EK_i = sk_i.G_{Enc}$ is the key of the i-th auditor/mediator

5.1.1.1 Encryption The setllement creator wants to ensure that the same leg encyption is given to all parties and thus uses twisted Elgamal.

1. Since 4 items need to be encrypted, picks $r_1, r_2, r_3, r_4 \in \mathbb{Z}_p$ and creates: $CT_s = r_1.G_{Enc} + AK_s$ $CT_r = r_2.G_{Enc} + AK_r$ $CT_v = r_3.G_{Enc} + v.H$ $CT_{at} = r_4.G_{Enc} + at.H$
2. Above can be used by the sender/reciver to decrypt but they also need to prove (during affirmations) that CT_s(or CT_r) has AK_s(or AK_r) and they know its secret key. To make it efficient, the creator picks r_i such that both sender and receiver can recover them. It does that using Diffie-Hellman as: Pick a random $y \in \mathbb{Z}_p$ and create a shared secret $ss = y.G_{Enc}$ Use hash-to-field to get $(r_1, r_2, r_3, r_4) = h2f(ss)$ Now create $Eph_s = y.EK_s$ and $Eph_r = y.EK_r$
3. Now sender (or receiver) can recover ss as $ss = ek_s^{-1}.Eph_s$ (or $ss = ek_r^{-1}.Eph_r$) and then $(r_1, r_2, r_3, r_4) = h2f(ss)$
4. For each $keys_i$, create $Eph_{k_i} = [r_1.EK_i, r_2.EK_i, r_3.EK_i, r_4.EK_i]$

The leg encryption is

$$(CT_s, CT_r, CT_v, CT_{at}, Eph_s, Eph_r, [Eph_{k_0}, Eph_{k_1}, ...])$$

5.1.1.2 Decryption If a sender/receiver knows which leg corresponds to his key (from off-chain communication), it first recovers r_i as described above. 1. Then it decrypts CT_s, CT_r

to get its and ensure that its the correct key $AK_s = CT_s - r_1.G_{Enc}$ $AK_r = CT_r - r_2.G_{Enc}$ 2. To decrypt amount and asset-id, does the Elgamal decryption followed by solving discrete log $v.H = CT_v - r_3.G_{Enc}$ $at.H = CT_{at} - r_4.G_{Enc}$

Since v and at are small, 48 and 32 bits respectively, they can be recovered in reasonable ammount of time solving discrete log.

if the sender/receiver didn't know which legs corresponds to him, then he has to scan the chain and repeat step 1 for each leg to check if its the sender or receiver. Note that the sender/receiver's affirmation secret key is not needed to decrypt so they can delegate the descryption task to another service without that service being able to spend their assets but it can only track them.

For auditors/mediators, who dont have r_i but only their key, they proceed by taking inverse of secret key as sk^{-1} and:

1. $AK_s = CT_s - sk^{-1}.Eph_{k_i}[0] = CT_s - r_1.G_{Enc}$
2. $AK_r = CT_s - sk^{-1}.Eph_{k_i}[1] = CT_r - r_2.G_{Enc}$
3. $v.H = CT_v - sk^{-1}.Eph_{k_i}[2] = CT_v - r_3.G_{Enc}$
4. $at.H = CT_{at} - sk^{-1}.Eph_{k_i}[3] = CT_{at} - r_4.G_{Enc}$

v and at are recovered by solving discrete log.

5.1.2 Leg creation proof

The proof for leg creation needs to hide: - The sender and reciever's identity (or public key) - The amount being transferred - The asset being transacted. This is the most complicated part as the it needs to prove that all the transaction details are being encrypted for the asset's auditors and mediators without revealing the asset or identity of auditors and mediators.

Recall from the "System model" model section that an asset's data, i.e. its id and public keys of all its auditors and mediators are stored in a leaf of the asset curve tree. It is Pedersen commitment to the "asset-id" and x-coordinates of the public keys of auditors and mediators as

$$Leaf_{asset\_id} = (AT.x).\widetilde{G} + (EK_1.x).\widetilde{G_1} + (EK_2.x).\widetilde{G_2} + ... + (EK_n.x).\widetilde{G_n}, \quad Leaf_{asset\_id} \in \mathbb{G}_q$$

But there is a slight difference in the implementation where we include the role of the EK_i as well so the leaf actually looks like this

$$Leaf_{asset\_id} = (AT.x).\widetilde{G} + ((role_1.J + EK_1).x).\widetilde{G_1} + ((role_2.J + EK_2).x).\widetilde{G_2} + ... + ((role_n.J + EK_n).x).\widetilde{G_n}, \quad Lea$$

Here - $(AT.x)$ refers to taking the x-coordinate of point $AT = asset\_id.J$, $A \in \mathbb{G}_p$ - $(role_i.J + EK_i.x)$ refers to taking the x-coordinate of $role_i.J + EK_i$ where $role_i$ and EK_i are role and public key of the i-th auditor/mediator. If role is auditor, $role_i = 1$ else 0. The x-coordinates are taken using the same approach as done by curve tree paper: adding a public element Δ to EK_i and taking the x-coordinate of the result

Now this pushes $Leaf_{asset\_id}$ in a different group $\mathbb{G}_q$ (Vesta curve) whereas all public keys are in $\mathbb{G}_p$ (Pallas curve). We use a similar technique used by Curve tree paper. The key idea is that the asset-id and encryption keys for all auditors and mediators is committed in a single Pedersen commitment where the x-coordinate of the item "represents" the item and then we randomize each item and can treat as a group element which can be revealed. Since the $asset\_id \in \mathbb{Z}_p$, we shift it to $\mathbb{G}_p$ by multiplying it by a public $J \in \mathbb{G}_p$. Following describes the general protocol

Pedersen Commitment to Curve Points Commits to elliptic curve points by committing to their x-coordinate the same way curve trees do. Then the knowledge of those committed points can be proven along with generating a re-randomized version of each point. Given points $P_i \in \mathbb{G}_p$ as

$$A = [P_0, P_1, ..., P_n]$$

Commit to A in a Pedersen commitment of x-coordinate of each P_i, i.e.

$$P_i.x \in C = PedCom(P_0.x, P_1.x, ..., P_n.x) = \sum_{\widetilde{G}_i * P_i.x}$$

where $C \in \mathbb{G}_q$ and $P_i.x \in \mathbb{Z}_q$ where $\mathbb{G}_p, \mathbb{G}_q$ are on 2 curves which form a 2-cycle (base field of one equals scalar field of other).

1. Prover randomizes each P_i to get

$$A_r = [P_{r0}, P_{r1}, ..., P_{rn}]$$

where

$$A_r[i] = P_{r_i} = P_i + bl_i * B$$

$$B \in \mathbb{G}_p, \quad bl_i \xleftarrow{\$} \mathbb{Z}_p$$

.

2. Prover proves $\forall i, P_i \in \mathbb{G}_p$, i.e. $P_i.x, P_i.y$ are x and y coordinates of a point which lies in group $\mathbb{G}_p$ and $P_i.x, P_i.y \in \mathbb{Z}_q$.
3. Prover proves $\forall i, A_r[i] = A[i] + bl_i * B = P_i + bl_i * B$

The implementation adds a public element Δ to each P_i as mentioned in the curve tree paper.

In our usage of the above protocol, A corresponds to $[AT, role_1.J + EK_1, role_2.J + EK_2, .., role_n.J + EK_n]$

A prover (settlement creator) wants to prove following relations when creating a leg with amount v:

1. CT_v, CT_{at} correctly encrypt the amount and asset-id
2. $Eph_{k_i} = [r_1.EK_i, r_2.EK_i, r_3.EK_i, r_4.EK_i]$ where EK_i is the key of the i-th auditor and mediator. This also involves a curve tree membership proof as the creator needs to prove that the asset-id its proving about is a valid one (in the tree).
3. $v <= MAX_BALANCE$

Instance: CT_v, CT_{at}, $Path_r$, $Root$, G_{Enc}, G_i, H, H_i, J, $\widetilde{G}$, $\widetilde{G_i}$, $\widetilde{H}$, $\widetilde{H_i} \in \mathbb{G}_q$

Witness: $v, at, r_1, r_2, r_3, r_4, EK_i, Path$.

Above are the high level instance and witness values known before the protocol starts and don't include instance and witness created during protocol execution like the various blindings and T values.

5.1.2.1 Prover

1. Prover creates list of points for asset-id and *keys* as $A = [AT, role_1.J + EK_1, role_2.J + EK_2, .., role_n.J + EK_n]$
2. Create blindings $blindings = [bl_0, bl_1, bl_2, ...bl_n]$ and randomize A as $A_r = [AT_r, E_1, E_2, ..., E_n]$ where $AT_r = AT + bl_0.H_0 = at.J + bl_0.H_0$ and $E_i = role_i.J + EK_i + bl_i.H_0$. A_r will be shared with the verifier since all its items are randomized.
3. Take coordinates of all points in A as $x = [(AT + \Delta).x, (E_1 + \Delta).x, (E_2 + \Delta).x, ..., (E_n + \Delta).x]$ and $y = [(AT + \Delta).y, (E_1 + \Delta).y, (E_2 + \Delta).y, ..., (E_n + \Delta).y]$
4. Enforces constraints that for each i:

- (x_i, y_i) is a valid curve point
- $(x_i, y_i) + bl_i.H_0 = (x_{r_i}, y_{r_i}) + \Delta$ where $(x_{r_i}, y_{r_i}) = A_r[i]$. The RHS is completely public but the LHS involves a fixed point (H_0) scalar multiplication - $bl_i.H_0$ and a curve point addition - $(x_i, y_i) + bl_i.H_0$.

5. Now that there is a public E_i corresponding to each public key EK_i as $E_i = role_i.J + EK_i + bl_i.H_0$, we can state EK_i in terms of E_i as $E_i - role_i.J - bl_i.H_0 = EK_i$ since the LHS is a "blinded version" of EK_i and is different for the same EK_i in different proofs (as long as fresh bl_i is chosen). Now use it for proving relations about Eph_{k_i} as

   $Eph_{k_i} = [r_1.(E_i - role_i.J - bl_i.H_0), r_2.(E_i - role_i.J - bl_i.H_0), r_3.(E_i - role_i.J - bl_i.H_0), r_4.(E_i - role_i.J - bl_i.H$

   which is doable using folklore techniques as all group elements, E_i, H_0, J are public. Infact, $role_i.J$ is public since $role_i$ is public.
6. Now prover has to use Sigma protocol for 2 witnesses for each item of Eph_{k_i} which is expensive so we express
   - $Eph_{k_i}[0]$ in the 2 witness relation as $Eph_{k_i}[0] = r_1.(E_i - role_i.J) + r_1.bl_1.(-H_0)$
   - $Eph_{k_i}[1] = (r_2.r_1^{-1}).Eph_{k_i}[0]$ since $r_1^{-1}.Eph_{k_i}[0] = E_i - role_i.J - bl_i.H_0 = EK_i$
   - Similarly $Eph_{k_i}[2] = (r_3.r_1^{-1}).Eph_{k_i}[0]$ and
   - $Eph_{k_i}[3] = (r_4.r_1^{-1}).Eph_{k_i}[0]$
7. Another trick is that prover doesn't need to prove that the scalars $r_2.r_1^{-1}, r_3.r_1^{-1}, r_4.r_1^{-1}$ are well formed in the sigma protocols. It can enforce that in Bulletproof using multiplication relations like $r_1.r_2.r_1^{-1} = r_2$ and so on. In the sigma protocol, prover treats $r_2.r_1^{-1}, r_3.r_1^{-1}, r_4.r_1^{-1}$ as α, β, γ.
8. For proving $AT_r = at.J + bl_0.H_0$, prover picks random $at_{blinding}, bl_{0_{blinding}} \in \mathbb{Z}_p$ and creates: $T_{AT_r} = at_{blinding}.J + bl_{0_{blinding}}.H_0, \in \mathbb{G}_p$
9. For the unified Bulletproof commitment, prover creates:

   $$C_{BP} = b.H_0 + r_1.H_1 + r_2.H_2 + r_3.H_3 + r_4.H_4 + \alpha.H_5 + \beta.H_6 + \gamma.H_7 + v.H_8, \quad C_{BP} \in \mathbb{G}_p$$

   where $\alpha = r_2.r_1^{-1}$, $\beta = r_3.r_1^{-1}$, $\gamma = r_4.r_1^{-1}$.
10. Bulletproof constraints enforce:
    - $r_1.\alpha = r_2$
    - $r_1.\beta = r_3$
    - $r_1.\gamma = r_4$
    - $v <= MAX_BALANCE$ (range proof)
11. For each auditor/mediator i, prover proves $Eph_{k_i}[0] = r_1.(E_i - role_i.J) + r_1.bl_i.(-H_0)$. Prover picks random $r_{1_{blinding}}, bl_{i_{blinding}} \in \mathbb{Z}_p$ and creates: $T_{Eph_{k_i}[0]} = r_{1_{blinding}}.(E_i - role_i.J) + bl_{i_{blinding}}.(-H_0), \in \mathbb{G}_p$
12. For $Eph_{k_i}[1]$, prover picks random $\alpha_{blinding} \in \mathbb{Z}_p$ and creates: $T_{Eph_{k_i}[1]} = \alpha_{blinding}.Eph_{k_i}[0], \in \mathbb{G}_p$
13. For $Eph_{k_i}[2]$, prover picks random $\beta_{blinding} \in \mathbb{Z}_p$ and creates: $T_{Eph_{k_i}[2]} = \beta_{blinding}.Eph_{k_i}[0], \in \mathbb{G}_p$

14. For $Eph_{k_i}[3]$, prover picks random $\gamma_{blinding} \in \mathbb{Z}_p$ and creates: $T_{Eph_{k_i}[3]} = \gamma_{blinding}.Eph_{k_i}[0], \in \mathbb{G}_p$
15. For the Bulletproof commitment, prover picks random $r_{BP} \in \mathbb{Z}_p$ and creates:

    $T_{BP} = r_{BP}.H_0 + r_{1_{blinding}}.H_1 + r_{2_{blinding}}.H_2 + r_{3_{blinding}}.H_3 + r_{4_{blinding}}.H_4 + \alpha_{blinding}.H_5 + \beta_{blinding}.H_6 + \gamma_{blindi}$
16. For proving $CT_v = r_3.G_{Enc} + v.H$ and $CT_{at} = r_4.G_{Enc} + at.H$, prover picks random $r_{3_{blinding}}, v_{blinding}, r_{4_{blinding}}, at_{blinding} \in \mathbb{Z}_p$ and creates: $T_v = r_{3_{blinding}}.G_{Enc} + v_{blinding}.H, \in \mathbb{G}_p$ $T_{at} = r_{4_{blinding}}.G_{Enc} + at_{blinding}.H, \in \mathbb{G}_p$
17. Prover hashes the following to create challenge c as:

    $c = Hash(A_r, Leaf_{asset\_id_r}, Path_r, Root, CT_v, CT_{at}, C_{BP}, T_{AT_r}, T_{Eph_{k_i}[0]}, T_{Eph_{k_i}[1]}, T_{Eph_{k_i}[2]}, T_{Eph_{k_i}[3]}, T_B$
18. Prover creates responses for $Resp_{AT_r}$: $Resp_{AT_r} = [at_{blinding} + at.c, \quad bl_{0_{blinding}} + bl_0.c]$
19. For each i, prover creates responses for $Resp_{Eph_{k_i}[0]}$: $Resp_{Eph_{k_i}[0]} = [r_{1_{blinding}} + r_1.c, \quad bl_{i_{blinding}} + r_1.bl_i.c]$
20. For $Eph_{k_i}[1]$, prover creates response: $Resp_{Eph_{k_i}[1]} = \alpha_{blinding} + \alpha.c$
21. For $Eph_{k_i}[2]$, prover creates response: $Resp_{Eph_{k_i}[2]} = \beta_{blinding} + \beta.c$
22. For $Eph_{k_i}[3]$, prover creates response: $Resp_{Eph_{k_i}[3]} = \gamma_{blinding} + \gamma.c$
23. For CT_v, prover creates responses: $Resp_v = [r_{3_{blinding}} + r_3.c, \quad v_{blinding} + v.c]$ $Resp_{at} = [r_{4_{blinding}} + r_4.c, \quad \perp]$
24. For relation C_{BP}, prover creates response for blinding b: $Resp_{BP} = [r_{BP} + b.c, \quad \perp, \quad r_{2_{blinding}} + r_2.c, \quad \perp,$
25. The proof is

    $(C_{BP}, A_r, Leaf_{asset\_id_r}, Path_r, CT_v, CT_{at}, T_{AT_r}, T_{Eph_{k_i}[0]}, T_{Eph_{k_i}[1]}, T_{Eph_{k_i}[2]}, T_{Eph_{k_i}[3]}, T_{BP}, T_v, T_{at}, Resp_$

5.1.2.2 Verifier

1. Verifier hashes the following to create challenge c as:

   $c = Hash(A_r, Leaf_{asset\_id_r}, Path_r, Root, CT_v, CT_{at}, C_{BP}, T_{AT_r}, T_{Eph_{k_i}[0]}, T_{Eph_{k_i}[1]}, T_{Eph_{k_i}[2]}, T_{Eph_{k_i}[3]}, T_B$
2. Verifier enforces the Bulletproof constraints for curve point validity: for each i, (x_i, y_i) satisfies the curve equation.
3. Verifier enforces the Bulletproof constraints for randomization consistency: $(x_i, y_i) + bl_i.H_0 = (x_{r_i}, y_{r_i}) + \Delta$ for all i.
4. Verifier enforces the Bulletproof constraints for curve tree membership to verify that $Path_r$ leads to $Root$.
5. Verifier enforces the Bulletproof constraints for the multiplication relations:
   - $r_1.\alpha = r_2$
   - $r_1.\beta = r_3$
   - $r_1.\gamma = r_4$
6. Verifier enforces the Bulletproof constraint for range proof: $v <= MAX_BALANCE$.
7. Verifies correctness of AT_r by checking: $Resp_{AT_r}[0].J + Resp_{AT_r}[1].H_0 \stackrel{?}{=} T_{AT_r} + AT_r.c$
   $\implies (at_{blinding} + at.c).J + (bl_{0_{blinding}} + bl_0.c).H_0 \stackrel{?}{=} T_{AT_r} + AT_r.c$
8. For each i, verifies correctness of $Eph_{k_i}[0]$ by checking:

   $$Resp_{Eph_{k_i}[0]}[0].(E_i - role_i.J) + .Resp_{Eph_{k_i}[0]}[1].(-H_0) \stackrel{?}{=} T_{Eph_{k_i}[0]} + Eph_{k_i}[0].c$$

   $\implies (r_{1_{blinding}} + r_1.c).(E_i - role_i.J) + (bl_{i_{blinding}} + r_1.bl_i.c).(-H_0) \stackrel{?}{=} T_{Eph_{k_i}[0]} + Eph_{k_i}[0].c$

9. For each i, verifies correctness of $Eph_{k_i}[1]$ by checking: $Resp_{Eph_{k_i}[1]}.Eph_{k_i}[0] \stackrel{?}{=} T_{Eph_{k_i}[1]} + Eph_{k_i}[1].c \implies (\alpha_{blinding} + \alpha.c).Eph_{k_i}[0] \stackrel{?}{=} T_{Eph_{k_i}[1]} + Eph_{k_i}[1].c$
10. For each i, verifies correctness of $Eph_{k_i}[2]$ by checking: $Resp_{Eph_{k_i}[2]}.Eph_{k_i}[0] \stackrel{?}{=} T_{Eph_{k_i}[2]} + Eph_{k_i}[2].c \implies (\beta_{blinding} + \beta.c).Eph_{k_i}[0] \stackrel{?}{=} T_{Eph_{k_i}[2]} + Eph_{k_i}[2].c$
11. For each i, verifies correctness of $Eph_{k_i}[3]$ by checking: $Resp_{Eph_{k_i}[3]}.Eph_{k_i}[0] \stackrel{?}{=} T_{Eph_{k_i}[3]} + Eph_{k_i}[3].c \implies (\gamma_{blinding} + \gamma.c).Eph_{k_i}[0] \stackrel{?}{=} T_{Eph_{k_i}[3]} + Eph_{k_i}[3].c$
12. Verifier checks response for C_{BP} as:

    $$Resp_{BP}[0].H_0 + Resp_{Eph_{k_i}[0]}[0].H_1 + Resp_{BP}[2].H_2 + Resp_v[0].H_3 + Resp_{at}[0].H_4 + Resp_{Eph_{k_i}[1]}.H_5 + Resp_{E}$$

    $$\implies (r_{BP} + b.c).H_0 + (r_{1_{blinding}} + r_1.c).H_1 + (r_{2_{blinding}} + r_2.c).H_2 + (r_{3_{blinding}} + r_3.c).H_3 + (r_{4_{blinding}} + r_4.c).H_4 +$$

13. Verifies correctness of CT_v by checking: $Resp_v[0].G_{Enc} + Resp_v[1].H \stackrel{?}{=} T_v + CT_v.c \implies (r_{3_{blinding}} + r_3.c).G_{Enc} + (v_{blinding} + v.c).H \stackrel{?}{=} T_v + CT_v.c$
14. Verifies correctness of CT_{at} by checking: $Resp_{at}[0].G_{Enc} + Resp_{at}[1].H \stackrel{?}{=} T_{at} + CT_{at}.c \implies (r_{4_{blinding}} + r_4.c).G_{Enc} + (at_{blinding} + at.c).H \stackrel{?}{=} T_{at} + CT_{at}.c$
15. Verifier finally checks the Bulletproofs proof for all constraints i.e. curve point validity, randomization consistency, curve tree membership, multiplication relations, and range proof

6 Sender/Receiver Affirmations

6.1 Affirmations

Once has settlment has been posted on chain, sender/receiver send their agreement along with a proof which proves that they are the party to that leg and they have done a correct account state transition, invalidating previous state by submitting the nullifier and their revealing their new state. Note that they don't point to their old state in the accumulator.

For an account state to transition from $State_{old}$ to $State_{new}$, - secret key sk, identity id and asset-id at should not change - nullifer should be revealed as $\rho^i.G_5$ - nullifier secret key ρ^i and randomness s^j should change as: $\rho^{i+1} = \rho.\rho^i$, $s^{2.j} = s^j.s^j$

Old state

$$State_{old} = sk.G_{Aff} + balance.G_1 + counter.G_2 + at.G_3 + \rho.G_4 + \rho^i.G_5 + s^j.G_6 + id.G_7 \quad State_{old} \in \mathbb{G}_p$$

New state

$$State_{new} = sk.G_{Aff} + balance'.G_1 + counter'.G_2 + at.G_3 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 + id.G_7 \quad State_{new} \in \mathbb{G}_p$$

balance might or might not be same as *balance′* depending on who is affirming or what kind of transaction it is. *counter′* might be 1 more or 1 less than *counter* depending on the kind of transaction. Also, it must be proved that the $State_{old}$ has same asset-id at that is in the leg and has the public key for the same secret key as in the account.

For a single leg settlement of amount v 1. When a sender affirms a leg, its new account state's balance should be v less than the old account's and its counter should increase by 1. 2. When a receiver affirms a leg, its new account state's balance should be same as the old account's and its counter should increase by 1. 3. Sender can revert its affirmation by creating a new account state which has the balance increased by v and counter decreased by 1. 4. Receiver can revert its affirmation by creating a new account state with counter decreased by 1. 5. Once a settlement has been confirmed, i.e. it cant be reverted, - sender can send a counter update txn to decrease its counter by 1 - receiver can send a claim funds txn to increases its balance by v and decrease its counter by 1.

Table describing changes to balance and counter for various txns. $_s$ and $_r$ refer to txn sent by sender and receiver respectively.

Transaction Type	Description	Balance Change	Counter Change
Affirm_s	Sender affirms a leg of amount v	**-v**	**+1**
Affirm_r	Receiver affirms a leg	**0**	**+1**
Claim_r	Receiver claims funds after settlement	**+v**	**-1**
CntUpd_s	Sender updates counter after settlement	**0**	**-1**
Reverse_s	Sender reverses their affirmation	**+v**	**-1**
Reverse_r	Receiver reverses their affirmation	**0**	**-1**

6.1.1 Protocol

This section describes a unified account state transition proof which can handle all the above scenarios and parts of this proof can be omitted when balance doesn't change Here the prover (sender/receiver) wants to prove following relations when sending affirmation for leg with amount v:

1. $State_{old} = sk.G_{Aff} + bal_0.G_1 + cnt_0.G_2 + at.G_3 + \rho.G_4 + \rho^i.G_5 + s^j.G_6 + id.G_7$
2. Similarly $State_{new} = sk.G_{Aff} + bal_1.G_1 + cnt_1.G_2 + at.G_3 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 + id.G_7$
3. $State_{old}$ exists in the accumulator (account curve tree)
4. $N = \rho^i.G_5$
5. $\rho^{i+1} = \rho.\rho^i$
6. $s^{2.j} = s^j.s^j$
7. If balance change is expected,
   - $bal_1 = bal_0 - v$ (for **Affirm_s**)
   - $bal_1 = bal_0 + v$ (for **Claim_r** or **Reverse_s**)
8. Counter change:
   - $cnt_1 = cnt_0 + 1$ (**Affirm_s**, **Affirm_r**)
   - $cnt_1 = cnt_0 - 1$ (**Claim_r**, **CntUpd_s**, **Reverse_s**, **Reverse_r**)
9. Since the change to counter is public, relation for $State_{new}$ can be expressd as:
   - $State_{new} - G_2 = sk.G_{Aff} + bal_1.G_1 + cnt.G_2 + at.G_3 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 + id.G_7$ if counter increased by 1.
   - $State_{new} + G_2 = sk.G_{Aff} + bal_1.G_1 + cnt.G_2 + at.G_3 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 + id.G_7$ if counter decreased by 1.
10. $bal_1 <= MAX_BALANCE$ to avoid overflows
11. Prove that asset id in CT_{at} is at
12. If transition involves a balance change, prove that amount in CT_v is v
13. Prove that sk is the secret key of public key in:
    - CT_s if they are sender
    - CT_r if they are receiver

Instance: $State_{old_r}$, $State_{new}$, N, $Path_r$, $Root$, CT_s, CT_r, CT_v, CT_{at}, G_{Aff}, G_i, H, H_i, $\widetilde{G}$, $\widetilde{G_i}$, $\widetilde{H}$, $\widetilde{H_i} \in \mathbb{G}_q$

Witness: $sk, at, id, v, bal_0, bal_1, cnt, \rho, \rho^i, \rho^{i+1}, s^j, s^{2.j}, Path$.

Above are the high level instance and witness values known before the protocol starts and don't include instance and witness created during protocol execution like the various blindings and T values.

6.1.1.1 Prover

1. Prover first wants to prove knowledge of $sk, at, bal_0, cnt, \rho, \rho^i, s^j, id$ in the relation $State_{old} = sk.G_{Aff} + bal_0.G_1 + cnt_0.G_2 + at.G_3 + \rho.G_4 + \rho^i.G_5 + s^j.G_6 + id.G_7$
2. Similarly, prover wants to prove knowledge of $sk, at, bal_1, cnt, \rho, \rho^{i+1}, s^{2.j}, id$ in $State_{new}$ from relation 10.
   - If counter increased by 1: $State_{new} - G_2 = sk.G_{Aff} + bal_1.G_1 + cnt_0.G_2 + at.G_3 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 + id.G_7$
   - If counter decreased by 1: $State_{new} + G_2 = sk.G_{Aff} + bal_1.G_1 + cnt_0.G_2 + at.G_3 + \rho.G_4 + \rho^{i+1}.G_5 + s^{2.j}.G_6 + id.G_7$
3. Prover gets the path of the leaf $State_{old}$ as $Path$, an array of nodes (curve points), and randomizes it by adding blindings to all nodes. This will result in the leaf $State_{old}$ being transformed into $State_{old_r} = State_{old} + b_0.H_0$ and $Path$ transforms into $Path_r$. Prover enforces the constraints for curve tree membership using $Path, Path_r, b_i$.
4. Since prover knows the opening of $State_{old}$ and b_0, it can prove the knowledge of opening of $State_{old_r}$ as well. So relation 1 can be transformed as:

   $$sk.G_{Aff} + bal_0.G_1 + cnt_0.G_2 + at.G_3 + \rho.G_4 + \rho^i.G_5 + s^j.G_6 + id.G_7 + b_0.H_0 = State_{old_r}$$

5. Prover picks random $sk_{blinding}, at_{blinding}, bal_{0_{blinding}}, cnt_{blinding}, \rho_{blinding}, \rho_{i_{blinding}}, \rho_{i+1_{blinding}}, s^j_{blinding}, s^2_{b}$ $\mathbb{Z}_p$ and creates:

$$T_{State_{old_r}} = sk_{blinding}.G_{Aff} + bal_{0_{blinding}}.G_1 + cnt_{blinding}.G_2 + at_{blinding}.G_3 + \rho_{blinding}.G_4 + \rho_{i_{blinding}}.G_5 + s^j_{bli}$$

6. For the new state, create:

$$T_{State_{new}} = sk_{blinding}.G_{Aff} + bal_{1_{blinding}}.G_1 + cnt_{blinding}.G_2 + at_{blinding}.G_3 + \rho_{blinding}.G_4 + \rho_{i+1_{blinding}}.G_5 + s^j_{}$$

Same blindings for sk, at, cnt, ρ, id are used in both old and new state since these values don't change. If balance doesn't change, also use same blinding so $bal_{0_{blinding}} = bal_{1_{blinding}}$.

7. For the correctness of nullifier, prover creates:

$$T_{null} = \rho_{i_{blinding}}.G_5, \in \mathbb{G}_p$$

8. For enforcing $\rho^{i+1} = \rho.\rho^i$ and $s^{2.j} = s^j.s^j$, prover sets up the constraints in Bulletproof as both are multiplications. Prover uses Sigma protocol (Chaum Pedersen) to prove equality of committed values in Bulletproof's commitment and in the state commitments. Let $C_{BP_{\rho,s}}$ be the commitment:

$$C_{BP_{\rho,s}} = b'.H_0 + \rho.H_1 + \rho^i.H_2 + \rho^{i+1}.H_3 + s^j.H_4 + s^{2.j}.H_5, \in \mathbb{G}_p$$

9. If the balance changes, prover sets up constraint in Bulletproof based on transaction type:
   - If balance change is $-v$: $bal_1 = bal_0 - v$
   - If balance change is $+v$: $bal_1 = bal_0 + v$

Let $C_{BP_{bal}}$ be the commitment:

$$C_{BP_{bal}} = b''.H_0 + v.H_1 + bal_0.H_2 + bal_1.H_3, \in \mathbb{G}_p$$

10. Prover creates commitment for $C_{BP_{\rho,s}}$ by picking random $r_{BP_{\rho,s}} \in \mathbb{Z}_p$:

$$T_{BP_{\rho,s}} = r_{BP_{\rho,s}}.H_0 + \rho_{blinding}.H_1 + \rho_{i_{blinding}}.H_2 + \rho_{i+1_{blinding}}.H_3 + s^j_{blinding}.H_4 + s^{2.j}_{blinding}.H_5, \in \mathbb{G}_p$$

11. Prover creates commitment for $C_{BP_{bal}}$ by picking random $v_{blinding}, r_{BP_{bal}} \in \mathbb{Z}_p$:

$$T_{BP_{bal}} = r_{BP_{bal}}.H_0 + v_{blinding}.H_1 + bal_{0_{blinding}}.H_2 + bal_{1_{blinding}}.H_3, \in \mathbb{G}_p$$

12. For proving at in CT_{at}, picks random $r_{4_{blinding}} \in \mathbb{Z}_p$ and creates: $T_{at} = r_{4_{blinding}}.G_{Enc} + at_{blinding}.H$ 13. For proving the correctness of public key in leg, picks random $r_{pk_{blinding}} \in \mathbb{Z}_p$ and creates: $T_{pk} = r_{pk_{blinding}}.G_{Enc} + sk_{blinding}.G_{Aff}$ if sender 14. If balance changes, picks random $r_{3_{blinding}} \in \mathbb{Z}_p$ and creates: $T_v = r_{3_{blinding}}.G_{Enc} + v_{blinding}.H$ 15. Prover hashes the following to create challenge c as:

$$c = Hash(State_{old_r}, State_{new}, N, Path_r, Root, CT_s, CT_r, CT_v, CT_{at}, C_{BP_{\rho,s}}, C_{BP_{bal}}, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T$$

16. Prover creates responses for $Resp_{state_{old_r}}$:

$$Resp_{state_{old_r}} = [sk_{blinding} + sk.c, \quad bal_{0_{blinding}} + bal_0.c, \quad cnt_{blinding} + cnt_0.c, \quad at_{blinding} + at.c, \quad \rho_{blinding} + \rho.c$$

17. Prover creates responses for $Resp_{state_{new}}$. If balance changes, create response for bal_1, otherwise mark as $\perp$:

$$Resp_{state_{new}} = [\perp, \quad bal_{1_{blinding}} + bal_1.c \text{ or } \perp, \quad \perp, \quad \perp, \quad \perp, \quad \rho_{i+1_{blinding}} + \rho_{i+1}.c, \quad s^{2.j}_{blinding} + s^{2.j}.c, \quad \perp, \quad \perp]$$

18. Response for nullifier:

$$Resp_{null} = \rho_{i_{blinding}} + \rho_i.c$$

19. For relation $C_{BP_{\rho,s}}$, only the response for blinding b' needs to be created: $Resp_{b'} = r_{BP_{\rho,s}} + b'.c$ $Resp_v = v_{blinding} + v.c$ 20. For relation $C_{BP_{bal}}$, only the response for blinding b'' and amount v needs to be created: $Resp_{b''} = r_{BP_{bal}} + b''.c$ 21. Response for asset-id in leg: $Resp_{r_4} = r_{4_{blinding}} + r_4.c$ 22. Response for public key in leg: $Resp_{pk} = r_{pk_{blinding}} + r_{pk}.c$ 23. If balance changes, response for amount in leg: $Resp_{r_3} = r_{3_{blinding}} + r_3.c$ 18. The proof is

$$(C_{BP_{\rho,s}}, C_{BP_{bal}}, State_{old_r}, State_{new}, Path_r, N, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{BP_{\rho,s}}, T_{BP_{bal}}, T_{at}, T_{pk}, T_v, Resp_{state_{old}}$$

Verifier

1. Verifier knows the transaction type txn_type which determines counter direction and balance change expectations.
2. Verifier hashes the following to create challenge c as:

   $$c = Hash(State_{old_r}, State_{new}, N, Path_r, Root, txn_type, C_{BP_{\rho,s}}, C_{BP_{bal}}, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{BP_{\rho,s}}, T$$

   3. Verifier enforces the Bulletproof constraints for the 2 multiplications: $\rho^{i+1} = \rho.\rho^i$ and $s^{2.j} = s^j.s^j$.
3. Verifier enforces the Bulletproof constraint for the balance change based on txn_type:
   - If $txn_type \in \{Affirm_s\}$: $bal_1 = bal_0 - v$
   - If $txn_type \in \{Claim_r, Reverse_s\}$: $bal_1 = bal_0 + v$
4. Verifier enforces the Bulletproof constraints for curve tree membership to verify that $Path_r$ leads to $Root$.
5. Verifies correctness of $State_{old_r}$ by checking:

   $$Resp_{state_{old_r}}[0].G_{Aff} + Resp_{state_{old_r}}[1].G_1 + Resp_{state_{old_r}}[2].G_2 + Resp_{state_{old_r}}[3].G_3 + Resp_{state_{old_r}}[4].G_4 + I$$

   $$\implies (sk_{blinding} + sk.c).G_{Aff} + (bal_{0_{blinding}} + bal_0.c).G_1 + (cnt_{blinding} + cnt_0.c).G_2 + (at_{blinding} + at.c).G_3 + (\rho_{bl}$$

6. Verifies correctness of $State_{new}$ by checking:
   - If counter increased by 1:

     $$Resp_{state_{old_r}}[0].G_{Aff} + Resp_{state_{new}}[1].G_1 + Resp_{state_{old_r}}[2].G_2 + Resp_{state_{old_r}}[3].G_3 + Resp_{state_{old_r}}[4].G$$

   - If counter decreased by 1:

     $$Resp_{state_{old_r}}[0].G_{Aff} + Resp_{state_{new}}[1].G_1 + Resp_{state_{old_r}}[2].G_2 + Resp_{state_{old_r}}[3].G_3 + Resp_{state_{old_r}}[4].G$$

   If balance doesn't change, $Resp_{state_{new}}[1]$ is $\perp$ and verifier uses $Resp_{state_{old_r}}[1]$ for both old and new balance responses.

$$\implies (sk_{blinding} + sk.c).G_{Aff} + (bal_{1_{blinding}} + bal_1.c).G_1 + (cnt_{blinding} + cnt_0.c).G_2 + (at_{blinding} + at.c).G_3 + (\rho_{blinding}$$

8. Verifies correctness of nullifier by checking: $Resp_{null}.G_5 \stackrel{?}{=} T_{null} + N.c \implies (\rho_{i_{blinding}} + \rho_i.c).G_5 \stackrel{?}{=} T_{null} + N.c$
9. Verifier checks response for $C_{BP_{\rho,s}}$ as:

   $$Resp_{b'}.H_0 + Resp_{state_{old_r}}[4].H_1 + Resp_{state_{old_r}}[5].H_2 + Resp_{state_{new}}[5].H_3 + Resp_{state_{old_r}}[6].H_4 + Resp_{state_{ne}}$$

   $$\implies (r_{BP_{\rho,s}} + b'.c).H_0 + (\rho_{blinding} + \rho.c).H_1 + (\rho_{i_{blinding}} + \rho_i.c).H_2 + (\rho_{i+1_{blinding}} + \rho_{i+1}.c).H_3 + (s^j_{blinding} + s^j.c)$$

10. Verifier checks response for $C_{BP_{bal}}$ as: $Resp_{b''}.H_0 + Resp_v.H_1 + Resp_{state_{old_r}}[1].H_2 + Resp_{state_{new}}[1].H_3 \stackrel{?}{=} T_{BP_{bal}} + C_{BP_{bal}}.c \implies (r_{BP_{bal}} + b''.c).H_0 + (v_{blinding} + v.c).H_1 + (bal_{0_{blinding}} + bal_0.c).H_2 + (bal_{1_{blinding}} + bal_1.c).H_3 \stackrel{?}{=} T_{BP_{bal}} + C_{BP_{bal}}.c$

11. Verifier checks response for CT_{at} by checking: $Resp_{r_4}.G_{Enc} + Resp_{state_{old_r}}[3].H \stackrel{?}{=} T_{at} + CT_{at}.c \implies (r_{4_{blinding}} + r_4.c).G_{Enc} + (at_{blinding} + at.c).H \stackrel{?}{=} T_{at} + CT_{at}.c$

12. Verifier checks response for public key in leg by checking: $Resp_{pk}.G_{Enc} + Resp_{state_{old_r}}[0].G_{Aff} \stackrel{?}{=} T_{pk} + CT_s.c$ (if sender) $\implies (r_{pk_{blinding}} + r_{pk}.c).G_{Enc} + (sk_{blinding} + sk.c).G_{Aff} \stackrel{?}{=} T_{pk} + CT_s.c$

$Resp_{pk}.G_{Enc} + Resp_{state_{old_r}}[0].G_{Aff} \stackrel{?}{=} T_{pk} + CT_r.c$ (if receiver) $\implies (r_{pk_{blinding}} + r_{pk}.c).G_{Enc} + (sk_{blinding} + sk.c).G_{Aff} \stackrel{?}{=} T_{pk} + CT_s.c$

13. If balance changes, verifier checks response for amount in leg by checking: $Resp_{r_3}.G_{Enc} + Resp_v.H \stackrel{?}{=} T_v + CT_v.c$

$$\implies (r_{3_{blinding}} + r_3.c).G_{Enc} + (v_{blinding} + v.c).H \stackrel{?}{=} T_v + CT_v.c$$

14. Verifier finally checks the Bulletproofs proof for all constraints i.e. curve tree membership, multiplications, and balance change

6.2 Fee

Fee accounts are different from regualar accounts and their states are tracked in a different curve tree. Fee accounts also transition from one state to another, invalidating old state by revealing the nullifier. Fee can be paid in designated assets and these are different from non-fee paying assets. These assets don't have auditors or mediators. Fee accounts support minting and spend. During both the amount minted or spent is public but the balance of new state is kept private. Also, for fee payment asset-id is always revealed. Nullifier secret key ρ and commitment randomness s are not generated deterministically but chosen randomly. They don't have an id or counter. Following shows a state transition in fee account.

$$State_{old} = sk.G_{Aff} + balance.G_1 + at.G_3 + \rho.G_5 + s.G_6, \quad State_{old} \in \mathbb{G}_p$$

$$State_{new} = sk.G_{Aff} + balance'.G_1 + at.G_3 + \rho'.G_5 + s'.G_6, \quad State_{new} \in \mathbb{G}_p$$

6.2.1 Account registration

A user can register fee account with a non-zero balance. The asset-id, balance and public key are public during registration. A successful completion of registration results in the following:

1. A state $State_0$ being inserted as a leaf in the accounts curve tree where
$$State_0 = sk.G_{Aff} + balance.G_1 + at.G_3 + \rho.G_5 + s.G_6$$
2. Pair asset-id, public key ($asset\_id$, $PK_{Aff}(= sk.G_{Aff})$) is added to the Asset-account mapping

6.2.1.1 Protocol

Here the prover (investor) wants to prove following relations:

1. $State_0 = sk.G_{Aff} + balance.G_1 + at.G_3 + \rho.G_5 + s.G_6$. This is equivalent to proving $State_0 - AK - balance.G_1 - at.G_3 = \rho.G_5 + s.G_6$ since $AK, balance, at$ are revealed to the verifier.
2. $AK = sk.G_{Aff}$

6.2.1.1.1 Prover

1. Prover wants to prove knowledge of ρ, s in the relation $\rho.G_5 + s.G_6 = State_0 - AK - balance.G_1 - at.G_3$ where the RHS is public.
2. Similarly, prover wants to prove knowledge of sk in $AK = sk.G_{Aff}$.
3. Prover picks random:
$$sk_{blinding}, \rho_{blinding}, s_{blinding} \xleftarrow{\$} \mathbb{Z}_p$$
and creates:
$$T_{state} = \rho_{blinding}.G_5 + s_{blinding}.G_6, \in \mathbb{G}_p$$
4. For proving knowledge of secret key, prover creates: $T_{pk} = sk_{blinding}.G_{Aff}, \in \mathbb{G}_p$
5. Prover hashes the following to create challenge c as: $c = Hash(State_0, AK, balance, at, T_{state}, T_{pk})$
6. Prover creates responses for $Resp_{state}$:
$$Resp_{state} = [\rho_{blinding} + \rho.c, \quad s_{blinding} + s.c]$$

7. Response for secret key: $Resp_{pk} = sk_{blinding} + sk.c$
8. The proof is
$$(State_0, AK, balance, at, T_{state}, T_{pk}, Resp_{state}, Resp_{pk})$$

6.2.1.1.2 Verifier

1. Verifier hashes the following to create challenge c as: $c = Hash(State_0, AK, balance, at, T_{state}, T_{pk})$
2. Verifies correctness of $State_0$ by checking:

$$Resp_{state}[0].G_5 + Resp_{state}[1].G_6 \stackrel{?}{=} T_{state} + (State_0 - AK - balance.G_1 - at.G_3).c$$

$$\implies (\rho_{blinding} + \rho.c).G_5 + (s_{blinding} + s.c).G_6 \stackrel{?}{=} T_{state} + (State_0 - AK - balance.G_1 - at.G_3).c$$

3. Verifies knowledge of secret key by checking: $Resp_{pk}.G_{Aff} \stackrel{?}{=} T_{pk} + AK.c \implies (sk_{blinding} + sk.c).G_{Aff} \stackrel{?}{=} T_{pk} + AK.c$

6.2.2 Account top-up

This is similar to regular account's minting txn where the balance in $State_{new}$ is increased by a public amount v and the public key $sk.G_{Aff}$ is revealed as well.

A successful completion of top-up txn results in the following:

1. The user's old account state $State_{old}$ is invalidated and new state $State_{new}$ is created. However, $State_{old}$ cannot be revealed as it identifies which account state is being invalidated. So a randomized version of it $State_{old_r}$ is used in relations.
2. Nullifier $N = \rho.G_5$ is revealed by issuer and chain adds it to the nullifier set. Here ρ refers to the nullifier secret key of $State_{old}$
3. The balance in $State_{new}$ is more than balance in $State_{old}$ by the minted amount
4. $State_{new}$ is inserted as a new leaf in the curve tree.

6.2.2.1 Protocol Here the prover (user) wants to prove following relations when minting amount v:

1. $State_{old} = sk.G_{Aff} + bal_0.G_1 + at.G_3 + \rho.G_5 + s^j.G_6$. This is equivalent to proving $State_{old} = AK + bal_0.G_1 + at.G_3 + \rho.G_5 + s^j.G_6$ since public key AK is revealed to the verifier. Since $bal_0 + v = bal_1$, this is equivalent to $State_{old} + v.G_1 = AK + bal_1.G_1 + at.G_3 + \rho.G_5 + s^j.G_6$
2. Similarly $State_{new} = AK + bal_1.G_1 + at.G_3 + \rho'.G_5 + s'.G_6$.
3. $State_{old}$ exists in the accumulator (account curve tree)
4. $N = \rho.G_5$
5. $AK = sk.G_{Aff}$
6. $bal_1 = bal_0 + v$
7. $bal_1 <= MAX_FEE_BALANCE$

Instance: $State_{old_r}, State_{new}, AK, at, N, v, Path_r, Root, G_{Aff}, G_i, H_i$

Witness: $sk, bal_0, bal_1, \rho, \rho', s, s', Path$.

Above are the high level instance and witness values known before the protocol starts and don't include instance and witness created during protocol execution like the various blindings and T values.

6.2.2.1.1 Prover

1. Prover wants to prove knowledge of bal_1, ρ, s in the relation $bal_1.G_1 + \rho.G_5 + s.G_6 = State_{old_r} + v.G_1 - AK - at.G_3$.
2. Similarly, prover wants to prove knowledge of bal_1, ρ', s' in $bal_1.G_1 + \rho'.G_5 + s'.G_6 = State_{new} - AK - at.G_3$ where the RHS is public.
3. Prover gets the path of the leaf $State_{old}$ as $Path$, an array of nodes (curve points), and randomizes it by adding blindings to all nodes. This will result in the leaf $State_{old}$ being transformed into $State_{old_r} = State_{old} + b_0.H_0$ and $Path$ transforms into $Path_r$. Prover enforces the constraints for curve tree membership using $Path, Path_r, b_i$.
4. Since prover knows the opening of $State_{old}$ and b_0, it can prove the knowledge of opening of $State_{old_r}$ as well. So relation 1 can be transformed as:

$$bal_1.G_1 + \rho.G_5 + s.G_6 + b_0.H_0 = State_{old_r} + v.G_1 - AK - at.G_3$$

   where the RHS is public.
5. Prover picks random

$$sk_{blinding}, bal_{1_{blinding}}, \rho_{blinding}, \rho'_{blinding}, s_{blinding}, s'_{blinding}, b_{0_{blinding}} \xleftarrow{\$} \mathbb{Z}_p$$

   and creates:

$$T_{State_{old_r}} = bal_{1_{blinding}}.G_1 + \rho_{blinding}.G_5 + s_{blinding}.G_6 + b_{0_{blinding}}.H_0, \in \mathbb{G}_p$$

6. For the new state, create

$$T_{State_{new}} = bal_{1_{blinding}}.G_1 + \rho'_{blinding}.G_5 + s'_{blinding}.G_6, \in \mathbb{G}_p$$

   Same blinding $bal_{1_{blinding}}$ is used for both old and new state as the equation of $State_{old_r}$ has been adjusted to keep balance witness same.
7. For the correctness of nullifier and knowledge of secret key, prover creates: $T_{null} = \rho_{blinding}.G_5, \in \mathbb{G}_p$ $T_{pk} = sk_{blinding}.G_{Aff}, \in \mathbb{G}_p$
8. For enforcing $bal_1 <= MAX_FEE_BALANCE$, prover sets up constraints in Bulletproof. Let $C_{BP_{bal}}$ be the commitment:

$$C_{BP_{bal}} = b'.H_0 + bal_1.H_1, \in \mathbb{G}_p$$

9. Prover creates commitment for $C_{BP_{bal}}$ by picking random $r_{BP_{bal}} \in \mathbb{Z}_p$:

$$T_{BP_{bal}} = r_{BP_{bal}}.H_0 + bal_{1_{blinding}}.H_1, \in \mathbb{G}_p$$

10. Prover hashes the following to create challenge c as:

    $c = Hash(State_{old_r}, State_{new}, AK, at, N, v, Path_r, Root, C_{BP_{bal}}, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{pk}, T_{BP_{bal}})$

11. Prover creates responses for $Resp_{state_{old_r}}$:

$$Resp_{state_{old_r}} = [bal_{1_{blinding}} + bal_1.c, \quad \rho_{blinding} + \rho.c, \quad s_{blinding} + s.c, \quad b_{0_{blinding}} + b_0.c]$$

11. Prover creates responses for $Resp_{state_{new}}$:

$$Resp_{state_{new}} = [\perp, \quad \rho'_{blinding} + \rho'.c, \quad s'_{blinding} + s'.c]$$

The symbol $\perp$ indicates that no response is created for bal_1 since a response has been created for the same witness in $Resp_{state_{old_r}}$.

12. Response for nullifier: $Resp_{null} = \rho_{blinding} + \rho.c$
13. Response for secret key: $Resp_{pk} = sk_{blinding} + sk.c$
14. For relation $C_{BP_{bal}}$, only the response for blinding b' needs to be created: $Resp_{b'} = r_{BP_{bal}} + b'.c$
15. The proof is

$(C_{BP_{bal}}, State_{old_r}, State_{new}, Path_r, N, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{pk}, T_{BP_{bal}}, Resp_{state_{old_r}}, Resp_{state_{new}}, R$

6.2.2.1.2 Verifier

1. Verifier hashes the following to create challenge c as:

$c = Hash(State_{old_r}, State_{new}, AK, at, N, v, Path_r, Root, C_{BP_{bal}}, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{pk}, T_{BP_{bal}})$

2. Verifier enforces the Bulletproof constraint for the balance change: $bal_1 <= MAX_FEE_BALANCE$. (Constraints in the appendix)

2. Verifier enforces the Bulletproof constraints for curve tree membership to verify that $Path_r$ leads to $Root$. (Constraints in the appendix)
3. Verifies correctness of $State_{old_r}$ by checking:

$Resp_{state_{old_r}}[0].G_1 + Resp_{state_{old_r}}[1].G_5 + Resp_{state_{old_r}}[2].G_6 + Resp_{state_{old_r}}[3].H_0 \stackrel{?}{=} T_{State_{old_r}} + (State_{old_r} +$

$\implies (bal_{1_{blinding}} + bal_1.c).G_1 + (\rho_{blinding} + \rho.c).G_5 + (s_{blinding} + s.c).G_6 + (b_{0_{blinding}} + b_0.c).H_0 \stackrel{?}{=} T_{State_{old_r}} + (State_{ol}$

5. Verifies correctness of $State_{new}$ by checking:

$Resp_{state_{old_r}}[0].G_1 + Resp_{state_{new}}[1].G_5 + Resp_{state_{new}}[2].G_6 \stackrel{?}{=} T_{State_{new}} + (State_{new} - AK - at.G_3).c$

$\implies (bal_{1_{blinding}} + bal_1.c).G_1 + (\rho'_{blinding} + \rho'.c).G_5 + (s'_{blinding} + s'.c).G_6 \stackrel{?}{=} T_{State_{new}} + (State_{new} - AK - at.G_3).c$

Note that for the bal_1 component, verifier reuses the response from $Resp_{state_{old_r}}$ as indicated by $\perp$ in $Resp_{state_{new}}$.

6. Verifies correctness of nullifier by checking: $Resp_{null}.G_5 \stackrel{?}{=} T_{null} + N.c \implies (\rho_{blinding} + \rho.c).G_5 \stackrel{?}{=} T_{null} + N.c$
7. Verifies knowledge of secret key by checking: $Resp_{pk}.G_{Aff} \stackrel{?}{=} T_{pk} + AK.c \implies (sk_{blinding} + sk.c).G_{Aff} \stackrel{?}{=} T_{pk} + AK.c$
8. Verifier checks response for $C_{BP_{bal}}$ as: $Resp_{b'}.H_0 + Resp_{state_{old_r}}[0].H_1 \stackrel{?}{=} T_{BP_{bal}} + C_{BP_{bal}}.c$

$\implies (r_{BP_{bal}} + b'.c).H_0 + (bal_{1_{blinding}} + bal_1.c).H_1 \stackrel{?}{=} T_{BP_{bal}} + C_{BP_{bal}}.c$

9. Verifier finally checks the Bulletproofs proof for all constraints i.e. curve tree membership and balance change

6.2.3 Fee payment

To pay of amount v, the old account state is transitioned to a new one whose balance is v less than the previous one. The user must always reveal the asset-id for fee.

A successful completion of fee payment results in the following:

1. The user's old account state $State_{old}$ is invalidated and new state $State_{new}$ is created. However, $State_{old}$ cannot be revealed as it identifies which account state is being invalidated. So a randomized version of it $State_{old_r}$ is used in relations.
2. Nullifier $N = \rho.G_5$ is revealed by issuer and chain adds it to the nullifier set. Here ρ refers to the nullifier secret key of $State_{old}$
3. The balance in $State_{new}$ is less than balance in $State_{old}$ by the fee amount
4. $State_{new}$ is inserted as a new leaf in the curve tree.

6.2.3.1 Protocol Here the prover (user) wants to prove following relations when minting amount v:

1. $State_{old} = sk.G_{Aff} + bal_0.G_1 + at.G_3 + \rho.G_5 + s^j.G_6$. Since $bal_0 - v = bal_1$, this is equivalent to $State_{old} + v.G_1 = sk.G_{Aff} + bal_1.G_1 + at.G_3 + \rho.G_5 + s^j.G_6$
2. Similarly $State_{new} = sk.G_{Aff} + bal_1.G_1 + at.G_3 + \rho'.G_5 + s'.G_6$.
3. $State_{old}$ exists in the accumulator (account curve tree)
4. $N = \rho.G_5$
5. $bal_1 = bal_0 - v$
6. $bal_1 <= MAX_FEE_BALANCE$

Instance: $State_{old_r}, State_{new}, at, N, v, Path_r, Root, G_{Aff}, G_i, H_i$

Witness: $sk, bal_0, bal_1, \rho, \rho', s, s', Path$.

Above are the high level instance and witness values known before the protocol starts and don't include instance and witness created during protocol execution like the various blindings and T values.

6.2.3.1.1 Prover

1. Prover wants to prove knowledge of sk, bal_1, ρ, s in the relation $bal_1.G_1 + \rho.G_5 + s.G_6 = State_{old_r} + v.G_1 - at.G_3 - sk.G_{Aff}$.
2. Similarly, prover wants to prove knowledge of sk, bal_1, ρ', s' in $bal_1.G_1 + \rho'.G_5 + s'.G_6 = State_{new} - at.G_3 - sk.G_{Aff}$ where the RHS is public.
3. Prover gets the path of the leaf $State_{old}$ as $Path$, an array of nodes (curve points), and randomizes it by adding blindings to all nodes. This will result in the leaf $State_{old}$ being transformed into $State_{old_r} = State_{old} + b_0.H_0$ and $Path$ transforms into $Path_r$. Prover enforces the constraints for curve tree membership using $Path, Path_r, b_i$.
4. Since prover knows the opening of $State_{old}$ and b_0, it can prove the knowledge of opening of $State_{old_r}$ as well. So relation 1 can be transformed as:

$$sk.G_{Aff} + bal_1.G_1 + \rho.G_5 + s.G_6 + b_0.H_0 = State_{old_r} + v.G_1 - at.G_3$$

   where the RHS is public.

5. Prover picks random:

$$sk_{blinding}, bal_{1_{blinding}}, \rho_{blinding}, \rho'_{blinding}, s_{blinding}, s'_{blinding}, b_{0_{blinding}} \xleftarrow{\$} \mathbb{Z}_p$$

and creates:

$$T_{State_{old_r}} = sk_{blinding}.G_{Aff} + bal_{1_{blinding}}.G_1 + \rho_{blinding}.G_5 + s_{blinding}.G_6 + b_{0_{blinding}}.H_0, \in \mathbb{G}_p$$

6. For the new state, create:

$$T_{State_{new}} = sk_{blinding}.G_{Aff} + bal_{1_{blinding}}.G_1 + \rho'_{blinding}.G_5 + s'_{blinding}.G_6, \in \mathbb{G}_p$$

Same blindings for sk, bal_1 are used in both old and new state since these values don't change.

7. For the correctness of nullifier, prover creates: $T_{null} = \rho_{blinding}.G_5, \in \mathbb{G}_p$
8. For enforcing $bal_1 <= MAX_FEE_BALANCE$, prover sets up constraints in Bulletproof. Let $C_{BP_{bal}}$ be the commitment: $C_{BP_{bal}} = b'.H_0 + bal_1.H_1, \in \mathbb{G}_p$
9. Prover creates commitment for $C_{BP_{bal}}$ by picking random $r_{BP_{bal}} \in \mathbb{Z}_p$: $T_{BP_{bal}} = r_{BP_{bal}}.H_0 + bal_{1_{blinding}}.H_1, \in \mathbb{G}_p$
10. Prover hashes the following to create challenge c as:

$$c = Hash(State_{old_r}, State_{new}, at, N, v, Path_r, Root, C_{BP_{bal}}, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{BP_{bal}})$$

11. Prover creates responses for $Resp_{state_{old_r}}$:

$$Resp_{state_{old_r}} = [sk_{blinding} + sk.c, \quad bal_{1_{blinding}} + bal_1.c, \quad \rho_{blinding} + \rho.c, \quad s_{blinding} + s.c, \quad b_{0_{blinding}} +$$

11. Prover creates responses for $Resp_{state_{new}}$:

$$Resp_{state_{new}} = [\perp, \quad \perp, \quad \rho'_{blinding} + \rho'.c, \quad s'_{blinding} + s'.c]$$

The symbol $\perp$ indicates that no response is created for sk, bal_1 since responses have been created for the same witnesses in $Resp_{state_{old_r}}$.

12. Response for nullifier: $Resp_{null} = \rho_{blinding} + \rho.c$
13. For relation $C_{BP_{bal}}$, only the response for blinding b' needs to be created: $Resp_{b'} = r_{BP_{bal}} + b'.c$
14. The proof is

$$(C_{BP_{bal}}, State_{old_r}, State_{new}, Path_r, N, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{BP_{bal}}, Resp_{state_{old_r}}, Resp_{state_{new}}, Resp_r$$

6.2.3.1.2 Verifier

1. Verifier hashes the following to create challenge c as:

$$c = Hash(State_{old_r}, State_{new}, at, N, v, Path_r, Root, C_{BP_{bal}}, T_{State_{old_r}}, T_{State_{new}}, T_{null}, T_{BP_{bal}})$$

2. Verifier enforces the Bulletproof constraint for the balance range: $bal_1 <= MAX_FEE_BALANCE$. (Constraints in the appendix)
3. Verifier enforces the Bulletproof constraints for curve tree membership to verify that $Path_r$ leads to $Root$. (Constraints in the appendix)
4. Verifies correctness of $State_{old_r}$ by checking:

$$Resp_{state_{old_r}}[0].G_{Aff} + Resp_{state_{old_r}}[1].G_1 + Resp_{state_{old_r}}[3].G_5 + Resp_{state_{old_r}}[4].G_6 + Resp_{state_{old_r}}[5].H_0 \stackrel{?}{=}$$

$\implies (sk_{blinding}+sk.c).G_{Aff}+(bal_{1_{blinding}}+bal_1.c).G_1+(\rho_{blinding}+\rho.c).G_5+(s_{blinding}+s.c).G_6+(b_{0_{blinding}}+b_0.c).$

5. Verifies correctness of $State_{new}$ by checking:

$Resp_{state_{old_r}}[0].G_{Aff}+Resp_{state_{old_r}}[1].G_1+Resp_{state_{new}}[3].G_5+Resp_{state_{new}}[4].G_6 \stackrel{?}{=} T_{State_{new}}+(State_{new}$

$\implies (sk_{blinding}+sk.c).G_{Aff}+(bal_{1_{blinding}}+bal_1.c).G_1+(\rho'_{blinding}+\rho'.c).G_5+(s'_{blinding}+s'.c).G_6 \stackrel{?}{=} T_{State_{new}}+(St$

Note that for the sk, bal_1 components, verifier reuses responses from $Resp_{state_{old_r}}$ as indicated by $\perp$ in $Resp_{state_{new}}$. 6. Verifies correctness of nullifier by checking: $Resp_{state_{old_r}}[3].G_5 \stackrel{?}{=} T_{null} + N.c \implies (\rho_{blinding} + \rho.c).G_5 \stackrel{?}{=} T_{null} + N.c$ 7. Verifier checks response for $C_{BP_{bal}}$ as: $Resp_{b'}.H_0 + Resp_{state_{old_r}}[1].H_1 \stackrel{?}{=} T_{BP_{bal}} + C_{BP_{bal}}.c \implies (r_{BP_{bal}} + b'.c).H_0 + (bal_{1_{blinding}} + bal_1.c).H_1 \stackrel{?}{=} T_{BP_{bal}} + C_{BP_{bal}}.c$ 8. Verifier finally checks the Bulletproofs proof for all constraints i.e. curve tree membership and balance range

7 Appendix

7.1 Appendix

7.1.1 Constraints for refreshing ρ and s

Once $\rho, \rho_i, \rho_{i+1}, s_i, s_{i+1}$ are committed in Bulletproof, we get circuit variables for each one of them. Then we enforce multiplication relations among these variables.

```
These variables are guaranteed to have these values
var_rho = \rho
var_rho_i = \rho_i
var_rho_i_plus_1 = \rho_{i+1}
var_s_i = s_i
var_s_i_plus_1 = s_{i+1}

// Assign circuit variable var_rho_i_plus_1_ to the product of var_rho and var_rho_i
var_rho_i_plus_1_ <- var_rho * var_rho_i

// Assign circuit variable var_s_i_plus_1_ to the product of var_s_i and var_s_i
var_s_i_plus_1_ <- var_s_i * var_s_i

// Enforce that values of var_rho_i_plus_1 and var_s_i_plus_1_ are same
var_rho_i_plus_1 == var_s_i_plus_1_

// Enforce that values of var_s_i_plus_1 and var_s_i_plus_1_ are same
var_s_i_plus_1 == var_s_i_plus_1_
```

7.1.2 Constraints for range proof

To prove that a value v is of n bits, i.e. $v \in [0, 2^n - 1)$, decompose v into bits and prove that each bit is indeed a bit, i.e. 0 or 1 and the appopriate linear combination of the bits is the original value v.

```
// bits is an array bits of v in little-endian representation.
bits = decompose(v)
// This will be set to different powers of 2 during execution of the loop
m = 1
for bit in bits {
  // Allocate circuit varaibles a and b and assign them values 1-bit and bit
  a <- 1 - bit
  b <- bit

  // Enforce a * b = 0, so one of (a,b) is zero
  a * b == 0

  // Enforce that a = 1 - b, so they both are 1 or 0.
  a + (b - 1) == 0

  // Add `-bit*2^i` to the linear combination
  v = v - bit * m;

  m = m * 2
}
```

```
// Enforce v = 0
v == 0
```