Orchard Update

By

Muhammad Shahid

Contents

1 Introduction

In this document, we discuss the Orchard protocol in detail, as well as the use of secure elements for shielded transactions within the Orchard protocol. The Orchard protocol was deployed as part of the Zcash Network Upgrade 5 (NU5), which was activated on the maisnnet at block height 1,687,104 on May 31, 2022. The NU5 upgrade introduced several enhancements, notably the Orchard shielded protocol. This new protocol simplifies and strengthens Zcash privacy features by improving the efficiency and security of shielded transactions. The document begins with an abstraction of the different functions used in the main protocol of Orchard. Next, we discuss the procedure for spending a valid Orchard coin. Finally, we explain how a secure element can be used for shielded Orchard transactions.

2 Abstraction

Before discussing the key components of Orchard, we will first define some terms and abstractions that are later used in the Orchard protocol.

2.1.1 Pallas and Vesta

Pallas and Vesta are the elliptic curves used in the Orchard. Vesta is used in the Orchard for the proof system, while Pallas is used in the application circuit. Both curves are designed to be efficiently implemented in ZK-SNARK circuits; however, Pallas is the curve used for the ZK-SNARK application in the Orchard.

In this document, we use the notation $\mathbb{P}$ for the group of points (x, y) that satisfy the equation of the Pallas curve $y^2 = x^3 + 5 \; mod \; q_{\mathbb{P}}$ alonge with the zero element $\mathcal{O}_{\mathbb{P}}$. Similarly, the notation $\mathbb{V}$ is used for set of points that satisfies the Vesta curve equation $y^2 = x^3 + 5 \; mod \; q_{\mathbb{V}}$, where $q_{\mathbb{P}} = 2^{254} + 45560315531419706090280762371685220353$ and $q_{\mathbb{V}} = 2^{254} + 45560315531506369815346746415080538113$ are the prime numbers. The order of $\mathbb{P}$ is $q_{\mathbb{V}}$ and the order of $\mathbb{V}$ is $q_{\mathbb{P}}$.

2.1.2 Extract Function (Extract$_{\mathbb{P}}$)

The Extract function is a mapping from the curve $\mathbb{P}$ to the field $\mathbb{Z}_{q_{\mathbb{P}}}$, denoted by $\mathrm{Extract}_{\mathbb{P}}$, defined as follows;

$$\mathrm{Extract}_{\mathbb{P}}: \mathbb{P} \cup \perp \rightarrow \mathbb{Z}_{q_{\mathbb{P}}}$$

$$\mathrm{Extract}_{\mathbb{P}}(Q) = \begin{cases} x & if \quad Q = (x, y) \\ \perp & if \quad Q = \perp \\ 0 & if \quad Q = \mathcal{O}_{\mathbb{P}} \end{cases}$$

2.1.3 Hash to Field

Hash to Field is a function defined as $hash_{to_{field}}: \mathbb{B}^n \times \mathbb{B}^m \rightarrow \mathbb{F}_{q_G}^2$, where $\mathbb{B}^n$ denote the sequence of bytes of orbitrary length. The function for the input $hash_{to_{field}}(msg, DST) = (u_0, u_1)$ is defined as follows;

- Let $\mathrm{DST}' = \mathrm{DST}||length(\mathrm{DST})$.
- Let $msg' = 0x00^{128}||\; msg \;||\; [0,128] \;||\; [0] \;||\; DST'$
- Let $b_0 = BLAKE2b - 512([0x00]^{16},\; msg')$
- Let $b_1 = BLAKE2b - 512\left([0x00]^{16},\; b_0||[1]||DST'\right)$
- Let $b_2 = BLAKE2b - 512\left([0x00]^{16},\; b_0 \oplus b_1||[2]||DST'\right)$

- Return $u_0 = b_1 \; mod \; q_G$ and $u_1 = b_2 \; mod \; q_G$.

2.1.4 Group Hash

The Group Hash is a function defined as $GroupHash^{\mathbb{G}}: \mathbb{B}^n \times \mathbb{B}^m \rightarrow \mathbb{G}$. The input to $GroupHash^{\mathbb{G}}$ consists of a pair: the first element of the pair is the domain separator, which distinguishes the usage of the function for different purposes, and the second element is the message. Let (D, M) be the input pair the $GroupHash^{\mathbb{G}}$ can be calculated as follows;

i. Let $DST = D||" - "||Curve\ name||$_XMD:BLACK_SSWU_RO_.
ii. Let $(u_0, u_1) = hash_{to_{filed}}(M, DST)$.
iii. Let $Q_0 = map\_to\_curve\_simple\_swu(u_0)$
iv. Let $Q_1 = map\_to\_curve\_simple\_swu(u_1)$

Return $iso_{map}(Q_0 + Q_1)$

2.1.5 Sinsemilla Hash Function

The Sinsemilla Hash Function is a collision-resistant hash function based on the discrete logarithm problem over elliptic curves. This hash function is specifically designed for Zcash Orchard, optimizing the use of lookups available in recent proof systems. The Sinsemilla Hash function can be denoted by $SinsemillaHashToPoint: \mathbb{B}^n \times \mathbb{B}^m \rightarrow \mathbb{P} \cup \{\perp\}$ defined as follows;

i. Compute $n = ceiling\left(\frac{length(M)}{k}\right)$
ii. Let $r = (n \times k) - length(M)$
iii. Concatenate 0^r with the message M, i.e., $M' = M||0^r$
iv. Dived the message M' into n sub blocks of size$k$, i.e., $m_1, m_2, \dots, m_n$.
v. Let $Q(D) = GroupHash^{\mathbb{P}}("z.cash\text{: SinsemillaQ"}, D)$
vi. Let $\mathcal{S}(m) = GroupHash^{\mathbb{P}}("z.cash\text{: SinsemillaS"}, m)$
vii. Define a binary operation

$$(x,y) * (x',y') = \begin{cases} (x,y) + (x',y') & if \quad (x,y) \neq \mathcal{O}_{\mathbb{P}} \neq (x',y') \\ & \qquad and \\ & \quad (x,y) \neq \perp \neq (x',y') \\ \perp & \quad otherwise \end{cases}$$

viii. Let $Acc = Q(D)$.
ix. For i form 1 upto n:
$$Acc = (Acc * \mathcal{S}(m_i)) * Acc$$

Return Acc.

2.1.6 Sinsemilla Commitments

The Sinsemilla commitment is a commitment function that is based on Sinsemilla hash function, with additional randomized point on the Pallas curve. Mathematically the commitment can be written as;

$$SinsemillaCommit_r(D, M) = \begin{cases} M' + r \cdot GroupHash^{\mathbb{P}}(D||\text{"-r"}, \text{""}) & if\ M' \neq \perp \\ \perp & otherwise \end{cases}$$

In the above equation, $M' = SinsemillaHashToPoint(D||\text{"-M"}, M)$. The Commit function is defined as follows;

$$\text{Commit}^{i_{vk}}_{\text{r}_{\text{ivk}}}(x, y) = \text{Extract}_{\mathbb{P}}\left(SinsemillaCommit_r(\text{"z.cash:Orchard-CommitIvk"}, x||y)\right).$$

2.1.7 Orchard Note Commitment

When a note is created through a transaction, only a commitment to its content is publicly disclosed in the transaction's Action description. This commitment is added to the note commitment tree when the transaction is recorded on the block chain. This ensures that the value and recipient remain private, while the ZK-SNARK proof verifies the note's existence on the block chain when it is spent. In the Orchard to create a note Sinsemilla Commitment has been used, the detail is given as follows;

$$NoteCommit_{rcm}^{Orchard}(x,y) = SinsemillaCommit_r(\text{"z.cash:Orchard-NoteComit"}, x||y)$$

2.1.8 Derive Internal FVK

The function to derive internal FVK is denoted by $\text{DeriveInternalFVK}^{\text{Orchard}}$ defined as follows;

i. Let $K = r_{ivk}$ represented in little-endian order.
ii. $r_{ivk_{internal}} = Black2b - 512(\text{"Zcash\_Expend",K,0x83}||a_k||n_k) \ mod \ r_{\mathbb{P}}$
iii. Return $(a_k, n_k, r_{ivk_{internal}})$.

2.1.9 Diversify Hash

Let $GroupHash^{\mathbb{P}}$ be as defined in 6.1.5, which is a function that map a string of bytes into the point of Pallas and Vesta Elliptic curve Point. Using the group hash the diversify hash can be calculated as follows;

$$DiversifyHash^{Orchard}(d) = \begin{cases} GroupHash^{\mathbb{P}}(\text{"z.cash:Orchard-gd"}, \text{""}) & if \ P = \mathcal{O}_{\mathbb{P}} \\ P & otherwise \end{cases}$$

Where $P = GroupHash^{\mathbb{P}}(\text{"z.cash:Orchard-gd"}, d)$.

2.1.10 $repr_{\mathbb{G}}$ Function

Let $\mathbb{G}$ be an Elliptic, then the $repr_{\mathbb{G}}$ is function from $\mathbb{G}$ to the set of bytes of length $l_{\mathbb{G}}$, defined as follows;

$$repr_{\mathbb{G}}(\mathcal{O}_{\mathbb{G}}) = 0$$

$$repr_{\mathbb{G}}((x,y)) = \begin{cases} x \ mod \ q_{\mathbb{G}} + 2^{255} & if \ y \equiv 1 \ mod \ 2 \\ x \ mod \ q_{\mathbb{G}} & if \ y \equiv 0 \ mod \ 2 \end{cases}$$

3 Orchard Key Component.

A new Orchard spending key can be generated by choosing a random sequence sk . From the spending key sk , generate the following keys, generate the spend authorization key a_{sk}, nullifier deriving key n_k and the key for commitment randomness given as follows

$$a_{sk} = Black2b - 512(\text{"Zcash\_Expend",}||sk||6) \ mod \ r_{\mathbb{P}}$$
$$n_k = Black2b - 512(\text{"Zcash\_Expend",}||sk||7) \ mod \ q_{\mathbb{P}}$$
$$r_{ivk} = Black2b - 512(\text{"Zcash\_Expend",}||sk||8) \ mod \ r_{\mathbb{P}}$$

From the spend authorization, compute the public key that validates the spend authorization, called the "validate spend authorization key" $a_k^{\mathbb{P}}$ defined as follows

$$a_k^{\mathbb{P}} = a_{sk} \cdot G^{orchard}$$
$$a_k = \text{Extract}_{\mathbb{P}}(a_{sk} \cdot G^{orchard})$$

From the n_k and a_k compute the incoming viewing key i_{vk} using the Commit function defined as follows;

$$i_{vk} = \text{Commit}_{\text{r}_{\text{ivk}}}^{i_{vk}}(a_k, n_k)$$

Let $K = r_{ivk}$ represented in little-endian order and suppose

$$R = Black2b - 512(\text{"Zcash\_Expend"},||K||0x82||a_k||n_k).$$
$$(a_k n_k, r_{ivk_{internal}}) = \text{DeriveInternalFVK}^{\text{Orchard}}(a_k, n_k, r_{ivk})$$
Let d_k be the first 32 bytes of R and o_{vk} be reaming 32 bytes of R and $K_{internal} = r_{ivk_{internal}}$ represented in little-endian order.
$$R_{internal} = Black2b - 512(\text{"Zcash\_Expend"},||K_{internal}||0x82||a_k||n_k).$$
Let $d_{k_{internal}}$ be the first 32 bytes of $R_{internal}$ and $o_{vk_{internal}}$ be reaming 32 bytes of $R_{internal}$. Afterward create a new diversified payment address from the given incoming viewing key (d_k, i_{vk}). To do this first choose a diversifier index uniformly and calculate the diversifier d and the diversified transmission key pk_d, the procedure is given as follows;
$$\mathrm{d} = \mathrm{FF1} - \mathrm{AES256}_{d_k}(\text{“”}, \mathrm{Index})$$
$$g_d^{\mathbb{P}} = DiversifyHash(d)$$
$$pk_d^{\mathbb{P}} = i_{vk} \cdot g_d^{\mathbb{P}}$$
FF1-AES256 is a format-preserving encryption algorithm that uses AES-256. It provides a secure pseudo-random permutation for a fixed empty string “” as a tweak. The relationship between the key components of the Orchard is depicted in Fig 4.

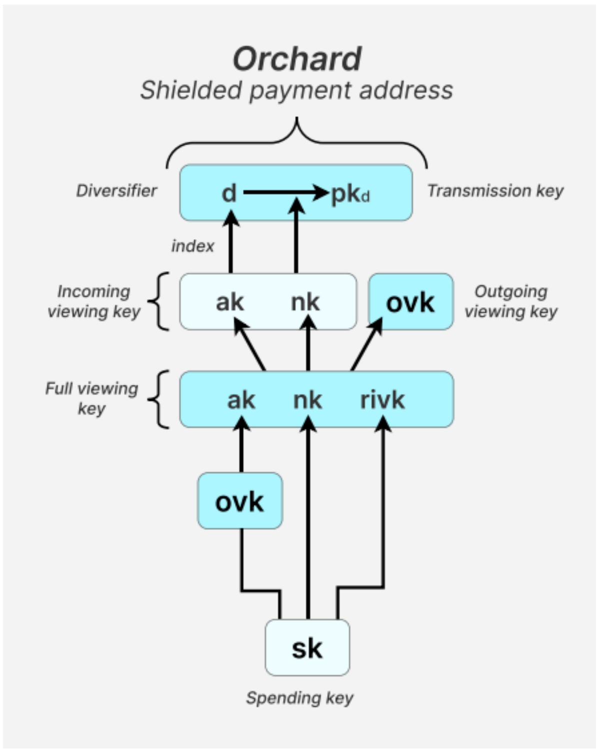


Figure 1 Orchard Key Components

4 Note

The orchard note is the set $(d, pk_d, v, \rho, \psi, rcm)$, where d is the diversifier, pk_d is diversifier public key address, v is the value of the coin, ρ and ψ is the value to compute the nullifier and rcm is the random commitment trapdoor.

5 Spending a Valid Coin (Orchard)

Let A be user with and orchard shielded payment address $(d^A, pk_{d^A}^{\mathbb{P}}, dk^A, n_{sk}^A, o_{vk}^A, i_{vk}^A, a_k^A, n_k^A, s_k^A, r_{ivk}^A)$ who wishes to send a valid note $n^A = (d^A,\ pk_{d^A}^{\mathbb{P}}, v^A, \rho^A, \psi^A, r^m)$ to a user B with orchard shielded payment addresses$(d^B, pk_{d^B}^{\mathbb{P}})$. Initially, the sender A constructs a transaction with one or more Action descriptions. For each description, the sender A chose a value v^B and the distention payment address $(d^B, pk_{d^B}^{\mathbb{P}})$, then perform the following steps.

i. Calculate that $pk_{d^B}^{\mathbb{P}}$is a type of orchard public key.
ii. Calculate $g_{d^B}^{\mathbb{P}} = DiversifyHash^{orchard}(d^B)$.
iii. Let $\rho^B = nf^A$, where nf^A, the nullifier of the input note.
iv. Derive $e_{sk} = Black2b - 512(\text{"Zcash\_Expend"}||rseed||4||\rho)\ mod\ r_{\mathbb{P}}$.

If $e_{sk} \equiv 0$, repeat the above steps.

v. Compute $rcm^B = Black2b - 512(\text{"Zcash\_Expend"}||rseed||5||\rho^B)\ mod\ r_{\mathbb{P}}$.
vi. Compute $\psi^B = Black2b - 512(\text{"Zcash\_Expend"}||rseed||9||\rho^B)\ mod\ r_{\mathbb{P}}$.

Let cv^{net} be the commitment note, which is the input note v^A minus v^B of the input note for this action transfer using the r_{cv}.

vii. Let $cm_x^B = Extract_{\mathbb{P}}\left(NoteCommit_{rcm}^{Orchard}\left(g_{d^B}^{\mathbb{P}}, v^B, \rho^B, \psi^B\right)\right)$.
viii. Let $n^B = (0x02, d^B, v^B, rseed, memo)$

In the above $memo$ is 512 byte optional part of the transection that allow the user to attached arbitrary data to the transaction. The sender then encrypt the note n^B to the recipient diversified transmission key $pk_{d^B}^{\mathbb{P}}$ with diversified base $g_d^{\mathbb{P}}$, and to the outgoing viewing key o_{vk}, resulting the transmitted note ciphertext $(e_{pk}^{\mathbb{P}}, C^{enc}, C^{out})$. The note cipher is then included in the Action description. The details of the Action description are provided in the following section.

6 Action Description.

Orchard introduces the notion of Action transfer, each of which can optionally perform an input optionally perform an output. An Action description consist of data $(cv^{net}, rt^B, nf^A, r_{k^A}, SpenAuthSig^A, cm^B, epk^A, C_{enc}^B,\ C_{enc}^B, enableSpend, enableOutput, \pi)$ included in a transaction that describes the action transfer. The detail of the data are provided as follows;

i. cv^{net}: is the value commitment to the spent note minus output note.
ii. rt^A: denote the anchor for the output treestate of the previous block.
iii. nf^A: is the nullifier for the input note n^A.
iv. rk^A: is validation key for the $SpendAuthSig^A$.
v. $SpendAuthSig^A$: is the spend authorization signature.
vi. cm^B: is the note commitment to the output note.
vii. e_{pk}: is the ephemeral key that is used shared a secret for encryption.
viii. C^{enc}: is the ciphertext component for the encrypted output note.
ix. C^{out}: is the ciphertext component that allow the holder of the outgoing cipher key to recover the recipient diversified transmission key $pk_{d^B}^{\mathbb{P}}$ and the ephemeral private key e_{sk}.

x. The $enableSpend$ is the flag that is set in order to enable the non-zero valued spends in this action.

xi. $enableOutput$: is the flag that is set to enable non-zero valued outputs in this action.

xii. π: is the zero-knowledge proof with primary input $(cv^{net}, nf^{A}, rk^{A}, cm_{x}^{A}, enableSpends, enableOutputs)$ for the action statement.

In the following subsections, we discuss the encryption and decryption procedures for encrypting the note's plaintext and ciphertext, as well as the Zero-Knowledge Proof, Binding Signature, and Authorized Spend Signature in more detail.

6.1 Encryption

In Orchard, the note n^Bshould be sent to user B securely, so that the user can later spend it. Therefore, user A encrypts the data n^Busing symmetric key encryption scheme. The symmetric algorithm AEAD_CHACHA20_POLY1305 is used in both the Sapling and Orchard protocols for encryption and decryption. Since we know that for symmetric key algorithms, the same key is used for both encryption and decryption, so, there must be a secure channel for sharing the secret key that will be used for both operations. To achieve this, both the Sapling and Orchard protocols use the Diffie-Hellman key exchange protocol to securely share the secret key. The complete details of the key exchange protocol and the encryption procedure are provided as follows:

i. Compute the shared secret $sk_{AB}^{\mathbb{P}} = e_{sk} \cdot pk_{d^B}^{\mathbb{P}}$, where $pk_{d^B}^{\mathbb{P}}$ is the point of ctEdward curve.

ii. The user A compute ephemeral public key $e_{pk}^{\mathbb{P}} = e_{sk} \cdot g_{d^B}^{\mathbb{P}}$

iii. Derive a symmetric key $K_{AB} = \text{BLAKE2b} - 256(\text{“Zcash\_OrchardKDF”}, sk_{AB}^{\mathbb{P}} || e_{pk}^{\mathbb{P}})$.

iv. Next encrypt the data $C^{enc} = ENC_{K_{AB}}(n^B)$

If $o_{vk} = \perp$

Choose a random o_{ck} and op from the set of bytes.

vi. Let $cv = repr_{\mathbb{G}}(cv)$.

vii. $cm^* = \text{Extract}_{\mathbb{G}}(cm)$.

viii. Let $o_{ck} = \text{BLAKE2b} - 256(\text{“Zcash\_Orchardock”}, o_{vk}||\text{cv}||\text{cm}^*||\text{e}_{pk}^{\mathbb{P}})$.

ix. Let $op = (pk_{d^B}^{\mathbb{P}}||e_{sk})$.

x. Let $C^{out} = ENC_{o_{ck}}(op)$.

6.2 Decryption using incoming Viewing Key

Let $(e_{pk}^{\mathbb{P}}, C^{enc}, C^{out})$ be the transmitted ciphertext from the output description. The recipient B must decrypt C^{enc} using the ephemeral key. However, only the holder of o_{vk} can decrypt the ciphertext C^{out}. The step-by-step decryption procedure is as follows:

i. Compute the share secret $sk_{AB}^{\mathbb{P}} = i_{vk}^{B} \cdot e_{pk}^{\mathbb{P}}$.

ii. Derive symmetric key $K_{AB} = \text{BLAKE2b} - 256(\text{“Zcash\_OrchardKDF”}, sk_{AB}^{\mathbb{P}} || e_{pk}^{\mathbb{P}})$.

iii. Decrypt the note ciphertext $n^B = DEC_{K_{AB}}(C^{enc})$.

iv. Extract $n^B = (0x02, d^B, v^B, rseed, memo)$.

v. Compute $g_{d^B}^{\mathbb{P}} = DiersifyHash(d^B)$

vi. Derive the public key $pk_{d^B}^{\mathbb{P}} = i_{vk}^{B} \cdot g_{d^B}^{\mathbb{P}}$.

vii. Let $\rho^B = nf^A$

viii. Compute $\psi^B = Black2b - 512(\text{"Zcash\_Expend"}||rseed||9||\rho^B) \; mod \; r_{\mathbb{P}}$.

vii. Compute $rcm^B = Black2b - 512(\text{"Zcash\_Expend"}||rseed||5||\rho^B) \; mod \; r_{\mathbb{P}}$.

ix. The note that receives B consist of $n^B = (pk_d^B, d^B, v^B, \psi^B, rcm^B)$.

The o_{vk} can only decrypt the ciphertext C^{out}. To decrypt the ciphertext C^{out}, the user have perform the following steps.

i. Let $o_{ck} = \text{BLAKE2b} - 256(\text{“Zcash\_Orchardock”}, o_{vk}||\text{cv}||\text{cm}^*||\text{e}_{pk}^{\mathbb{P}})$.

ii. Compute $op = DEC_{ock}(C^{out})$.

6.3 Action Statement π^A

The spend statement π^A assure that for a given primary input $(rt^A, cv^{net}, nf^A, rk^A, cm_x^A, enableSpend, enableOutput)$ the prover know the auxiliary inputs $(Path, Position, g_{d^A}^{\mathbb{P}}, pk_{d^B}^{\mathbb{P}}, v^A, \rho^A, \psi^A, rcm^A, cm^A, \alpha^A, n_k, rivk^A, g_{d^B}^{\mathbb{P}}, pk_{d^B}^{\mathbb{P}}, v^B, \psi^B, rcm^B)$ such that the following conditions hold;

i. Note Commitment integrity: $cm_x^A = Extract_{\mathbb{P}}\left(NoteCommit_{rcm}^{Orchard}(g_{d^A}^{\mathbb{P}}, v^A, \rho^A, \psi^A)\right)$.

ii. The path and position $(path, position)$ of cm^A in the Markle tree is valid.

iii. Value commitment integrity: $cv^{net} = ValueCommit_{rcv}^{Orchard}(v^A - v^B)$.

iv. Nullifier: $nf^A = Extract_{\mathbb{P}}(\text{ PoseidonHash}(\text{nk}_\text{A}, \rho^A) + \psi^A \; mod \; q_p + cm^A \,)$.

v. Randomized public key: $r_{k^A}^{\mathbb{P}} = \alpha^A \cdot G^{orchard} + a_{sk^A}^{\mathbb{P}}$.

vi. Diversified address: $pk_{d^A}^{\mathbb{P}} = i_{vk^A} \cdot g_{d^A}^{\mathbb{P}}$.

vii. Incoming viewing key $i_{vk}^A = \text{ Commit}_{\text{r}_{\text{ivk}}^{\text{A}}}^{i_{vk}^A}(a_k^A, n_k^A)$.

viii. New note commitment $cm^A = NoteCommit_{rcm}^{Orchard}\left(g_{d^B}^{\mathbb{P}} \;\middle\|pk_{d^B}^{\mathbb{P}}\middle\|\; v^B \;||\rho^B||\psi^B \right)$,

ix. Enable spend flag $v^A = 0$ or $enableSpends = 1$.

x. Enable Output flag $v^B = 0$ or $enableOutputs = 1$.

6.4 Balance and Binding Signature

The net value of orchard spend minus output in a transaction is called the orchard balancing value denoted by $v^{balanceOrchard}$. The consistency of $v^{balanceOrchard}$ with value commitment in Action description is enforced by the Orchard binding signature. The role of this signature in the Orchard pool is to prove that the net value spend by Action transfer is consistent with the $v^{balanceOrchard}$ field of the transaction. For the binding signature the notion of Homomorphic Pedersen commitment is introduced. Let $V^{orchard} \in \mathbb{P}^*$ and $R^{orchard} \in \mathbb{P}^*$ be the base elements. Let $\boxplus$ be the binary operation addition of private keys defined as:

$$\boxplus: \text{Sign. Privat} \times \text{Sign. Privat} \to \text{Sign. Privat}$$

Suppose $\boxminus$ be the additive inverse operation defined on the set of private key i.e., $sk \boxplus (\boxminus \, sk) = \mathcal{O}_{\boxplus}$. Let $\oplus$ be the binary operation addition defined on the set of public key:

$$\oplus: \text{Sign. Public} \times \text{Sign. Public} \to \text{Sign. Public}$$

Let $\ominus$ be additive inverse binary operation defined on the set of public key i.e., $pk \oplus (\ominus \, pk) = \mathcal{O}_{\ominus}$. Now that a transaction has n Action description with value commitment

$cv_1^{net}, \dots, cv_n^{net}$ committing to a value $v_1^{net}, \dots, v_n^{net}$ with randomness $rcv_1^{net}, \dots, rcv_n^{net}$. The orchard balancing value $v^{balanceOrchard} = \sum_{i=1}^{n} v_i^{net}$, but the validator cannot check it directly because the value are hidden by the commitment, therefore validator calculate the transection binding validating key:

$$b_{vk}^{orchard} = (\bigoplus_{i=1}^{n} cv_i^{net}) \ominus ValueCommit_0^{orchard}(v^{balanceorchard})$$

In the above equation $ValueCommit_0^{orchard}$ is a function defined as

$$ValueCommit_0^{orchard}(v^{balanceorchard}) = [\boxplus_{i=1}^{n} v_i^{net}] \cdot V^{orchard}$$

$$cv_i^{net} = [\boxplus_{i=1}^{n} v_i^{net}] \cdot V^{orchard} \oplus [\boxplus_{i=1}^{n} rcv_i^{net}] \cdot R^{orchard}$$

Implies

$$b_{vk}^{orchard} = [\boxplus_{i=1}^{n} v_i^{net}] \cdot V^{orchard} \oplus [\boxplus_{i=1}^{n} rcv_i^{net}] \cdot R^{orchard} \ominus [\boxplus_{i=1}^{n} v_i^{net}] \cdot V^{orchard}$$

$$b_{vk}^{orchard} = [\boxplus_{i=1}^{n} rcv_i^{net}] \cdot R^{orchard}$$

Since the signer know$rcv_1^{net}, rcv_2^{net}, \dots, rcv_n^{net}$, so they can calculate the corresponding signing key

$$b_{sk}^{orchard} = \boxplus_{i=1}^{n} rcv_i^{net}$$

In order to check the implementation the signer should check that either the public key $b_{vk}^{orchard}$ is equal to creating the public key from the private key $b_{sk}^{orchard}$ mathematically defined as

$$b_{vk}^{orchard} = b_{sk}^{orchard} \cdot R^{orchard}$$

Let SigHash be a transaction hash containing action description using SIGHASH type SIGHASH_ALL. So the validator check the balance by validating

$$\text{BindingSig}^{Orchard}.\text{Validate}_{b_{vk}^{Orchard}}(SigHash, \text{bindingSigOrchard}) = 1.$$

Thus checking the orchard binding signature ensure that the action transfer in the transection balance without their individual net value being revealed.

6.5 Spending Authorization Signature

In Orchard, the concept of SpendAuthSig is used to prove knowledge of the spending key authorizing the spending of an input note. In this document, the notation $\text{SpendAuthSig}^{\text{Orchard}}$ refers to the spend authorization signature scheme. The knowledge of the spending key could have been proven directly in the action statement; however, the reason for using a separate signature is to allow devices with limited resources, such as hardware wallets, to authorize shielded spends. These devices cannot create, and may not be able to verify, zk-SNARK proofs for a statement of the size needed using the Halo 2 proving system. The validating key of the signature must be revealed in the Action description so that the signature can be checked by the validator. To ensure that the validating key cannot be linked to the spending key a_{sk} from which the note was spent, a signature scheme with re-randomizable keys is used in Zcash. In the Action statement, it is proven that this validating key is a re-randomization of the spend authorization key a_k using a randomizer known to the signer. The spend authorization signature is applied over the SIGHASH transaction hash, ensuring that it cannot be reused in other transactions.

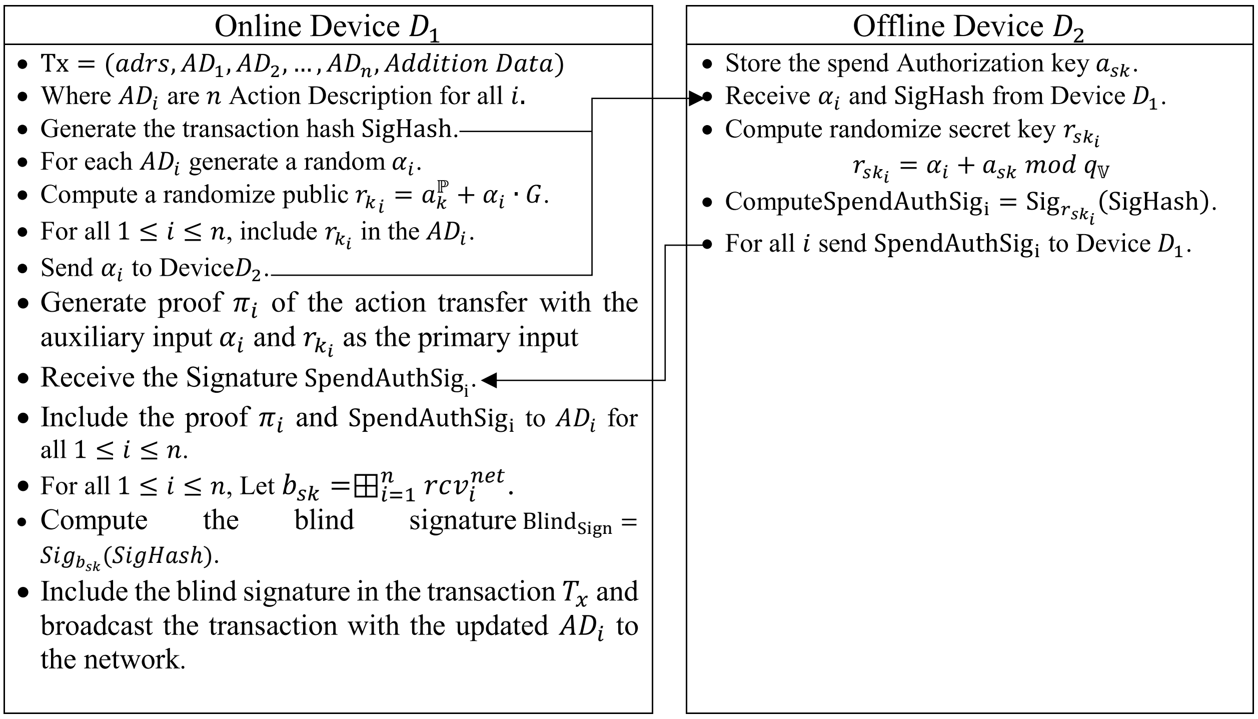
Let SigHash be the SIGHASH transaction hash using the SIGHASH type SIGHASH_ALL. Let a_{sk}^{A} be the spend authorization key. The detail is given as follows;

i. For each action description the signer choose a fresh randomizer α.
ii. Compute $r_{sk} = \alpha + a_{sk}$.
iii. Let $rk = \alpha \cdot G^{orchard} + a_{k}^{\mathbb{P}}$.
iv. Generate a proof π of the action statement with α in the auxiliary input and r_k in the primary input.
v. Let $\text{SpendAuthSig} = \text{Sig}_{\text{r}_{\text{sk}}}(SigHash)$

The resulting SpendAuthSig and the proof π are included in the Action description.

7 Use of Secure Element

In this section, we discuss how to use a secure element for securing shielded transactions. As we know, in order to send a value, the sender initially constructs a transaction with one or more action descriptions. The process of producing the action description has already been discussed. Suppose we have two devices, D_1 and D_2. Device D_1 is the online device responsible for constructing the transaction, which includes n action descriptions, while device D_2 is the offline device, that stores the secret key a_{sk} and is responsible for signing the SIGHASH and generating theSpendAuthSig, which should be included in the action description. The detail is provided below.



Online Device D_1

- $\text{Tx} = (adrs, AD_1, AD_2, \dots, AD_n, Addition\ Data)$
- Where AD_i are n Action Description for all i.
- Generate the transaction hash SigHash.
- For each AD_i generate a random α_i.
- Compute a randomize public $r_{k_i} = a_k^{\mathbb{P}} + \alpha_i \cdot G$.
- For all $1 \leq i \leq n$, include r_{k_i} in the AD_i.
- Send α_i to DeviceD_2.
- Generate proof π_i of the action transfer with the auxiliary input α_i and r_{k_i} as the primary input
- Receive the Signature $\text{SpendAuthSig}_\text{i}$.
- Include the proof π_i and $\text{SpendAuthSig}_\text{i}$ to AD_i for all $1 \leq i \leq n$.
- For all $1 \leq i \leq n$, Let $b_{sk} = \boxplus_{i=1}^{n} rcv_i^{net}$.
- Compute the blind signature $\text{Blind}_{\text{Sign}} = Sig_{b_{sk}}(SigHash)$.
- Include the blind signature in the transaction T_x and broadcast the transaction with the updated AD_i to the network.

Offline Device D_2

- Store the spend Authorization key a_{sk}.
- Receive α_i and SigHash from Device D_1.
- Compute randomize secret key r_{sk_i}
  $r_{sk_i} = \alpha_i + a_{sk}\ mod\ q_{\mathbb{V}}$
- $\text{ComputeSpendAuthSig}_\text{i} = \text{Sig}_{r_{sk_i}}(\text{SigHash})$.
- For all i send $\text{SpendAuthSig}_\text{i}$ to Device D_1.

From the above protocol, it can be seen that the offline device, Device D_2 , which stores the spending authorization key, performs just two steps. First, it performs modular addition to combine the spending authorization key a_{sk} with the randomized element α_i to generate randomize the secret key r_{sk_i}. Then, the device D_2 signs the transaction hash (SigHash) using the RedDSA signature and outputs the signature SpendAuthSig_i for each Action description AD_i . All other steps for the transaction, such as generating the proof π_i and blind signature, can be carried out on device D_1 without revealing the spending authorization key a_{sk} to device D_1.