Zcash Overview

By

Muhammad Shahid

For Cryptnox SA

https://cryptnox.com/

November 2024

Contents

1 Introduction

The aim of this document is to provide an in-depth overview of the Zcash protocol, detailing both the protocol itself and the underlying cryptographic primitives. Zcash supports two types of payment schemes: the transparent payment scheme and the shielded payment scheme. The difference between these two payment schemes, is that in the transparent payment scheme, all user information's, such as transaction volume and user addresses are publicly accessible. In contrast, the shielded payment scheme conceals this information, making it unavailable to the public.

Zcash offers users the flexibility to choose between privacy and transparency, with two types of addresses: shielded addresses and transparent addresses. Shielded addresses are used for shielded payments, while transparent addresses are used for payments where user information is public. In addition, there is flexibility in transactions between shielded and transparent addresses. Transactions from shielded addresses to transparent addresses and vice versa are supported. The transection details remain private from the shielded addresses to transparent addresses, while the transaction details are revealed when transferring from transparent to shielded addresses. The complete detail is illustrated in Figure 1.

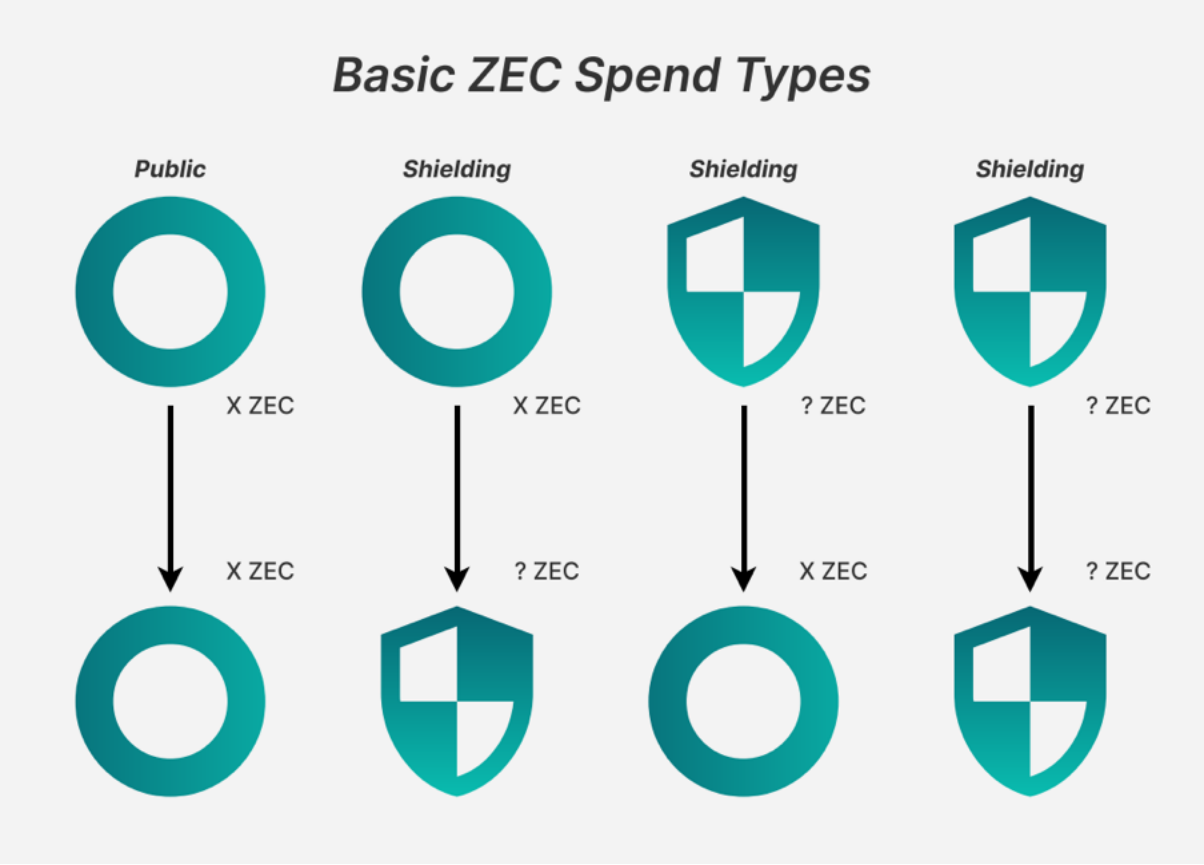

Figure 1 Sending Between Shielded and Transparent Addresses

In the following sections of this document, we discuss both the payment schemes, how these payment schemes work, and the cryptographic primitives used within them, especially the shielded payment scheme. Section 2 provides a brief overview of the transparent payment scheme. The first ZeroCash protocol for the shielded payment scheme is described in Section 3. Section 4 covers the Sprout shielded payment scheme. The Sapling update is discussed in Section 5. Section 6 is devoted to the Orchard update of the Zcash shielded payment scheme.

2 Transparent Transection.

Transparent payment scheme uses transparent addresses for the transparent transaction. So, in this payment scheme after the transection the address and the associated value are publicly recorded on the Zcash block chain just like bitcoin. The addresses of transparent payment start with the letter t, and it does not use Zero Knowledge proof (ZKP) to protect transection data for value sent or received it. However, for authentication the transparent input signatures use **ECDSA over the secp256k1 curve**, as in Bitcoin.

3 ZeroCash

There are two types of shielded transactions, named mint and pour. The mint transaction is used to mint a new coin, denoted by$t$ x_{mint}. The pour transaction is used to transfer a coin from one user to another, denoted by tx_{pour}. In the Shielded Payment system, each user has shielded addresses$(addr_{pk}, addr_{sk})$, called public and private addresses, and some public (pk_{sig}, vk_{pour}) and private $(sk_{sig},\ pk_{pour})$ parameters. The public address consists of the tuple $addr_{pk} = (a_{pk}, pk_{enc})$, where pk_{enc} is the encryption key for an asymmetric encryption scheme and the a_{pk} is a user address used to receive coins. The private address consists of the tuple $addr_{sk} = (a_{sk}, sk_{dec})$, where sk_{dec} is the secret decryption key for an asymmetric encryption scheme. a_{sk} is a spending key; without knowledge of it, no one can spend the coins. The public parameters consist of a signature verification public key pk_{sig} that can be used to verify signatures used in the protocol and a zero-knowledge verification key k_{pour}. Similarly, the private parameters consist of a signature private key sk_{sig} and a zero-knowledge private key k_{pour}, used in the proof of the ZKP. Additionally, Zcash uses three pseudorandom functions, i.e., PRF_x^{addr}, PRF_x^{sn} and PRF_x^{pk} with input seed x, where PRF_x^{sn} is a collision-resistant function. In the following subsection, we discuss the algorithms for generating addresses, mint transactions, pour transactions, the verification algorithm, and the receiving algorithm in detail.

3.1 Generate Address

For a security perimeter λ generate public and private $(pk_{enc}, sk_{dec}) \longleftarrow Gen(1^{\lambda})$. Next generate a random number a_{sk} and compute a_{pk} i.e., $a_{pk} = PRF_{a_{sk}}^{addr}(0)$. The public address $addr_{pk} = (a_{pk}, pk_{enc})$ and the private $addr_{sk} = (a_{sk}, sk_{dec})$. The output of address generate algorithm is $(addr_{pk}, addr_{sk})$.

3.2 Mint

To mint a coin with the desired value $v \in \{0,1, \dots, v_{max}\}$, the user U with public address a_{pk} initially generates three random number sequences $\rho,\ r,$ and s. Then, compute $k = COM_r(a_{pk} \,||\, \rho)$ and $cm = COM_s(v||\, k)$. The coin is $c = (a_{pk},\ v, \rho, r, s, cm)$ and the

transaction $tx_{mint} = (cm\ v, k, s)$. The transaction tx_{mint} is accepted to the ledger when the correct amount is deposited.

3.3 Mint Transection Verification

To verify the mint transection $tx_{mint} = (cm, v, k, s)$, compute $cm' = COM_s(v||k)$ and out $b = 1$ if $cm' == cm$ else $b = 0$. It mean the transection tx_{mint} is valid if $b = 1$, otherwise the transaction is not valid.

3.4 Pour Transection

The pour transaction is used to spend a valid coin by transferring it to another user. The pour operation consumes the input coin along with the secret address a_{sk} and public parameter of the user who spends the coin, and the public address a_{pk} and public parameter of the users who receive the coin.

Step 1. Suppose a user A with the address key pair $(addr^A_{pk}, addr^A_{sk})$ wishes to send his coin $c^A = \left(a^A_{pk}, v, \rho^A, r^A, s^A, cm^A\right)$ to the target addresses $addr^B_{pk}$ and $addr^C_{pk}$ belonging to users B and C. Initially, the user A produces two new coins c^B and c^C, with total value $v = v^B + v^C$. For new coins user A generates a set of random numbers $\{\rho^B, \rho^C, r^B, r^C, s^B, s^C\}$. Then, it computes $k^B = COM_{r^B}(a^B_{pk}||\ \rho^B)$ and $k^C = COM_{r^C}(a^C_{pk}||\rho^C)$. Afterwards, user A computes $cm^B = COM_{s^B}(v^B||\ k^B$ and $cm^C = COM_{s^C}(v_C||\ k^C)$. This yields two new coins $c^B = (a^B_{pk}, v^B, \rho^B, r^B, s^B, cm^B)$ and $c^C = (a^C_{pk}, v^C, \rho^C, r^C, s^C, cm^C)$.

Step 2. Now, in order to allow users B and C to spend their coins c^B and c^C, user A needs to send the secret values corresponding to the new coins to B and C securely. For that, in step two, user A encrypts the secret values: $C_B = ENC_{pk^B}(v_B, \rho^B, r^B, s^B)$ and $C_C = ENC_{pk^C}(v_C, \rho^C, r^C, s^C)$ using an asymmetric key encryption scheme with the public keys pk^B_{enc} and pk^C_{enc} of users B and C.

Step 3. In step four, user A generates a signature key pair (pk_{sig}, sk_{sig}) and computes the hash of the public signature key, i.e., $h_{sig} = H(pk_{sig})$, and then generates a random sequence using $h = PRF^{pk}_{a_{sk}}(h_{sig})$.

Step 4. In step 4, user A generates proof and verification keys (pk_{pour}, vk_{pour}) to produce a ZKP proof π^A for the following NP statement.

Statement: The instance is in the form $x = (rt, sn^A, cm^B, cm^C, v^A, h_{sig}, h)$. The instance x consists of the Merkle tree root rt, serial number sn^A, coin commitments cm^B and cm^C, the value of the coin v^A,h_{sig} and h.

Witness: The witness is in the form $a = (path, c^A, a^A_{sk}, c^B, c^C)$. It consists of the authentication path, the information about the old and new coins, and the address secret key.

The user A generates a $\pi^A = ZKP_{proof}(pk_{pour}, x, a)$. The proof $ZKP_{verify}(pk_{pour}, \pi^A)$ is valid if the following conditions are met.

i. The coin commitment cm^A of c^A appears in the ledger and the path is a valid authentication path for the leaf cm^A with respect to the root rt.
ii. The address secret key matches the address public key, i.e., $a^A_{pk} = PRF^{addr}_{a^A_{sk}}(0)$.

iii. The serial number sn^A of c^A is computed correctly, i.e., $sn^A = PRF^{sn}_{a^A_{sk}}(\rho^A)$.

iv. For c^A, it holds that $k^A = COM^A_r(a^A_{pk}||\rho^A)$ and $cm^A = COM_{s^A}(v_A||k^A)$. Similarly, for c^B, it holds that $k^B = COM_{r^B}(a_{pk^B}||\rho^B)$ and $cm^B = COM_{s^B}(v_B||k^B)$ and for c^C, it holds that $k^C = COM_{r^C}(a_{pk^C}||\rho^C)$ and $cm^C = COM_{s^C}(v_C||k^C)$.

v. The random sequence h is generated using the address secret key a^A_{sk} and h_{sig}, i.e., $h = PRF^{pk}_{a^A_{sk}}(h_{sig})$.

vi. Balance is preserved, i.e., $v^A = v^B + v^C$.

Step 6: Next, user A signs a set $m = \{x, \pi^A, C_A, C_B\}$ using the secret signing key, i.e., $\sigma = Sig(sk_{sig}, m)$.

Step 7: In this step, the user sets a transaction pour $tx_{pour} = (rt, sn^A, cm^B, cm^C, v^A, pk_{sig}, h, \pi^A, C_A, C_B\ \sigma)$.

As a result, the pour transaction $tx_{pour} = (rt, sn^A, cm^B, cm^C, \pi^A)$ is appended to the ledger. Since A does not know the pair of secret addresses a_{sk^B} and a_{sk^C} corresponding to the public addresses a_{pk^B} and a_{pk^C}, Therefore, A cannot spend the coins c^B and c^C as A cannot provide a_{sk^B} and a_{sk^C} as part of the witness for subsequent pour operations. In addition, to prevent double spending, if sn^A is appears in a ledger then reject the transection else sn^A add to the list.

3.5 Pour Transection Verification

To verify the pour transaction $tx_{pour} = (rt, sn^A, cm^B, cm^C, v^A, pk_{sig}, h, \pi^A, C_A, C_B, \sigma)$, check whether sn^A appears in the ledger L. If it does, output $b = 0$; otherwise, output b = 1. This step prevents double spending. Next, check the Merkle root rt in the ledger L. If the rt does not appear in the ledger L, output $b = 0$; otherwise, output $b = 1$. Next, compute $h_{sig} = H(pk_{sig})$ and set $x = (rt, sn^A, cm^B, cm^C, v^A, h_{sig}, h)$. Then set $m = (x, \pi^A, C_A, C_B)$ and verify the signature $V_{sig}(pk_{sig}, m, \sigma)$. If the signature σ is verified, output $b = 1$; otherwise, output $b = 0$. In the next step, verify the zero-knowledge proof $Verify(vk_{pour}, x, \pi^A)$. If the verification is true, output $b' = 1$; otherwise, output $b' = 0$. In the last step, if $b \wedge b' = 1$, it means that the transaction tx_{pour} is verified; otherwise, reject the transaction tx_{pour}.

3.6 Receive Transection

In this subsection, the steps to receive the spent coins are discussed in detail. Suppose user B receives the pour transaction $tx_{pour} = (rt, sn^A, cm^B, cm^C, v^A, pk_{sig}, h, \pi^A, C_B, C_C, \sigma)$. First, decrypt C_B using their private key: $(v_B, \rho^B, r^B, s^B) = DEC_{sk^B}(C_B)$. Then verify the output of the decryption by checking $cm^B = COM_{s^B}\left(v^B || COM_{r^B}(a_{pk^B} || \rho^B)\right)$. Next, check whether $sn^B == COM_{r^B}(a_{pk^B} || \rho^B)$ does not appear in the ledger L. If both conditions are true, $c^B = (addr_{pk}, v_B, \rho^B, r^B, s^B, cm^B)$ is the new coin for user B to spend.

Setup

Input: Security Parameters λ.

Output: Public parameters pp and sp.

1. Generate $(sk, vk) \leftarrow Gen(1^{\lambda})$.
2. Generate $(sk_{sig}, pk_{ver}) \leftarrow Gen(1^{\lambda})$.
3. Generate $(sk_{dec}, pk_{enc}) \leftarrow Gen(1^{\lambda})$.
4. Public parameters $pp = (vk, pk_{ver}, pk_{dec})$.
5. Secret parameters $sp = (sk, sk_{sig}, sk_{dec})$.

Create Addresses

Input: Public parameters $\boldsymbol{pp}$

Output: Address key pair $(addr_{pk}, addr_{sk})$.

1. Randomly sample PRF seed a_{sk}.
2. Compute $a_{pk} = PRF_{a_{sk}}(0)$.
3. $addr_{pk} = (a_{pk}, pk_{enc})$.
4. $addr_{sk} = (a_{sk}, sk_{dec})$.

Mint

Input: Public parameter pp,coin value v and $addr_{pk}$

Output: Coin c and mint transection Tx_{mint}

1. Randomly select a PRF seed ρ.
2. Random select two trapdoors r and s.
3. Compute $k = COMM_r(a_{pk}||\rho)$.
4. Compute $cm = COMM_s(v||k)$.
5. Set $c = (addr_{pk}, v, \rho, r, s, cm)$.
6. Set $Tx_{mint} = (cm, v, k, s)$

Verification Transection

Input: Public Parameters pp.

- Transection Tx
- The current ledger

Output: $b = 1$ if the transaction Tx is valid.

- $b = 0$ if the transaction Tx is not valid.

a) If given a mint transection Tx_{mint}.

1. Parse $Tx_{mint} = (cm, v, k, s)$.
2. Set $cm' = COMM_s(v||k)$.
3. If $cm' = 1$ Output $b = 1$ else $b = 0$.

b) If given a Pour Transection Tx_{pour}.

1. Parse $x_{pour} = (rt, nf, cm, v, pk_{sig}, h, \pi, C, \sigma)$.
2. If nf appear in the ledger output $b = 0$.
3. If rt does not appear in the ledger output $b = 0$.
4. Compute $h_{sig} = Hash(pk_{sig})$.
5. Set $x = (rt, nf, cm, v, h_{sig}, h,)$
6. Set $m = (x, \pi, C)$.
7. Compute $V_{sig}(pk_{sig}, m, \sigma)$ is not valid output $b = 0$.
8. Compute $V_{zk}(vk, x, \pi)$ not valid output $b = 0$.

Pour

Input: Public parameters pp.

- Markle root rt^A.
- User A coin c^A.
- User A address secret key $addr_{sk}^A$.
- $Path^A$ from commitment cm^A to root rt^A.
- New coin value v^B.
- User B public address $addr_{pk}^B$.
- User A coin value v^A.

Output: New coin c^B for B and transection Tx^B.

1. Parse $c^A = (addr_{pk}^A, v^A, \rho^A, r^A, s^A, cm^A)$.
2. Parse the secret address $addr_{sk}^A$.
3. Compute nullifier $nf^A = PRF_{a_{sk}^A}(\rho^A)$.
4. Randomly select a PRF seed ρ^B.
5. Randomly sample random numbers r^B and s^B.
6. Compute $k^B = COMM_{r^B}(a_{pk}^B||\rho^B)$.
7. Compute $cm^B = COMM_{s^B}(v^B||k^B)$.
8. Set $c^B = (addr_{pk}^B, v^B, \rho^B, r^B, cm^B)$.
9. Encrypt $C^B = ENC_{pk_{enc}^B}(v^B, \rho^B, r^B, s^B)$
10. Compute $h_{sig} = Hash(pk_{sig}^A)$
11. Compute $h = PRF_{a_{sk}^A}(1||h_{sig})$.
12. Set $x = (rt^A, sn^A, cm^B, v^A, h_{sig}, h)$.
13. Set $a = (path^A, c^A, addr_{sk}^A, c^B)$.
14. Compute $\pi^A = Prove(vk^A, x, a)$.
15. Set $m = (x, \pi, C^B)$.
16. Compute $\sigma = Sign(sk_{sig}, m)$.
17. Set $Tx_{pour} = (rt^A, nf^A, cm^B v^A, pk_{sig}^A, h, \pi^A, C^B, \sigma)$

Receive Transection

Input: Public Parameter pp.

- Recipient Address $(addr_{pk}^B, addr_{sk}^B)$.
- Current Ledger

Output: New coin c^B.

1. Parse $addr_{pk} = (a_{pk}^B, pk_{enc}^B)$.
2. Parse $addr_{sk} = (a_{sk}^B, sk_{dec}^B)$.
3. Parse $Tx_{pour}(rt^A, nf^A, cm^B v^A, pk_{sig}^A, h, \pi^A, C^B, \sigma)$.
4. Decrypt $DEC_{sk_{dec}}(C^B) = (v^B, \rho^B, r^B, s^B)$
   - If $cm^B = COMM_{s^B}\left(v^B||COMM_r(a_{pk}^B||\rho^B)\right)$
   - If $nf^B = PRF_{a_{sk}}(\rho^B)$ does not appear in ledger L.
5. $c^B = (addr_{pk}^B, v^B, \rho^B, s^B, cm^B)$

4 Sprout Shielded Transection

4.1 Key Generation

To generate a new Sprout spending key, choose a uniformly random sequence of bits a_{sk}. From the spending key a_{sk}, generate the public address $a_{pk} = PRF_{a_{sk}}^{addr}(0)$, where the pseudo random function is defined as;

$$PRF_{a_{sk}}^{addr}(0) = SHA256Compress(1100||a_{sk}||0^8||0^{248}),$$

Afterward, generate the public and private keys $(pk, sk) \leftarrow KeyGen_{Sprout}(1^{\lambda})$ for Diffie-Hellman key exchange over Curve25519. The method of generating the public and private keys is as: let q be the order of the group. For the private key sk, choose a random number from the set $\{2, 3, \ldots, q-1\}$ and compute the public key $pk = sk \cdot G$, where G the generator of the group Curve25519 is. The keys (a_{pk}, pk) are the public addresses used to receive the coin, and (a_{sk}, sk) are the private addresses used to spend the coin.

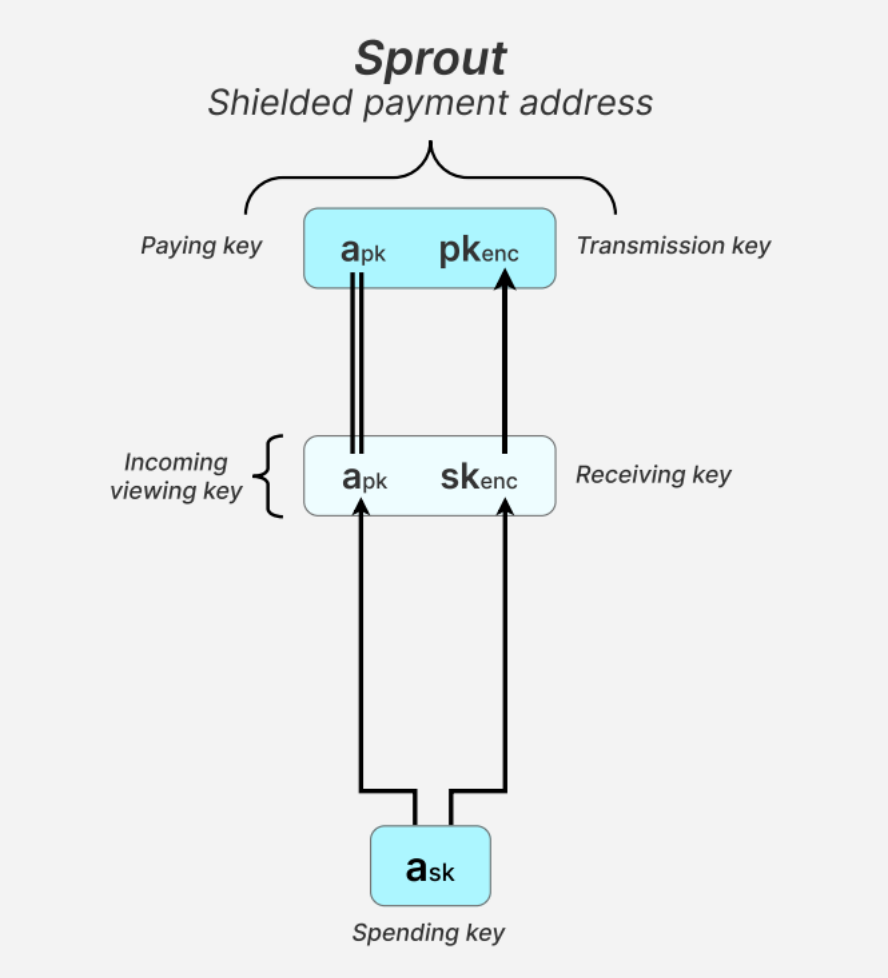

Figure 2 Sprout Key Components

4.2 Note

A Sprout note is a tuple $n = (a_{pk}, v, \rho, rcm, memo)$, where a_{pk} is the paying key of the recipient's shielded payment address is, v is an integer that represents the value of the coin, ρ is the input parameter for the function $PRF_{a_{sk}}^{nf} = SHA256Compress(1110||a_{sk}||0^8||\rho)$ to derive the nullifier of the note. The value rcm is the note commitment trapdoor used for generating the note commitment, and $memo$ is the sequence of random bytes, called the memo field.

4.3 Spending Note

Let A and B be two entities, and A wishes to send his coin $n_p^A = (a_{pk}^A, v^A, rcm^A, memo)$ to entity B. Let (a_{pk}^B, pk^B) be the public address of B. The transaction from A to B consists of a data, called JoinSplit description. For the JoinSplit description, initially A generates public and private keys for signing the transaction, called the JoinSplitSig key pair.

$$\text{JoinSplitPrivKey}^{\text{A}} \leftarrow \text{JoinSplitSig.GenPrivate}()$$

$$\text{JoinSplitPubKey}^{\text{A}} \leftarrow \text{JoinSplitSig.DerivePublic}(\text{JoinSplitPrivKey})$$

Next the sender chooses a random seed randomseed and selects the input node np^A. Afterward the sender A compute

$$h_{Sig} = \text{Blacke2b-256}(\text{"ZcashComputehSig"}||\text{hSigInput})$$

Where in the above equation hSigInput is the string given as follows;

$$\text{hSigInput} = \text{RandomSeed}||\text{nf}^{\text{A}}||\text{JoinSplitPrivKey}^{\text{A}}$$

Subsequently, the sender A choose random number φ^B and create output note np^B. The step by step procedure of crating the note is given as follows;

Choose uniformly random $rcm^B \leftarrow \text{NoteCommit}^{\text{Sprout}}.\text{GenTrapdoor}()$.

Compute $\rho^B = SHA256COMPRESS(000||\varphi^B||\ h_{Sig}\ )$.

Compute $cm^B = SHA256(10110000||a_{pk}^B||v^B||\rho^B||rcm^B\ )$.

The output note is $n_p^B = (0x00, v^B, \rho^B, rcm^B, memo^B)$. The sender A then encrypts n_p^B using a symmetric encryption scheme. Since the same secret key is used for both encryption and decryption in symmetric encryption, a key derivation function (KDF) is used. The details of the key derivation function and the procedures for encryption, decryption, signature, and zero-knowledge proofs are given in the following subsection.

4.3.1 Key Derivation function.

For the secret key derivation the entity A initially chose a secret number s^A from the set $\{2,3,\dots,q-1\}$ and compute ephemeral private key $e_{sk}^{AB} = s^A pk^B$ ($s^A pk^B = s^A sk^B G$) over the Curve25519. Afterward A computes a public ephemeral key $e_{pk}^A = s^A G$. Next, A uses the ephemeral private key e_{sk}^{AB} to derive a secret key for the symmetric encryption scheme to encrypt the data n_p^B and sends it to B, who can use it later. The key derivation function is given as follows:

K_{enc}^{AB} = Black-256("ZcashKDF" $||0^{56}||h_{sig}^{A}$ $||e_{sk}^{AB}$ $||e_{pk}^{A}$ $||pk^{B}$)

4.3.2 Encryption (Sprout)

To encrypt the note plaintext n_p^B, the sender A used a symmetric key encryption scheme AEAD CHACHA20 POLY1305 using the derived key K_{enc}. For sprout we will denote the encryption function by $ENC_{K_{enc}^{AB}}$ and the decryption function by $DEC_{K_{enc}^{AB}}$. So the ciphertext for the JointSplit description is given a follows;

$$C^B = ENC_{K_{enc}^{AB}}(n_p^B).$$

4.3.3 ZK-SNARK Statements

For the JointSplit description the entity A generate a ZK_SNARK statement $\pi_{ZK_{JointSplit}}^{B}$ that assure that for given output parameters $(rt^A, nf^A, cm^B, v^A, v^B, h_{sig}^B, h^B)$ the prover i.e., A knows the inputs $(path^A, position^A, n^A, a_{sk}^A, n^B, enforceMarklePath)$ such that the following conditions holds;

i. The note $n^A = (a_{pk}^A, v^A, \rho^A, rcm^A, memo^A)$ and $n^B = (a_{pk}^B, v^B, \rho^B, rcm^A, memo^A)$.
ii. For the note n^A the $path^A$ and $position^A$ is a valid Merkle path of depth from note commitment cm^A to the anchor root rt^A.
iii. The balance for input v^A and output v^B notes satisfied the equation i.e., $v^A - v^B \geq 0$
iv. The nullifier $nf^A = SHA256Compress(1110||a_{sk}||\rho^B)$.
v. The public address $SHA256Compress(1100||a_{sk}||0^{256})$.
vi. The non-malleability $h^A = SHA256Compress(0||0||000||a_{sk}||h_{sig}^A)$.
vii. The uniqueness of $\rho^B = SHA256COMPRESS(000||\varphi^B||\ h_{Sig}^A\ )$.
viii. The note Commitment integrity $cm^B = SHA256(10110000||a_{pk}^B||v^B||\rho^B||rcm^B)$.

4.4 JointSplit Transfers and Description

Each transection in Sprout consist of zero or more JoinSplit description. A JoinSplit description consist of the data that describe a shielded value transfer. The data comprises $\left(v^A, v^B, rt^A, nf^A, cm^B e_{pk}^A, randomSeed, h_{sig}^A, \pi_{ZK_{JointSplit}}^B, C^B\right)$, where v^A denote the value of a spending input coin is and v^B denote the value of the output coin. The anchor of the spending coin symbolize by rt^A. The nf^A is the nullifier of the spending coin and cm^B is the note commitment of the output coin.

4.5 Signature

Since we know that each transaction consists of one or more JoinSplit descriptions, any transaction that has at least one JoinSplit description must have a JoinSplit signature using Ed25519. Let `dataToBeSigned` be the hash value of the transaction. In this step, the sender of the coin computes the signature σ^A by signing `dataToBeSigned` with the signature private key `JoinSplitPrivKey$^A$` and includes the public validating key `JoinSplitPublicKey$^A$` and the signature σ^A in the transaction. Since the signing keys used for computing the signature are ephemeral, the user generates new signature key pairs for every transaction. For each key pair, the value h_{sig}^A given in the JoinSplit description and its integrity proof, provided in the ZK-SNARK statement, verify that the owner of the private address a_{sk}^A is authorized to use the private key. The transection $T_x^A = (JoinSplit, \sigma^A, \text{JoinSplitPublicKey}^A)$ included JoinSplitSig submitted to the peer to peer network.

4.6 Receiving Note.

The entity B receive the transaction data consist of JointSplit descriptions. Since the JointSplit description consist of the note data in encrypted from that entity B will spend letter. So before the decryption the entity B validate the signature and proof of the transaction. For validating the signature B used the public signature key of A and validate the signature. $Verify_{\text{JoinSplitPublicKey}^{\text{A}}}(\sigma^A) = 1$. Next, the receiver validate the proof of the Transection using proof validation public key. After, validating the proof the receiver derived the secret key and decrypt the note. The key derivation function and the decryption function is given in the following subsections.

4.6.1 Key Derivation Function

To derive the secret key the entity B used the ephemeral key e_{pk} and his secret key to compute a shared secret key $e_{sk}^{AB} = sk^B e_{sk}^A = sk^B sk^A G$ the Curve25519. Next, B uses the ephemeral private key e_{sk}^{AB} to derive a secret key for the symmetric encryption scheme to decrypt the note data n_p^B received from B, who can use it later. The key derivation function is given as follows:

$$K_{enc}^{AB} = \text{Black-256}(\text{“ZcashKDF”} \,||0^{56}||h_{sig}^A \,||e_{sk}^{AB} \,||e_{pk}^A \,||pk^B)$$

4.6.2 Decryption (Sprout)

To decrypt the note ciphertext n_p^B, the reciver B used a symmetric key encryption scheme AEAD CHACHA20 POLY1305 using the derived key K_{enc}^{AB}. Since, we used the notation $ENC_{K_{enc}^{AB}}$ for the encryption and $DEC_{K_{enc}^{AB}}$ for the decryption function. The note plaintext for the JointSplit description is given a follows;

$$n_p^B = DEC_{K_{enc}^{AB}}(C^B).$$

After decrypting the note the receiver validate the note commitment that weather cm^B is equal to cm' or not where $cm' = SHA256(10110000||a_{pk}^B||v^B||\rho^B||rcm^B)$ and check the nullifier nf^A in the nullifier set.

5 Sapling Shielded Transection

In this section, we have presented complete detail of the sapling Protocol, used by the Zcash for shielded transection as of the sapling network upgrade, which satisfies certain security properties.

5.1 Key Generation

For generating the Sapling key components, a new random Sapling spending key sk is generated by selecting a random number. From the spending key, derive spend authorizing key a_{sk}, proof authorization key n_{sk}, and outgoing viewing key o_{vk}, defined as follows:

$$a_{sk} = Black2b - 512(\text{"}Zcash\_Expend\text{"}||sk||0)$$

$$n_{sk} = Black2b - 512(\text{"}Zcash\_Expend\text{"}||sk||1)$$

$$o_{vk} = Black2b - 512(\text{"}Zcash\_Expend\text{"}||sk||2)$$

Afterward, use the spend authorizing key to generate $a_k = a_{sk} \cdot G$, and similarly, use the proof authorization key to generate $n_k = n_{sk} \cdot H$, where a_k and n_k are points on the elliptic curve, with generators G and H over the Jubjub curve, used for various purposes. Then, generate the incoming viewing key from a_k and n_k as follows:

$$i_{vk} = Black2s - 256(\text{"}Zcashivk\text{"}||a_k||n_k)$$

Now, to create a new diversified payment address from the given incoming viewing key i_{vk}, repeatedly choose a diversifier d uniformly at random until the diversifier base g_d is not equal to $\perp$ (i.e., not invalid). Mathematically, g_d can be derived as:

$$g_d = DiversifyHash^{Sapling}(d)$$

$$DiversifyHash^{Sapling}(d) = GroupHash_U(\text{"}Zcash\_gd\text{"}, d)$$

The Hash function $GroupHash_U$ can be calculated as follows;

$$H = Black2s - 256(\text{"Zcash\_gd"}, U||M)$$

$$P = abst_j(H)$$

Compute $Q = 8P$ if $Q = \mathcal{O}_j$ return $\perp$. Else return Q. Afterward, compute the diversified transmission key pk_d=$i_{vk} \cdot g_d$. Thus, the diversified Sapling payment address is d. The set of parameters $(sk, a_{sk}, n_{sk}, o_{vk}, i_{vk}, a_k, n_k, pk_d)$ is the key component in sapling. Furthermore, the relationship between the Sapling key component parameters is depicted in Fig. 1.

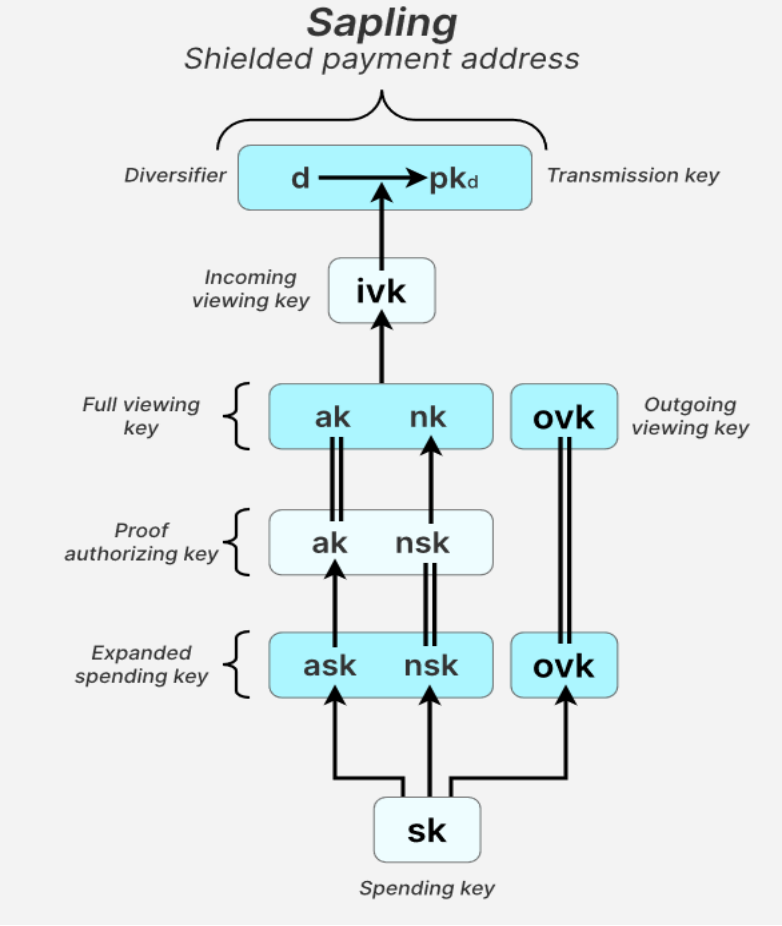

Figure 3 Sapling Key Components

5.2 Note

A sapling note is tuple $n = (d, pk_d, v, cm, r)$, basically it represents the value $v \in \{0,1,2, \dots, v_{max}\}$ that is spendable by the user who hold the spending key Where d of the recipient shielded payment address. pk_d is the diversified transmission key of the recipient shielded payment address, v is the value of the note in zatoshi and r is the random commitment trapdoor number.

5.3 Spend a Valid Coin

Let user A have a Sapling shielded payment address $(sk^A, a_{sk}^A, n_{sk}^A, o_{vk}^A, i_{vk}^A, a_k^A, n_k^A, pk_d^A)$ and wish to send his valid coin $n^A = (d_A, pk_d^A, v^A, cm^A, r^A, \rho^A, rcm^A)$ to user B, who has the Sapling key components $(sk^B, a_{sk}^B, n_{sk}^B, o_{vk}^B, i_{vk}^B, a_k^B, n_k^B, pk_d^B)$. The transaction to spend the coin consists of a spend transfer and an output transfer. The spend transfer validates the coin, and with the output transfer, the recipient receives the coin and can then spend it. In the following subsection, we discuss spend and output descriptions, which include all the data that describe spend and output transfer.

5.4 Spend Description

The spend description consist of $(cv^A, cm^A, rt^A, nf^A, r_{pk}^A, \pi_{ZKSpend}^A, SpendAutSig)$ where cv^A is value commitment integrity $cv^A = v^A V^{sapling} + rcv^A R^{sapling}$, for the base elements

$V^{sapling}$ and $R^{sapling}$ over JubJub curve. The parameter cm^A is the note commitment. The parameter $nf^A = \text{BLAKE2s} - 256(\text{“Zcash\_nf”}||\text{nk}||\rho^A)$ is the nullifier. The $r_{pk}^A = \alpha \cdot G + a_k$ is the randomized validating key that should be used to validate $SpendAutSig$ and $\pi_{ZKSpend}^A$ is the ZKP statement and the $SpendAutSig$ is the spend authorization signature.

4.3.1.1 Spend Statement $\pi_{ZKSpend}^A$

The spend statement $\pi_{ZkSpend}$ assure that for a given primary input (rt^A, cv^A, nf^A, rk), the prover know the auxiliary inputs $(Path, Position, g_d^A, pk_d^A, v^A, rcv^A, \alpha, nsk^A, cm^A, rcm^A)$ such that the following conditions hold;

i. The integrity of the note Commitment i.e., $cm^A = \text{NoteCommit}_{rcm}^{\text{Sapling}}(g_d^A||pk_d^A||v^A)$.
ii. The path and position $(path, position)$ of cm^A in the Markle tree is valid.
iii. The value commitment integrity is valid i.e., $cv^A = v^A V^{sapling} + rcv^A R^{sapling}$, .
iv. The order of the group containing g_d and ak is not the small.
v. The nullifier is validated i.e., $n_{f^A} = \text{BLAKE2s} - 256(\text{“Zcash\_nf”}||\text{nk}||\rho^A)$.
vi. To prove that rk is randomized public key $r_{pk}^A = \alpha^A + a_{sk}^A G$.
vii. Diversified address integrity $pk_d^A = i_{vk}^A \cdot g_d^A$.

4.3.1.2 Spend Authorization Signature

The spend authorization signature ($SpendAuthSig$) is used in Sapling to prove the knowledge of the spending key that authorizes the spending of the input note. In Sapling, the knowledge of the spending key cannot be proven directly in the spend statement. The motivation for keeping the signature separate is to allow devices that are limited in memory and computational capacity, such as hardware wallets, to authorize a Sapling shielded spend. The randomized signature RedDSA over the JubJub curve is used for signing. The complete details of the spend authorization signature are as follows:

i. For each spend description the signer chooses a fresh signer randomizer α.
ii. Let r_G be the order of the group over JubJub Curve. So, compute a random secret $r_{sk}^A = a_{sk}^A + \alpha^A \ mod \ r_G$ using spending authorization key a_{sk}^A.
iii. In the third step compute the private key $r_{pk}^A = a_{sk}^A G$ where G is the generator of the group.
iv. The $SpendAuthSig = RedDSA_{r_{sk}^A}^{sign}(SigHash)$, the $SigHash$ is the transection hash not associated with input.

5.4.1 Output Description

To send a note n^A to user B, the sender A initially selects a value v^B from the set $\{0,1,2, \ldots, v_{max}\}$ and constructs an output description. The output description consists of the data $(cv^B, cm^B, e_{pk}^A, C_{enc}^B, C_{out}^B, \pi_{ZKoutput}^B)$. Let (d^B, pk_d^B) be the public addresses of user B. The user A performs the following steps to construct the output description:

i. The user first checks that (d^B, pk_d^B) is of the type $KA_{sapling}$ public prime subgroup, i.e., (d^B, pk_d^B) should be a valid ctEdwards curve point on the JubJub curve.
ii. Then, user A chooses a random commitment trapdoor rcv^B.
iii. In step 4, user A chooses a uniformly random ephemeral key e_{sk}^A .
iv. In this step, user A also chooses a uniformly random commitment trapdoor rcm^B.

v. Afterward, user A computes $cv^B = v^B V^{sapling} + rcv^B R^{sapling}$.

vi. Next, user A computes $cm^B = \text{NoteCommit}_{rcm}^{\text{Sapling}}(g_d^B || pk_d^B || v^B)$.

The note plaintext $np^B = (leadBytes, d^B, v^B, rcm^B, rcv^B)$. Subsequently, user A encrypts np^B through a derived secret key. In the following subsections we have discussed the key derivation function, encryption and decryption.

5.4.2 Key Derivation Function

To derive the secret key the entity B used the ephemeral key e_{pk} and his secret key to compute the secret key $e_{sk}^{AB} = sk^B e_{sk}^A = sk^B sk^A G$ the Curve25519. Next, B uses the ephemeral private key e_{sk}^{AB} to derive a secret key for the symmetric encryption scheme to decrypt the note data n_p^B received from B, who can use it later. The key derivation function is given as follows:

$$K_{enc}^{AB} = \text{Black-256}(\text{“Zcash\_SaplingKDF”} || e_{sk}^{AB} || e_{pk}^A)$$

4.3.1.3 Encryption

In Sapling, the note plaintext np^B should be sent to user B securely so that the user can spend it later. Therefore, user A encrypts the data np^B. For encryption and decryption, a symmetric algorithm is used in the Sapling protocol. Since we know that symmetric key algorithms use the same key for both encryption and decryption, so, there must be a secure channel for sharing the secret key. To achieve this, the Sapling protocol deploys 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 given as follows:

i. First, select an ephemeral private key e_{sk}^A randomly.

ii. Compute the shared secret $sk^{AB} = e_{sk}^A \cdot pk_d$ where pk_d^B is a point on the ctEdwards curve.

iii. User A then computes the ephemeral public key $e_{pk}^A = e_{sk}^A \cdot g_d$.

iv. Apply the key derivation function that we have discussed in the previous subsection and generate a secret key K^{AB}.

v. Next, encrypt the data $C_{enc}^B = Enc_{K^{AB}}(np^B)$.

vi. Let $o_{ck}^A = BLAKE2b - 256(\text{“Zcash\_Derive\_ock"}, ||cv^B||cm^B||e_{pk}^A)$.

vii. Finally, compute $C_{out} = Enc_{ock}(pk_d^B \parallel e_{sk}^A)$

2.1.1.1 Output Sapling Statement $\pi_{ZKoutput}$

A valid instance of an output statement $\pi_{ZKoutput}$ assures that given a primary input (cv^B, cm^B, e_{pk}^A) the prover has the auxiliary input $(g_d^B, pk_d^B, v^B, rc^B, rcm_{new}, e_{sk}^A)$ such that the following conditions hold;

vii. The note commitment integrity $cm_u^B = \text{NoteCommit}_{rcm}^{\text{Sapling}}(g_d^B || pk_d^B || v^B)$.

i. The value commitment integrity is $cv^B = v^B V^{sapling} + rcv^B R^{sapling}$.

ii. The order of g_d^B is small.

iii. The ephemeral key is public key $e_{pk}^A = e_{sk}^A g_d^B$.

4.3.1.4 Decryption using incoming Viewing Key

Let $(epk^A, C^B_{ens}, C^B_{out})$ be the transmitted note ciphertext components. The recipient must decrypt the ciphertext components C^B_{enc} and C^B_{out} of the transmitted note data. The step-by-step decryption procedure is as follows:

i. Compute the share secret $sk^{AB} = i^B_{vk} e^A_{pk}$.
ii. $K^{AB} = KDF(sk^B)$
iii. $np^B = DEC(C^B_{enc})$
iv. $np^B = (d^B, v^B, rcm^B, memo)$
v. $g^B_d = DiersifyHash(d^B)$
vi. $pk^B_d = ivk^B * g^B_d$
vii. $n = (d^B, pk^B_d, v^B, rcm^B)$
viii. Let $cm^B = \text{NoteCommit}^{\text{Sapling}}_{rcm}(g^B_d || pk^B_d || v^B)$.
ix. Compute $o^A_{ck} = BLAKE2b - 256(\text{“}Zcash\_Derive\_ock\text{"} || cv || cm^B || e^A_{pk})$
x. Compute $op = DEC_{ock}(C^{out})$.

6 Orchard Shielded Transection

The Orchard protocol was deployed as part of the Zcash Network Upgrade 5 (NU5), which was activated on the mainnet at block height 1,687,104 on May 31, 2022. The NU5 upgrade includes several enhancements, notably the introduction of the Orchard shielded protocol. This new protocol simplifies and enhances the privacy features of Zcash by improving the efficiency and security of shielded transactions. A detailed explanation of the Orchard protocol is provided in the following subsections.

6.1. Abstraction

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

6.1.1 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 $q_{\mathbb{P}}$ and the order of $\mathbb{V}$ is $q_{\mathbb{P}}$.

6.1.2 Extract Function ($\text{Extract}_{\mathbb{P}}$)

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

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

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

6.1.3 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 $\text{DST}' = \text{DST}||length(\text{DST})$.
- Let $msg' = 0x00^{128}||\; msg \;||\; [0,128] \;||\; [0] \;||\; DST'$
- Let $b_0 = BLAKE2b - 512([0x00]^{16},\; msg')$
- Let $b_1 = BLAKE2b - 512\big([0x00]^{16},\; b_0\big|\big|[1]\big|\big|DST'\big)$
- Let $b_2 = BLAKE2b - 512\big([0x00]^{16},\; b_0 \oplus b_1\big|\big|[2]\big|\big|DST'\big)$
- Return $u_0 = b_1 \; mod \; q_G$ and $u_1 = b_2 \; mod \; q_G$.

6.1.4 Group Hash

The Group Hash is a function defined as $GroupHash^{\mathbb{G}}: \mathbb{B}^n \times \mathbb{B}^m \to \mathbb{G}$. 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 $iso_{map}(Q_0 + Q_1)$

6.1.5 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 $SinsemillaHashToPoint: \mathbb{B}^n \times \mathbb{B}^m \to \mathbb{P} \cup \{\perp\}$ 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 $S(m) = GroupHash^{\mathbb{P}}("z.cash: \text{SinsemillaS}", m)$
vii. Define a binary operation

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

viii. Let $Acc = Q(D)$.
ix. For i form 1 upto n:

$$Acc = (Acc * S(m_i)) * Acc$$

Return Acc.

6.1.6 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;

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

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

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

6.1.7 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;

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

6.1.8 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. $r_{i_{vk_{internal}}} = Black2b - 512(\text{"Zcash\_Expend"}, K, 0x83||a_k||n_k)\ mod\ r_{\mathbb{P}}$
iii. Return $(a_k, n_k, r_{i_{vk_{internal}}})$.

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

6.1.10 $repr_{\mathbb{G}}$ 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;

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

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

6.2 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;

$$\text{d} = \text{FF1} - \text{AES256}_{d_k}(\text{“”}, \text{Index})$$
$$g_d^{\mathbb{P}} = DiversifyHash(d)$$
$$pk_d^{\mathbb{P}} = i_{vk} \cdot g_d^{\mathbb{P}}$$

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.

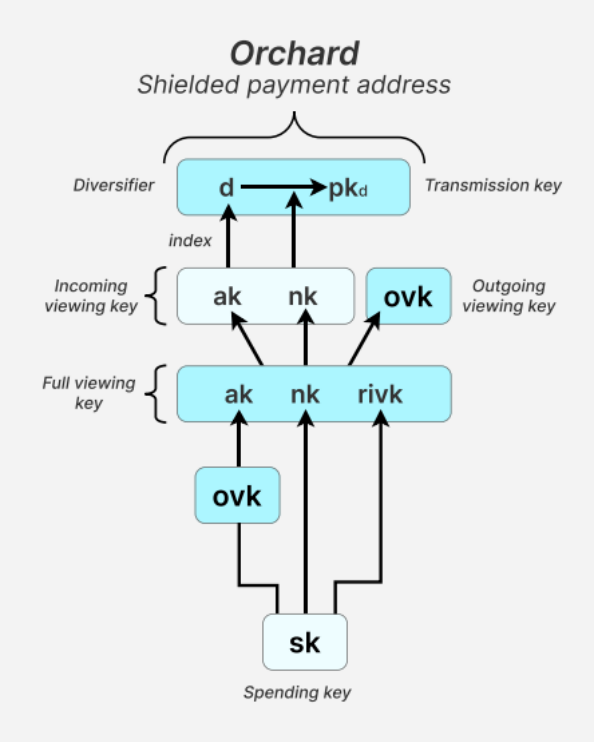

Figure 4 Orchard Key Components

6.3 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.

6.4 Spending a Valid Coin (Orchard)

Let A be user with orchard shielded payment address $(d^A, pk_{d^A}^{\mathbb{P}}, dk^A, n_{sk}^A, o_{vk}^A, i_{vk}^A, a_k^A, n_k^A, s_k^A, r_{ivk}^A)$ wishes to send his valid coin $n^A =$

$(d^A, pk_{d^A}^{\mathbb{P}}, v^A, \rho^A, \psi^A, r^m)$ to a user B with orchard shielded payment addresses$(d^B, pk_{d^B}^{\mathbb{P}})$. Initially, the sender A construct a transaction with one or more Action descriptions. For each description the sender A chose a value v^B and the distention payment address $(d^B, pk_{d^B}^{\mathbb{P}})$ and 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 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 user to attached arbitrary part of the transection. 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 procedure is given in the following subsection.

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

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

6.5 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 $(cv^{net}, rt^B, nf^A, rk^A, SpenAuthSig^A, cm^B, epk^A, C_{enc}^B, C_{enc}^B, enableSpend, enableOutput, \pi)$ 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.

We have already discussed the encryption and decryption procedures for encrypting the note's plaintext and ciphertext. In the following subsections, the Zero-Knowledge Proof and Binding Signature are discussed in more detail.

5.3.2.1 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: $cm_x^A = Extract_{\mathbb{P}}\left(NoteCommit_{rcm}^{Orchard}\left(g_{d^A}^{\mathbb{P}}, v^A, \rho^A, \psi^A\right)\right)$.
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: $nf^A = Extract_{\mathbb{P}}\big(\text{PoseidonHash}(\text{nk}_\text{A}, \rho^A) + \psi^A \bmod q_p + cm^A\big)$.
v. Randomized public key: $r_{k^A}^{\mathbb{P}} = (\alpha^A + a_{sk}^A)\mathbb{P}$.
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 $cm^A = NoteCommit_{rcm}^{Orchard}\left(g_{d^B}^{\mathbb{P}} \left|\left|pk_{d^B}^{\mathbb{P}}\right|\right| v^B \left|\left|\rho^B\right|\right|\psi^B\right)$,
ix. Enable spend flag $v^A = 0$ or $enableSpends = 1$.
x. Enable Output flag $v^B = 0$ or $enableOutputs = 1$.

6.5.1 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:

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

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:

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

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 $cv_1^{net}, \ldots, cv_n^{net}$ committing to a value $v_1^{net}, \ldots, v_n^{net}$ with randomness $rcv_1^{net}, \ldots, rcv_n^{net}$. 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.

6.5.2 Spending Authorization Signature

In Orchard the concept of SpendAuthSig has been used in order to prove the knowledge of the spending key authorizing spending of an input note. In this document the notation $\text{SpendAuthSig}^{\text{Orchard}}$ is used for spend authorization signature scheme. The knowledge of spending could have been proven directly in the action statement, however the reason behind a separate signature is to allow devises that a limited to resources such as Hardware wallet authorize the shielded spend, as these devices cannot create and may not be verified zk-SNARK proof for a statement of the size needed using the Hola 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 with the spending key a_{sk} from which the note was spent, in zcash a signature scheme has been used with re-randomizable keys. In the Action statement prove that this validating key is a re-randomization of the spend authorization key a_k with a randomizer known to the signer. The spend authorization signature is over the SIGHASH transaction has, so that it cannot be replied in other transection.

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 $rk = \alpha \cdot Ga_k^{\mathbb{P}} + a_k^{\mathbb{P}}$.
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 Cryptographic primitive

In the previous sections, we discussed a generalized overview of various shielded payment protocols. We have seen the cryptographic primitive such as random number generators, hash functions, signature algorithms, zero-knowledge proof algorithms, and encryption algorithms are used. In this section, we provide a detailed explanation of these algorithms

i. The pseudo rando function $PRF_k(X)$ is the $SHA256(1110||k_{252-bit}||X_{256-bit})$.
ii. The note commitment function $COMM_r(X) = SHA256(X||r)$.
iii. The $Diversify$ is a function that diversify an element into the base element of the Elliptic Curve.
iv. The Key Derivation Function (KDF) is used to securely share a secret key. In Zcash, for all shielded payment schemes, i.e., Sprout, Sapling, and Orchard, the **Diffie-Hellman key exchange protocol** is used over the elliptic curve.
v. We have seen that for secure node parameter transmission, a symmetric encryption scheme is used. The symmetric encryption scheme is **AEAD_CHACHA20_POLY1305**, which is an authenticated encryption scheme with associated data. In Zcash, the algorithm is used with empty associated data and an all-zero nonce.
vi. For signing a transaction, Zcash uses multiple signature algorithms: one for transparent transactions and three for shielded payment schemes (Sprout, Sapling, and Orchard).
    - The transparent input signatures use ECDSA over the secp256k1 curve, as in Bitcoin.
    - For Sprout, the signing procedure is called JoinSplitSig, which is used to sign transactions that contain at least one JoinSplit description. The signature algorithm used for JoinSplitSig is Ed25519.
    - In Sapling, the signature algorithms used are SpendAuthSig, for signing the authorization of spend transfers, and BindingSig, for enforcing the balance between spend and output transfers. The signature algorithm used for both SpendAuthSig

and BindingSig is RedDSA over the JubJub curve. The parameters for RedDSA over the JubJub curve are as follows:

$p = 52435875175126190479447740508185965837690552500527637822603658699938581184513$

$a = 52435875175126190479447740508185965837690552500527637822603658699938581184512$

$d = 19257038036680949359750312669786877991949435402254120286184196891950884077233$

$x = 80762466406628849098818017587043067140346099874558698045205220918555166029 23$

$y = 13262374693698910701929044844600465831413122818447359594527400194675274060458$

$q = 65544843968907738099309675635232457297059212658723172813653591623921832541 99$

Where p is the prime number, the integers a and d are the parameter the equation of the following ctEdward curve

$$ax^2 + y^2 = 1 + dx^2y^2$$

- The generator G generate a subgroup of order q.

- In orchard binding signature is used to enforce balance of action transfer and prevent their reply. The signing algorithm used for the signature is the **RedDSA over the pallas curve**.

The group generated by the pallas curve is the set of points that satisfied the equation of short Weierstrass equation given as follows;

$$y^2 = x^3 + ax + b \; mod \; p$$

$p = 28948022309329048855892746252171976963363056481941560715954676764349967630337,$

$a = 0$

$b = 5$

$G = (28948022309329048855892746252171976963363056481941560715954676764349967630336,$
$28948022309329048855892746252171976963363056481941647379679742748393362948097)$

vii. In the shielded payment scheme, zero-knowledge proofs are used for all three payment schemes, i.e., Sprout, Sapling, and Orchard. For each payment scheme, Zcash uses a specific proving system; therefore, there are a total of three proving systems in Zcash.

- In the Sprout shielded payment scheme, zk-SNARKs are generated by a fork of libsnark using the BCTV14 proving system and BN-254 pairing to prove and verify Sprout JoinSplit statements.
- For Sapling, the Groth16 proving system is used with BLS12-381 pairing to prove and verify Sapling spend and output statements.
- For Orchard, the Halo 2 proving system is used with the Vesta curve to prove and verify Orchard action statements.