Sapling keys hierarchy

[

2394×1514 309 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/939ae9c65b0b7b64e3e385ed4e11985d67945520.jpeg)

id

name

full name

type

description

1.

sk

spending key

scalar

A private key associated with a shielded address. Authorizes spending of a note. Is generated randomly. Used to generate nearly all of the other keys

Nullifier

id

name

full name

type

description

2.

nsk

nullifier private key

scalar

Used to derive the nullifier. Known to the prover of the Spend statement (a note is spent by proving knowledge of (\rho, ak, nsk)

). nsk = PRF_{sk}(1)

3.

nk

nullifier deriving key

point

nk = [nsk]*H

, H

is some generator. Used to derive the nullifier: nf = PRF_{nk}(\rho)

.

Spend authorization signature

id

name

full name

type

description

4.

ask

spend authorizing key

scalar

$ask = PRF_{sk}(0)$

5.

ak

spend validating key

point

A public key derived from ask

. Used to validate signatures. Known to the prover of the Spend statement. $ak = [ask]*P_{G}$

6.

rsk

-

scalar

Randomization of ask

. Used to sign the hash of a transaction. $rsk = ask + \alpha$

, α

is the randomness

7.

rk

validating key

point

Used to validate a signature created on the corresponding randomized private key.$rk = [rsk]P_G = [ask + \alpha]P_G = ak + \alpha*P_G$

, P_G

is some generator

Binding signature

Check here to learn more about binding signatures and how the keys are computed

id

name

full name

type

description

8.

bsk

binding signing key

scalar

Computed from the value commitment randomness rcv

.

9.

bvk

binding validating key

point

Computed from value commitments cv_i

. Not encoded in the transaction explicitly, must be recalculated. bvk = [bsk]*R

(R

is some generator) (it is not how the key is computed in practice (check the link above to learn more) but the relationship holds for correct key pairs).

Encryption

id

name

full name

type

description

10.

ivk

incoming viewing key

scalar

Used to derive pk_d

→ decryption of notes → blockchain scanning. ivk = H(ak, nk)

11.

ovk

outgoing viewing key

scalar

Encryption/decryption of outgoing notes. ovk = PRF_{sk}(2)

12.

pk_d

diversified transmission key

point

Used to derive a note encryption key. Is a part of a diversified (shielded) payment address (d, pk_d)

. pk_d = [ivk] * g_d = [ivk]* H(d)

. For each sk

, there is also a default difersified payment address (d, pk_d)

with a “random-looking” diversifier. The value d

is picked randomly so that g_d = H(d)

is not empty

13.

K_{enc}

-

scalar

A symmetric encryption key used to encrypt np

. K_{enc}= KDF([esk]*pk_d, epk)

14.

esk

ephemeral secret key

scalar

Randomly generated, used to derive K_{enc}

15.

epk

ephemeral public key

point

epk = [esk]*g_d

[esk]*pk_d =[esk]*([ivk] * g_d) = [ivk] * epk

. Used to derive K_{enc}

16.

ock

outgoing cipher key

scalar

Symmetric encryption key used to encrypt pk_d

and esk

. ock = PRF_{ovk}(cv, cm, epk)

17.

-

receiving key

-

Allows scanning of the blockchain for incoming notes and decrypt them. Just another name for an existing key type emphasizing the key's role

18.

fvk (ak, nk, ovk)

full viewing key

-

Is enough to both encrypt & decrypt notes, but not enough to spend

- Note n = (d, v, pk_d, rcm)
- Note plaintext np = (leadByte, d, v, memo)

Encrypt(np

, pk_d

, ovk

):

1. Generate esk
2. epk = [esk]*g_d
3. K_{enc} = KDF([esk]*pk_d, epk)
4. C_{enc} = E_{K_{enc}}(np)
5. ock = PRF_{ovk}(cv, cm, epk)
6. C_{out} = E_{ock}(pk_d || esk)

(if ovk

is None

, C_{out}

is garbage encrypted on garbage → not used)

→ ct = (epk, C_{enc}, C_{out})

Decrypt

If the user has the incoming viewing key

ivk

, they decrypt the note directly deriving K_{enc}

from ivk

:

1. K_{enc} = KDF([ivk]*epk, epk)
2. np = D_{K_{enc}}(C_{enc})
3. pk_d = [ivk]*g_d

If the user has the full viewing key

(though we only use the ovk

component of it), they use it to decrypt the keys C_{out} and then use the decrypted keys to decrypt the note

1. ock = PRF_{ovk}(cv, cm, epk)

(cv

and cm

are parts of the Output description)

1. pk_d, esk = D_{ock}(C_{out})
2. K_{enc} = KDF([esk]*pk_d, epk)
3. np = D_{K_{enc}}(C_{enc})

ZIP-32

id

name

full name

type

description

19.

(ask, nsk, ovk, dk, c)

extended spending key (ExtSK)

-

Chain code c

allows to avoid the situation where the child keypair solely depends on the parent key

20.

(ak, nk, ovk, dk, c)

extended viewing key (ExtVK)

-

Same as above

21.

dk

diversifier key

scalar

PRF(sk_m, 10)

. Used to derive diversifiers (~same way as in Orchard): d_j = PRP(dk, j)

Misc

id

name

full name

type

description

22.

(ask, nsk, ovk)

expanded spending key

-

Enough to spend a note

23.

(ak, nsk)

proof authorizing key

-

As a part of the spending action, one has to prove knowledge of (\rho, ak, nsk)

and disclose the nullifier