Ledger Device for Monero

v1.5+

Cédric Mesnil (cedric@ledger.fr)

Ledger SAS

August 3, 2022

Contents

1 License **4**

2 Introduction **5**

3 Notation **6**

4 Commands overview **7**
4.1 Introduction . . . 7
4.2 Common command format . . . 7

5 Provisioning And Key Management **8**
5.1 Overview . . . 8
5.2 Commands . . . 8
5.2.1 Reset . . . 8
5.2.2 Put keys . . . 9
5.2.3 Get Public Key . . . 9
5.2.4 Get Private View Keys . . . 10
5.2.5 Display Address . . . 11

6 Low level crypto commands **12**
6.1 Overview . . . 12
6.2 Commands . . . 12
6.2.1 Verify Keys . . . 12
6.2.2 Get ChaCha8 PreKey . . . 13
6.2.3 Generate Key Derivation . . . 13
6.2.4 Derivation To Scalar . . . 14
6.2.5 Derive Public Key . . . 15
6.2.6 Derive Secret Key . . . 16
6.2.7 Derive Subaddress Public Key . . . 17
6.2.8 Get Subaddress Spend Public Key . . . 17
6.2.9 Get Subaddress Secret Key . . . 18
6.2.10 Get Subaddress . . . 19
6.2.11 Generate Key Image . . . 20
6.2.12 Derive View Tag . . . 21
6.2.13 Generate Keypair . . . 21
6.2.14 Secret Key To Public Key . . . 22
6.2.15 Secret Add . . . 23
6.2.16 Secret Scalar Mult Key . . . 24
6.2.17 Secret Scalar Mult Base . . . 24
6.2.18 Stealth . . . 25
6.2.19 Unblind . . . 26

7 High Level Transaction command **27**
7.1 Transaction process overview . . . 27
7.2 Transaction State Machine . . . 29

7.3 Transaction Commands . . . 30
7.3.1 Open TX . . . 30
7.3.2 Set Signature Mode . . . 31
7.3.3 Hash Prefix . . . 31
7.3.3.1 Hash prefix init . . . 31
7.3.3.2 Hash prefix update . . . 32
7.3.4 Generate Commitment Mask . . . 32
7.3.5 Blind . . . 33
7.3.6 Generate TX output keys . . . 34
7.3.7 Validate and Pre Hash . . . 36
7.3.7.1 Initialize MLSAG-prehash . . . 36
7.3.7.2 Update MLSAG-prehash . . . 37
7.3.7.3 Finalize MLSAG-prehash . . . 38
7.3.8 MLSAG . . . 39
7.3.8.1 MLSAG prepare . . . 39
7.3.8.2 MLSAG hash . . . 41
7.3.8.3 MLSAG sign . . . 42

8 Conclusion **43**

9 Annexes **43**
9.1 References . . . 43
9.2 Helper functions . . . 43

1 License

Author: Cédric Mesnil <cslashm@gmail.com>

License:

2 Introduction

We want to enforce key protection, transaction confidentiality and transaction integrity against potential malware on the Host. To achieve that we propose to use a Ledger Nano S/X as a 2nd factor trusted device. Such a device has small amount of memory and is not capable of holding the entire transaction or building the required proofs in RAM. So we need to split the process between the host and the NanoS. This draft note explain how.

To summarize, the signature process is:

. Generate a TX key pair (r, R)

. Process Stealth Payment ID

. For each input T_{in} to spend:

- Compute the input public derivation data $\mathfrak{D}_{\mathrm{in}}$
- Compute the spend key (x_{in}, P_{in}) from R_{in} and b
- Compute the key image I_{in} of x_{in}

. For each output T_{out} :

- Compute the output secret derivation data $\mathfrak{D}_{\mathrm{out}}$
- Compute the output public key P_{out}

. For each output T_{out} :

- compute the range proof
- blind the amount
- compute the view tag

. Compute the final confidential ring signature

. Return TX

3 Notation

Elliptic curve points, such as pubic keys, are written in italic upper case, and scalars, such as private keys, are written in italic lower case:

- spk : protection key
- (r, R) : transaction key pair
- (a, A) (b, B) : sender main view/spend key pair
- (c, C) (d, D) : sender sub view/spend key pair
- A_{out} B_{out} : receiver main view/spend public keys
- C_{out} D_{out} : receiver sub view/spend public key
- h : 2nd group generator, such $H = h.G$ and h is unknown
- amount : amount to send/spend
- mask : secret amount mask factor
- C_v : commitment to a with v such $C_v = k.G + v.H$
- α_{in} : secret co-signing key for ith input
- x_{in} : secret signing key for ith input
- P_{in} : public key of ith input
- P_{out} : public key of ith output
- $\mathfrak{D}_{\text{out}}$ $\mathfrak{D}_{\text{in}}$: first level derivation data

Hash and encryption function:

- `AES` : [k](m) AES encryption of m with key k
- `AES`$^{-1}$: [k](c) AES decryption of c with key k

Others:

- $PayID$: Stealth payment ID
- `ENC_PAYMENT_ID_TAIL` : 0x82

4 Commands overview

4.1 Introduction

Hereafter are the code integration and application specification.

The commands are divided in three sets:

- Provisioning
- Low level crypto command
- High level transaction command

The low level set is a direct mapping of some crypto Monero function. For such command the Monero function will be referenced.

The high level set encompasses functions that handle the confidential/sensitive part of full transaction

4.2 Common command format

All command follow the generic ISO7816 command format, with the following meaning:

byte	length	description
CLA	01	Protocol version
INS	01	Command
P1	01	Sub command
P2	01	Command/Sub command counter
LC	01	byte length of data
data	01	options
	var	additional data

When a command/sub-command can be sent repeatedly, the counter must be increased by one at each command. The flag `last sub command indicator` must be set to indicate another command will be sent.

Common option encoding

`x--------`	Last sub command indicator
`1--------`	More identical subcommand forthcoming
`0--------`	Last sub command

5 Provisioning And Key Management

5.1 Overview

There is no provisioning in a standard setup. Both key pairs (a, A) and (b, B) should be derived under BIP44 path.

The general BIP44 path is :

```
/ purpose' / coin_type' / account' / change / address_index
```

and is defined as follow for any Monero main address:

```
/44'/128'/account'/0/0
```

so in hexa:

```
/0x8000002C/0x80000080/0x8......./0x00000000/0x00000000
```

The *address_index* is set to *0* for the main address and will be used as sub-address index according to kenshi84 fork.

In case an already existing key needs to be transferred, an optional dedicated command may be provided. As there is no secure messaging for now, this transfer shall be done from a trusted Host. Moreover, as provisioning is not handled by Monero client, a separate tool must be provided.

5.2 Commands

5.2.1 Reset

Description

Restart the application and check client/application versions compatibility.

Command

CLA	INS	P1	P2	LC
03	02	00	00	ll

Command data

Length	Value
01	00
var	string version, without trailing null byte

Response data

Length	Value
01	Application major version
01	Application minor version
01	Application micro version

5.2.2 Put keys

Description

Put sender key pairs.

This command allows to set specific key on the device and should only be used for testing purpose.

The application shall:

check $A == a.G$
check $B == b.G$
store a, A, b, B

Command

CLA	INS	P1	P2	LC
03	22	00	00	e0

Command data

Length	Value
01	00
20	a
20	A
20	b
20	B
5f	Base58 encoded public key

Response data

Length	Value

5.2.3 Get Public Key

Description

Retrieves public base58 encoded public key.

Command

CLA	INS	P1	P2	LC
03	20	01	00	01

Command data

Length	Value
01	00

Response data

Length	Value
20	"A" view public key
20	"B" view spend key
5f	Base58 encoded public key

5.2.4 Get Private View Keys

Description

Retrieves the private view key in order to accelerate the blockchain scan.

The device should ask the user to accept or reject this export. If rejected the client will use the device for scanning the blockchain.

Command

CLA	INS	P1	P2	LC
03	20	02	00	01

Command data

Length	Value
01	00

Response data

Length	Value
20	"a" secret view key

5.2.5 Display Address

Monero

Description

Display requested main address, sub address or integrated address.

compute $x = \texttt{AES}^{-1}[spk](\widetilde{x})$

if payment ID is provided:

compute $xP = x.G$
check $xP == P$

Command

CLA	INS	P1	P2	LC
03	21	xx	00	11

if P1 is '00' display non-integrated address.

if P1 is '01' display integrated address.

Any other value will be rejected.

Command data

Length	Value
01	00
08	index (Major.minor) *index*
08	Payment ID, (or '0000000000000000')

Response data

Length	Value

6 Low level crypto commands

6.1 Overview

This section describe lowlevel commands that can be used in a transaction or not.

6.2 Commands

6.2.1 Verify Keys

Monero

device_default::verify_keys.

Description

Verify that the provided private key and public key match.

compute $x = \texttt{AES}^{-1}[spk](\widetilde{x})$
compute $xP = x.G$
check $xP == P$

Command

CLA	INS	P1	P2	LC
03	26	xx	00	41

if P1 is '00' the provided public key will be used.

if P1 is '01' the public view is key will be used and the provided private key will be 'ignored'

if P1 is '02' the public spend is key will be used and the provided private key will be 'ignored'

Any other value will be rejected.

Command data

Length	Value	
01	00	
20		secret key $\widetilde{x}$
20		public key or '00'*32 P

Response data

Length	Value

6.2.2 Get ChaCha8 PreKey

Monero

Description

compute $s = h(A \mid B \mid \texttt{ENC\_PAYMENT\_ID\_TAIL})$

return the full internal state (200 bytes) of Keccak.

Command

CLA	INS	P1	P2	LC
03	24	00	00	00

Command data

Length	Value

Response data

Length	Value
C8	ChaCha8 prekey

6.2.3 Generate Key Derivation

Monero

crypto::generate_key_derivation.

Description

Compute the secret key derivation and return it encrypted.

compute $x = \texttt{AES}^{-1}[spk](\widetilde{x})$
compute $\mathfrak{D}_{\mathrm{in}} = \texttt{KeyDerivation}(x,P)$
compute $\widetilde{\mathfrak{D}_{\mathrm{in}}} = \texttt{AES}[spk](\mathfrak{D}_{\mathrm{in}})$

return $\widetilde{\mathfrak{D}_{\mathrm{in}}}$.

Command

CLA	INS	P1	P2	LC
03	32	00	00	41 or 61

Command data

Length	Value
01	00
20	public key P
20	secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)

Response data

Length	Value
20	encrypted key derivation $\widetilde{\mathfrak{D}_{\text{in}}}$
20	ephemeral hmac (optional, only during active transaction)

6.2.4 Derivation To Scalar

Monero

crypto::derivation_to_scalar.

Description

Transform a secret derivation data to a secret scalar according to its index.

compute $\mathfrak{D}_{\text{in}} = \texttt{AES}^{-1}[spk](\widetilde{\mathfrak{D}_{\text{in}}})$
compute $s = \texttt{HashPointToScalar}(\mathfrak{D}_{\text{in}}, \mathit{index})$
compute $\widetilde{s} = \texttt{AES}[spk](s)$

return $\widetilde{s}$.

Command

CLA	INS	P1	P2	LC
03	34	00	00	25 or 45

Command data

Length	Value
01	00
20	encrypted key derivation $\widetilde{\mathfrak{D}_{\text{in}}}$

Length	Value
20	ephemeral hmac (optional, only during active transaction)
04	index

Response data

Length	Value
20	encrypted scalar $\widetilde{s}$
20	ephemeral hmac (optional, only during active transaction)

6.2.5 Derive Public Key

Monero

crypto::derive_public_key.

Description

Compute a new public key from some secret derivation data, a parent public key and its index.

compute $\widetilde{\mathfrak{D}_{\text{in}}} = \texttt{AES}^{-1}[spk](\widetilde{\mathfrak{D}_{\text{in}}})$

derivation_to_scalar:

compute $s = \texttt{HashPointToScalar}(\mathfrak{D}_{\text{in}}, \mathit{index})$

then:

compute $P' = P{+}s.G$

return P'.

Command

CLA	INS	P1	P2	LC
03	36	00	00	25 or 45

Command data

Length	Value
01	00
20	encrypted key derivation $\widetilde{\mathfrak{D}_{\text{in}}}$
20	ephemeral hmac (optional, only during active transaction)
04	index
20	public key P

Response data

Length	Value
20	public key P'

6.2.6 Derive Secret Key

Monero

crypto::derive_secret_key.

Description

Compute a new secret key from some secret derivation data, a parent secret key and its index.

compute $\widetilde{\mathfrak{D}_{\text{in}}} = \mathtt{AES}^{-1}[spk](\widetilde{\mathfrak{D}_{\text{in}}})$
compute $x = \mathtt{AES}^{-1}[spk](\widetilde{x})$

derivation_to_scalar:

compute $s = \mathtt{HashPointToScalar}(\mathfrak{D}_{\text{in}},\ index)$

then:

compute $x' = (x{+}s)\ \%\ \#\mathtt{n}$
compute $\widetilde{x}' = \mathtt{AES}[spk](x)$

return $\widetilde{x}$.

Command

CLA	INS	P1	P2	LC
03	38	00	00	65 or 85

Command data

Length	Value
01	00
20	encrypted key derivation $\widetilde{\mathfrak{D}_{\text{in}}}$
20	ephemeral hmac (optional, only during active transaction)
04	index
20	encrypted secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)

Response data

Length	Value
20	encrypted derived secret key $\widetilde{x}$'
20	ephemeral hmac (optional, only during active transaction)

6.2.7 Derive Subaddress Public Key

Monero

crypto_ops::derive_subaddress_public_key.

Description

compute $\mathfrak{D}_{\text{in}} = \texttt{AES}^{-1}[spk](\widetilde{\mathfrak{D}_{\text{in}}})$
compute $s = \texttt{HashPointToScalar}(\mathfrak{D}_{\text{in}}, \textit{index})$
compute $P' = P - s.G$

return P'

Command

CLA	INS	P1	P2	LC
03	46	00	00	45 or 65

Command data

Length	Value
01	00
20	public key P
20	encrypted derivation key $\widetilde{\mathfrak{D}_{\text{in}}}$
20	ephemeral hmac (optional, only during active transaction)
04	index *index*

Response data

Length	Value
20	sub public key P'

6.2.8 Get Subaddress Spend Public Key

Monero

device_default::get_subaddress_spend_public_key.

Description

get_subaddress_secret_key:

compute $s = h(\text{"SubAddr"} \mid A \mid \mathit{index}\ )$
compute $x = s \ \% \ \#\mathtt{n}$

then:

compute $d = B + x.G$

return d

Command

CLA	INS	P1	P2	LC
03	4A	00	00	09

Command data

Length	Value
01	00
08	index (Major.minor) *index*

Response data

Length	Value
20	sub spend public key d

6.2.9 Get Subaddress Secret Key

Monero

get_subaddress_secret_key

Description

compute $x = \mathtt{AES}^{-1}[spk](\widetilde{x})$
compute $s = h(\text{"SubAddr"} \mid x \mid \mathit{index}\ )$
compute $d = s \ \% \ \#\mathtt{n}$
compute $\widetilde{d_i} = \mathtt{AES}^{-1}[spk](d)$

return $\widetilde{d_i}$

Command

CLA	INS	P1	P2	LC
03	4C	00	00	39 or 59

Command data

Length	Value
01	00
20	secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)
08	index (Major.minor) *index*

Response data

Length	Value
20	sub secret key $\widetilde{d_i}$
20	ephemeral hmac (optional, only during active transaction)

6.2.10 Get Subaddress

Monero

device_default::get_subaddress_secret_key.

Description

> compute $s = h(\texttt{"SubAddr"} \mid A \mid index\ )$
> compute $x = s \ \% \ \#\mathtt{n}$

then:

> compute $d = B + x.G$
> compute $c = A.d$

return c, d

Command

CLA	INS	P1	P2	LC
03	48	00	00	09

Command data

Length	Value
01	00
08	index (Major.minor) *index*

Response data

Length	Value
20	sub view public key c
20	sub spend public key d

6.2.11 Generate Key Image

Monero

crypto::generate_key_image.

Description

Compute the key image of a key pair.

compute $x = \mathtt{AES}^{-1}[spk](\widetilde{x})$
compute $P' = \mathtt{HashToPoint}(P)$
compute $Img(P) = x.P'$

return $Img(P)$.

Command

CLA	INS	P1	P2	LC
03	3A	00	00	41 or 61

Command data

Length	Value
01	00
20	public key P
20	secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)

Response data

Length	Value
20	key image $Img(P)$

6.2.12 Derive View Tag

Monero

crypto::derive_view_tag.

Description

Derive the view tag of an output.

> compute $\mathfrak{D}_{\text{in}} = \texttt{AES}^{-1}[spk](\widetilde{\mathfrak{D}_{\text{in}}})$
> compute $view\_tag\_full = \texttt{HashToScalar}(\text{"view\_tag"} \mid, \mathfrak{D}_{\text{in}}, index)$
> compute $view\_tag = view\_tag\_full[0{:}1]$

return $view\_tag$.

Command

CLA	INS	P1	P2	LC
03	3B	00	00	25 or 45

Command data

Length	Value
01	00
20	encrypted key derivation $\widetilde{\mathfrak{D}_{\text{in}}}$
20	ephemeral hmac (optional, only during active transaction)
04	index

Response data

Length	Value
01	view tag $view\_tag$

6.2.13 Generate Keypair

Monero

crypto::generate_keys.

Description

Generate a new keypair and return it. The secret key is returned encrypted.

generate x
compute $xP = x.P$
compute $\widetilde{x} = \texttt{AES}[spk](x)$

return P, $\widetilde{x}$.

Command

CLA	INS	P1	P2	LC
03	40	00	00	01

Command data

Length	Value
01	00

Response data

Length	Value
20	public key P
20	encrypted secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)

6.2.14 Secret Key To Public Key

Monero

crypto::secret_key_to_public_key.

Description

Compute a public key from secret a secret key.

compute $x = \texttt{AES}^{-1}[spk](\widetilde{x})$
compute $P = x.G$

return P.

Command

CLA	INS	P1	P2	LC
03	30	00	00	21 or 41

Command data

Length	Value
01	00
20	encrypted secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)

Response data

Length	Value
20	public key P

6.2.15 Secret Add

Monero

sc_add

Description

compute $x_1 = \mathtt{AES}^{-1}[spk](\widetilde{x_1})$
compute $x_2 = \mathtt{AES}^{-1}[spk](\widetilde{x_2})$
compute $x = x_1 + x_2$
compute $\widetilde{x} = \mathtt{AES}[spk](x)$

return $\widetilde{x}$.

Command

CLA	INS	P1	P2	LC
03	3C	00	00	41 or 61

Command data

Length	Value
01	00
20	secret key $\widetilde{x_1}$
20	ephemeral hmac (optional, only during active transaction)
20	secret key $\widetilde{x_2}$
20	ephemeral hmac (optional, only during active transaction)

Response data

Length	Value
20	secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)

6.2.16 Secret Scalar Mult Key

Monero

rct::scalarmultKey.

Description

Multiply a secret scalar with a public key.

> compute $x = \texttt{AES}^{-1}[spk](\widetilde{x})$
> compute $xP = x.P$

return xP

Command

CLA	INS	P1	P2	LC
03	42	00	00	41 or 61

Command data

Length	Value
01	00
20	public key P
20	secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)

Response data

Length	Value
20	new public key xP

6.2.17 Secret Scalar Mult Base

Monero

rct::scalarmultBase.

Description

Multiply a secret scalar with the publis base point G.

compute $x = \texttt{AES}^{-1}[spk](\widetilde{x})$
compute $xG = x.G$

return xG

Command

CLA	INS	P1	P2	LC
03	44	00	00	21 or 41

Command data

Length	Value
01	00
20	secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)

Response data

Length	Value
00	
20	new public key xG

6.2.18 Stealth

Monero

Description

Encrypt payment ID

compute $x = \texttt{AES}^{-1}[spk](\widetilde{x})$
compute $\mathfrak{D}_{\text{in}} = \texttt{KeyDerivation}(P, x)$
compute $s = \texttt{HashToScalar}(\ \mathfrak{D}_{\text{in}} \mid \texttt{ENC\_PAYMENT\_ID\_TAIL})$
compute $PayID = \widetilde{PayID} \hat{}\, s$

return $PayID$

Command

CLA	INS	P1	P2	LC
03	76	00	00	61 or 81

Command data

Length	Value
01	00
20	public key P
20	encryped secret key $\widetilde{x}$
20	ephemeral hmac (optional, only during active transaction)
20	encryped payment ID $\widetilde{PayID}$

Response data

Length	Value
20	payment ID $PayID$

6.2.19 Unblind

Monero

Description

Unblind amount and his mask.

First:

compute $\mathcal{AK}_{\text{amount}} = \texttt{AES}^{-1}[spk](\widetilde{\mathcal{AK}_{\text{amount}}})$

If blind V1:

compute $s = \texttt{HashToScalar}(\mathcal{AK}_{\text{amount}})$
compute $\widetilde{\text{mask}} = \text{mask-}s$
compute $s = \texttt{HashToScalar}(A)$
compute $\widetilde{\text{amount}} = \text{amount-}s$

If blind V2: compute mask = $\texttt{HashToScalar}(\text{"commitment\_mask"} \mid \mathcal{AK}_{\text{amount}})\ \%\ \#\texttt{n}$
compute $s = \texttt{HashToScalar}(\text{"amount"} \mid \mathcal{AK}_{\text{amount}})$
compute $\text{amount}[0{:}7] = \widetilde{\text{amount}}[0{:}7]\,\hat{}\,s[0{:}7]$

return $\widetilde{\text{mask}}, \widetilde{\text{amount}}$

Command

CLA	INS	P1	P2	LC
03	7A	00	00	61 or 81

specific options

`-------xx`	Commitment scheme version
`-------10`	Blind V2
`-------00`	Blind V1

Command data

Length	Value
01	xx
20	encryped blinding factor $\mathcal{AK}_{\text{amount}}$
20	ephemeral hmac (optional, only during active transaction)
20	blinded mask $\widetilde{\text{mask}}$
20	blinded amount $\widetilde{\text{amount}}$

Response data

Length	Value
20	mask $\widetilde{\text{mask}}$
20	amount $\widetilde{\text{amount}}$

7 High Level Transaction command

7.1 Transaction process overview

The transaction is mainly generated in construct_tx_and_get_tx_key (or construct_tx) and construct_tx_with_tx_key functions.

First, a new transaction keypair (r, R) is generated.

Then, the stealth payment id is processed if any.

Then, for each input transaction to spend, the input key image is retrieved.

Then, for each output transaction, the epehemeral destination key and the blinding key amount $\mathcal{AK}_{\text{amount}}$ are computed.

Once T_{in} and T_{out} keys are set up, the genRCT/genRctSimple function is called.

First a commitment C_v to each amount amount and its associated range proof are computed to ensure the amount amount confidentiality. The commitment and its range proof do not imply any secret and generate C_v, mask such $C_v = k.G + v.H$.

Then mask and amount are blinded by using the $\mathcal{AK}_{\text{amount}}$ which is only known in an encrypted form by the host.

After all commitments have been setup, the confidential ring signature happens. This signature is performed by calling proveRctMG which then calls MLSAG_Gen.

At this point the amounts and destination keys must be validated on the NanoS. This information is embedded in the message to sign by calling get_pre_mlsag_hash, prior to calling ProveRctMG. So the get_pre_mlsag_hash function will have to be modified to serialize the rv transaction to NanoS which will validate the tuple <amount,dest> and compute the prehash. The prehash will be kept inside NanoS to ensure its integrity. Any further access to the prehash will be delegated.

Once the prehash is computed, the proveRctMG is called. This function only builds some matrix and vectors to prepare the signature which is performed by the final call MLSAG_Gen.

During this last step some ephemeral key pairs are generated : α_{in}, $\alpha_{in}.G$. All α_{in} must be kept secret to protect the x_{in} keys. Moreover we must avoid signing arbitrary values during the final loop.

In order to achieve this validation, we need to approve the original destination address $A_{out}B_{out}$, which is not recoverable from P out . Here the only solution is to pass the original destination with the mask, amount, $\mathcal{AK}_{\text{amount}}$.

Unblind mask and amount and then verify the commitment $C_v = k.G + v.H$. If C_v is verified and user validate A_{out},B_{out} and amount, continue.

7.2 Transaction State Machine

During a transaction the following state machine is enforced:

```
OPEN_TX{1} -----------------------------------------------------
                                                               |
----------------------------------------------------------------
|
----> STEALTH{1} -----------------------------------------------
                                                               |
----------------------------------------------------------------
|
 ----> GEN_TXOUT_KEYS{*} ---------------------------------------
                                                               |
------------------------------------------------------------ ---
|
----> PREFIX_HASH{1} ---> PREFIX_HASH{*} ---> PREFIX_HASH{1} ---
        (ph_init)          (ph_update)        (ph_finalize)    |
                                                               |
----------------------------------------------------------------
|
----> GEN_COMMITMENT_MASK{*} -----------------------------------
        only for real TX                                       |
                                                               |
----------------------------------------------------------------
|
----> BLIND ----------------------------------------------------
                                                               |
----------------------------------------------------------------
|
----> VALIDATE{1} ---> VALIDATE{*} --- VALIDATE{*} <------------
  mlsag_ph_init     mlsag__update    mlsag__finalize           |
                                                               |
---------------------------------------------------------------
|
----> MLSAG{1} ------> MLSAG{*} ------> MLSAG{1} ---------------
  --> mlsag_prepare    mlsag_hash       mlsag_sign --          |
  |                                                |           |
  ---------------------------------------------------          |
                                                               |
----------------------------------------------------------------
|
----> CLOSE_TX
```

Note this state machine assume the multi-signature is not supported. For multi-signature the INS_MLSAG/mlsag_prepare and INS_MLSAG/mlsag_sign may be received several time.

7.3 Transaction Commands

7.3.1 Open TX

Description

Open a new transaction. Once open the device impose a certain order in subsequent commands:

- OpenTX
- Stealth
- Get TX output keys
- Blind *
- Initialize MLSAG-prehash
- Update MLSAG-prehash *
- Finalize MLSAG-prehash
- MLSAG prepare
- MLSAG hash *
- MLSAG sign
- CloseTX

During this sequence low level API remains available, but no other transaction can be started until the current one is finished or aborted.

Initialize $\mathcal{H}_{\text{outkeys}}$
compute initial transaction key pair (r, R)

return (r, R)

Command

CLA	INS	P1	P2	LC
03	70	01	cnt	05

Command data

Length	Value
01	options
04	account identifier (ignored, RFU)

Response data

Length	Value
20	public transaction key R
20	encrypted private transaction key $\widetilde{r}$
20	ephemeral hmac

Length	Value
20	ephemeral hmac of view key
20	ephemeral hmac of spend key

7.3.2 Set Signature Mode

Description

Set the signature to 'fake' or 'real'. In fake mode a random key is used to signed the transaction and no user confirmation is requested.

Command

CLA	INS	P1	P2	LC
03	72	01	00	02

Command data

Length	Value
01	options
01	'1' aka 'real' or '2' aka 'fake'

Response data

Length	Value

7.3.3 Hash Prefix

7.3.3.1 Hash prefix init

Description

Init prefix hash and ask user to validate time lock

Command

CLA	INS	P1	P2	LC
03	7D	01	cnt	05

Command data

Length	Value
01	options
varint	TX version
varint	TX timelock

Response data

Length	Value

7.3.3.2 Hash prefix update

Description

Update prefix hash with raw data. Options fields tells if there is more data to come or not.

Command

CLA	INS	P1	P2	LC
03	7D	02	cnt	05

Command data

Length	Value
01	options
var	raw data to hash

Response data

Length	Value

7.3.4 Generate Commitment Mask

Description

compute $s = \texttt{HashToScalar}(\text{"commitment\_mask"} \mid \mathcal{AK}_{\text{amount}})$

Return s

Command

CLA	INS	P1	P2	LC
03	77	00	00	21

Command data

Length	Value
01	00
20	encryped blinding factor $\mathcal{AK}_{\text{amount}}$
20	ephemeral hmac

Response data

Length	Value
20	commitment mask s

7.3.5 Blind

Monero

Description

Blind amount and his mask.

First:

> compute $\mathcal{AK}_{\text{amount}} = \texttt{AES}^{-1}[spk](\widetilde{\mathcal{AK}_{\text{amount}}})$

If blind V1:

> compute $s = \texttt{HashToScalar}(\mathcal{AK}_{\text{amount}})$
> compute $\widetilde{\text{mask}} = \text{mask}+s$
> compute $s = \texttt{HashToScalar}(A)$
> compute $\widetilde{\text{amount}} = \text{amount}+s$

If blind V2:

> set $\widetilde{\text{mask}}$ to 32 zero bytes
> compute $s = \texttt{HashToScalar}(\text{"amount"} \mid \mathcal{AK}_{\text{amount}})$
> compute $\widetilde{\text{amount}} = \text{amount}[0{:}7] \hat{} s[0{:}7]$

return $\widetilde{\text{mask}}$,$\widetilde{\text{amount}}$

Command

CLA	INS	P1	P2	LC
03	78	00	00	81

specific options

`-------xx`	Commitment scheme version
`-------10`	Blind V2
`-------00`	Blind V1

Command data

Length	Value
01	xx
20	encryped blinding factor $\mathcal{AK}_{\mathrm{amount}}$
20	ephemeral hmac
20	mask mask
20	amount amount

Response data

Length	Value
20	blinded mask $\widetilde{\text{mask}}$
20	blinded amount $\widetilde{\text{amount}}$

7.3.6 Generate TX output keys

Description

Compute additional key P if needed, amount key blinding and ephemeral destination key.

if *need_additional_key* :
  if *is_subaddress* :
    compute R' = $additional\_key.B_{out}$
  else
    compute R' = $additional\_key.G$

if *is_change_address* :
  compute $\mathfrak{D}_{\mathrm{in}}$ = `KeyDerivation`(A,R)
else
  if *need_additional_key* and *is_subaddress*:

compute $\mathfrak{D}_{\text{in}} = \texttt{KeyDerivation}(\textit{additional\_key}, A_{out})|$
else:
compute $\mathfrak{D}_{\text{in}} = \texttt{KeyDerivation}(R, A_{out})$

compute $\mathcal{AK}_{\text{amount}} = \texttt{HashPointToScalar}(\mathfrak{D}_{\text{in}}, \textit{index})$
compute $\widetilde{\mathcal{AK}}_{\text{amount}} = \texttt{AES}[spk](\mathcal{AK}_{\text{amount}})$

compute $s = \texttt{HashPointToScalar}(\mathfrak{D}_{\text{in}}, \textit{index})$
compute $P = B_{out} + s.G$

update $\mathcal{H}_{\text{outkeys}}$: $\texttt{H}_{\texttt{update}}(A_{out}, B_{out}, \text{is\_change}, \mathcal{AK}_{\text{amount}})$
if option 'last' is set:
finalize $\mathcal{H}_{\text{outkeys}}$

The application returns

Command

CLA	INS	P1	P2	LC
03	7B	01	cnt	EC

Command data

Length	Value
01	options
04	tx version
20	secret tx key R
20	ephemeral hmac
20	public tx key R
20	destination public view key A_{out}
20	destination public spend key B_{out}
04	output index *index*
01	is change address
01	is subaddress
01	need additional key *need_additional_key*: 1 if yes, 0 else
20	encrypted additional key *additional_key*, if *need_additional_key* == 1, 0*32 else
20	ephemeral hmac

Response data

Length	Value
20	encrypted amouny key blinding $\widetilde{\mathcal{AK}_{\text{amount}}}$
20	ephemeral hmac
20	ephemeral destination key P
20	additional Key R' if *need_additional_key* == 1, not present else

7.3.7 Validate and Pre Hash

7.3.7.1 Initialize MLSAG-prehash

Description

During the first step, the application updates the $\mathcal{H}$ with the transaction header:

if cnt == 1

> Finalize $\mathcal{H}_{\text{outkeys}}$
> Initialize $\mathcal{H}_{\text{outkeys}}$'
> Initialize $\mathcal{H}_{\text{commitment}}$
> Initialize $\mathcal{H}$
> update $\mathcal{H}$: $\mathtt{H}_{\mathtt{update}}(txnFee)$
> request user to validate $txnFee$

else

> update $\mathcal{H}$: $\mathtt{H}_{\mathtt{update}}(pseudoOut)$

Command

CLA	INS	P1	P2	LC
03	7C	01	cnt	var |

Command data

if `cnt==1` :

Length	Value
01	options
01	type
varint	txnFee

if `cnt>1` :

Length	Value
01	options

Length	Value
20	pseudoOut

7.3.7.2 Update MLSAG-prehash

Description

On the second step the application receives amount and destination and check values. It also re-compute the $\mathcal{H}_{\text{outkeys}}$ value to ensure consistency with steps 3 and 4. So for each command received, do:

compute $\mathcal{AK}_{\text{amount}} = \texttt{AES}^{-1}[spk](\widetilde{\mathcal{AK}_{\text{amount}}})$

update $\mathcal{H}_{\text{outkeys}}$" : $\texttt{H}_{\texttt{update}}(A_{out} \mid B_{out} \mid \text{is\_change} \mid \mathcal{AK}_{\text{amount}})$

if blind v1
  compute mask = $\widetilde{\text{mask}}$ - $\texttt{HashToScalar}(\mathcal{AK}_{\text{amount}})$
  compute amount = $\widetilde{\text{amount}}$ - $\texttt{HashToScalar}(\texttt{HashToScalar}(\mathcal{AK}_{\text{amount}}))$

if blind v2
  compute mask = $\texttt{HashToScalar}(\text{"commitment\_mask"}|\mathcal{AK}_{\text{amount}}))$
% #n
  compute $s = \texttt{HashToScalar}(\text{"amount"}||\mathcal{AK}_{\text{amount}})$
  compute amount[0:7] = $\widetilde{\text{amount}}$[0:7] ^ s[0:7]

check C_v == mask.G + amount.h |
update $\mathcal{H}_{\text{commitment}}$: $\texttt{H}_{\texttt{update}}(C_v)$

if last command:
  finalize $\mathcal{H}_{\text{outkeys}}$'
  check $\mathcal{H}_{\text{outkeys}}$' == $\mathcal{H}_{\text{outkeys}}$
  finalize $\mathcal{H}_{\text{commitment}}$

update $\mathcal{H}$: $\texttt{H}_{\texttt{update}}(ecdhInfo)$

ask user validation of A_{out}, B_{out}, amount

Command

CLA	INS	P1	P2	LC
03	7C	02	cnt	E3

Command data

Length	Value
01	options
01	1 if sub-address, 0 else
01	1 if change-address, 0 else
20	Real destination public view key A_{out}
20	Real destination public spend key B_{out}
20	encrypted amount key blinding $\widetilde{\mathcal{AK}}_{\text{amount}}$
20	ephemeral hmac
20	C_v of amount,mask

40	one serialized ecdhInfo :
	{
	bytes[32] mask ($\widetilde{\text{mask}}$)
	bytes[32] amount ($\widetilde{\text{amount}}$)
	}

specific options

`-------xx`	Mask scheme version
`-------10`	Blind V2
`-------00`	Blind V1

Note: Whatever the mask scheme is, amount is always transmitted as 32 bytes.

7.3.7.3 Finalize MLSAG-prehash

Description

Finally the application receives the last part of data:

if cnt == 1
  Initialize $\mathcal{H}_{\text{commitment}}$'

if last command:
  finalize $\mathcal{H}_{\text{commitment}}$'
  check $\mathcal{H}_{\text{commitment}} == \mathcal{H}_{\text{commitment}}$'
  s = finalize $\mathcal{H}$
  compute $\mathcal{H} = \texttt{HashToScalar}\ (message \mid s \mid proof)$

else
  update $\mathcal{H}_{\text{commitment}}$': $\texttt{H}_{\texttt{update}}(C_v)$
  update $\mathcal{H}$: $\texttt{H}_{\texttt{update}}(C_v)$

Keep $\mathcal{H}$

Command

CLA	INS	P1	P2	LC
03	7C	03	cnt	21

Command data

not last:

Length	Value
01	options

20	one serialized commitment :
	{
	bytes[32] mask (C_v)
	}

last:

Length	Value
01	options
20	message (rctSig.message)
20	proof (proof range hash)

Response data

Length	Value

7.3.8 MLSAG

7.3.8.1 MLSAG prepare

Description

Generate the matrix ring parameters:

generate α_{in} ,
compute $\alpha_{in}.G$
if real key:
check the order of H_i
compute $\alpha_{in}.H_i$

compute $\widetilde{\alpha_{in}} = \mathtt{AES}[spk](\alpha_{in})$
if not option_clear_xin:
    compute $x_{in} = \mathtt{AES}^{-1}[spk](\widetilde{x_{in}})$
compute $II_{in} = x_{in}.H_i$

return $\widetilde{\alpha_{in}}$, $\alpha_{in}.G$ $[\alpha_{in}.H_i,\ II_{in}]$

Command

CLA	INS	P1	P2	LC
03	7E	01	cnt	61

specific options

`------x--`	Mask scheme version
`------1--`	unencrypted x_{in}
`------0--`	encryted $\widetilde{x_{in}}$

Command data

for real key:

Length	Value
01	options
20	point
20	secret spend key $\widetilde{x_{in}}$
20	ephemeral hmac

for random ring key

Length	Value
01	options

Response data

for real key:

Length	Value
20	encrypted α_{in} : $\widetilde{\alpha_{in}}$
20	ephemeral hmac
20	$\alpha_{in}.G$
20	II_{in}
20	$\alpha_{in}.H_i$

for random ring key

Length	Value
20	encrypted α_{in} : $\widetilde{\alpha_{in}}$
20	ephemeral hmac
20	$\alpha_{in}.G$

7.3.8.2 MLSAG hash

Description

Compute the last matrix ring parameter:

if cnt == 1:
  replace the inputs by the previously computed MLSAG-prehash
  initialize $\mathcal{H}$

update $\mathcal{H}$: `HashToScalar`(inputs)

if last command:
 c = finalize $\mathcal{H}$ % #`n`

Command

CLA	INS	P1	P2	LC
03	7E	02	cnt	21

Command data

Length	Value
01	options
20	inputs

Response data

if last command

Length	Value
20	c

else

Length	Value

7.3.8.3 MLSAG sign

Description

Finally compute all signatures:

compute $\alpha_{in} = \texttt{AES}^{-1}[spk](\widetilde{\alpha_{in}})$
compute $x_{in} = \texttt{AES}^{-1}[spk](\widetilde{x_{in}})$
compute $ss = (\alpha_{in}$ - c * x_{in}) % l

return ss

Command

CLA	INS	P1	P2	LC
03	7E	03	cnt	81

Command data

Length	Value
01	options
20	$\widetilde{x_{in}}$
20	ephemeral hmac
20	$\widetilde{\alpha_{in}}$
20	ephemeral hmac

Response data

Length	Value
20	signature ss

8 Conclusion

Let's Go

9 Annexes

9.1 References

[1] https://github.com/monero-project/monero/tree/v0.15.0.1
[2] https://github.com/monero-project/monero/pull/2056
[3] https://github.com/kenshi84/monero/tree/subaddress-v2
[4] https://www.reddit.com/r/Monero/comments/6invis/ledger_hardware_wallet_monero_integration
[5] https://github.com/moneroexamples

9.2 Helper functions

|keyDrv| (`KeyDerivation`)

input : r, P
output: $\mathfrak{D}$
Monero: generate_key_derivation

$$\mathfrak{D} = r.P$$
$$\mathfrak{D} = 8.\mathfrak{D}$$

|Hs| (`HashToScalar`)

input: raw
output: s

$$s = h(raw)$$

|Hps| (`HashPointToScalar`)

input: D, idx
output: s

$$data = point2bytes(D)|varint(idx)$$
$$s = h(data)\ \%\ \#\mathtt{n}$$

|Hp| (`HashToPoint`)

input: P
output: Q

$data = point2bytes(P)$
$s = h(data)\ \%\ \#\texttt{n}$
$Q = ge\_from\_fe(s)$

DeriveAES

input: R, a, b
output: spk

$seed = sha256(R|a|b|R)$
$data = sha256(seed)$
$spk = lower16(data)$