RCTTypeCLSAG (Type 5) conceptual (not serialized) components

Input (an instance of this section **for each actual** input UTXO)

	decoy UTXOs **offsets**	**actual** UTXO **offset**	decoy UTXOs **offsets**	
level j=1 CLSAG pubkeys: $\mathbf{X}_i$	$\mathbf{X}_{\dots}$	$\mathbf{X}_{i=\pi}$	$\mathbf{X}_{\dots}$	CLSAG **signature** $(\mathbf{c}_1, \mathbf{r}_i)$ whole TX is signed except signatures' fields
level j=2 CLSAG pubkeys: $\mathbf{C}_i - \mathbf{C}_{pseudo}$	$\mathbf{C}_{\dots}$	$\mathbf{C}_{i=\pi} = \mathbf{bG} + \mathbf{aJ}$	$\mathbf{C}_{\dots}$	
		$\mathbf{C}_{pseudo} \triangleq \mathbf{b'G} + \mathbf{aJ}$		
key images:		effective $\mathbf{X}^*$ artifact $\mathbf{C}^*$ ($\Longrightarrow$ CLSAG's pseudo $\mathbf{W}^*$)		

$(C_{i=\pi} - C_{pseudo}) = (b - b')\,G$

being able to sign with this pub/priv EC keys pair hence proves that $C_{i=\pi}$ and C_{pseudo} commit to the same "a"

Output UTXO "t"

X stealth address

C ≜ bG + aJ commitment

with:

a : moneroj in the UTXO (**plain** value)

b ≜ H("commitment_mask" , H(r V_i , t))
= H("commitment_mask" , H(v_0 R , t))

usual payer/payee "exchange trick" via transaction and view keys

amount

encrypted value:
≜ **a** ⊕ H("amount" , H(r V_i , t))
= **a** ⊕ H("amount" , H(v_0 R , t))

TX Fee

f
plain value, so to make miners able to evaluate it

expressed, as "**a**" in commitments, in **atomic units** ($1/10^{12}$ XMR)

Outputs Unlock Time

absolute time, expressed as:

- **Lock disabled** (0)
- **Block height** (< 500.000.000)
- **Epoch time** (≥ 500.000.000)

protocol enforces a default relative lock of **10 blocks**, too

Bulletproof

output commitments' "**a**" range proofs to avoid:

$5\mathbf{J} + 6\mathbf{J} = 21\mathbf{J} + (\ell - 10)\mathbf{J} = (21 - 10)\mathbf{J} + \ell\mathbf{J}$
(inputs — outputs)

inflation by cyclic group ↻ *overflow*

to be continued…

Extra

- **R** transaction pubkey(s)
- encrypted **payID** (if any)
- **. . .**

NOTE: loosely structured field, signed but not part of consensus ($\Longrightarrow$ wallets duties)

Pedersen Commitments 101

Commitments are a way to bind to a value, without revealing it (maybe postponing the disclosure). Pedersen ones in EC form:

C ≜ bG + aJ

where:

a : committed value

b : random blinding factor, introducing entropy to make hard to get "a" by means of a rainbow table

G : common generator point

J : an EC point for which "j" in: J ≜ j G is unknown (DLP)

PROPERTIES

Theoretically hiding : many (a,b) couples can give the same commitment value C

Computationally binding : pretending to have committed a fake different value is equivalent to solving DLP: $C(a_2, b_2) = C(a, b) \Longrightarrow j = (b - b_2)/(a_2 - a) \Longrightarrow$ DLP solved

Homomorphism : $C(a_1, b_1) + C(a_2, b_2) = C(a_1 + a_2, b_1 + b_2)$

CLSAG recap & now

- CLSAG is a sort of multisignature schema like MLSAG, given that each "squeezed" layer features its own actual key and a set of decoys
- Differently from MLSAG, effective key image protecting against double-spending is available only for layer 1, so each input UTXO needs a separate CLSAG (but MLSAGs were used this way as well, due to anonymity concerns)
- In RingCT two-levels-CLSAGs are used, where layer 2 signature is built to be a proof of equivalence between committed values of the actual UTXO and of the pseudo output commitment C_{pseudo} : the equivalence subsists *iff* a new specific public/private elliptic keys pair exists, so signing the CLSAG's 2nd layer with that pair proves the pair exists and so the equivalence

Secret value flow via Monero's UTXOs

Monero UTXOs encode their value in two fields: the commitment **C** and the **amount**:

- **amount** is the way payer and payee secretly share the value: the payer calculates it from the plain value **a** by the given definition; the payee can get back and verify the plain value **a** by just XORing the **amount** with the same hash function used in its definition (two equal XORs elide themselves). To have an hash computable by both payer and payee, transaction and view keys are used in the same Diffie-Hellman-like way they are in stealth addresses generation
- commitments **C** are the way by which every network node can verify that the sum of transaction input UTXOs values are equal to the sum of transaction output UTXOs values plus fee, all without knowing the actual values (apart from fee's, which is plain by design)

In RingCT schema:

- the payee verifies that output **amount** and **C** express the same value **a** (binding the payer/payee exchanged value to the one validated by the network), thanks to the -already documented- way by which **amount** and **b** are defined
- for each input, payer defines a pseudo output commitment C_{pseudo} bound to same value "a" of the actual UTXO (without revealing its position in the ring, being decoupled by CLSAG signature)
- to validate the value flow, network nodes hence only have to also check balance & bulletproof:

$\sum_{inputs} C_{pseudo} = \sum_{outputs} C + fJ$

$\sum_{inputs} b' = \sum_{outputs} b$ (imposed by the payer)

$\Longrightarrow \sum_{inputs} aJ = \sum_{outputs} aJ + fJ$

$\sum_{inputs} a = \sum_{outputs} a + f$ ← Bullet proof check

Credits & useful sources

- This cheatsheet relies on previous ones dealing with Monero Addresses and Ring Signatures
- Zero to Monero: 2nd Edition chapters 5, 6 and appendices A, B (even if still about MLSAG-signed TXs, CLSAG case is straightforward once you master differences between the two Rings flavours)
- From Zero (Knowledge) to Bulletproofs (first 6 pages dealing with commitments)
- Many Monero Stack Exchange posts (e.g. "Complete extra field structure …")
- MoneroBlocks APIs (e.g. this call to inspect a TX – JSON parsing via jq suggested)
- Monero CLI Source Code (e.g. /src/ringct/rctTypes.h with new signature type 5), hopefully explored via a code-inspector software (e.g. Sourcetrail)