

Confidential Transactions - Initial Overview

Alessandro Miola

September, 2018

Building a RP - 1st issue: how Borromean ring signatures get into

Recall: RP is an additional piece of data needed to prove the committed amount lies in a range of positive values (the amount must neither be negative, nor exceed the field order), without disclosing the committed amount itself (Zero Knowledge RP).

- With the tools presented, it's now possible to prove it.
- The prominent role played by Borromean ring signatures will be soon evident. It was somehow previewed before, slide ??.

Example

Consider again the tx presented before, slide ??.

Let's say that Alice creates a single output (for simplicity), of value 2,010 BTC.

- 1 Need to define what an appropriate range would be.
 - As amounts in Bitcoin transactions are encoded in 32-bit integer data, the definition of the range is straightforward.
Minimum amount: **0** BTC
Maximum amount: **42.94967295** BTC
 (i.e. $4294967295 \text{ satoshis} = 2^{32} - 1 \text{ satoshis}$, maximum amount storable in a 32-bit integer data).
 - This also explains the anticipation at slide ??, **range = $[0, 2^{32})$** .

- 2 Create a Pedersen commitment to the tx value ($v = 201000000$ satoshis).

$$C = rG + vH$$

- 3 Construct an associated Range Proof.
 - 1 Write the output amount in its *binary* expansion (although an optimization considering base-4 encoding will come later).

$$v = v_0 \cdot 2^0 + v_1 \cdot 2^1 + v_2 \cdot 2^2 + \dots + v_{31} \cdot 2^{31}$$

Decimal amount: 201000000 satoshis.

Binary amount: 00001011111110110000010001000000 satoshis.

2 Ring-sign (with a Borromean-style ring signature) **over each digit.**

- **1 ring per digit, 2 verification pubkeys per ring** (a pubkey committing to 0, the other committing to 1).
 \Rightarrow 32 rings, 2 pubkeys each corresponding to digit values (0,1).
- **For each digit create a Pedersen commitment**, making sure the sum of the commitments corresponds to C.

$$C_i = r_i G + v_i 2^i H$$

which is either a commitment to **0** or to **2^i** , without revealing which.

Example: fifth digit from the left (value = 1) in the binary representation \rightarrow amount = $2^{27} = 134217728$
 \Rightarrow Construct the following commitment:

$$C_4 = r_4 G + 134217728 H$$

- **Arrange for one pubkey per digit to be “signable-for”**. That is, provide a Borromean ring signature over the ring:

$$\{r_i G + v_i 2^i H, r_i G + v_i 2^i H - 2^i H\}$$

\Rightarrow if $v_i \notin \{0,1\}$, then neither of the keys in the ring will be a commitment to 0, thus preventing the ring to be signable for (see even slide ??).

Example: given the digit considered before, construct the following set of pubkeys: ¹

$$C_{40} = C_4 - 0H$$

$$C_{41} = C_4 - 134217728H$$

Then perform a Borromean ring signature and particularly perform a

real signature for $C_{41} = \underbrace{C_4}_{r_4 G + 134217728H} - 134217728H = r_4 G$

(for which r_4 is known).

- **Signature size:** the final RP of the value v is:

$$RP_v = (C_0, \dots, C_{31}, \underbrace{e_0, s_0, \bar{s}_0, s_1, \bar{s}_1, \dots, s_{31}, \bar{s}_{31}}_{\text{Borromean}}).$$

Thus, in total: 1 e-value + 32*2 s-values + 32 commitment pubkeys (about 3KB of data).

It would be possible however to save something in terms of space usage by means of a different encoding for the committed value.

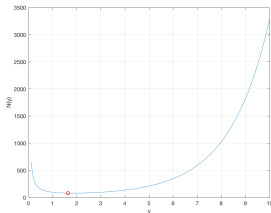
- 3 Consider to use a different encoding for the committed value.
32 bit numbers:

- base 2 \rightarrow 1 bit per digit: $\begin{cases} 0 \leftrightarrow 0 \text{ in bit} \\ 1 \leftrightarrow 1 \text{ in bit} \end{cases}$
- base 4 \rightarrow 2 bits per digit: $\begin{cases} 0 \leftrightarrow 00 \text{ in bits} \\ 1 \leftrightarrow 01 \text{ in bits} \\ 2 \leftrightarrow 10 \text{ in bits} \\ 3 \leftrightarrow 11 \text{ in bits} \end{cases}$

- base **8** \rightarrow 3 bits per digit:

$$\left\{ \begin{array}{ll} 0 & \leftrightarrow \text{000 in bits} \\ 1 & \leftrightarrow \text{001 in bits} \\ \vdots & \\ 6 & \leftrightarrow \text{110 in bits} \\ 7 & \leftrightarrow \text{111 in bits} \end{array} \right.$$
- Given y is the number of bits-per-digit, the total number of pubkeys + s-values is given by:

$$N(y) = \frac{32}{y} \cdot (1 + 2^y)$$



- It takes the minimum around $y=2$ (2 bits per digit \rightarrow base 4).
- Amounts are represented via a **base-4** encoding (i.e. rings of size 4) as this minimizes the number of commitments sent.

- 4 Write the output amount in its *base-4* expansion.

$$v = v_0 \cdot 4^0 + v_1 \cdot 4^1 + v_2 \cdot 4^2 + \dots + v_{15} \cdot 4^{15}$$

Decimal amount: 201000000 satoshis.

Base-4 amount: 0023332300101000 satoshis.

- 5 **Ring-sign** (with a Borromean-style ring signature) **over each digit**.
- Exactly like before. 1 ring per digit, 4 verification pubkeys per ring.
 - Create a Pedersen commitment for each digit, making sure the sum of the commitments corresponds to C.

$$C_i = r_i G + v_i 4^i H$$

Example: third digit from the left (value = 2) in the base-4 representation \rightarrow amount = $2 \cdot 4^{13} = 134217728$
 \Rightarrow Construct the following commitment:

$$C_2 = r_2 G + 134217728 H$$

- Provide a Borromean ring signature over the ring:

$$\{r_i G + v_i 4^i H, r_i G + v_i 4^i H - 4^i H\}$$

Example: given the digit considered before, construct the following set of pubkeys:

$$C_{20} = C_2 - 0H$$

$$C_{21} = C_2 - 67108864H$$

$$C_{22} = C_2 - 134217728H$$

$$C_{23} = C_2 - 201326592H$$

Then perform a Borromean ring signature and particularly perform a **real** signature for $C_{22} = \underbrace{C_2}_{r_2 G + 134217728H} - 134217728H = r_2 G$

(for which r_2 is known).

- **Signature size**: 1 e-value + 16*4 s-values + 16 commitment pubkeys, which is about 2560 bytes (less than it was before).

$$1_{0H} = 0 \cdot 2^{27} H$$

Building a RP - IInd issue: details of ECDH Key Exchange

Remark

What is left out is what concerns sender/receiver communication about:

- setting of the **blinding factors**
- transmission of the **committed amount value**.

Recall:

- From the point of view of the network, RPs just represent a proof that the amounts committed in a tx are valid, but amounts (together with blinding factors) stay secret and only known to participants in the tx.
- As partly previewed (slide ?? ff.), for security as well as efficiency issues, it is required for the transfer not to be interactive (no receiver's involvement). Moreover, it would be nice to have a **space-saving solution** to embed the secret information in the tx without leaking it.



The ultimate Range Proof's construction (at least the one involving Borromean ring signatures) enables the possibility to transfer safely tx value & blinding factors:

- **w/ no interactivity**
- **w/o adding more data in the tx** rather than commitments and RP.
 - ⇒ *Amounts* are embedded in RP w/o taking up more space.
 - ⇒ *Blinding factors* are part of a deterministic generation process; set by the sender and reproducible by sender and receiver only.

In addition to this, it even enables the possibility to **transfer arbitrary user-selected data** (always embedded in RP, no more space needed).

ECDH role

To point out something already anticipated,

- each party in a CT has a separate ephemeral pubkey for ECDH purpose only (**scanning key**),
- an instance of ECDH serves to define a **shared key**.
 - The scheme is the standard one, seen at slide ??.
 - a, b play the role of ECDH ephemeral private keys.

Then,

- this shared key is exploited by both tx sender and receiver to deduce the same blinding factors (and not only them).

Scanning key:

- CT addresses have a different format wrt standard Bitcoin addresses (they are longer than the latter).
- Each CT address embeds a scanning (pub)key for ECDH.
- Format:

base58check(<scanning key version byte> <standard address version byte> <scanning key> <standard address data bytes>)

RFC6979 deterministic generation

- **Blinding factors**
- **Random s-values** needed for Borromean ring signatures

are both part of a deterministic generation process via RFC6979, with the **shared key** derived from ECDH used to *seed* a RFC6979 prng.

Making this process *deterministic*, provides some sort of advantages (besides the ones already discussed).

Especially important among these advantages is the possibility to distinguish *where randomness resides*² and exploit this feature accordingly:

- (for the sender) possibility to **send data** over the predetermined and forged signature values:
 - send a specific information that needs transmission, the **committed tx output amount**
 - send other private messages to the designated tx receiver.
- (for the receiver) possibility to **“rewind” the proof**, which implies:
 - generating the same blinding factors of the sender
 - reading out the tx output amount
 - extract further messages the sender could have sent.

²The focus here is on fake signatures generation.

1 Blinding factors' setting

- The 32-byte shared key derived from ECDH is used as first seed to a RFC6979 prng.
- Then, each output is used as next seed.
- Blinding factors r_0, r_1, \dots, r_{15} are generated (although r_{15} e.g. is chosen by difference so as to match the sum of blinding factors from inputs).

2 Forged signatures & nonces' generation

Recall: $\frac{3}{4}$ of the 64 s-values needed to construct the Borromean ring signature are forged, but need to be indistinguishable from random.

- The RFC6979 prng is used to generate these **fake signature values**.

Moreover, even for what concerns the remaining $\frac{1}{4}$ of the signatures (the real ones), generation of random values is needed.

- The RFC6979 prng is used to generate the **nonces**.

3 Committed tx amount's transmission

- Idea: embed the tx amount somewhere in the construction without letting it occupy more space.
- Solution: embed it into one of the forged signatures (usually the last one, unless it is a real one) by **XORing**³ (\oplus) it **with the forged s-value for that signature**.
 - XOR preserves the entropy of the original pseudo-random s-value.
 - The amount is encoded with particular formatting details.

1 Sender & receiver's viewpoint:

Dealing with forged signatures only (and consequently forged s-values only), let's consider:

- *SCOMMON*: s-value deterministically generated from RFC6979 procedure.
- *SPUBLISH*: s-value actually published by the sender in RP construction.

Sender:

$$SPUBLISH = SCOMMON \oplus amount$$

Receiver:

$$SCOMMON \oplus SPUBLISH = SCOMMON \oplus SCOMMON \oplus amount = amount$$

Thus, the receiver, looking at the last (in general) signature and computing $SCOMMON \oplus SPUBLISH$ sees:

- the amount if the last signature is the one that has been tampered with the amount
- 0 if the last signature hasn't been tampered with the amount, while still being forged.

2 Output amount (8-bytes value) encoding:

Consider again the example run in the previous subsection:

Decimal amount: 201000000 satoshis

Hex amount (8 bytes) + single bytes decimal conversion⁴:

000000000BFB0440							
↑	↑	↑	↑	↑	↑	↑	↑
0	0	0	0	11	251	4	64

3 XOR the amount in 3 separate places into the last of the 64 signatures (if forged)

- XORing in 3 separate fixed places into the last 32 byte value.
- Need to XOR the number “128” with the 1st byte of the s-value of the considered signature to avoid an issue that would occur with null amounts.
- If the s-value into the last signature is not forged, do the same on the last but one.

Indeed, real signatures mustn't be tampered with and the reason why is clear: the real s-value is *necessary* to close the ring and validate the signature.

Remark ①: the amount takes 32 bytes only, all the rest (less the space corresponding to the real s-values) is left for the transmission of generic data (about the 80% of the proof's size).

Remark ②:

- **if** no forged signature has been XORed in, the tx receiver could potentially deduce the amount in another way. He could look at all the s-values and check which ones can be generated via RFC6979 procedure, the other being necessarily real ones. Knowing which are real, he can deduce the amount accordingly. However,
 - this is not the best solution as it would preclude sending other messages.

Remark ③: in the end, the forged s-values (at least some of these, 1 minimum) published as part of the RP are the result of a XORing.

4 Other user-defined data's transmission

- Basically, it works the same as before.
- The only difference is that some user-defined data rather than the tx amount is XORed in with the random s-values.

5 Proof's "rewind" by the receiver

Given the shared key, the receiver can run the same RFC6979 procedure of the sender and deduce:

- same blinding factors
- same forged s-values.

Then,

- he can perform the XOR operation as said before.

According to our example, if the considered signature is effectively the one tampered with the amount, the result of the XOR should be something like this (with the repetitions occurring in positions 9-16, 17-24, 25-32 e.g.):

128,0,0,0,0,0,0,0,0,0,0,0,0,11,251,4,64,0,0,0,0,11,251,4,64,0,0,0,0,11,251,4,64

The receiver would then:

- Convert each byte in its hexadecimal representation.
- Convert the whole hexadecimal number in base 10.

³XOR: bit-wise addition mod 2 (w/o carry). Properties:
 $A \oplus 0 = A$; $A \oplus A = 0$; $(A \oplus B) \oplus C = A \oplus (B \oplus C)$; $(B \oplus A) \oplus A = B \oplus 0 = B$.

⁴Necessary because XOR acts on single bytes. The idea is: XOR with single bytes decimal-converted number, pass to hexadecimal and then come back to decimal. It is built in such a way that the concatenation of bytes resulting from the XORing forms the hexadecimal number corresponding to the amount (in base 10).

Building a RP - IIIrd issue: explicit fees

- In a standard Bitcoin transaction the fee is *implicit*, what comes out from the difference between input and output values.
- In CT, amounts are not published to the network → the fee amount **f** is published **explicitly** as it cannot be deduced implicitly as before.
 - Published as a plaintext, 4-byte amount after the inputs and before the outputs.
- Consequences on the commitment scheme:
 - Construct a Pedersen commitment for the fee amount.
 - The necessity to keep fee amount explicit (which opposes to the Pedersen commitment's aim to obscure tx amount) makes it necessary for fees to be paid “**unmasked**”, just as **fH** rather than $rG + fH$.
 - Thus, it is built with blinding factor $r_f = 0$.
- The network just checks that $\sum_i C_i^{inp} - \sum_i C_i^{out} - fH = 0$.

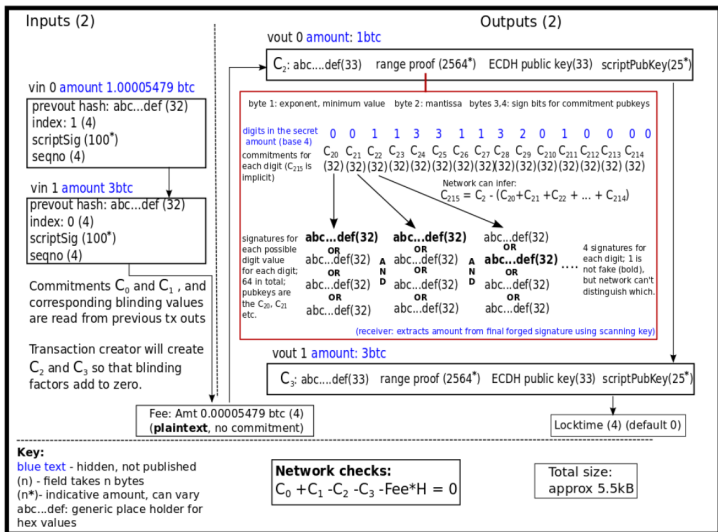





Figure 1: A near complete schematic of a CT transaction with 2 inputs and 2 outputs. The arrows show the path of serialization as it is published on the network. Some minor details are missing.


Figure: figure from "An investigation into Confidential Transactions" - A. Gibson

References

-  https://people.xiph.org/~greg/confidential_values.txt
-  <https://github.com/AdamISZ/ConfidentialTransactionsDoc>
-  https://github.com/Blockstream/borromean_paper/raw/master/borromean_draft_0.01_8c3f9e7.pdf
-  Slides from - Bulletproofs: Short Proofs for Confidential Transactions and More
B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille and G. Maxwell
<https://crypto.stanford.edu/~buenz/publications/>
-  <https://bitcointalk.org/index.php?topic=1085273.40>
-  <https://github.com/AdamISZ/bulletproofs-poc>

 <http://dihypl.us/wiki/transcripts/gmaxwell-confidential-transactions/>


 <https://ledgerjournal.org/ojs/index.php/ledger/article/viewFile/34/61>

 <http://cryptoservices.github.io/cryptography/2017/07/21/Sigs.html>

 <https://moderncrypto.org/mail-archive/curves/2015/000534.html>

 <https://bitcointalk.org/index.php?topic=285142.40>

 <https://tools.ietf.org/html/rfc6979#section-3.2>

 <https://bitcoin.stackexchange.com/questions/48064/sending-confidential-transaction-amount-to-the-receiver>
MANY MORE MISSING.....