Bitcoin–Monero Cross-chain Atomic Swap Implementation notes

v0.2 DRAFT

tecnovert*

2020-09-10

1 Introduction

Notes on implementing the protocol described by h4sh3d in Bitcoin–Monero Cross-chain Atomic Swap[5]

2 Discrete Logarithm Equality Across Groups

A Discrete Logarithm Equality Across Groups (DLEAG) proof can prove a discrete logarithm is equal for two points on different curves.
We use the scheme from MRL-0010[8], slightly modified to use a borromean[4] ring stack for a reduced proof size.
The curves considered are secp256k1[10] used by the Bitcoin[7] cryptocurrency and forks and ed25519[1] used by Monero[11].
As the order of the generator point of the ed25519 curve l is less than that of curve secp256k1 o the shared secret must be less than l.
To represent $l - 1$ in base2, we must prove 252 bits.

2.1 Definitions

- G - The base point of curve secp256k1
- $\hat{G}$ - An alternate base point of curve secp256k1 in the same order as G, chosen as G hashed to a point, proving it's discrete logarithm is unknown.
- B - The base point of curve ed25519
- $\hat{B}$ - An alternate base point of curve ed25519 in the same order as B, chosen as B hashed to a point, proving it's discrete logarithm is unknown.
- l - Order of B and $\hat{B}$
- o - Order of G and $\hat{G}$
- n - Number of bits proved
- HashG() - A hash function which outputs an integer $\in \mathbb{Z}_o$
- HashB() - A hash function which outputs an integer $\in \mathbb{Z}_l$

2.2 DLEAGProve(x)

1. Split x into an array of bits b so $\sum_{i=0}^{n-1} b_i = x$
2. For each $i \in [0, n-2]$ generate random blinding factors:
   $r_i \in \mathbb{Z}_o$ and $s_i \in \mathbb{Z}_l$

*tecnovert@tecnovert.net

3. Set the last blinding factors to:
   $r_{n-1} = (2^{n-1})^{-1}(-\sum_{i=0}^{n-2} r_i 2^i) \in \mathbb{Z}_o$
   and
   $s_{n-1} = (2^{n-1})^{-1}(-\sum_{i=0}^{n-2} s_i 2^i) \in \mathbb{Z}_l$
   Causing each set of weighted blinding factors to sum to 0
   $\sum_{i=0}^{n-1} r_i 2^i == 0$ and $\sum_{i=0}^{n-1} s_i 2^i == 0$

4. For each $i \in [0, n-1]$ compute two commitments: $C_i^G = b_i G + r_i \hat{G} \in \mathbb{Z}_o$
   $C_i^B = b_i B + s_i \hat{B} \in \mathbb{Z}_l$
   so the weighted commitment sums equal xG and xB
   $\sum_{i=0}^{n-1} C_i^G 2^i = xG$
   $\sum_{i=0}^{n-1} C_i^B 2^i = xB$

5. Construct the ring stack, for each $i \in [0, n-1]$:
   (a) Choose random $j_i \in \mathbb{Z}_o$ and $k_i \in \mathbb{Z}_l$
   (b) Set $J_{i,b_i} = j_i \hat{G}$ and $K_{i,b_i} = j_i \hat{B}$
   (c) For each $z \in [b_i + 1, 2]$:
      i. e_G =HashG$(C_i^G || C_i^B || J || K || i || z)$
         e_B =HashB$(C_i^G || C_i^B || J || K || i || z)$
      ii. Choose random $a_{i,z}^G \in \mathbb{Z}_o$ and $a_{i,z}^B \in \mathbb{Z}_l$
      iii. $J_{i,z} = a_{i,z}^G \hat{G} - e_G(C_i^G - zG)$
         $K_{i,z} = a_{i,z}^B \hat{B} - e_B(C_i^B - zB)$

6. $e_0^G = HashG(J_{0,2} || ... || J_{i-1,2})$ and
   $e_0^B = HashG(K_{0,2} || ... || K_{i-1,2})$

7. Sign the ring stack, For each $i \in [0, n-1]$:
   (a) e_G =HashG$(C_i^G || C_i^B || e_0^G || e_0^B || i || 0)$
   (b) e_B =HashB$(C_i^G || C_i^B || e_0^G || e_0^B || i || 0)$
   (c) For each $z \in [0, b_i]$:
      i. Choose random $a_{i,z}^G \in \mathbb{Z}_o$ and $a_{i,z}^B \in \mathbb{Z}_l$
      ii. $J_{i,z} = a_{i,z}^G \hat{G} - e_G(C_i^G - zG)$
         $K_{i,z} = a_{i,z}^B \hat{B} - e_B(C_i^B - zB)$
      iii. e_G =HashG$(C_i^G || C_i^B || J || K || i || z + 1)$
         e_B =HashB$(C_i^G || C_i^B || J || K || i || z + 1)$
   (d) Close the loop:
      $a_{i,b_i}^G = j_i + e_G r_i$
      $a_{i,b_i}^B = k_i + e_B s_i$

8. Return the proof $(xG, xB, \{C_i^G\}, \{C_i^B\}, e_0^G, e_0^B, \{a_{0,i}^G\}, \{a_{1,i}^G\}, \{a_{0,i}^B\}, \{a_{1,i}^B\})$
   Proof length is $33 + 32 + 32 + 32 + (33 + 32 + (32)4)n == 129 + 193n$ bytes.

2.3 DLEAGVerify(proof)

$(xG, xB, \{C_i^G\}, \{C_i^B\}, e_0^G, e_0^B, \{a_{0,i}^G\}, \{a_{1,i}^G\}, \{a_{0,i}^B\}, \{a_{1,i}^B\})$ =proof

1. Verify the weighted commitment sums equal xG and xB
   $\sum_{i=0}^{n-1} C_i^G 2^i = xG$
   $\sum_{i=0}^{n-1} C_i^B 2^i = xB$

2. Verify the ring stack, for each $i \in [0, n-1]$:
   (a) $J_{i,z} = a_{i,z}^G \hat{G} - e_G(C_i^G - zG)$
      $K_{i,z} = a_{i,z}^B \hat{B} - e_B(C_i^B - zB)$

(b) e_G =HashG($C_i^G||C_i^B||J||K||i||z+1$)
e_B =HashB($C_i^G||C_i^B||J||K||i||z+1$)

3. $e_0^{'G} = HashG(J_{0,2}||...||J_{i-1,2})$ and
$e_0^{'B} = HashG(K_{0,2}||...||K_{i-1,2})$
4. return 1 if $e_0^{'G} == e_0^G$ and $e_0^{'B} == e_0^B$

3 Non-Interactive Zero Knowledge proof of Discrete Logarithm Equality

The One-Time VES scheme requires a NIZK DLEQ from [2]

Given two generator points and two points on the same curve a DLEQ proves the discrete logarithm of the two points is equal.

3.1 DLEQProve

Take points B_1, B_2 on the same curve and of the same group order and a scalar $x \in \mathbb{Z}_o$ where o is the group order.
Hash is a function returning a number $\in \mathbb{Z}_o$

1. Calculate $P_1 = xB_1$ and $P_2 = xB_2$
2. Generate a random scalar $k \in \mathbb{Z}_o$
3. Calculate $K_1 = kB_1$ and $K2 = kB_2$
4. Set $c = Hash(P_1||P_2||K_1||K2)$
5. Set $r = k - cx$
6. The proof is the tuple (K_1, K_2, r)

3.2 DLEQVerify

Take points P_1, P_2 and the proof tuple (K_1, K_2, r) as inputs.

1. Set $c = Hash(P_1||P_2||K_1||K_2)$
2. Calculate $R_1 = rB_1$ and $R_2 = rB_2$
3. Calculate $C_1 = cP_1$ and $C_2 = cP_2$
4. Output 1 if $K_1 == R_1 + C_1$ and $K2 == R_2 + C_2$ else 0

4 One-Time Verifiably Encrypted Signatures

Also known as Adaptor Signatures.
A One-Time VES is a signature made invalid by mixing it with the public key of an encrypting key pair, a valid signature can be decrypted with knowledge of the private encrypting key and the private encrypting key can be recovered with knowledge of both the encrypted and plaintext signatures. We use a VES constructed to function with ECDSA signatures as described by Fournier et al. in [3]

4.1 EncSign(p_S, P_E, m)

On input of a secret signing key p_S, a public encryption key P_E , and a message m, output a ciphertext $\hat{\sigma}$ which is an encrypted signature of m by p_S.

1. Generate a random scalar $r \in \mathbb{Z}_o$
2. Calculate $R_1 = rG$ and $R_2 = rP_E$
3. π = DLEQProve(G, P_E, r)
4. Set R_{2x} to the x coord of R_2 mod o
5. Calculate $\hat{s} = r^{-1}(Hash(m) + R_{2x}p_S)$
6. Return the tuple $(R_1, R_2, \hat{s}, \pi)$ as $\hat{\sigma}$

4.2 EncVrfy($P_S, P_E, m, \hat{\sigma}$)

On input of a public signing key P_S, a public encryption key P_E , a message m and a ciphertext $\hat{\sigma}$ output 1 if $\hat{\sigma}$ is a valid encryption of a signature on m for P_S under P_E.

1. Set $(R_1, R_2, \hat{s}, \pi) = \hat{\sigma}$
2. Fail if DLEQVerify(R_1, R_2, π) != 1
3. Set R_{2x} to the x coord of R_2 mod o
4. Output 1 if $R_1 = (Hash(m)G * R_{2x}P_S)\hat{s}^{-1}$ else 0

4.3 DecSig($p_E, \hat{\sigma}$)

Decrypt the signature with the private encryption key p_E.

1. Set $(R_1, R_2, \hat{s}, \pi) = \hat{\sigma}$
2. Calculate $s = \hat{s}p_E^{-1}$
3. Set R_{2x} to the x coord of R_2 mod o
4. Return the tuple (R_{2x}, s) as σ

4.4 RecEncKey($P_E, \hat{\sigma}$, σ)

Recover the private encryption key p_e from the plaintext and encrypted signatures.

1. Set $(R_1, R_2, \hat{s}, \pi) = \hat{\sigma}$
2. Set $(R_{2x}, s) = \sigma$
3. $p = s^{-1}\hat{s}$
4. Return p if $pG == P_E$ else $-p$

5 Atomic Swap Protocol

5.1 Basic Idea

Two parties want to exchange coins between blockchains in a such a way that neither can be cheated.

The swap leader exchanges coin from the script-enabled blockchain and sends the first transaction.
The leader wants to exchange an amount of coinA for an amount of coinB.
The follower wants to exchange an amount of coinB for an amount of coinA.

1. Leader and follower exchange details necessary for the swap.

2. Once they agree on the details the leader publishes a lock transaction which locks up the amount of coinA being exchanged by sending it to an output that can only be spent if both sides cooperate until the time agreed for the exchange expires when the leader can reclaim it.

3. The follower waits for the coinA lock transaction to be confirmed in it's blockchain. If he's satisfied the transaction was formed correctly he publishes a transaction locking up the amount of coinB being exchanged. The coinB lock tx can only be spent with knowledge of a private key from both the leader and follower.

4. The leader waits for the coinB lock transaction to be confirmed in it's blockchain. If he's satisfied the transaction was formed correctly he reveals information to the follower which allows him to spend from the coinA lock transaction.

5. By spending from the coinA lock transaction the follower reveals information to the leader allowing him to spend from the coinB lock transaction.

6. The leader spends from the coinB lock transaction and the exchange is completed successfully.

Leader:
- Loses: coinA locked value, 1x coinA tx fees and 1x coinB tx fees.
- Gains: coinB locked value.
Follower:
- Loses: coinB locked value, 1x coinB tx fees and 1x coinA tx fees.
- Gains: coinA locked value.

5.2 Details

Step 2:
In addition to the coinA lock tx the parties craft a transaction spending from the coinA lock tx refund path and spendable either by both parties cooperating or by the follower after a second locktime has expired. Before the leader publishes the coinA lock tx he will know the information to spend the coinA refund tx immediately after it's published however spending from the coinA refund tx cooperative path reveals information to the follower allowing him to from the coinB lock transaction.
The refund path could be built into the coinA lock tx, but would increase the size of the coinA lock tx significantly and should seldom be required.

Step 3:
The coinB lock tx outputs to a key which is the sum of a key from the leader and a key from the follower. The One-Time Verifiably Encrypted Signatures are crafted with this key as the encryption key, and the information distributed so the follower reveals his key to the leader by spending from the coinA lock tx and the leader reveals his key to the follower by spending from the coinA lock refund tx.

If something goes wrong and the leader or follower stop responding:

After step 2, before step 3:
If the follower drops out:
The leader can retrieve his coin from the coinA lock tx by waiting for the exchange time to expire and publishing the coinA refund tx. Then spending from the coinA refund tx cooperative path. The follower has not yet locked any coinB.
Leader loses 2x coinA tx fees, follower loses nothing.

If the leader drops out:
The follower can wait for the first locktime to expire, then publish the coinA refund tx. Once the coinA refund tx locktime expires the follower can spend from it with only his signature, thus claiming the amount of coinA while still retaining the coinB.
Leader:
- Loses: 1x coinA fees and locked coinA value.
Follower:
- Loses: 1x coinA fees, 1x coinB fees.

- Gains: Locked coinA value

After step 3:
If the follower drops out:
As at step 2, by spending from the coinA refund tx cooperative path the leader reveals information allowing the follower to spend from the coinB lock tx.
Leader:
- Loses: 2x coinA tx fees.
Follower:
- Loses: 2x coinB tx fees.

If the leader drops out:
As at step 2 except the follower has published the coinB lock tx.
Neither party can spend the coinB lock tx without assistance from the other.
Leader:
- Loses: 1x coinA fees and locked coinA value.
Follower:
- Loses: 1x coinA fees, 1x coinB fees and locked coinB value.
- Gains: locked coinA value.

5.2.1 Transaction scripts

Listing 1: Lock tx output script

```
OP_IF
    OP_SHA256 {secret_hash} OP_EQUALVERIFY
    OP_2 {pk_leader} {pk_follower} OP_2 OP_CHECKMULTISIG
OP_ELSE
    {lock_for_1} OP_CHECKSEQUENCEVERIFY OP_DROP
    OP_2 {pk_leader_refund} {pk_follower_refund} OP_2 OP_CHECKMULTISIG
OP_ENDIF
```

Listing 2: Lock refund tx output script

```
OP_IF
    OP_2 {pk_leader_refund} {pk_follower_refund} OP_2 OP_CHECKMULTISIG
OP_ELSE
    {lock_for_2} OP_CHECKSEQUENCEVERIFY OP_DROP
    {pk_follower} OP_CHECKSIG
OP_ENDIF
```

5.2.2 Checking the secret length

Checking the secret length as required in Decred style atomic swaps [6] is not strictly necessary as the secret value and hash will only be used on one blockchain, it may help to prevent unexpectedly large secret values from being used to throw off the transaction fee.

6 Future Enhancements

- coinA lock and refund scripts can be joined into one more efficiently on chains where the taproot bip-0341[9] improvement is active.
- DLEAG proof could prove multiple bits per ring layer, reducing the proof size.
- Improve encoding of VES data to reduce size.

References

[1] Daniel J. Bernstein et al. "High-Speed High-Security Signatures". In: *CHES*. Vol. 6917. Lecture Notes in Computer Science. Springer, 2011, pp. 124–142. DOI: 10.1007/978-3-642-23951-9_9. URL: https://www.iacr.org/archive/ches2011/69170125/69170125.pdf.

[2] David Chaum and Torben P. Pedersen. "Wallet Databases with Observers". In: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '92. London, UK, UK: Springer-Verlag, 1993, pp. 89–105. ISBN: 3-540-57340-2. URL: http://dl.acm.org/citation.cfm?id=646757.705670.

[3] Lloyd Fournier. *One-Time Verifiably Encrypted Signatures, A.K.A. Adaptor Signatures*. 2019. URL: https://github.com/LLFourn/one-time-VES/blob/master/main.pdf.

[4] A Poelstra G Maxwell. *Borromean Ring Signatures*. URL: https://github.com/Blockstream/borromean_paper/.

[5] h4sh3d. *Bitcoin–Monero Cross-chain Atomic Swap*. 2020. URL: https://github.com/h4sh3d/xmr-btc-atomic-swap/blob/master/whitepaper/xmr-btc.pdf.

[6] Mark B Lundeberg. *Advisory: secret size attack on cross-chain hash lock smart contracts*. URL: https://gist.github.com/markblundeberg/7a932c98179de2190049f5823907c016 (visited on 2020).

[7] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Dec. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[8] Sarang Noether. *MRL-0010: Discrete logarithm equality across groups*. 2018. URL: https://web.getmonero.org/es/resources/research-lab/pubs/MRL-0010.pdf.

[9] Anthony Towns Pieter Wuille Jonas Nick. *Taproot: SegWit version 1 spending rules*. URL: https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki (visited on 2020).

[10] Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters*. 2010. URL: http://www.secg.org/sec2-v2.pdf.

[11] Nicolas Van Saberhagen. *CryptoNote v 2.0*. 2013.