Bitcoin–Monero Cross-chain Atomic Swap

June 16, 2020 – DRAFT

h4sh3d[1]

h4sh3d@protonmail.com

Abstract. In blockchains where hashed timelock contracts are possible atomic swaps are already deployed, but when one blockchain doesn't have this capability it becomes a challenge. This protocol describes how to achieve atomic swaps between Bitcoin and Monero with two transactions per chain without trusting any central authority, servers, nor the other swap participant. We propose a swap between two participants, one holding bitcoin and the other monero, in which when both follow the protocol their funds are not at risk at any moment. The protocol does not require timelocks on Monero side nor script capabilities but does require two proofs of knowledge of equal discrete logarithm across the `edward25519` and the `secp256k1` groups and `ECDSA` one-time VES.

Keywords: Bitcoin, Monero, atomic swaps, ECDSA one-time VES, discrete logarithm equality across groups

1 Introduction

We describe a protocol for an on-chain atomic swap between Monero and Bitcoin, but the protocol can be generalized for any cryptocurrency that fulfills the same requirements as Monero to any other cryptocurrency that fulfills the same requirements as Bitcoin, see chapter 3.

Participants send funds into a specific address generated during the process (the lock) on each chain (cross-chain) where each party can take control of the funds on the other chain (swap) atomically (i.e. claiming of funds on either chain is mutually exclusive from the ability to claim funds from the other chain.)

1.1 Known limitations

To provide liveness (if at least one participant is still online) we allow for the worst case scenario in which a participant may end up loosing funds (by not being able to claim on the other chain). This can happen in the case where they do not follow the protocol, e.g. remaining online during pending swap or claiming funds in time. The rationale behind this design is explained in 2.3.

Fees are different from one chain to the other partly because of internal blockchain parameters & transaction complexity, and also due to external factors such as demand for blockspace. Note that within this protocol the Bitcoin blockchain is used as a decision engine, where we use advanced features of bitcoin—although we try to put as much logic as we can off-chain—, which causes bigger transactions on the bitcoin side. These two factors combined make the Bitcoin transactions more expensive in general than those on the Monero chain.

Instant user feedback in a cross-chain atomic swap is hard to achieve. The slowest chain and the number of confirmations required for transaction finality dictate the speed of the protocol, allowing front running in some cases. The protocol can be extended to prevent front running within certain setups however. It is worth noting that front running cannot be enforced by one participant on the other, thus making the worst case scenario the loss of transaction fees on each of participants' respective blockchain.

2 Scenario

We describe the participants and their incentives. Alice, who owns monero (XMR), and Bob, who owns bitcoin (BTC), want to swap funds. We assume that they already have negotiated the price in advance (i.e. amount of bitcoin for amount of monero to swap.) This negotiation can also be integrated into the protocol, for example by swap services who provide a price to their customers.

Both participants wish to only have two possible execution paths (which are mutually exclusive to each other) when executing the protocol: (1) the protocol succeeds and Alice gets bitcoin, Bob gets monero, or (2) the protocol fails and both keep their original funds minus the minimum transaction fees possible.

2.1 Successful swap

If both participants follow the protocol there will be four transactions broadcast in total—only three if the Monero are not moved directly after completion, which is not a problem—, two on the Bitcoin blockchain and two on the Monero blockchain. The first ones on each chain lock the funds and makes them ready for the trade on each chain. The second ones unlock the funds for one participant only and gives knowledge to the other participant who takes control of the output on the other chain.

This is the optimal execution of the protocol, requiring no timelocks, the minimum number of transactions and only locking funds for the minimum confirmation on each chain depending on the level of security expected by each participant, i.e. how many confirmations each expects for the funding transaction to be considered final and continue to the next step of the process.

2.2 Swap correctly aborted

When locking the bitcoin, after a timelock, Alice or Bob can start the process of refunding the locked funds. At that moment the monero might not be locked yet, if no monero are locked the refund process will just refund the bitcoin, otherwise Alice will learn enough information to refund her monero too.

When the refund transaction is broadcast Bob must spend the refund before some timelock, otherwise he might end up loosing his bitcoin without getting any monero. Because of this we can describe this as an interactive protocol from Bob's perspective, i.e. Bob cannot go offline – he must react to such a situation during the swap. Alice on the other hand, can remain offline.

2.3 Worst case scenario

If the swap is canceled with the refund process and Bob does not spend his refund before the timelock, Alice can claim the refund without revealing the knowledge needed to Bob to claim on the other chain. Thus one participant, Bob, ends up disadvantaged and three Bitcoin transactions are needed instead of two.

Rationale This choice is made to avoid the following case: if monero are locked, Alice will be able to refund them only if Bob refunds his bitcoin first. We need an incentive mechanism to force Bob to spend his refund to prevent a deadlock in the refund process or compensate Alice if Bob does not follow the protocol correctly.

3 Prerequisites

As previously described, conditional execution must be possible in order to achieve a swap with atomicity. Bitcoin has a simple stack-based script language that allows for conditional execution and timelocks. On the other hand, at the moment, Monero with its privacy oriented RingCT design provides only signatures to unlock UTXOs. Meaning that control of UTXOs is only related to who controls the associated private keys. The challenge is then to move control of funds only with knowledge of some private keys.

3.1 Monero

Monero does not require any particular primitives on-chain (hashlocks, timelocks), all building blocks are off-chain primitives. Thus we need to provide proofs of the correct initialization of the protocol, which might be the hardest part, proofs are described in chapter 4.3.

Secret shares, to enable a basic two-path execution in Monero. The Monero private spend key is split into two secret shares k_a^s, k_b^s. Participants will not use any multi-signing protocol, instead, the private spend key shares are distributed during initialization of the swap process where one participant will gain knowledge of the full key $k^s \equiv k_a^s + k_b^s \pmod{l}$ at the end of the protocol execution, either for a completed swap or for an aborted swap.

3.2 Bitcoin

The bitcoin transactions in this protocol require the fix for transaction malleability provided by the `SegWit` upgrade. This allows us to chain transactions without necessarily broadcasting them. This protocol is only compatible with cryptocurrencies which use a bitcoin style UTXO model and have an equivalent malleability fix such as Litecoin (i.e. Bitcoin Cash is not compatible.)

Timelock, to enable new execution paths after some predefined amount of time, e.g. start the refund process after having locked funds on-chain without creating a race condition. It is worth noting that we do not require timelocks for Monero.

2-of-2 multisig, to create a common path accessible only by the two participants if both agree. In this protocol we use the on-chain option but off-chain multi-signature scheme can replace it.

Semi-Scriptless protocol, to reveal a secret and allow the protocol to continue execution without requiring complex scripting capabilities. It is worth noting that full scriptless script protocol should be able to achieve the same results.

3.3 Equal discrete logarithm across groups zero-knowledge proof of knowledge

Equal discrete logarithm across groups zero-knowledge proof of knowledge proves that a valid discrete logarithm α, given two groups with fixed generators G and H, where $\alpha \leq min(|G|, |H|)$, exists such that given xG and yH: $x = y = \alpha$. Groups are `edward25519` with $|G| = l$ and `secp256k1` with $|H| = n$.

Curves parameters Bitcoin and Monero do not use the same elliptic curves. Bitcoin uses the `secp256k1` curve as defined in *Standards for Efficient Cryptography (SEC)* with the `ECDSA` algorithm while Monero, based on the second version of CryptoNote [4], uses `curve25519`, hereinafter also `edward25519`, from Daniel J. Bernstein [3].

We denote curve parameters for

`edward25519` as

$$\begin{aligned} &q : \text{a prime number; } q = 2^{255} - 19 \\ &d : \text{an element of } \mathbb{F}_q\text{; } d = -121665/121666 \\ &\mathcal{E} : \text{an elliptic curve equation; } -x^2 + y^2 = 1 + dx^2y^2 \\ &G : \text{a base point; } G = (x, -4/5) \\ &l : \text{the base point order; } l = 2^{252} + 27742317777372353535851937790883648493 \end{aligned} \tag{1}$$

`secp256k1` as

$$\begin{aligned} &p : \text{a prime number; } p = 2^{256} - 2^{32} - 977 \\ &a : \text{an element of } \mathbb{F}_p\text{; } a = 0 \\ &b : \text{an element of } \mathbb{F}_p\text{; } b = 7 \\ &\mathcal{E}' : \text{an elliptic curve equation; } y^2 = x^3 + bx + a \\ &H : \text{a base point; } H = \\ &\big(\texttt{0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798}, \\ &\texttt{0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8}\big) \\ &n : \text{the base point order; } n = 2^{256} - 432420386565659656852420866394968145599 \end{aligned} \tag{2}$$

4 Protocol

The overall protocol is as follows: Alice moves the monero into an address where each participant controls half of the private spend key (later referred to as key "shares"). The Bitcoin scripting language is then used with semi-scriptless protocols to reveal one of the halfs of the private spend key depending on which participant claims the bitcoin. Depending on who reveals their half of the private spend key, the locked monero changes ownership. Bitcoin transactions are designed in such a way that if a participant follows the protocol they can't terminate with a loss.

If the deal goes through, Alice spends the bitcoin by revealing her private key share, thus allowing Bob to spend the locked monero. If the deal is canceled, Bob spends the bitcoin after the first timelock by revealing his private key share thus allowing Alice to spend the monero, in both cases minus transaction fees.

Key exchange is performed with `ECDSA` one-time VES—also adaptor signature—, as demonstrated in [1], and equal discrete logarithm across groups zero-knowledge proof of knowledge[2]. One-time VES are constructed such that given the ciphertext and the decrypted signature, the decryption key is easily recoverable. By setting the private key share—one half of the Monero full private spend key—as the decryption key we have a way to atomically sell the private key share to the other participant. Thus, because Bitcoin and Monero use different curves, we need to prove the relation between a point on `edward25519` and `secp256k1` to ensure a trustlessness.

4.1 Non-interactive refund

If Alice and or Bob locked their funds but one abort the swap, or stop communicating at some point, the protocol must not require interactivity to complete the refund procedure for both participant. Otherwise Alice can hostage Bob by not responding and wait for the second timelock to get free bitcoin. In a gracefully aborted swap Bob should reveal his Monero private key share allowing Alice to get her monero back through one one-time VES.

`ECDSA` one-time VES are interactive, one has to provide the encrypted signature and if the verification succeeds the counterparty provides a valid signature for the 2-of-2 Bitcoin multisig back, thus allowing the former to decrypt and publish two valid signatures for the 2-of-2 multisig and the latter to learn—on-chain—the decrypted signature and recover the decryption key.

As mentionned before the refund process must be non-interactive, the protocol is design in such a way that Alice learns Bob's refund encrypted signature and provides to Bob a valid refund 2-of-2 multisig signature before locking the funds. Bob can, in the case of a refund, decrypt and publish the signature without Alice cooperation.

4.2 Monero private keys

Monero private keys are pairs of `edward25519` scalars. One is called the private view key and one is called the private spend key. We use small letters to denote private keys and capital letters for public keys such that

$$X = xG$$

Where G is the generator element of the curve. We denote

(i) the private key k^v as the full private view key,

(ii) K^v as the full public view key,

(iii) k_a^v as the private view key share of Alice and k_b^v of Bob,

(iv) the private key k^s as the full private spend key,

(v) K^s as the full public spend key,

(vi) and k_a^s as the private spend key share of Alice and k_b^s of Bob.

Partial keys We denote private key shares as k_a^s and k_b^s such that

$$k_a^s + k_b^s \equiv k^s \pmod{l}$$

And then

$$\begin{aligned} k_a^s G &= K_a^s \\ k_b^s G &= K_b^s \\ K_a^s + K_b^s = (k_a^s + k_b^s)G &= k^s G = K^s \end{aligned} \tag{3}$$

The same holds for k^v with k_a^v and k_b^v.

4.3 Zero-Knowledge proofs

Zero-knowledge proofs are required at the beginning to make the protocol trustless. The protocol uses one-time VES to reveal private key shares, but we cannot check the discrete logarithm equality between the Monero public key share and Bitcoin public decryption key of the other participant before it goes on-chain. Thus we need to provide a proof that the discrete logarithm is the same across the two groups.

Equal discrete logarithm across groups Alice and Bob must prove to each other with

$$\begin{aligned} k_i^s &\leftarrow \text{valid scalar on } \texttt{edward25519} \text{ and } \texttt{secp256k1} \\ K_i^s &= k_i^s G \\ B_i^s &= k_i^s H \end{aligned} \tag{4}$$

for $i \in \{a, b\}$, given K_i^s and B_i^s that

$$\exists k_i^s \mid K_i^s = k_i^s G \wedge B_i^s = k_i^s H \wedge k_i^s < \mathsf{min}(l, n) \tag{5}$$

4.4 Time parameters

Two timelocks t_0, t_1 are defined during the initialization phase. t_0 sets the time window during which it is safe to execute the trade, after t_0 the refund process may start, making the trade unsafe to complete because of a potential race condition (even if hard to exploit in reality). t_1 sets the response time during which Bob is required to react and reveal his private Monero share to get his bitcoin back and allow Alice to redeem her monero (if moneroj have been locked). After t_1 Alice is able to claim the bitcoin unilaterally.

4.5 Bitcoin scripts

`SWAPLOCK` is a script used to lock funds and defines the two base execution paths: (1) swap execution—success—and (2) refund execution—fail—. We define the `SWAPLOCK` script as:

```
OP_IF
    2 <B_a> <B_b> 2 OP_CHECKMULTISIG
OP_ESLE
    <t_0> OP_CHECKSEQUENCEVERIFY OP_DROP
    2 <B_a^r> <B_b^r> 2 OP_CHECKMULTISIG
OP_ENDIF
```

Buy `SWAPLOCK`, Alice takes control of bitcoin and reveals her Monero key share to Bob with σ_1, a one-time VES leaking k_a^s, thus allowing Bob to take control of the monero. $\texttt{BTX}_{claim}$ redeem the `SWAPLOCK` with:

```
OP_0 <σ_1> <σ_2> OP_TRUE <SWAPLOCK script>
```

Refund `SWAPLOCK`, signed by both participants and moves the funds into the `REFUND` script. $\texttt{BTX}_{refund}$ redeem the `SWAPLOCK` with:

```
OP_0 <σ'_r> <σ''_r> OP_FALSE <SWAPLOCK script>
```

REFUND is a script used in case the swap already started on-chain but is cancelled. This refund script is used to move the funds out of the SWAPLOCK script with the 2-of-2 timelocked multisig. We define the REFUND script as:

```
OP_IF
    2 <B_a^r> <B_b^r> 2 OP_CHECKMULTISIG
OP_ESLE
    <t_1> OP_CHECKSEQUENCEVERIFY OP_DROP
    <B_a> OP_CHECKSIG
OP_ENDIF
```

Spend REFUND, Bob cancels the swap and reveals his Monero private share with σ_1', a one-time VES leaking k_b^s, thus allowing Alice to regain control over her Monero. $\texttt{BTX}_{spend}$ redeem the REFUND with:

```
OP_0 <σ'_1> <σ'_2> OP_TRUE <REFUND script>
```

Claim REFUND, Alice takes control of bitcoin after both timelocks without revealing her Monero key share, ending up with Bob loosing money for not following the protocol. $\texttt{BTX}_{claim}$ redeem the REFUND with:

```
<sig_a> OP_FALSE <REFUND script>
```

4.6 Transactions

We describe and name the Bitcoin and Monero transactions that are needed for the entire protocol.

$\texttt{BTX}_{lock}$, Bitcoin transaction with ≥ 1 inputs from Bob and the first output (vout: 0) to SWAPLOCK script and optional change outputs.

$\texttt{BTX}_{buy}$, Bitcoin transaction with 1 input consuming the SWAPLOCK script ($\texttt{BTX}_{lock}$, vout: 0) with the 2-of-2 semi-scriptless multisig and ≥ 1 outputs.

$\texttt{BTX}_{refund}$, Bitcoin transaction with 1 input consuming the SWAPLOCK script ($\texttt{BTX}_{lock}$, vout: 0) with the 2-of-2 timelocked multisig and exactly one output to REFUND script.

$\texttt{BTX}_{spend}$, Bitcoin transaction with 1 input consuming the REFUND script ($\texttt{BTX}_{refund}$, vout: 0) with the 2-of-2 semi-scriptless multisig and ≥ 1 outputs.

$\texttt{BTX}_{claim}$, Bitcoin transaction with 1 input consuming the REFUND script ($\texttt{BTX}_{refund}$, vout: 0) with Alice signature and ≥ 1 outputs.

$\texttt{XTX}_{lock}$, Monero transaction that sends funds to the address (K^v, K^s).

$\texttt{XTX}_{buy}$, Monero transaction that spend funds from the address (K^v, K^s).

Alice (XMR→BTC)		Bob (BTC→XMR)
$k_a^v, k_a^s \xleftarrow{R} [1, l-1]$		$k_b^v, k_b^s \xleftarrow{R} [1, l-1]$
$K_a^s \leftarrow k_a^s G$		$K_b^s \leftarrow k_b^s G$
$B_a^s \leftarrow k_a^s H$		$B_b^s \leftarrow k_b^s H$
$b_a, b_a^r \xleftarrow{R} [1, n-1]$		$b_b, b_b^r \xleftarrow{R} [1, n-1]$
$B_a \leftarrow b_a H$		$B_b \leftarrow b_b H$
$B_a^r \leftarrow b_a^r H$		$B_b^r \leftarrow b_b^r H$
$z_a \leftarrow \mathsf{DLProve}(k_a^s, K_a^s, B_a^s)$		$z_b \leftarrow \mathsf{DLProve}(k_b^s, K_b^s, B_b^s)$
	$\xleftarrow{\langle k_i^v, K_i^s, B_i, B_i^s, B_i^r, z_i \rangle \ \forall i \in \{a,b\}}$	
	$k^v \equiv k_a^v + k_b^v \pmod{l}$	
	$K^v = k^v G, \ \ K^s = K_a^s + K_b^s$	
$\mathsf{DLVrfy}(K_b^s, B_b^s, z_b) \stackrel{?}{=} 1$		$\mathsf{DLVrfy}(K_a^s, B_a^s, z_a) \stackrel{?}{=} 1$
		$(\mathtt{BTX}_{lock}, \mathtt{BTX}_{refund}) \leftarrow \mathsf{InitTx}(B_a, B_b, B_a^r, B_b^r)$
		$\sigma_r' \leftarrow \mathsf{Sign}(b_b^r, \mathtt{BTX}_{refund})$
		$\hat{\sigma}_1' \leftarrow \mathsf{EncSign}(b_b^r, B_b^s, \mathtt{BTX}_{spend})$
	$\xleftarrow{\langle \mathtt{BTX}_{lock}, \mathtt{BTX}_{refund}, \mathtt{BTX}_{spend}, \sigma_r', \hat{\sigma}_1' \rangle}$	
$\mathsf{VrfyTx}(\mathtt{BTX}_{lock}, \mathtt{BTX}_{refund}, B_a, B_b, B_a^r, B_b^r) \stackrel{?}{=} 1$		
$\mathsf{Vrfy}(B_b^r, \mathtt{BTX}_{refund}, \sigma_r') \stackrel{?}{=} 1$		
$\mathsf{EncVrfy}(B_b^r, B_b^s, \mathtt{BTX}_{spend}, \hat{\sigma}_1') \stackrel{?}{=} 1$		
$\sigma_2' \leftarrow \mathsf{Sign}(b_a^r, \mathtt{BTX}_{spend})$		
$\delta' \leftarrow \mathsf{RecKey}(B_b^s, \hat{\sigma}_1')$		
$\sigma_r'' \leftarrow \mathsf{Sign}(b_a^r, \mathtt{BTX}_{refund})$		
	$\xrightarrow{\langle \sigma_r'', \sigma_2' \rangle}$	
		$\mathsf{Vrfy}(B_a^r, \mathtt{BTX}_{refund}, \sigma_r'') \stackrel{?}{=} 1$
		$\mathsf{Vrfy}(B_a^r, \mathtt{BTX}_{spend}, \sigma_2') \stackrel{?}{=} 1$
		$\mathsf{PubTx}(\mathtt{BTX}_{lock})$
$\mathsf{WatchTx}(\mathtt{BTX}_{lock}) \stackrel{?}{=} 1$		
	$\cdots$	
$\mathtt{XTX}_{lock} \leftarrow \mathsf{InitTx}(K^v, K^s)$		
$\mathsf{PubTx}(\mathtt{XTX}_{lock})$		
$\mathtt{BTX}_{buy} \leftarrow \mathsf{InitTx}(\mathtt{BTX}_{lock})$		
$\hat{\sigma}_1 \leftarrow \mathsf{EncSign}(b_a, B_a^s, \mathtt{BTX}_{buy})$		
	$\xrightarrow{\langle \mathtt{BTX}_{buy}, \hat{\sigma}_1 \rangle}$	
		$\mathsf{WatchTx}(K^v, K^s)$ w/ $(k^v, K^s) \stackrel{?}{=} 1$
	$\cdots$	
		$\mathsf{EncVrfy}(B_a, B_a^s, \mathtt{BTX}_{buy}, \hat{\sigma}_1) \stackrel{?}{=} 1$
		$\sigma_2 \leftarrow \mathsf{Sign}(b_b, \mathtt{BTX}_{buy})$
		$\delta \leftarrow \mathsf{RecKey}(B_a^s, \hat{\sigma}_1)$
	$\xleftarrow{\langle \sigma_2 \rangle}$	
$\mathsf{Vrfy}(B_b, \mathtt{BTX}_{buy}, \sigma_2) \stackrel{?}{=} 1$		
$\sigma_1 \leftarrow \mathsf{DecSig}(k_a^s, \hat{\sigma}_1)$		
$\sigma := (\sigma_1, \sigma_2)$		
$\mathsf{PubTx}(\mathtt{BTX}_{buy}, \sigma)$		
		$\mathsf{WatchTx}(\mathtt{BTX}_{buy}) \stackrel{?}{=} 1$
	$\cdots$	
		$(\sigma_1, \sigma_2) \leftarrow \mathsf{RecSig}(\mathtt{BTX}_{buy})$
		$k_a^s \leftarrow \mathsf{Rec}(\sigma_1, \delta)$
		$k^s \equiv k_a^s + k_b^s \pmod{l}$
		$\langle k^v, k^s \rangle$

5 Acknowledgement

TrueLevel SA collaborators and Sarang Noether from the Monero Research Lab are acknowledged for their helpful contribution and comments during the completion of this work.

References

[1] 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`.

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

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

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