Bitcoin–Monero Cross-chain Atomic Swap

March 18, 2019 – DRAFT

h4sh3d[1]

TrueLevel
h4sh3d@truelevel.io

Abstract. Cross-chain atomic swaps have been discussed for a very long time and are very useful tools. 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 parties, one holding bitcoin and the other monero.

Keywords: Bitcoin, Monero, Cross-chain swaps

1 Introduction

We describe a protocol for an on-chain atomic swap between Monero and Bitcoin, but the protocol can be generalized for Monero and any other cryptocurrencies that fulfill the same requirements as Bitcoin.

This protocol is heavily based on a Monero StackExchange post discussing if it's possible to trade Monero and Bitcoin in a trustless manner [1]. The concept is roughly the same, with some changes in the Bitcoin part and is explained in more detail; they send funds to a special location on each chain (cross-chain) where each party can take control of the funds on the other chain (swap) only (atomic; i.e. claiming of funds on either chain is mutually exclusive from the ability to claim funds from another chain).

1.1 Known limitations

To provide finality (if at least one participant is still online) we allow for the worst case scenario in which a participant may end up loosing his funds, but only in the case where they do not follow the protocol (which includes remaining online during pending swap). 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 complex features of bitcoin, which causes transactions on the bitcoin side to be more expensive. 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 by each participant to consider a transaction final, dictates the speed of the protocol, making front runs possible in some setups.

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.

2.1 Successful swap

If both participants follow the protocol there will be 4 transactions broadcast in total, 2 on Bitcoin blockchain and 2 on 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 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 (how many confirmations each expects for the funding transaction) and no timelocks.

2.2 Swap correctly aborted

When locking the bitcoin, after a timelock Alice or Bob can start the process of refunding the locked funds. In this case 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 has to 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 Bobs perspective.

2.3 Worst case scenario

If the swap is cancelled with the refund process and Bob does not spend his funds before the timelock, Alice can claim the refund. Thus one participant, Bob, ends up underwater and 3 Bitcoin transactions are needed instead of 2.

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/stalemate.

3 Prerequisites

As previously described, conditional execution must be possible in order to achieve a swap with atomicity. Bitcoin has a small stack-based script language that allows for conditional execution and timelocks. On the other hand 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.

Secret share, to enable a basic two-path execution in Monero. The Monero swap 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 $k^s \equiv k_a^s + k_b^s$ at the end of the protocol execution, either for a completed swap or for an aborted swap.

Pre-image non-interactive zero-knowledge proofs of knowledge, to prove to the other participant that a valid pre-image α to a given hash $h = \mathcal{H}(\alpha)$ is known and within a specific range, e.g. $\alpha \in [1, l-1]$ where l is related to `edward25519` curve and $\mathcal{H}$ is `SHA256`. The scalar α must be serialized in a standard way.

`Edward25519` **private key non-interactive zero-knowledge proofs of knowledge** to prove to the other participant that a valid private key x is known for a public key X, e.g. signatures are non-interactive zero-knowledge proofs given some public keys.

3.2 Bitcoin

The bitcoin transactions in this protocol require SegWit activation in order to be able to chain unbroadcast transactions (i.e. not Bitcoin Cash). These requirements must be fulfilled for any other cryptocurrencies with a bitcoin style UTXO model [such as Litecoin] which wish to compatible with this protocol.

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.

Hashlock, to reveal secrets to the other participant. Hashlocks are primitives that require one to reveal some data (a pre-image) associated with a given hash to be allowed to spend the associated UTXO.

2-out-of-2 multisig, to create a common path accessible only by the two participants if both agree.

3.3 Curves parameters

Bitcoin and Monero does not use the same elliptic curves. Bitcoin uses the `secp256k1` curve from *Standards for Efficient Cryptography (SEC)* with `ECDSA` while Monero, based on the second version of CryptoNote, uses `Curve25519`, hereinafter also `edward25519`, from Daniel J. Bernstein [2, 3]. We denote curve parameters for

`edward25519` as

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

`secp256k1` as

$$\begin{aligned} p &: \text{a prime number; } p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \\ a &: \text{an element of } \mathbb{F}_p;\ a = 0 \\ b &: \text{an element of } \mathbb{F}_p;\ b = 7 \\ E' &: \text{an elliptic curve equation; } y^2 = x^3 + bx + a \\ G' &: \text{a base point; } G' = \\ &\big(\texttt{0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798}, \\ &\ \texttt{0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8}\big) \end{aligned} \tag{2}$$

4 Protocol

In this chapter we describe the protocol for Alice and Bob to swap their funds atomically and without trusting each other.

Alice (XMR)		Bob (BTC)
$k_a^v, k_a^s \xleftarrow{R} [1, l-1]$		$k_b^v, k_b^s \xleftarrow{R} [1, l-1]$
$K_a^s \leftarrow k_a^s G$		$K_b^s \leftarrow k_b^s G$
$b_a \xleftarrow{R} [1, p-1]$		$b_b \xleftarrow{R} [1, p-1]$
$B_a \leftarrow b_a G'$		$B_b \leftarrow b_b G'$
$h_a \leftarrow \mathcal{H}(k_a^s)$		$h_b \leftarrow \mathcal{H}(k_b^s)$
		$s \xleftarrow{R} \mathbb{Z}_{2^{256}}$
		$h_s \leftarrow \mathcal{H}(s)$
$z_a \leftarrow \texttt{zkp}(k_a^s)$		$z_b \leftarrow \texttt{zkp}(k_b^s)$
	$\overset{\langle k_a^v, k_b^v, K_a^s, K_b^s, B_a, B_b, h_a, h_b, h_s, z_a, z_b \rangle}{\longleftrightarrow}$	
verify $[k_b^v, K_b^s, B_b, z_b]$		verify $[k_a^v, K_a^s, B_a, z_a]$
	$k^v \equiv k_a^v + k_b^v \pmod{l}$	
	$K^v = k^v G$	
	$K^s = K_a^s + K_b^s$	
		create $[\text{BTX}_{lock}, \text{BTX}_{refund}]$
		sign BTX_{refund}
	$\xleftarrow{\langle \text{BTX}_{lock}, \text{BTX}_{refund} \rangle}$	
verify $[\text{BTX}_{lock}, \text{BTX}_{refund}]$		
sign BTX_{refund}		
	$\xrightarrow{\langle \text{BTX}_{refund}^{signed} \rangle}$	
		verify $\text{BTX}_{refund}^{signed}$
		sign BTX_{lock}
		broadcast $\text{BTX}_{lock}^{signed}$
	$\xleftarrow{(\text{watch BTX}_{lock})}$	
create XTX_{lock} w/ (K^v, K^s)		
sign XTX_{lock}		
broadcast $\text{XTX}_{lock}^{signed}$		
	$\xrightarrow{(\text{watch w/ } (k^v, K^s))}$	
		verify (K^v, K^s) w/ (k^v, K^s)
	$\xleftarrow{\langle s \rangle}$	
spend BTX_{lock} w/ $\langle k_a^s, b_a, s \rangle$		
	$\xrightarrow{\text{watch BTX}_{lock}}$	
		$k^s \equiv k_a^s + k_b^s \pmod{l}$
		spend XTX_{lock} w/ $\langle k^v, k^s \rangle$

Fig. 1. Cross-chain atomic swap protocol for Bitcoin–Monero

The overall protocol is as follows: Alice moves the monero into an address where each participant controls half of the private spend key. The Bitcoin scripting language is then used to reveal one of the halfs of the private spend key depending on which participant claims the bitcoin. Depending on who reveals the key share, the locked monero changes ownership. Bitcoin transactions are designed in such a way that if a participant follows the protocol he 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 cancelled, 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.

4.1 Time parameters

Two timelocks t_0, t_1 are defined. t_0 defines 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 reallity). t_1 defines 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 she locked monero).

4.2 Monero private keys

Monero private keys are pairs of `edward25519` private keys. The first key is called view key and the second is called spend key. We use small letters to denote private keys and caps for public keys such that

$$X = xG$$

Where G is the generator element of the curve (see 3.3). We denote

(i) the private key k^v as the private view key and K^v as the public view key,

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

(iii) the private key k^s as the private spend key and K^s as the public spend key,

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

Partial keys We denote partial private keys 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 of the protocol for trustlessness. The protocol uses hashes as commitments of private keys, but we cannot check the hash of the other participant before it goes on-chain. Thus we need to provide a proof that the hash used in the Bitcoin script is the serialized private share and not other random data.

ZKP Alice and Bob must prove to each other with

$$\begin{aligned} k^s &= \text{valid scalar on } \texttt{edward25519} \text{ curve} \\ K^s &= k^s G \\ h &= \mathcal{H}(k^s) \end{aligned} \tag{4}$$

that given K^s and h

$$\exists k^s \mid K^s = k^s G \wedge h = \mathcal{H}(k^s) \wedge k^s \in [1, l-1] \tag{5}$$

where $\mathcal{H}(k^s)$ is the hash of the 256-bit little-endian integer representation of k^s.

4.4 Bitcoin scripts

`SWAPLOCK` is a `P2SH` used to lock funds and defines the two base execution paths: (1) swap execution and (2) refund execution.

```
OP_IF
    OP_SHA256 <h_a> OP_EQUALVERIFY
    OP_SHA256 <h_s> OP_EQUALVERIFY
    <B_a> OP_CHECKSIG
OP_ESLE
    <t_0> OP_CHECKSEQUENCEVERIFY OP_DROP
    2 <B_a> <B_b> 2 OP_CHECKMULTISIG
OP_ENDIF
```

Buy `SWAPLOCK`, Alice takes control of bitcoin and reveals her Monero key share, thus allowing Bob to take control of the monero. Alice can redeem the `SWAPLOCK` with:

```
<sig_a> <s> <k_a^s> OP_TRUE <SWAPLOCK script>
```

Refund `SWAPLOCK`, signed by both participants and moves the funds into `REFUND P2SH`. BTX_{refund} uses the following redeem script:

```
OP_0 <sig_a> <sig_b> OP_FALSE <SWAPLOCK script>
```

REFUND, P2SH used in case the swap already started on-chain but is cancelled. This refund script is used to lock the only output of the transaction which spends the SWAPLOCK output with the 2-out-of-2 timelocked multisig.

```
OP_IF
    OP_SHA256 <h_b> OP_EQUALVERIFY
    <B_b> OP_CHECKSIG
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 in order to redeem the REFUND P2SH, thus allowing Alice to regain control over her Monero. Bob can redeem it with:

```
<sig_b> <k_b^s> 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. Alice can redeem the REFUND P2SH with:

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

4.5 Transactions

BTX$_{lock}$, Bitcoin transaction with ≥ 1 inputs from Bob and the first output (vout: 0) to SWAPLOCK P2SH and optional change outputs.

BTX$_{refund}$, Bitcoin transaction with 1 input consuming SWAPLOCK P2SH (BTX$_{lock}$, vout: 0) with the 2-out-of-2 timelocked multisig and exactly one output to REFUND P2SH.

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

5 Acknowledgement

Daniel Lebrecht and Tom Shababi are acknowledged for their helpful contribution and comments during the completion of this work.

References

[1] PyRulez. *Can you trustlessly trade Monero for Bitcoin?* 2016. URL: https://monero.stackexchange.com/questions/894/can-you-trustlessly-trade-monero-for-bitcoin/895#895.

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

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