# Bitcoin–Monero Cross-chain Atomic Swap

**March 28, 2020 – DRAFT**

h4sh3d[1]

TrueLevel
h4sh3d@protonmail.com

**Abstract.** Cross-chain atomic swaps have been discussed for a very long time and are very useful tools. In blockchains where hashed timelock contracts are doable 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 Edward25519 and secp256k1 groups.

**Keywords:** Bitcoin, Monero, atomic 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, see 3.2.

This protocol is heavily based on a Monero StackExchange post discussing if it's possible to trade Monero and Bitcoin in a trustless manner [2]. The concept is roughly the same, with changes in Bitcoin construction, less prerequisites in Monero, and more detailed explainations.

Participants send funds into a specific address generated during the initialization process 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 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. not remaining online during swap process or not 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, 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 by each participant to consider a transaction final, dictates the speed of the protocol, making front runs possible in some setups. But the protocol can be extended to avoid them within some setups. It is worth noting that front runs cannot be enforced to the other participant, thus making the worst case scenario a redund with fee costs on each chain.

## 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, two on Bitcoin blockchain and two on Monero blockchain. The first ones on each chain lock the funds and makes them ready for the trade. 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 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 Bob's perspective, i.e. Bob cannot go offline or not react during the swap but Alice can.

### 2.3 Worst case scenario

If the swap is cancelled 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 underwater 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 small 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.

**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 $k^s \equiv k_a^s + k_b^s \pmod{l}$ at the end of the protocol execution, either for a completed swap or for an aborted swap.

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

### 3.2 Bitcoin

The bitcoin transactions in this protocol require `SegWit` activation in order to be able to chain unbroadcast transactions. 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 (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.

**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. In this protocol we use the on-chain option but off-chain multi-signature scheme can replace it.

**Signed message,** to create a valid signature $(r, s)$ with the private spend key share on `secp256k1` on a given message. Signature's $r$ is used to fix the random parameter $k$ in `ECDSA` and allow private key disclosure when signing the bitcoin transaction.

### 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 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

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

`secp256k1` as

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

## 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. 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.

Key exchange is performed, as demonstated in [1], by exploiting weaknesses of ECDSA when the nonce $k$ is reused. The full protocol is shown in Figure 1.

### 4.1 Zero-Knowledge proofs

Zero-knowledge proofs are required at the beginning of the protocol for trustlessness. The protocol uses common discrete logarithms as commitments of private keys. Thus we need to provide a proof that a common scalar is known for the Bitcoin public key and the Monero public key.

**ZKP** Alice and Bob must prove to each other with $i \in \{a, b\}$

$$
\begin{aligned}
k_i^s &\leftarrow \text{valid scalar on } \texttt{edward25519} \text{ and } \texttt{secp256k1} \\
K_i^s &= k_i^s G \\
B_i^s &= k_i^s G'
\end{aligned}
\tag{3}
$$

that given $K_i^s$ and $B_i^s$

$$
\exists k_i^s \mid K_i^s = k_i^s G \wedge B_i^s = k_i^s G' \wedge k_i^s < min(l, n) \tag{4}
$$

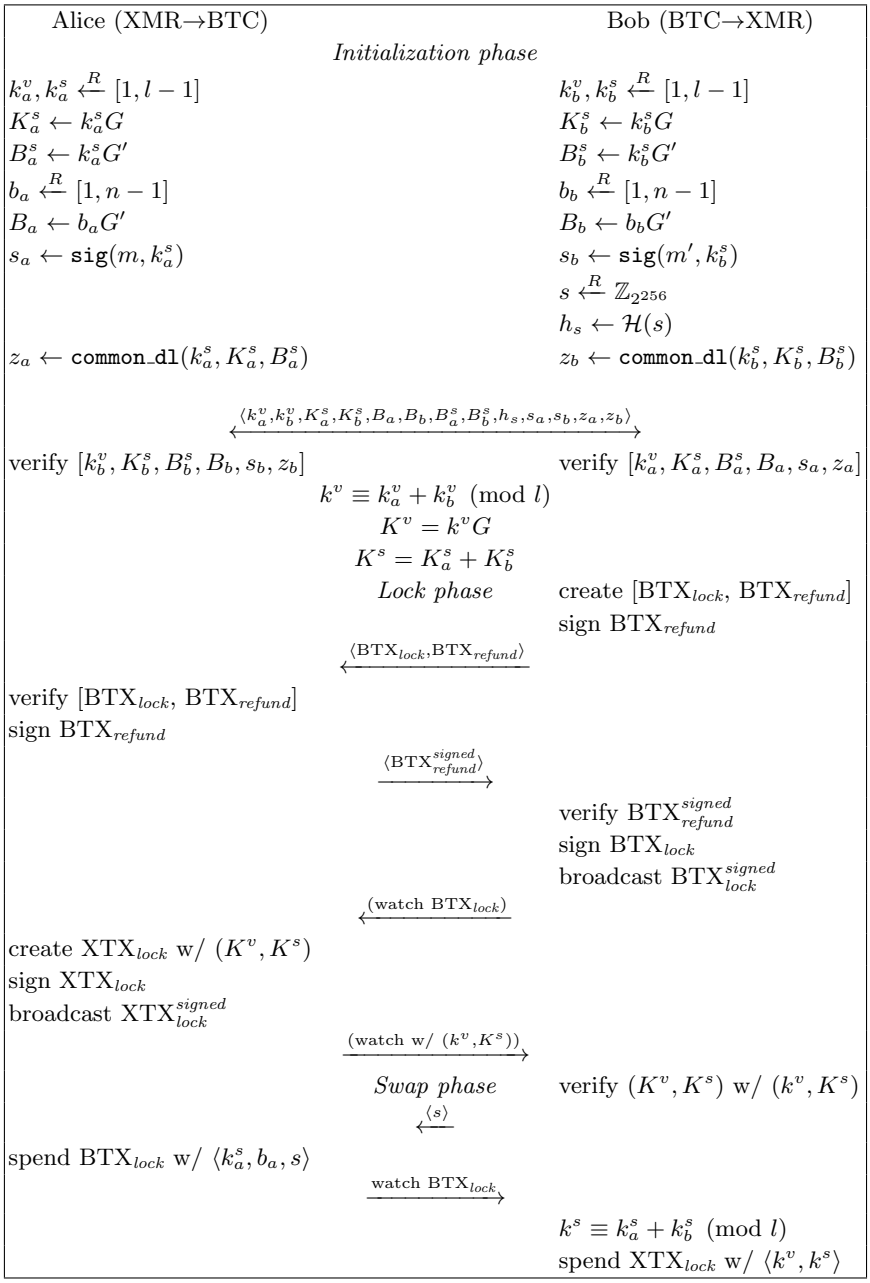


**Fig. 1.** Full protocol of cross-chain atomic swap for Bitcoin–Monero between two participants Alice and Bob with initialization, lock, and swap phases.

### 4.2 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 reallity). $t_1$ 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 monero have been locked).

### 4.3 Monero private keys

Monero private keys are pairs of `edward25519` scalars. The first scalar is called private view key and the second is called private spend key. We use small letters to denote private keys and cappital letters 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 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

$$\begin{aligned} k_a^s G &= K_a^s \\ k_b^s G &= K_b^s \\ K_a^s + K_b^s = (k_a^s + k_b^s)G &= k^s G = K^s \end{aligned} \tag{5}$$

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

### 4.4 Bitcoin scripts

`SWAPLOCK` is a `P2SH` 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
    OP_DUP <B_a^s> OP_CHECKSIGVERIFY
    OP_SIZE <0x47> OP_EQUALVERIFY
    <sigmask> OP_AND <rprev> OP_EQUALVERIFY
    OP_SHA256 <h_s> OP_EQUAL
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:

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

**Refund** `SWAPLOCK`**,** signed by both participants and moves the funds into `REFUND P2SH`. $\text{BTX}_{refund}$ uses the following redeem script:

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

`REFUND` is a `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. We define the `REFUND` script as:

```
OP_IF
    OP_DUP <B_b^s> OP_CHECKSIGVERIFY
    OP_SIZE <0x47> OP_EQUALVERIFY
    <sigmask> OP_AND <rprev> OP_EQUAL
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^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 $\geq 1$ 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` script.

**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] Sergi Delgado-Segura et al. *Bitcoin Private Key Locked Transactions*. Cryptology ePrint Archive, Report 2016/1184. `https://eprint.iacr.org/2016/1184`. 2016.

[2] 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`.

[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.