Bitcoin–Monero Cross-chain Atomic Swap

Joël Gugger

h4sh3d@protonmail.com

Abstract. In blockchains where hashed timelock contracts are possible atomic swaps are already deployed, but when one of the blockchains 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 the 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: Blockchain, Atomic Swap, Cross-Chain Transactions, Bitcoin, Monero

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. For an exhaustive list of prerequisites, 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.)

During this process participants do not expose their funds if they follow the protocol accordingly, meaning that the swap is trustless and does not require any sort of collateral, allowing two strangers to trade without risks nor the help of a third-party.

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 make them ready for the trade on each chain. The second ones unlock the funds for one participant only and give 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 following step of the protocol.

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. 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 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 for 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 the monero are locked, Alice will be able to refund them if and 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.

Otherwise Bob, with all the information already learned, can go offline and move his bitcoin a year after the swap, forcing Alice to consistently monitor the chain until she sees Bob's transaction and learns the last piece of knowledge she needs to unlock her monero.

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's privacy oriented RingCT design provides only signatures to unlock UTXOs. 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.

In this chapter we discuss all the required pieces needed on both chains and off-chain to achieve an atomic swap under the previously exposed scenario.

3.1 Monero

Monero does not require any particular on-chain primitives (hashlocks, timelocks), all building blocks are off-chain primitives. Thus we need to provide proofs of the correct initialization of the protocol such as described in chapter 4.3: those proofs will ensure the swap atomicity for each participant.

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 and k_b^s. Participants will not use any multi-signature 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 the other chain.

Hashlock, to syncronize both chains before allowing one to start the swap.

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 in the context of Bitcoin, but off-chain multi-signature schemes are more efficient and should be preferred in other setups.

Semi-Scriptless protocol, to reveal a secret and allow the protocol to continue execution without requiring complex scripting capabilities. We use ECDSA one-time VES such as described in 3.4. It is worth noting that full scriptless script [5] protocol should be able to achieve the same results with better efficiency.

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

Equal discrete logarithm across groups zero-knowledge proof of knowledge, as described in technical note [2], allow the verification of a common discrete logarithm α given two groups with fixed generators $G \in \mathbb{G}$ and $H \in \mathbb{H}$, where $\alpha \leq min(|G|, |H|)$, such that given xG and yH: $x = y = \alpha$.

In this context we focus on groups `edward25519` with $|G| = l$ and `secp256k1` with $|H| = n$.

Definition 1 (Equal discrete logarithm across groups scheme). *An equal discrete logarithm across groups scheme is defined with two algorithms under the parameter tuple* $(\mathbb{G}, \mathbb{H}, G, G', H, H')$*:*

- $\mathsf{DLProve}(\alpha) \to (\phi, A, B)$*: A probabilistic proving algorithm which on input of a discrete logarithm* α *outputs a proof* ϕ*, a point* $A \in \mathbb{G}$*, and a point* $B \in \mathbb{H}$*.*
- $\mathsf{DLVrfy}(\phi, A, B) \to \{0, 1\}$*: A deterministic proof verification algorithm which on input of a proof* ϕ*, a point* $A \in \mathbb{G}$*, and a point* $B \in \mathbb{H}$ *outputs 1 if and only if* (A, B) *share a common discrete logarithm across their respective groups.*

Curve 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. Monero, based on the second version of CryptoNote [9], uses `curve25519`, hereinafter also `edward25519`, from Daniel J. Bernstein [6].

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

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

3.4 ECDSA one-time VES

ECDSA one-time VES are introduced by Fournier et al. in [1] as a generalization of the concept of adaptor signatures under Schnorr [7] and ECDSA. One-time Verifiably Encrypted Signatures, hereinafter one-time VES, are VES constructed such that with knowledge of the ciphertext and the plaintext, it is possible to recover the encrypting key.

We recall some of the algorithms defined in [1] as:

Definition 2 (ECDSA one-time VES). *An ECDSA one-time VES scheme contains:*

- $\mathsf{EncSign}(sk_S, pk_E, m) \xrightarrow{\$} \hat{\sigma}$*: A possibly probabilistic encrypted signing algorithm, which on input of a secret signing key* sk_S*, a public encryption key* pk_E*, and a message* m*, outputs a ciphertext* $\hat{\sigma}$*.*
- $\mathsf{EncVrfy}(pk_S, pk_E, m, \hat{\sigma}) \to \{0,1\}$*: A deterministic encrypted signature verification algorithm which — on input of a public signing key* pk_S*, a public encryption key* pk_E*, a message* m *and a ciphertext* $\hat{\sigma}$ *— outputs 1 if and only if* $\hat{\sigma}$ *is a valid encryption of a signature on* m *for* pk_S *under* pk_E*.*
- $\mathsf{DecSig}(sk_E, \hat{\sigma}) \to \sigma$*: A (usually) deterministic signature decryption algorithm which — on input of a decryption key* sk_E *and a valid ciphertext* $\hat{\sigma}$ *under that encryption key — outputs a valid signature* σ*.*
- $\mathsf{RecKey}(pk_E, \hat{\sigma}) \to \delta$*: A deterministic recovery key extraction algorithm which extracts a recovery key* δ *from the ciphertext* $\hat{\sigma}$ *and the public encryption key* pk_E*.*
- $\mathsf{Rec}(\sigma, \delta) \to sk_E$*: A deterministic decryption key recovery algorithm which — when given a decrypted signature* σ *and the recovery key* δ *associated with the original ciphertext — returns the secret decryption key* sk_E*.*

3.5 Generalization

As shown in the Monero section, there are no on-chain requirements. We only make use of private key generation and addresses, meaning that this scheme can be generalized to any pair of cryptocurrencies where one fulfills the bitcoin prerequisites, making this scheme very chain agnostic (under the assumption of the above cryptographic primitives existence for the underlying chains' parameters). However, it is worth noting that other schemes are simpler and more efficient when, in the pair, both chains have on-chain capabilities, but such schemes are generally already deployed [3, 4, 8].

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 halves 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 change 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 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 with ECDSA one-time VES—also adaptor signatures—, and equal discrete logarithm across groups zero-knowledge proofs of knowledge. 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 points on edward25519 and secp256k1 to ensure protocol trustlessness.

4.1 Non-interactive refund

If Alice and or Bob locked their funds but one of them aborts the swap, or stop communicating at some point, the protocol must not require interactivity to complete the refund procedure for both participants. Otherwise Alice can keep Bob hostage 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—a.k.a semi-scriptless scripts—, 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 mentioned before, the refund process must be non-interactive: The protocol is designed 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's cooperation.

4.2 Monero private keys

Monero private keys are pairs of `edward25519` scalars: One is the private view key and the other is 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 group $\mathbb{G}$. 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 $\mathbb{G}$ and $\mathbb{H}$.

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

$$\begin{aligned} k_i^s &\leftarrow \text{scalars on } \texttt{edward25519} \text{ and } \texttt{secp256k1} \text{ with an equivalent bit representation} \\ K_i^s &= k_i^s G \in \mathbb{G} \\ B_i^s &= k_i^s H \in \mathbb{H} \end{aligned} \tag{4}$$

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 due to a potential race condition (even if it is hard to exploit in reality). t_1 sets the response time during which Bob is required to react, reveal his private Monero share to get his bitcoin back, and allow Alice to redeem her monero (if monero have been locked). After t_1, Alice is able to claim the bitcoin unilaterally.

4.5 Bitcoin scripts

Two scripts are needed on the bitcoin side: the first is used to complete the swap or start the refund process—a.k.a `SWAPLOCK`—, the second is used to complete the refund process—a.k.a `REFUND`—. In a successful swap, the second script does not go on-chain and is not used. Each script defines two possible paths, and we consequently explain the four possible ways (*buy*, *refund*, *spend*, and *claim*) of spending the two `UTXOs`.

`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
    OP_SHA256 <h_s> OP_EQUALVERIFY
    2 <B_a> <B_b> 2 OP_CHECKMULTISIG
OP_ELSE
    <t_0> OP_CHECKSEQUENCEVERIFY OP_DROP
    2 <B_a^r> <B_b^r> 2 OP_CHECKMULTISIG
OP_ENDIF
```

Buy `SWAPLOCK`, Alice takes control of the 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. $\texttt{BTX}_{buy}$ redeems the `SWAPLOCK` with:

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

Refund `SWAPLOCK`, signed by both participants, and moves the funds into the `REFUND` script. $\texttt{BTX}_{refund}$ redeems 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_ELSE
    <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 the bitcoin after both timelocks without revealing her Monero key share, resulting in Bob losing money for not following the protocol. $\texttt{BTX}_{claim}$ redeems 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.

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

$\texttt{BTX}_{buy}$, a 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.

$\texttt{BTX}_{refund}$, a 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 the `REFUND` script.

$\texttt{BTX}_{spend}$, a 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.

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

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

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

4.7 Full protocol sequence

We describe the full protocol execution to successfully complete a swap, while computing and sharing all necessary knowledge in case any participant stops responding at any time or any participant starts the refund process actively.

During the first communication round, since both share parameters used to initialize the protocol, we might avoid schemes such as commit-reveal even with $K^s = K^s_a + K^s_b$: thanks to the equal discrete logarithm zero-knowledge proofs, one cannot arbitrarily choose K^s and compute a valid proof z_i. However, to ensure getting a random view key for each execution, a commit-reveal must be added on k^v_i. Thus, without adding another round of communication, a commit-reveal might be added on k^s_i also.

We define some utility algorithms to initialize, sign, and verify Bitcoin and Monero transactions.

- InitTx(): A generic and deterministic algorithm which — on input of a set of parameters — outputs a valid initialized transaction.
- Sign(): A generic and probabilistic algorithm for siging a transaction which — on input of a private key and an initialized transaction — outputs a valid signature for the transaction.
- VrfyTx(): A generic and deterministic algorithm which — on input of a set of transactions and parameters — outputs 1 if and only if the transactions are valid under the protocol rules and blockchain consensus.
- Vrfy(): A generic and deterministic algorithm which — on input of a public key, a transactions and a signature — outputs 1 if and only if the signature is valid for transactions given the public key.
- PubTx(): A generic algorithm for publishing a transactions over the network.
- WatchTx(): A generic algorithm for waiting on a transaction to confirm.
- RecSig(): A generic algorithm for extracting transaction's signatures.

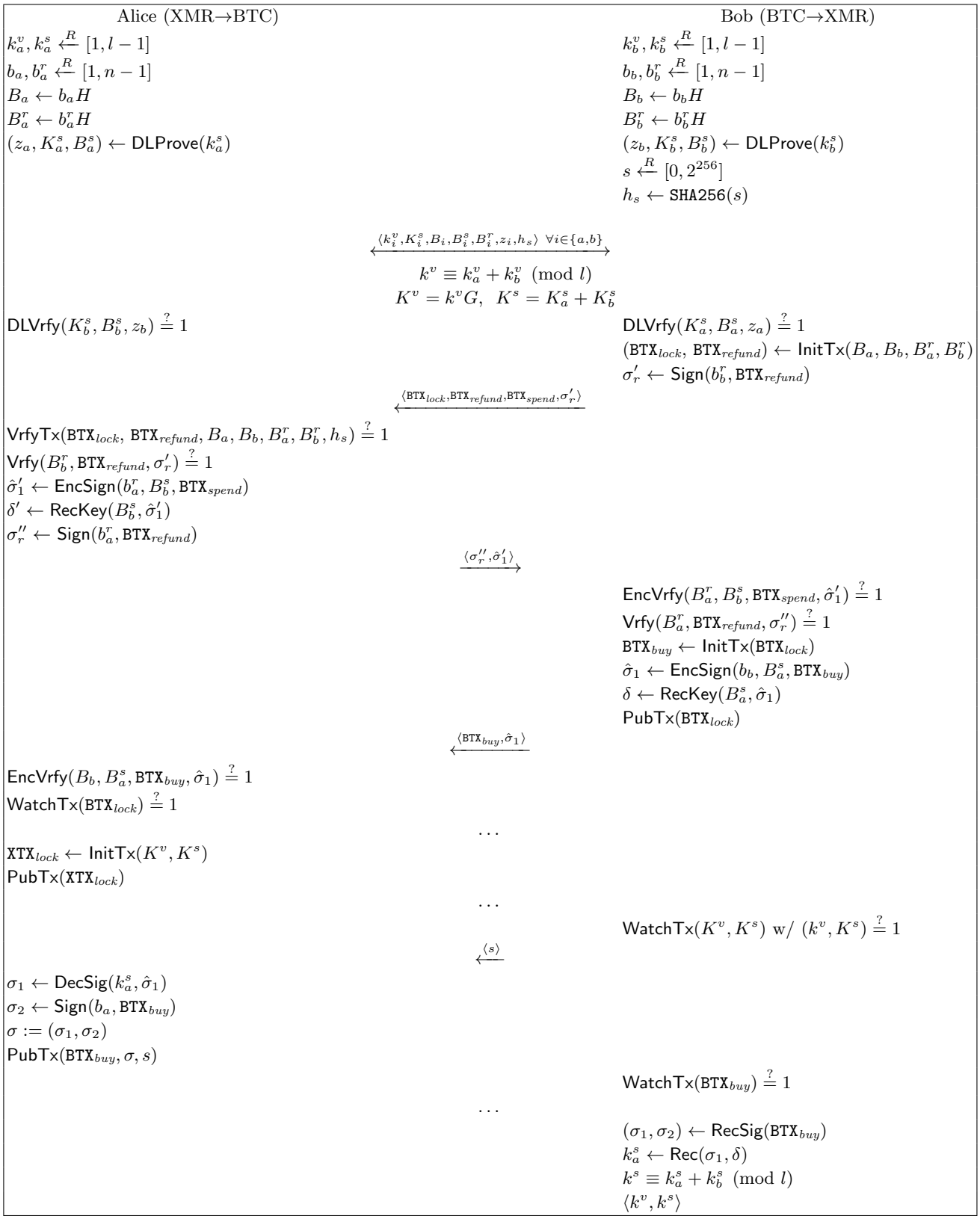

Fig. 1. Protocol execution between Alice and Bob for a successful swap

Dots represent synchronization timing during which one or more participants must check for transaction confirmations before continuing the protocol execution. The level of security — i.e. the number of confirmations required by each participants — is a local parameter, but must be set according to the timing parameters — which are global to both participants — to avoid the case where the timing is less or equal to the security parameter.

5 Further research

This protocol is implementable with today's state of Bitcoin and Monero, but further research is required to use `Schnorr` capabilities and simplicity to create a more efficient protocol with lighter on-chain footprints. This would improve anonymization of atomic swaps with respect to chain-analysis.

As discussed, this protocol can be adapted to other cryptocurrencies. Some of them might not have atomic swaps yet, and extending to other pairs might improve decentralization. Extending this protocol to privacy preserving cryptocurrency pairs only — like Monero with Mimblewimble — is the next challenge in atomic swaps.

Integration with services or decentralized exchanges would help democratizing trading without trusted institutions and might increase the market liquidity. Since the design presented here is asymmetric, building services might not be straightforward.

5.1 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 losing 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 scripting 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 the participants' respective blockchains.

6 Acknowledgement

The Monero Research Lab and Sarang Noether are acknowledged for their helpful comments during the completion of this work. This work has been supported and partially funded by the Monero Community — we extend a special thanks to all donators. Finally, thanks to TrueLevel SA collaborators for the initial funding and their helpful contribution and comments.

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] Tier Nolan. *Alt chains and atomic transfers*. URL: `https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949`.

[4] Andrew Poelstra. *Adaptor Signatures and Atomic Swaps from Scriptless Scripts*. 2017. URL: `https://github.com/ElementsProject/scriptless-scripts/blob/master/md/atomic-swap.md`.

[5] Andrew Poelstra. *Scriptless Scripts*. 2017. URL: https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-05-milan-meetup/slides.pdf.

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

[7] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '89. Berlin, Heidelberg: Springer-Verlag, 1990, pp. 239–252. ISBN: 3-540-97317-6. URL: http://dl.acm.org/citation.cfm?id=646754.705037.

[8] Lucas Sorianos del Pino and Lloyd Fournier. *Grin-Bitcoin Atomic Swap*. 2019. URL: https://github.com/comit-network/grin-btc-poc/blob/master/spec.pdf.

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