arXiv:2101.12332v2 [cs.CR] 2 Feb 2021

Atomic Swaps between Bitcoin and Monero

Philipp Hoenisch[1,2] and Lucas Soriano del Pino[1,2]

[1] COMIT, `{firstname}@comit.network`
[2] CoBloX Pty Ltd, `{firstname}@coblox.tech`

Abstract. Due to the evergrowing blockchain ecosystem, interoperability has become a matter of great importance. Atomic swaps allow connecting otherwise isolated blockchains while adhering to the core principles of censorship resistance and permissionlessnes. Up until recently, atomic swap protocols have mostly relied on complex script support, excluding certain types of blockchains. With advances in cryptography, it is now possible to build a bridge between almost any two blockchains. In this work, we give an explanation of one such protocol which applies adaptor signatures on Bitcoin to procure atomic swaps between Monero and Bitcoin. We dive into the cryptographic details, discuss its limitations and give an outlook on our current work where we use adaptor signatures on the Monero signature scheme.

Keywords: Blockchain · Atomic Swap · Bitcoin · Monero · Adaptor Signatures.

1 Introduction

Since the birth of Bitcoin in 2008[11], many other cryptocurrencies have been introduced. It is without a doubt that this flourishing ecosystem has evolved into an enormous financial market. Cryptocurrencies are traded against fiat (e.g. USD, AUD, EUR) or against each other. However, due to the lack of interoperability between different blockchains, most of the trades are executed on centralized exchanges. Due to regulations, these centralized exchanges have integrated complex KYC (Know Your Customer) procedures where traders have to go through lengthy processes to prove their identity. In addition, traders give up control over their hard-earned coins by depositing them in the exchange so that they can execute trades. The trader now has to trust the exchange to manage their funds according to the highest standards, to protect them against thieves or not lose them otherwise. This trust was misused more than once in the past and billions of dollars in user funds have been lost[5].

One could say that these centralized exchanges are now a relic of the past. A new era of decentralized exchanges has started, adhering to the core idea of Bitcoin: censorship resistance at all levels.

Decentralized exchanges powered by atomic swaps, first introduced in 2015 by TierNolan[17], can now promise more guarantees in terms of security and privacy to traders.

The original idea of atomic swaps uses HTLCs (Hash Time-Lock Contracts), imposing certain requirements on the underlying blockchains: (1) they must support scripts so that one can build hash locks; and (2) they must support timelocks.

Technology has evolved and, with advances in cryptography, a new way of cross-chain atomic swaps using adaptor signatures is gaining traction.

Atomic swaps using adaptor signatures (also referred to as Scriptless Scripts) have several advantages over traditional atomic swaps using HTLCs: (1) contrary to HTLCs where the same hash has to be used on each chain, transactions involved in an atomic swap using adaptor signatures cannot be linked; and (2) since no script is involved, the on-chain footprint is reduced which makes the atomic swap cheaper.

Within this work we present our current efforts on cross-chain atomic swaps using adaptor signatures. In particular, we show how adaptor signatures can be employed to swap between Monero and Bitcoin. Notably, the former does not support scripts or timelocks.

2   HTLC-based Atomic Swaps

Replacing centralized exchanges by decentralized ones is not new. The idea of using HTLCs for atomically swapping two assets across two chains has been around for a while[17]. Various companies have used this technology in their products and protocols for cross-chain trading[1, 6]. Moreover, HTLCs are also used in the Lightning Network for multi-hop payments[15].

In a nutshell, an HTLC-based atomic swap protocol works like this: we assume two parties, Alice and Bob found each other somehow and agreed on the amounts and assets (e.g. bitcoin and ether) which the two parties want to exchange. Alice generates a random secret s and uses a cryptographic hash function to generate hash h. She then creates an HTLC using h and locks up the bitcoin. These coins can either be redeemed (spent) using the secret s or are returned to her after time t has passed. Bob does the same thing on the other chain: he locks up his ether in an HTLC using the same hash h.

Since Alice knows the original secret s that was used to produce the hash h, she can redeem the ether from Bob's HTLC. By doing so, she reveals the secret s to Bob who can then take the bitcoin, completes the swap.

This apparently simple process has a few drawbacks:

- The requirements on the underlying blockchains are high. A certain script capability is required in order to support a hash function as well as timelocks. While many blockchains support these two features, some lack either one (e.g. Grin has no script support and hence no support for hash functions) or both (e.g. Monero).
- By definition, the same hash has to be used on both chains. This allows an independent third party to link those two transactions. Worse yet, since blockchain transactions are publicly available to everyone, this onlooker can now track where the parties move their newly acquired funds.

- The use of scripts (e.g. on Bitcoin, Litecoin, etc) or smart contracts (e.g. on Ethereum) results in an increased on-chain footprint and higher transaction fees in general.

With recent advancements in cryptography and the application of adaptor signatures to atomic swaps, it is now possible to overcome almost all of the aforementioned drawbacks. For example, Grin-Bitcoin swaps can be realized despite Grin's lack of a scripting language. Using Schnorr adaptor signatures and timelocks, an atomic swap protocol can be executed[4].

Recently, Gugger, J. (aka *h4sh3d*) came up with a protocol which enables atomic swaps between Monero and Bitcoin[10]. In the next section we discuss this protocol in detail; in Section 4, we present our current work, motivated by some of the limitations of [10].

3   BTC to XMR atomic swaps

The protocol described in this section is largely based on the work of Gugger[10]. We highlight key differences between the original and our instantiation of it[2] throughout.

3.1   Situation

Alice and Bob have agreed to a trade in which Alice will send $\mathsf{amt_{xmr}}$ to Bob, and Bob will send $\mathsf{amt_{btc}}$ to Alice. They require this exchange to be atomic, i.e. the change of ownership of one asset should effectively imply the change of ownership of the other. Additionally, should the exchange not come to fruition, they expect any committed assets to be returned to them.

3.2   Overview

Happy path After exchanging a set of addresses, keys, zero-knowledge proofs and signatures, Bob locks up $\mathsf{amt_{btc}}$ in a Point Time Locked Contract (PTLC)[14] locked using point S_a^{btc} by publishing $tx_{\mathsf{lock}}^{\mathsf{btc}}$. Being a PTLC, the output is also spendable in an alternative manner after time t_1.

Alice subsequently locks up $\mathsf{amt_{xmr}}$ in a shared output with public spend key $S_a^{xmr} + S_b^{xmr}$ and public view key $V_a + V_b$ by publishing $tx_{\mathsf{lock}}^{\mathsf{xmr}}$. The relationship between S_a^{xmr} and S_a^{btc} is that they share the same secret key s_a despite being points on different elliptic curve groups. The same relationship applies to S_b^{xmr} and S_b^{btc}. This output will be owned by the party with knowledge of both s_a and s_b.

Bob notices the publication of $tx_{\mathsf{lock}}^{\mathsf{xmr}}$ and sends to Alice an adaptor signature[8] $\hat{\sigma}_{\mathsf{redeem}}^{S_a^{btc}, B}$ which she is able to combine with s_a producing $\sigma_{\mathsf{redeem}}^B$. Provided there is enough time until t_1, she then publishes a $tx_{\mathsf{redeem}}^{\mathsf{btc}}$ signed with $\sigma_{\mathsf{redeem}}^B$ and her own $\sigma_{\mathsf{redeem}}^A$. Broadcasting this transaction moves $\mathsf{amt_{btc}}$ to an address owned by Alice.
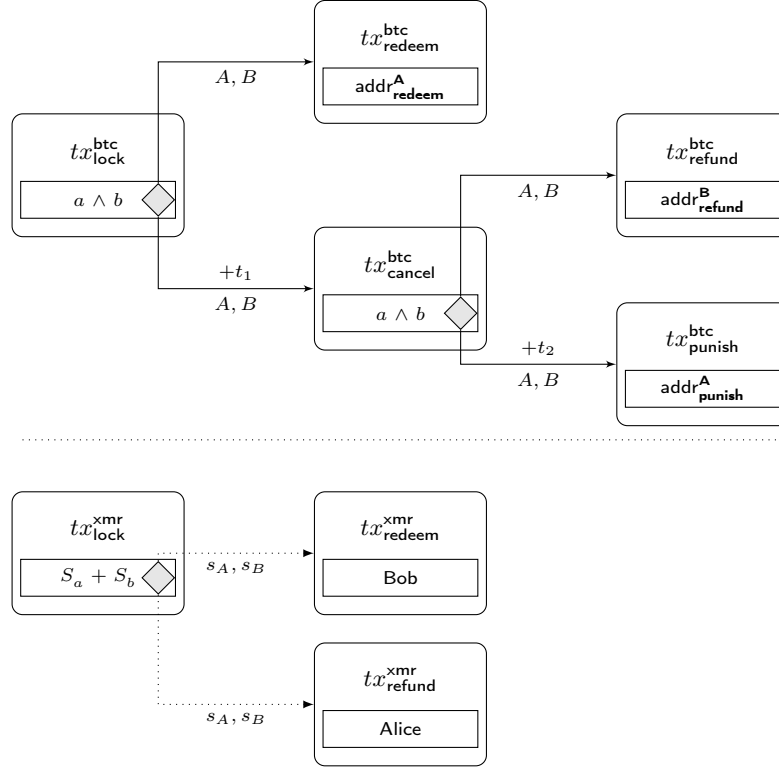
Fig. 1. Transaction schema for BTC to XMR atomic swaps. *Top*: Transaction schema for Bitcoin. *Bottom*: Transaction schema for Monero. *Note: Monero view keys are omitted for clarity.*

Finally, Bob sees $tx^{\mathsf{btc}}_{\mathsf{redeem}}$ on the blockchain. He finds $\sigma^B_{\mathsf{redeem}}$ in the witness stack and combines it with $\hat{\sigma}^{S^{btc}_a,B}_{\mathsf{redeem}}$ to learn s_a. With knowledge of both s_a and s_b, Bob is the de facto owner of $\mathsf{amt_{xmr}}$, which he is able to move to a different address of his at any time.

Cancel Once Bob has published $tx^{\mathsf{btc}}_{\mathsf{lock}}$, if time t_1 is reached, either party can elect to publish $tx^{\mathsf{btc}}_{\mathsf{cancel}}$, diverging from the "happy path". The transaction $tx^{\mathsf{btc}}_{\mathsf{cancel}}$ was constructed in such a way that it will only be mined after time t_1. The use of transaction-level timelocks is one of the ways in which this protocol deviates from the original[10].

Refund: With $tx^{\mathsf{btc}}_{\mathsf{cancel}}$ confirmed on the blockchain, Bob should immediately publish $tx^{\mathsf{btc}}_{\mathsf{refund}}$ to reclaim his $\mathsf{amt_{btc}}$ (minus some fees).

Alice would then spot $tx^{\mathsf{btc}}_{\mathsf{refund}}$ on the Bitcoin blockchain, giving her access to $\sigma^A_{\mathsf{refund}}$. Combining $\sigma^A_{\mathsf{refund}}$ with $\hat{\sigma}^{S^{btc}_b,A}_{\mathsf{refund}}$ would leak s_b to her. Knowing s_a and s_b,

Alice would effectively reclaim control over $\mathsf{amt_{xmr}}$, which she could eventually move back to one of her wallet addressess.

Punish: Should Bob remain inactive after $tx^{\mathsf{btc}}_{\mathsf{cancel}}$ is published, Alice still has a way to get compensation for the failed swap. After time t_2, Alice can punish Bob for not triggering the refund path in time by publishing $tx^{\mathsf{btc}}_{\mathsf{punish}}$. With this transaction Alice claims $\mathsf{amt_{btc}}$. The $\mathsf{amt_{xmr}}$ remains locked forever, but from Alice's perspective it is as if the trade went through.

The existence of $tx^{\mathsf{btc}}_{\mathsf{punish}}$ therefore incentivises Bob to publish $tx^{\mathsf{btc}}_{\mathsf{refund}}$ as soon as possible. Either way, Alice, the party who has no agency on whether refund will occur or not, remains protected.

3.3   Off-chain preparation

As hinted at in the previous section, before Alice and Bob can go on-chain they must exchange some data.

For simplicity we assume a fixed fee for all Bitcoin transactions that must be signed by both parties. In practice, the best way to handle transaction fees would be to adopt a Child-pays-for-parent (CPFP)[13] strategy, so that the parties do not have to commit to a particular fee rate ahead of time.

Key generation Firstly, they engage in a key generation protocol, as shown in Fig. 2.

Alice sends to Bob a Bitcoin public key A; a Monero private view key v_a; a Monero public spend key S^{xmr}_a; a Bitcoin public key S^{btc}_a; and a Discrete Logarithm Equality (DLEQ) π_{s_a} proof between S^{xmr}_a and S^{btc}_a. The characteristics of this kind of proof will be explained in greater detail in Section 3.5.

Similarly, Bob sends to Alice a Bitcoin public key B; a Monero private view key v_b; a Monero public spend key S^{xmr}_b; a Bitcoin public key S^{btc}_b; and a DLEQ proof π_{s_b} between S^{xmr}_b and S^{btc}_b.

If either party receives an invalid DLEQ proof, they must abort the protocol.

Address exchange Additionally, Alice sends to Bob two Bitcoin addresses $\mathsf{addr^A_{redeem}}$ and $\mathsf{addr^A_{punish}}$; and Bob sends to Alice one Bitcoin address $\mathsf{addr^B_{refund}}$. The $\mathsf{amt_{btc}}$ will end up in one of these depending on the protocol execution.

Expiries The value of the two timelocks t_1 and t_2 must be confirmed before Alice and Bob can sign any transactions. Timelock t_1 determines how long Alice will have to publish and confirm $tx^{\mathsf{xmr}}_{\mathsf{lock}}$, and safely redeem $tx^{\mathsf{btc}}_{\mathsf{lock}}$. Timelock t_2 determines how long Bob has to refund his bitcoin after $tx^{\mathsf{btc}}_{\mathsf{cancel}}$ is published by either party.
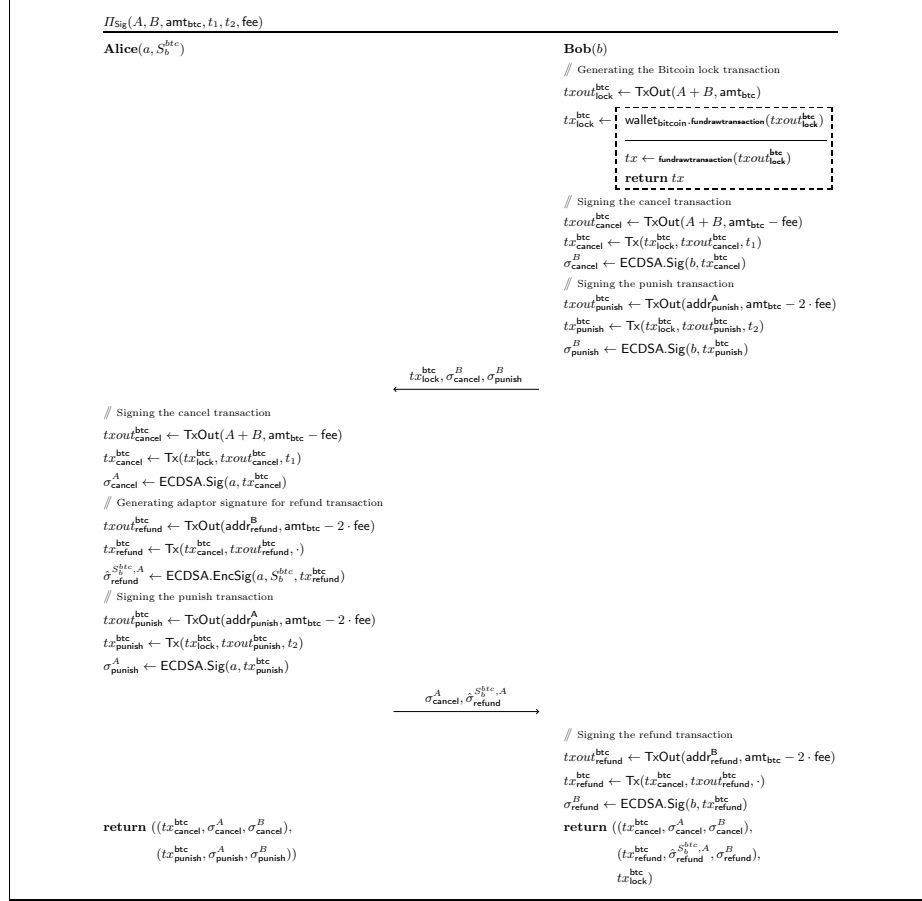
In this protocol we only use relative timelocks because they create consistent windows of action no matter when $tx^{\mathsf{btc}}_{\mathsf{lock}}$ and $tx^{\mathsf{btc}}_{\mathsf{cancel}}$ are included in a block.

Π_{KGen}

Alice	**Bob**

Alice:
$a \leftarrow_\$ \mathbb{Z}_q; A \leftarrow aG$
$v_a \leftarrow_\$ \mathbb{Z}_p$
$s_a \leftarrow_\$ \mathbb{Z}_p$
$S_a^{btc} \leftarrow s_a G; S_a^{xmr} \leftarrow s_a H$
$\pi_{s_a} \leftarrow \mathsf{P}_{\mathsf{DLEQ}((G,S_a^{btc}),(H,S_a^{xmr}),s_a)}$

$$\xrightarrow{\quad (A, v_a, S_a^{btc}, S_a^{xmr}, \pi_{s_a}) \quad}$$

Bob:
$\mathsf{V}_{\mathsf{DLEQ}((G,S_a^{btc}),(H,S_a^{xmr}),\pi_{s_a})} \stackrel{?}{=} 1$
$b \leftarrow_\$ \mathbb{Z}_q; B \leftarrow bG$
$v_b \leftarrow_\$ \mathbb{Z}_p$
$s_b \leftarrow_\$ \mathbb{Z}_p$
$S_b^{btc} \leftarrow s_b G; S_b^{xmr} \leftarrow s_b H$
$\pi_{s_b} \leftarrow \mathsf{P}_{\mathsf{DLEQ}((G,S_b^{btc}),(H,S_b^{xmr}),s_b)}$

$$\xleftarrow{\quad (B, v_b, S_b^{xmr}, S_b^{btc}, \pi_{s_b}) \quad}$$

Alice:
$\mathsf{V}_{\mathsf{DLEQ}((G,S_b^{btc}),(H,S_b^{xmr}),\pi_{s_b})} \stackrel{?}{=} 1$
return (a, A, B, v_a, v_b, s_a)

Bob:
return (b, A, B, v_a, v_b, s_b)

Fig. 2. Key generation protocol.

Signing phase This phase is a pre-requisite to Bob being able to lock up the bitcoin safely. It also ensures that Alice can safely lock up the monero herself, once she has confirmed that the bitcoin is on the blockchain.

Before either party can start signing the Bitcoin transactions, Bob must define what $tx_{\mathsf{lock}}^{\mathsf{btc}}$ looks like. Given what they both already know, they can construct the PTLC output: one which can be spent by providing signatures for A and B. Bob builds the rest of the transaction using a Bitcoin wallet which will contribute the necessary inputs and outputs. This is the first step of the signing protocol, which is depicted in Fig. 3. Bob sends the unsigned $tx_{\mathsf{lock}}^{\mathsf{btc}}$ to Alice, alongside the signatures $\sigma_{\mathsf{cancel}}^B$ and $\sigma_{\mathsf{punish}}^B$. He can safely share these signatures with Alice because $tx_{\mathsf{lock}}^{\mathsf{btc}}$ remains unpublished and unsigned.

With $tx_{\mathsf{lock}}^{\mathsf{btc}}$ Alice computes the signatures $\sigma_{\mathsf{cancel}}^A$ and $\sigma_{\mathsf{punish}}^A$. She also computes the adaptor signature $\hat{\sigma}_{\mathsf{refund}}^{S_b^{btc},A}$, which Bob would need to decrypt if he ever wants to refund his bitcoin. Using the corresponding decrypted signature $\sigma_{\mathsf{refund}}^A$ to publish $tx_{\mathsf{refund}}^{\mathsf{btc}}$ would leak s_b to Alice, allowing her to refund her own monero. Alice sends back $\sigma_{\mathsf{cancel}}^A$ and $\hat{\sigma}_{\mathsf{refund}}^{S_b^{btc},A}$ to Bob.

All that remains is for Bob to compute his own $\sigma_{\mathsf{cancel}}^B$.

$\Pi_{\mathsf{Sig}}(A, B, \mathsf{amt}_{\mathsf{btc}}, t_1, t_2, \mathsf{fee})$

Alice(a, S_b^{btc}) **Bob**(b)

          // Generating the Bitcoin lock transaction

          $txout_{\mathsf{lock}}^{\mathsf{btc}} \leftarrow \mathsf{TxOut}(A + B, \mathsf{amt}_{\mathsf{btc}})$

          $tx_{\mathsf{lock}}^{\mathsf{btc}} \leftarrow$ $\mathsf{wallet}_{\mathsf{bitcoin}.\mathbf{fundrawtransaction}}(txout_{\mathsf{lock}}^{\mathsf{btc}})$

                $tx \leftarrow \mathbf{fundrawtransaction}(txout_{\mathsf{lock}}^{\mathsf{btc}})$

                return tx

          // Signing the cancel transaction

          $txout_{\mathsf{cancel}}^{\mathsf{btc}} \leftarrow \mathsf{TxOut}(A + B, \mathsf{amt}_{\mathsf{btc}} - \mathsf{fee})$

          $tx_{\mathsf{cancel}}^{\mathsf{btc}} \leftarrow \mathsf{Tx}(tx_{\mathsf{lock}}^{\mathsf{btc}}, txout_{\mathsf{cancel}}^{\mathsf{btc}}, t_1)$

          $\sigma_{\mathsf{cancel}}^{B} \leftarrow \mathsf{ECDSA.Sig}(b, tx_{\mathsf{cancel}}^{\mathsf{btc}})$

          // Signing the punish transaction

          $txout_{\mathsf{punish}}^{\mathsf{btc}} \leftarrow \mathsf{TxOut}(\mathsf{addr}_b^{\mathsf{A}}{}_{\mathsf{punish}}, \mathsf{amt}_{\mathsf{btc}} - 2 \cdot \mathsf{fee})$

          $tx_{\mathsf{punish}}^{\mathsf{btc}} \leftarrow \mathsf{Tx}(tx_{\mathsf{lock}}^{\mathsf{btc}}, txout_{\mathsf{punish}}^{\mathsf{btc}}, t_2)$

          $\sigma_{\mathsf{punish}}^{B} \leftarrow \mathsf{ECDSA.Sig}(b, tx_{\mathsf{punish}}^{\mathsf{btc}})$

$$\xleftarrow{\quad tx_{\mathsf{lock}}^{\mathsf{btc}}, \sigma_{\mathsf{cancel}}^{B}, \sigma_{\mathsf{punish}}^{B} \quad}$$

// Signing the cancel transaction

$txout_{\mathsf{cancel}}^{\mathsf{btc}} \leftarrow \mathsf{TxOut}(A + B, \mathsf{amt}_{\mathsf{btc}} - \mathsf{fee})$

$tx_{\mathsf{cancel}}^{\mathsf{btc}} \leftarrow \mathsf{Tx}(tx_{\mathsf{lock}}^{\mathsf{btc}}, txout_{\mathsf{cancel}}^{\mathsf{btc}}, t_1)$

$\sigma_{\mathsf{cancel}}^{A} \leftarrow \mathsf{ECDSA.Sig}(a, tx_{\mathsf{cancel}}^{\mathsf{btc}})$

// Generating adaptor signature for refund transaction

$txout_{\mathsf{refund}}^{\mathsf{btc}} \leftarrow \mathsf{TxOut}(\mathsf{addr}_{\mathsf{refund}}^{\mathsf{B}}, \mathsf{amt}_{\mathsf{btc}} - 2 \cdot \mathsf{fee})$

$tx_{\mathsf{refund}}^{\mathsf{btc}} \leftarrow \mathsf{Tx}(tx_{\mathsf{cancel}}^{\mathsf{btc}}, txout_{\mathsf{refund}}^{\mathsf{btc}}, \cdot)$

$\hat{\sigma}_{\mathsf{refund}}^{S_b^{btc}, A} \leftarrow \mathsf{ECDSA.EncSig}(a, S_b^{btc}, tx_{\mathsf{refund}}^{\mathsf{btc}})$

// Signing the punish transaction

$txout_{\mathsf{punish}}^{\mathsf{btc}} \leftarrow \mathsf{TxOut}(\mathsf{addr}_{\mathsf{punish}}^{\mathsf{A}}, \mathsf{amt}_{\mathsf{btc}} - 2 \cdot \mathsf{fee})$

$tx_{\mathsf{punish}}^{\mathsf{btc}} \leftarrow \mathsf{Tx}(tx_{\mathsf{lock}}^{\mathsf{btc}}, txout_{\mathsf{punish}}^{\mathsf{btc}}, t_2)$

$\sigma_{\mathsf{punish}}^{A} \leftarrow \mathsf{ECDSA.Sig}(a, tx_{\mathsf{punish}}^{\mathsf{btc}})$

$$\xrightarrow{\quad \sigma_{\mathsf{cancel}}^{A}, \hat{\sigma}_{\mathsf{refund}}^{S_b^{btc}, A} \quad}$$

          // Signing the refund transaction

          $txout_{\mathsf{refund}}^{\mathsf{btc}} \leftarrow \mathsf{TxOut}(\mathsf{addr}_{\mathsf{refund}}^{\mathsf{B}}, \mathsf{amt}_{\mathsf{btc}} - 2 \cdot \mathsf{fee})$

          $tx_{\mathsf{refund}}^{\mathsf{btc}} \leftarrow \mathsf{Tx}(tx_{\mathsf{cancel}}^{\mathsf{btc}}, txout_{\mathsf{refund}}^{\mathsf{btc}}, \cdot)$

          $\sigma_{\mathsf{refund}}^{B} \leftarrow \mathsf{ECDSA.Sig}(b, tx_{\mathsf{refund}}^{\mathsf{btc}})$

return $((tx_{\mathsf{cancel}}^{\mathsf{btc}}, \sigma_{\mathsf{cancel}}^{A}, \sigma_{\mathsf{cancel}}^{B}),$ **return** $((tx_{\mathsf{cancel}}^{\mathsf{btc}}, \sigma_{\mathsf{cancel}}^{A}, \sigma_{\mathsf{cancel}}^{B}),$

        $(tx_{\mathsf{punish}}^{\mathsf{btc}}, \sigma_{\mathsf{punish}}^{A}, \sigma_{\mathsf{punish}}^{B}))$ $(tx_{\mathsf{refund}}^{\mathsf{btc}}, \hat{\sigma}_{\mathsf{refund}}^{S_b^{btc}, A}, \sigma_{\mathsf{refund}}^{B}),$

                                                   $tx_{\mathsf{lock}}^{\mathsf{btc}})$

Fig. 3. Signing protocol. Both parties must verify the signatures received, but this is left out for clarity.

3.4   On-chain protocol

In Section 3.3 we have explained how Alice and Bob set the stage for the swap to take place. The sequence diagram in Fig. 4 shows the rest of the steps towards a successful atomic swap.

With the ability to broadcast signed versions of $tx^{\mathsf{btc}}_{\mathsf{cancel}}$ and $tx^{\mathsf{btc}}_{\mathsf{refund}}$ to take his coins back, Bob can now proceed by publishing $tx^{\mathsf{btc}}_{\mathsf{lock}}$. He uses his Bitcoin wallet to sign each input and broadcasts it to the network.

Alice finds $tx^{\mathsf{btc}}_{\mathsf{lock}}$ on the blockchain by using the transaction ID which can be deterministically computed from $tx^{\mathsf{bitcoin}}_{\mathsf{lock}}$. With enough confirmations on it to consider it irreversible and sufficient time until t_1, Alice publishes $tx^{\mathsf{xmr}}_{\mathsf{lock}}$. The only requirement on this transaction is that it must pay $\mathsf{amt_{xmr}}$ to the address corresponding to the public spend key $S^{xmr}_a + S^{xmr}_b$ and the public view key $V_a + V_b$. Bob does not need to know any other details, because the parties do not need to sign transactions depending on $tx^{\mathsf{xmr}}_{\mathsf{lock}}$ ahead of time.

Bob finds $tx^{\mathsf{xmr}}_{\mathsf{lock}}$ on the blockchain by leveraging his knowledge of the private view key $v_a + v_b$. In Monero, only parties with knowledge of the private view key are privy to transactions involving the matching address. Once Bob considers that $tx^{\mathsf{xmr}}_{\mathsf{lock}}$ has garnered enough confirmations, he proceeds by sending $\hat{\sigma}^{S^{btc}_a,B}_{\mathsf{redeem}}$ to Alice. This adaptor signature can be decrypted by Alice to grant her the ability to redeem $\mathsf{amt_{btc}}$.

On receiving $\hat{\sigma}^{S^{btc}_a,B}_{\mathsf{redeem}}$ Alice first verifies that what she has received is useful to her by executing $\mathsf{ECDSA.EncVrfy}(B, S^{btc}_a, tx^{\mathsf{btc}}_{\mathsf{redeem}}, \hat{\sigma}^{S^{btc}_a,B}_{\mathsf{redeem}})$. This ensures that the adaptor signature commits to a valid signature on B for the transaction $tx^{\mathsf{btc}}_{\mathsf{redeem}}$, encrypted by S^{btc}_a. With knowledge of s_a Alice decrypts it by calling $\mathsf{ECDSA.Dec}(s_a, \hat{\sigma}^{S^{btc}_a,B}_{\mathsf{redeem}})$, obtaining $\sigma^B_{\mathsf{redeem}}$.

Alice now has the means to publish $tx^{\mathsf{btc}}_{\mathsf{redeem}}$, but she must only do so if there is enough time to confirm the transaction before t_1. Otherwise, Bob could front-run her transaction with $tx^{\mathsf{btc}}_{\mathsf{cancel}}$, ensuring his refund of $\mathsf{amt_{btc}}$ and still finding $tx^{\mathsf{btc}}_{\mathsf{redeem}}$ in the mempool, with which he would be able to also claim the $\mathsf{amt_{xmr}}$. Assuming there is enough time, she goes ahead and publishes $tx^{\mathsf{btc}}_{\mathsf{redeem}}$, claiming $\mathsf{amt_{btc}}$.

Finally, Bob can use the information obtained by the publication of $tx^{\mathsf{btc}}_{\mathsf{redeem}}$ to claim the $\mathsf{amt_{xmr}}$. He takes the transaction from the blockchain, extracts the signature $\sigma^B_{\mathsf{redeem}}$ from it and calls $\mathsf{ECDSA.Rec}(\sigma^B_{\mathsf{redeem}}, \hat{\sigma}^{S^{btc}_a,B}_{\mathsf{redeem}})$ to obtain s_a. As the sole owner of both s_a and s_b, Bob is the only one capable of moving $\mathsf{amt_{xmr}}$ to a different address. He does so at his own convenience, so that he can safely forget $s_a + s_b$.

3.5   Cross-chain DLEQ proof

The key generation phase depicted in Fig. 2 shows both Alice and Bob producing a so-called *cross-curve* DLEQ proof. This construct is used to non-interactively prove in zero-knowledge that the public key pair (S^{btc}_a, S^{xmr}_a) has a common secret key s_a, and that the public key pair (S^{btc}_b, S^{xmr}_b) has a common secret
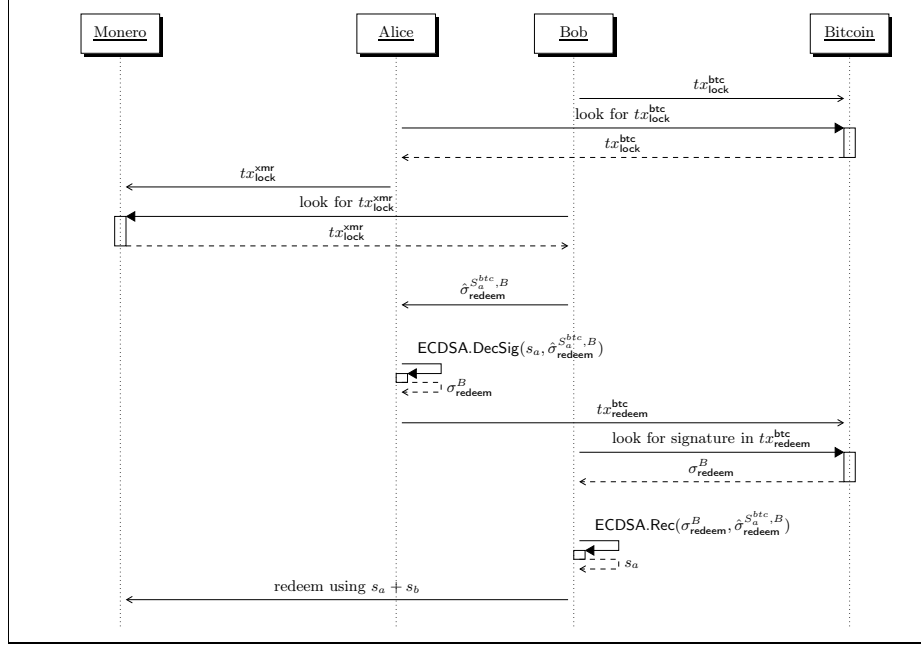
Fig. 4. Happy path on-chain protocol.

key s_b. Without these proofs, there would be no guarantee that the adaptor signatures $\hat{\sigma}_{\mathsf{redeem}}^{S_a^{btc},B}$ and $\hat{\sigma}_{\mathsf{refund}}^{S_b^{btc},A}$ which they later exchange actually commit to the expected signature-secret pairs.

Conventionally, this kind of proof can only be constructed for points on the same elliptic curve. The idea of proving discrete logarithm equality across different groups comes from [12]. The algorithm proposed in [12] is used in the original protocol by Gugger, but we elect to use something simpler based on the original idea so that it can be argued secure by the composition of sigma protocols[16].

We built an experimental implementation of this proof[3] to support our proof-of-concept implementation of this protocol[2]. We also contributed to a more general and efficient implementation of the same proof[9] which we intend to use in the future.

4   XMR to BTC atomic swaps

In the section above we described an atomic swap protocol between Bitcoin and Monero.

That protocol is appropriate for a use case in which the Service Provider (SP) is in the role of Alice, i.e. she offers buying XMR for BTC to her customers. Following the protocol as defined in Section 3, offering that kind of trade, allows

the SP to lock up $\mathsf{amt_{xmr}}$ (by publishing $tx_{\mathsf{lock}}^{\mathsf{xmr}}$) only after the other party has locked up $\mathsf{amt_{btc}}$ (by publishing $tx_{\mathsf{lock}}^{\mathsf{btc}}$). This is safe for the SP as she knows that she will either be able to redeem (publish $tx_{\mathsf{redeem}}^{\mathsf{btc}}$) or refund.

However, using that protocol to swap in the *opposite* direction is not feasible i.e. an SP should not offer buying BTC for XMR. The problem is that an SP (in the role of Bob) could be easily attacked: the taker (in the role of Bob) could agree on a trade with the SP, make him lock up funds on Bitcoin and then bail out at no cost. The SP could always refund his locked up BTC after some time, but he would have to pay for transaction fees to do so. The taker's ability to make the SP incur in transaction fees without penalty would expose the SP to running out of funds over time, which is why we refer to this as a *draining attack*.

We need a different protocol to allow an SP to offer BTC/XMR buy trades, since the original makes it a hard requirement for the party holding BTC to move first. In the following sections we propose a new protocol which instead requires the party holding XMR to move first. This depends on the development of adaptor signatures based on Monero's ring signature scheme, which is a work-in-progress and whose details are left out of the scope of this work.

4.1   Protocol definition
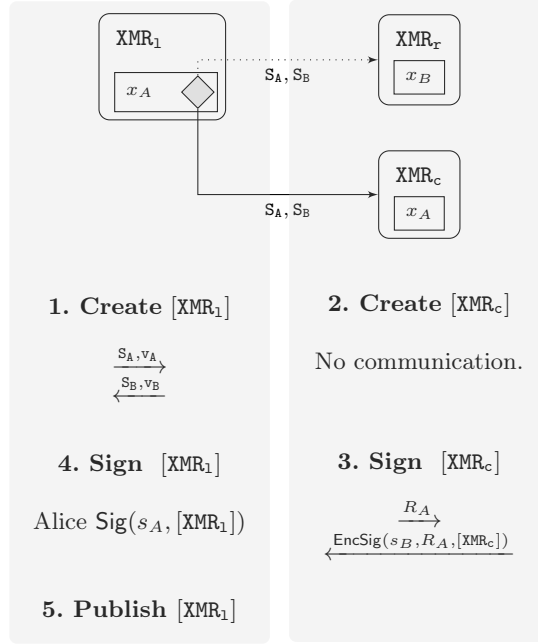
Fig. 5. Monero transaction schema.
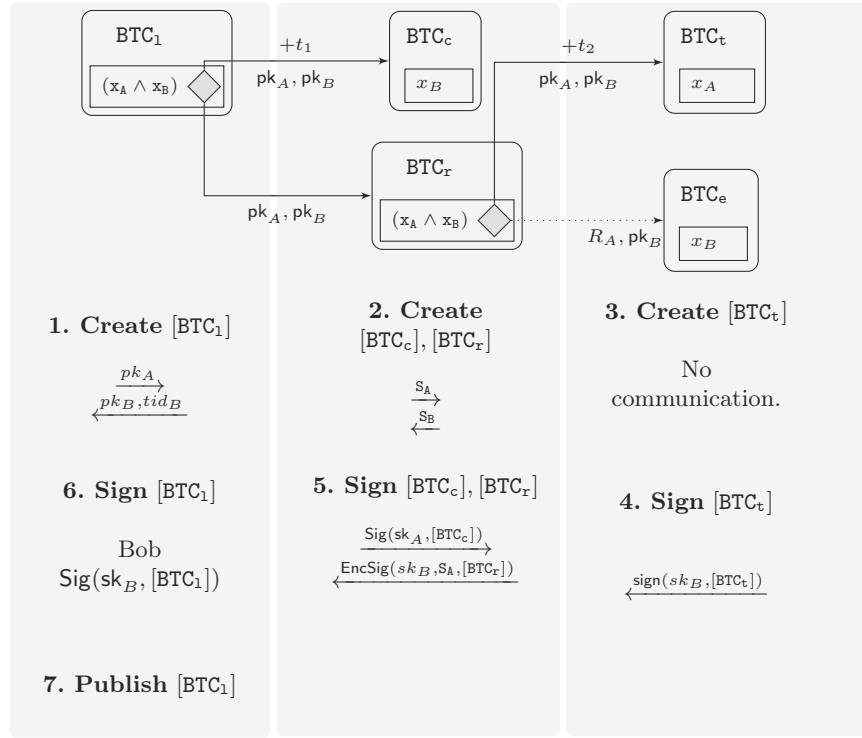
Fig. 6. Bitcoin transaction schema.

Fig. 5 and Fig. 6 show the transaction schema for Monero and Bitcoin respectively. These diagrams are used to illustrate the relationships between different transactions on the same blockchain. Transactions are represented as rectangles with rounded corners. Transaction outputs are depicted as boxes inside transactions. The value of the output is written inside the output box and their spending conditions are written above and below arrows coming out of the output. For example, x_A means the output holds x coins owned by party A and $(x_A \wedge x_B)$ means the output of amount x is controlled by party A and B.

With regard to the spending conditions, we define the following convention: the public keys of all required signatures are listed *below* the arrow; other conditions, such as timelocks, appear *above* the arrow.

4.2   Creating Monero transactions

The transaction schema for Monero can be found in Figure 5. Below we describe the naive 5-step protocol to create all transactions. An optimized implementation could reduce the number of steps by combining messages, but we refrain from doing so in the interest of clarity.

Step 1: To construct the locking transaction $\texttt{XMR}_1$ the parties need to exchange some keys: Alice shares with Bob a public spend key S_A and her private view key v_A, as well as her funding source tid_A. Bob shares with Alice his public spend key S_B and his private view key v_B. They can now create $\texttt{XMR}_1$ locally with input tid_A and an output with public spend key $S_A + S_B$, private view key $v_A + v_B$. Notably, Alice does not sign the transaction yet. Both parties now have a local copy of an unsigned $\texttt{XMR}_1$ which requires one signature from each party to spend its output.

Step 2: Both parties create the refund transaction $\texttt{XMR}_c$ which spends from $\texttt{XMR}_1$ and returns the funds back to Alice. Notably, they do not create the redeem transaction $\texttt{XMR}_r$ in the same way, because they do not have to exchange any signatures on it. The key idea is that Bob will learn s_A later on if Alice publishes $\texttt{BTC}_r$, allowing him to construct, sign and publish the redeem transaction $\texttt{XMR}_r$ by himself.

Step 3: Like in Section 3.3 for the old protocol, adaptor signatures are used but this time on Bitcoin *and* Monero. Alice generates a keypair (r_A, R_A) and constructs a DLEQ proof for it for the same reasons presented in Section 3.5. She sends R_A to Bob, which he uses as the encryption key to generate an adaptor signature on the refund transaction $\texttt{XMR}_c$. Bob sends this adaptor signature to Alice. If she were to ever publish $\texttt{XMR}_c$ she would need to use this adaptor signature, leaking r_A to Bob. This would allow him to execute an emergency refund on Bitcoin if Alice were misbehaving by attempting to take both the bitcoin and the monero.

Steps 4+5: Alice could now sign the locking transaction $\texttt{XMR}_1$ and publish it on the Monero blockchain with the assurance that she could get her funds back at any point by publishing $\texttt{XMR}_1$. But these steps are not carried out until the two parties have collaborated on creating the Bitcoin transactions.

4.3 Creating transactions for Bitcoin

The transaction schema for Bitcoin can be found in Figure 6.

Step 1: To prepare the locking transaction $\texttt{BTC}_1$, Alice shares a public key pk_A with Bob. Bob shares his funding source tid_B with Alice as well as a public key pk_B. Both parties can now create $\texttt{BTC}_1$ which spends from tid_B into a multisignature output requiring two signatures: one for pk_A and another one for pk_B.

Step 2: Knowing $\texttt{BTC}_1$, both parties can construct $\texttt{BTC}_c$, a transaction which returns the bitcoin back to Bob after time t_1. They also construct $\texttt{BTC}_r$. This transaction sets the stage for Alice to be able to take the bitcoin. It can be spent in two ways: (1) Alice can claim the coins after time t_2 by providing signatures for pk_A and pk_B, and (2) Bob can still refund if he learns Alice's refund secret r_A and uses it with his own public key pk_B. Bob would learn r_A if Alice publishes $\texttt{BTC}_c$, using the adaptor signature generated in step 3 of the Monero transaction creation protocol above.

Step 3: Having constructed $\texttt{BTC}_r$, both parties can create $\texttt{BTC}_t$, which spends from it and can be published after time t_2 giving the funds to Alice.

Step 4: For safety purposes, transactions are signed in reverse order of publication. To that end, Alice and Bob collaboratively sign BTC_t. Only Bob sends his signature to Alice because she is the one that would care to publish this transaction, since it benefits her. There is no need to create BTC_e which would require a signature from R_A and pk_B. Bob will be able to create and sign this transaction by himself if the situation allows.

Step 5: Alice and Bob sign BTC_c collaboratively. Only Alice shares her signature with Bob because he is the only one interested in ever being able to take his bitcoin back. Bob also generates an adaptor signature on BTC_r for his public key pk_B encrypted under S_A and sends it to Alice. This adaptor signature ensures the atomicity of the swap: if Alice publishes BTC_r she will need to decrypt and use the adaptor signature, leaking s_A, which he would use to take the monero.

Step 6+7: Bob is now ready to sign and publish BTC_l. He still must wait for Alice to lock her monero first by publishing XMR_l, finishing steps 4 and 5 of Section 4.2. Once Alice has committed her funds to the Monero blockchain, Bob is safe to do the same on Bitcoin.

4.4 Protocol execution

The content of this section is still work-in-progress. Hence we do not delve deeper into the cryptography which is needed to create adaptor signatures on Monero. Instead, we continue describing things on a high level.

Scenario The motivation behind this protocol is to allow the party holding XMR and wanting BTC to move first. In this scenario, Alice holds XMR and wants to receive BTC. Conversely, Bob holds BTC and wants to receive XMR.

After successfully building and signing transactions following the steps outlined in Section 4.2 and Section 4.3, Alice and Bob are ready to go on-chain.

Happy path Alice publishes her locking transaction XMR_l knowing that she can always receive her funds back by cancelling the swap and publishing XMR_c. Once Bob is happy with the amount of confirmations on XMR_l he follows suit and publishes the locking transaction on Bitcoin BTC_l. Given sufficient confirmations on BTC_l and enough time until t_1, Alice publishes the redeem transaction BTC_r. In doing so, she leaks s_A to Bob. Alice cannot immediately claim the bitcoin for herself but has to wait until time t_2. In the meantime, Bob has until time t_2 to safely take the monero by using s_A to create and sign XMR_r, and publishing it on the blockchain. Once time t_2 is reached. Alice can finally take the bitcoin by publishing BTC_t, completing the atomic swap.

One party is unresponsive At any point in time during the execution phase either party could become inactive. In order to prevent money loss, both parties have mechanisms at their disposal to refund.

For instance, Alice could publish her locking transaction XMR_1 and then see that Bob never moves forward with the publication of his locking transaction BTC_1. As depicted in Fig. 5, XMR_c requires signatures on S_A and S_B. Alice can use her own secret key s_A to produce one of the signatures, and decrypt Bob's adaptor signature using r_A to produce the other. She would then publish XMR_c, taking back her monero.

Similarly, if Bob does publish BTC_1, but Alice fails to continue by publishing BTC_r before time t_1, Bob can then take his bitcoin by publishing BTC_c, since he either has or can produce or the signatures needed for it to be valid.

Alice tries to cheat There exists an edge case in which Alice can attempt to take both assets. This is possible after both parties have published their respective locking transactions XMR_1 and BTC_1. Alice may attempt to redeem the bitcoin by publishing BTC_r *and* refund the monero by publishing XMR_c. Fortunately, the publication of XMR_c would leak s_A to Bob which would allow him to create, sign and publish BTC_e to execute an emergency refund, at least until time t_2. The result would be equivalent to having executed a *normal* refund. Bob therefore remains protected, but this possibility imposes a strong requirement for him to stay online at all times.

5   Conclusion

Atomic swaps constitute the main mechanism to bridge the gap between unrelated blockchains without violating the core principles of censorship resistance, permissionlessness and pseudonymity originally championed by Bitcoin. Up until recently, their application was believed to be exclusive to blockchains with very particular characteristics. Advances in cryptography have lowered the barrier to entry, allowing for new protocols to be devised in order to connect blockchains that were originally thought to be incompatible. One such example is the *Bitcoin–Monero Cross-chain Atomic Swap* by Gugger[10], which has inspired the development of applications such as [7] and [2].

In this work, we give a high-level sketch of a new protocol which expands on the ideas of the original to serve a new use case. In particular, by applying adaptor signatures to the Monero signature scheme, we make possible atomic swaps in which the party holding BTC is no longer the one vulnerable to draining attacks. A real-world service provider could therefore leverage both protocols to put up buy and sell BTC/XMR offers as a market maker.

This proposal hinges on the viability of using adaptor signatures on Monero, a topic which we do not discuss here, but one which is being researched at the time of writing.

References

1. COMIT: Comit. https://github.com/comit-network/comit-rs (2018), accessed: 2021-01-13

2. COMIT: Bitcoin-monero cross-chain atomic swap. `https://github.com/comit-network/xmr-btc-swap/tree/91fe18a79657e7d8ee100c931a2b2fcce0f1cd0f` (2020), accessed: 2021-01-27
3. COMIT: Cross-curve dleq. `https://github.com/comit-network/cross-curve-dleq/tree/eddcdea1d1f16fa33ef581d1744014ece535c920` (2020), accessed: 2021-01-27
4. COMIT: Grin-bitcoin atomic swap. `https://github.com/comit-network/grin-btc-poc/tree/38cf690fa65d115db7354bee1905cb2c694308fc` (2020), accessed: 2021-01-27
5. CryptoSec: Crypto exchange hacks. `https://cryptosec.info/exchange-hacks/` (2021), accessed: 2021-01-13
6. ExchangeUnion: Opendex. `https://opendex.network/` (2020), accessed: 2021-01-13
7. Farcaster: Farcaster project. `https://github.com/farcaster-project/RFCs` (2020), accessed: 2021-01-27
8. Fournier, L.: One-time verifiably encrypted signatures a.k.a adaptor signatures. `https://github.com/LLFourn/one-time-VES/blob/master/main.pdf` (2019)
9. Fournier, L.: Sigmafun! `https://github.com/LLFourn/secp256kfun/tree/fa25e7ee0b7bcc5d6d12550b8def9ab798dbadca` (2020), accessed: 2021-01-27
10. Gugger, J.: Bitcoin–monero cross-chain atomic swap. `https://eprint.iacr.org/2020/1126.pdf` (2020)
11. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf` (2008), accessed: 2021-01-13
12. Noether, S.: Discrete logarithm equality across groups. `https://web.getmonero.org/es/resources/research-lab/pubs/MRL-0010.pdf` (2018)
13. Optech, B.: Child-pays-for-parent (cpfp). `https://bitcoinops.org/en/topics/cpfp`, accessed: 2021-01-27
14. Optech, B.: Point time locked contracts (ptlcs). `/https:/bitcoinops.org/en/topics/ptlc`, accessed: 2021-01-27
15. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
16. Schoenmakers, B.: Lecture notes cryptographic protocols. `https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf` (2020)
17. TierNolan: Atomic swaps - bitcointalk forum. `https://bitcointalk.org/index.php?topic=193281.msg2224949\#msg2224949` (2013), accessed: 2021-01-13