LightSwap: An Atomic Swap Does Not Require Timeouts At Both Blockchains *

Philipp Hoenisch[1], Subhra Mazumdar[2], Pedro Moreno-Sanchez[3], and Sushmita Ruj[4]

[1] CoBloX Pty Ltd, Australia
philipp@coblox.tech
[2] TU Wien, Christian Doppler Laboratory Blockchain Technologies for the Internet of Things Vienna, Austria
subhra.mazumdar@tuwien.ac.at
[3] IMDEA Software Institute, Madrid, Spain
pedro.moreno@imdea.org
[4] School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
sushmita.ruj@unsw.edu.au

Abstract. Security and privacy issues with centralized exchange services have motivated the design of *atomic swap* protocols for decentralized trading across currencies. These protocols follow a standard blueprint similar to the 2-phase commit in databases: (i) both users first lock their coins under a certain (cryptographic) condition and a timeout; (ii-a) the coins are swapped if the condition is fulfilled; or (ii-b) coins are released after the timeout. The quest for these protocols is to minimize the requirements from the scripting language supported by the swapped coins, thereby supporting a larger range of cryptocurrencies. The recently proposed universal atomic swap protocol [IEEE S&P'22] demonstrates how to swap coins whose scripting language only supports the verification of a digital signature on a transaction. However, the timeout functionality is cryptographically simulated with verifiable timelock puzzles, a computationally expensive primitive that hinders its use in battery-constrained devices such as mobile phones. In this state of affairs, we question whether the 2-phase commit paradigm is necessary for atomic swaps in the first place. In other words, is it possible to design a secure atomic swap protocol where the timeout is not used by (at least one of the two) users?

In this work, we present LightSwap, the first secure atomic swap protocol that does not require the timeout functionality (not even in the form of a cryptographic puzzle) by one of the two users. LightSwap is thus better suited for scenarios where a user, running an instance of LightSwap on her mobile phone, wants to exchange coins with an online exchange service running an instance of LightSwap on a computer. We show how LightSwap can be used to swap Bitcoin and Monero, an interesting use case since Monero does not provide any scripting functionality support other than linkable ring signature verification.

* A full version of our paper is available in [2]

Keywords: Blockchain · Atomic swap · Bitcoin · Monero · Lightweight applications · Adaptor signatures.

1 Introduction

The functionality of atomic swaps [18] was introduced for trading assets between two parties such that each of them holds assets in a different blockchain. The concept of atomicity in such a setting is inspired by database systems where either a multi-step transaction gets committed or it is rolled back in its entirety. In the blockchain setting, it holds similar relevance guaranteeing that the swap either fully occurs or fails entirely [17,44].

As an illustrative example, consider that a user *Alice* has asset α in blockchain $\mathcal{B}_A$ and user *Bob* has asset β in blockchain $\mathcal{B}_B$. An atomic swap is said to be successful when *Bob* transfers asset β to *Alice* on $\mathcal{B}_B$ contingent to the transfer of asset α by *Alice* to *Bob* on $\mathcal{B}_A$. If *Alice* decides to cancel the swap, a refund will be initiated. Upon asset refund, *Alice* will retain α in $\mathcal{B}_A$ and *Bob* will retain β in $\mathcal{B}_B$. A successful swap thereby leads to an exchange of asset's ownership [42]. Hence both the parties need to have accounts in each of the blockchains to enable transfer of ownership [28].

While one can easily envision an atomic swap functionality leveraging a trusted server, the blockchain community has put significant efforts into decentralized protocols for atomic swaps [1,36,18,45,44,35,29,26,39,30]. In a nutshell, these different protocols follow a standard blueprint based on two building blocks: (i) a (cryptographic) locking mechanism that allows one user to locks coins for another user in a given blockchain; and (ii) a timeout mechanism that allows the creator of a lock to release it after a certain time has expired. With these building blocks, current atomic swap protocols are based on the following blueprint: first, *Alice* locks α in $\mathcal{B}_A$ for *Bob* and establishes an expiration time of T_A to such lock. Afterward, *Bob* locks β in $\mathcal{B}_B$ to *Alice* with an expiration time of $T_B : T_A > T_B$. At this point, the atomic swap has been committed and one of the following two outcomes can happen: (i) *Bob* allows *Alice* to unlock β in $\mathcal{B}_B$, which in turn "automatically" allows *Bob* to unlock α in $\mathcal{B}_A$; or (ii) both parties decide to abort the swap by allowing to release the locks at times T_B and T_A respectively.

This blueprint framework used by atomic swaps is based on two crucial properties. First, the (cryptographic) locks should allow to "relate" one to another in the sense that if one party opens one lock in one blockchain, such opening operation automatically reveals enough information to the other party to open her own lock in the other blockchain. Such "correlated locks" have been implemented in practice using different techniques such as leveraging the Turing-complete scripting language of blockchains like Ethereum [40] or more specific scripting functionality like Hash-time lock contract [18,7,12,30], using a third blockchain [21,22,41] as the coordinator or bridge of the two blockchains [43,23,24,34,3] used for the swap , leveraging trusted hardware [6], or designing cryptographic schemes crafted for this purpose such as adaptor signatures [39,13].

The second crucial property is that locked funds must be released to the original owner after a certain time has expired. Surprisingly, all alternative protocols previously mentioned share only two techniques with regard to handling the timelock functionality. They either (i) rely on the scripting language of the underlying blockchain to implement it; or (ii) rely on a cryptographic timelock puzzle [33,37,10] where a secret is saved under a cryptographic puzzle that can be solved after a certain number of serial cryptographic operations are executed. Unfortunately, both of these techniques clearly hinder the adoption of atomic swaps. On the one hand, timelock based on the scripting language restricts its use from those cryptocurrencies that do not have such support, such as Monero [31] or Zcash (shielded addresses) [20]. On the other hand, cryptographic puzzles impose a computation burden on the users that need to compute such a puzzle for each of the atomic swaps that they are involved in. Such a scheme is not suitable for lightweight applications as it would drain the battery of a smartphone or would add a non-trivial cost if outsourced to a third party (e.g., Amazon Web Services [11]).

In this state of affairs, we raise the following question: *Is the timelock functionality a necessary condition to design atomic swap protocols?* Or in other words, *is it possible to design an atomic swap protocol such that the timelock functionality is not required in (at least one of) the two involved blockchains?*

1.1 Our contribution

In this work, we present for the first time a secure, decentralized, and trustless atomic swap protocol that does not require any type of timelock in one of the cryptocurrencies. In particular, we present LightSwap, a lightweight atomic swap between Bitcoin and Monero. Similar to previous works, LightSwap leverages adaptor signatures to implement the cryptographic condition that correlates the locks over the committed coins. The crux of the contribution in LightSwap is to depart from the 2-phase paradigm. Instead, we propose a novel paradigm that maintains the security for the users (i.e., an honest user does not lose coins) while removing the need to use timeouts in any form for one of the two cryptocurrencies.

2 Notation and background

Transactions in UTXO model. In this work, we focus on the UTXO transaction model, as it is followed by both Bitcoin and Monero.

For readability, transaction charts are used to visualize the transactions, their ordering, and usage in any protocol. We follow the notation in [5]. The charts must be read from left to right as per the direction of the arrows. A transaction is represented as a rectangular box with a rounded corners, input to such transactions is denoted by incoming arrows and output by outgoing arrows. Each rectangular box has square boxes drawn within. These boxes represent the output of the transaction, termed as *output boxes*, and the value within represents

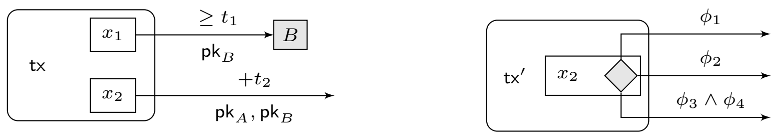

Fig. 1: (Left) Transaction tx has two outputs, one of value x_1 that can be spent by B (indicated by the gray box) with a transaction signed w.r.t. pk_B at (or after) round t_1, and one of value x_2 that can be spent by a transaction signed w.r.t. pk_A and pk_B but only if at least t_2 rounds passed since tx was accepted on the blockchain. (Right) Transaction tx' has one input, which is the second output of tx containing x_2 coins and has only one output, which is of value x_2 and can be spent by a transaction whose witness satisfies the output condition $\phi_1 \vee \phi_2 \vee (\phi_3 \wedge \phi_4)$. The input of tx is not shown.

the number of coins. Conditions for spending these coins are written on the output arrows going out of these boxes. The notations and the illustration of the transaction charts are provided in Figure 1.

The parties that can spend these coins present in the output box are represented below the outgoing arrows in form of a signature. Usually, these are represented as the public keys which can verify this signature. Additional conditions for spending the coins are written above the arrow. Conditions are encoded in a script supported by the underlying cryptocurrency. For our paper, we use the notation "$+t$" or $\texttt{RelTime}(t)$ which denotes the waiting time before a transaction containing an output can be published on-chain. This is termed as the *relative locktime*. If absolute locktime is used, then it is represented as "$\geq t$" or $\texttt{AbsTime}(t)$. It means the condition for spending the output is satisfied if the height of the blockchain is at least t. For representing multiple conditions, if it is a disjunction of several conditions, i.e. $\phi = \phi_1 \vee \phi_2 \vee \ldots \vee \phi_n$, a diamond-shaped box is used in the output box and each sub condition ϕ_i is written above the output arrow. The conjunction of several conditions is represented as $\phi = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_m$.

Adaptor signatures. We recall the functionality for generation and verification of adaptor signature with respect to a hard relation. This becomes one building block in our approach to substitute the functionality of HTLC. In more detail, given a hard relation $R : (x, X) \in R$, where X is the statement and x is a witness, public key pk having secret key sk, the language L_R and a signature scheme $\Sigma = (\texttt{Gen},\texttt{Sign},\texttt{Vrfy})$, an adaptor signature is defined using four algorithms $\Xi_{R,\Sigma} = (\texttt{pSign, pVrfy, Adapt, Ext})$ as follows [4]:

- $\texttt{pSign}(sk, m, X)$: A probabilisitc polynomial time algorithm which on input of secret key sk, message $m \in \{0,1\}^*$ and statement $X \in L_R$, outputs an a pre-signature $\hat{\sigma}$.
- $\texttt{pVrfy}(pk, m, X, \hat{\sigma})$: A deterministic polynomial time algorithm which on input the public key pk, the message $m \in \{0,1\}^*$, the statement $X \in L_R$, and pre-signature $\hat{\sigma}$, outputs a bit b. If $b = 1$, $\hat{\sigma}$ is a valid pre-signature on message m.

– `Adapt`$(\hat{\sigma}, x)$: A deterministic polynomial time algorithm which on input the witness for the statement X, i.e. x and the pre-signature $\hat{\sigma}$, outputs a signature σ.
– `Ext`$(\sigma, \hat{\sigma}, X)$: A deterministic polynomial time algorithm which on input signature σ, pre-signature $\hat{\sigma}$ and the statement $X \in L_R$, outputs a witness $x : (x, X) \in R$ or $\perp$.

In this work, we leverage the threshold adaptor signature for ECDSA [27] for the Bitcoin side and the instance defined in [38,29] for Monero. In a 2-of-2 threshold adaptor signature instance, each participant has a share of the secret key *sk*.

3 Problem Definition

Given a user *Alice* and the service provider *Bob*, the former holds x XMR in Monero blockchain and *Bob* holds y BTC in Bitcoin blockchain. *Alice* wants to exchange x XMR for *Bob*'s y BTC. A generic atomic swap protocol follows a 2-phase commit protocol similar to that used in databases: (i) each user commits their assets and (ii) each user claims the assets of the counterparty. To initiate an atomic swap, both parties need to lock their coins and set a timeperiod within which the swap must be completed. If *Alice* wants to cancel the swap, she will initiate a refund and the locked coins are refunded to the original owner after the designated timeperiod.

Existing atomic swap protocols and their drawbacks. We discuss existing approaches as solution for the problem defined above. We denote *Alice* as **A** and *Bob* as **B**.

(i) *Using HTLC based approach.* The simplest trustless exchange protocol widely used across several cryptocurrency exchange is based on *Hash Time-locked Contract or HTLC*. We discuss an HTLC based solution where both **A** and **B** hold their coins at time t_0. The script used in HTLC takes the tuple $(\alpha, h, t, \mathbf{A}, \mathbf{B})$, where α is the asset to be transferred, h is the hash value, and t is the contract's timeout period. The contract states that **A** will transfer α to **B** contingent to the knowledge r where $h = \mathtt{H}(r)$ where $\mathtt{H}$ is a standard cryptographic hash function if the contract is invoked within the timeout period t. If the timeperiod elapses and **B** fails to invoke the contract, the asset α is refunded to user **A**.

A can initiate exchange of x XMR in $\mathcal{B}_A$ for y BTC in $\mathcal{B}_B$ using HTLC. The former chooses a random value r and generates $h = \mathtt{H}(r)$. She next proceeds to lock x XMR in the contract $H_1 = HTLC(x, h, t_5, \mathbf{A}, \mathbf{B})$ at time t_1, where $t_1 > t_0$, and sends h, t_5 to **B**. The timeout period of the contract is t_5. Now **B** will reuse the same terms of the contract but set the timeperiod as $t_4 : t_4 < t_5$. We will explain why the timeout period must be less than the previous contract. **B** locks y BTC in the contract $H_2 = HTLC(y, h, t_4, \mathbf{B}, \mathbf{A})$ at time t_2, where $t_2 > t_1$. **A** knows the preimage of h and claim the coins from **B** by invoking H_2 at time $t_3 : t_2 < t_3 < t_4$. **B** gets the preimage r which he can use for claiming coins from

A. If he had used the timeout period t_5 for H_2, then it is quite possible that **A** delays and claims the coins from **B** just at time t_5. This would lead to a race condition and **B** might fail to acquire the coins from **A** if the time at which H_1 is invoked exceeds t_5. Hence he sets the timeout period of the contract H_2 less than the timeout period of contract H_1. **B** claims the coins from **A** by invoking H_1 at time $t_4 : t_3 < t_4 < t_5$. By time t_5, **A** holds y BTC in $\mathcal{B}_B$ and **B** holds x XMR in $\mathcal{B}_A$. This depicts the situation when the swap succeeds and the state transition from time t_0 to t_5 discussed above is termed as *happy path*. If either of the party decides not to co-operate then it will lead to failure of swap.

Incompatibility of HTLC in scriptless cryptocurrencies (e.g., Monero). HTLC-based approach requires the use of timelock on both the Monero side as well as the Bitcoin side. The timeout mechanism is essential to allow users to recover their assets in the case the swap does not go through. Thus we require two main building blocks to implement atomic swaps for cryptocurrencies: an atomic locking mechanism and a timeout. However, the main challenge is that Monero does not support hashlock and timelock. Without these two features, it will not be possible for **A** to lock her coins at time t_1. The use of timelock puzzles will make our protocol unsuitable for lightweight applications. Hence none of the paths can be initiated.

(ii) *Without using HTLC for Monero.* A fix for the challenges faced in HTLC based protocol would be to design a protocol without having any hashlock and timelock at Monero side, but **B** uses HTLC for locking y BTC in $\mathcal{B}_B$. In Monero, coins locked in the address can be spend only by the party possessing the private key of that particular address. The modified protocol allows **A** to lock her coins in an address say pk, whose secret key is solely possessed by her. This will allow **A** to initiate a refund at her will. Let the secret key be s. She locks x XMR in address pk at time t_1. Using this secret key, she generates $h_s : h_s = \mathtt{H}(s)$. She shares h_s with **B**. The latter locks y BTC into $HTLC(y, h_s, t_4, \mathbf{B},\mathbf{A})$ at time t_2. For a successful swap, **A** invokes HTLC using the secret s at time t_3 and claims y BTC. **B** uses the secret key s to spend x XMR locked in address pk at t_4 and transfers it to his address in $\mathcal{B}_A$.

Attack on this approach. Apparently, it might look like we can accomplish the swap using this approach. However, the problem is now **A** can initiate a refund at any time she wants. Even if she initiates a refund after t_2, she can still invoke the HTLC as $t_2 < t_4$, and claim y BTC from **B**. The service provider **B** will lose his coins. To counter this problem, we can resort to 2-of-2 secret sharing where each half of the secret key s of address pk will be shared with **A** and **B**. This will make **A** dependent on **B** for issuing a refund, violating our objective. If **B** does not lock his coins at t_2, **A**'s coins will remain locked forever.

From the above discussion, it is clear that designing an efficient protocol without any kind of timeout in one of the two chains is a challenging task. We provide a high-level overview of our proposed solution in the next section.

4 Our approach

4.1 Solution overview

Our protocol must ensure that the party moving first is allowed to issue a refund without depending on the counterparty. However, it must also be ensured that if the swap is canceled, both parties must get a refund. Since Monero does not support timelocks, we need to design a protocol that leverages the timelock used in the Bitcoin script. We use threshold adaptor signature for seamless redemption and refund of coins without any party suffering a loss in the process.

Signing refund transaction in Monero. Consider an atomic swap where *Alice* (or **A**) wants to exchange her monero for *Bob*'s (or **B**) bitcoin. If she locks her coins in an address whose secret key is known to her, she can spend the coins at any time. It is better if the secret key is shared where each half is possessed by **A** and **B**. However, this would mean that **A** has to depend on **B** for initiating a refund. If **B** does not cooperate, then **A**'s coin will remain locked forever. Hence both of them must collaborate and sign the refund transaction even before **A** locks her coins. The signature generated uses *threshold version of adaptor signature*. To generate such a signature, **B** uses his portion of the secret key as well as a cryptographic condition, say R, to generate the incomplete signature. **A** can complete the signature using her share of the secret key and upon fulfilling the hard relation R inserted by **B**. On the Bitcoin side, once **A** invokes the redeem transaction, the coins can be redeemed by her only after a certain timeperiod, say t, elapses. In the meantime, if **B** finds that **A** has refunded her coins but still invoked the redeem transaction at the Bitcoin side, then he can publish his refund transaction within the timeperiod t. A valid signature for a refund transaction can be generated by providing a witness to the relation R. Once **A** has published her refund transaction on $\mathcal{B}_A$, **B** will know the witness and hence, he can claim a refund easily.

We now describe our proposed two-party atomic swap protocol ensuring that none of the parties lose coins in the process.

4.2 Protocol description

We discuss a lightwieght atomic swap protocol where **A** wants to exchange x_A XMR for y_B BTC of **B**. The protocol consists of six phases: *setup, lock, redeem, cancel, emergency refund* and *punish*. The transaction schema for BTC to XMR atomic swap is shown in Figure 2. x_A coins are held in blockchain $\mathcal{B}_A$ and y_B coins are held in blokchain $\mathcal{B}_B$.

Setup phase. In this phase, **A** and **B** jointly create the public key pk in $\mathcal{B}_A$. **A** uses pk to generate an address for locking her coins. Each party will generate one-half of the secret key, i.e., **A** will generate s_A, and **B** will generate s_B. A linear combination of their secret keys will result in s. The latter serves as the private key of the address pk. Additionally, **A** samples an additional secret r_A and generates the statements R_A for $\mathcal{B}_A$ and R^*_A for $\mathcal{B}_B$ (For example, $R_A = r_a G$ and $R^*_A = r_a H$ for two different groups having generator G and H

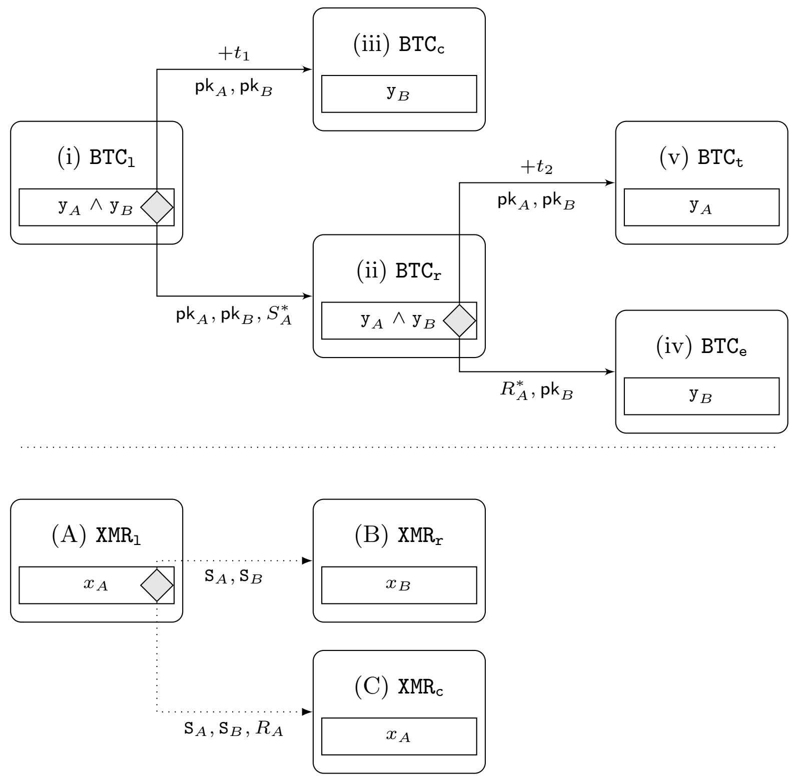

Fig. 2: New transaction schema for BTC to XMR atomic swaps. *Top*: Transaction schema for Bitcoin. *Bottom*: Transaction schema for Monero. Here x_A and x_B denotes the fact that x Monero coins belong to either Alice or Bob correspondingly. Similarly with y_A and y_B in Bitcoin.

respectively). **A** generates a proof π_{r_a} that proves r_A is the witness to both the statements R_A and R_A^*. Similarly, using one half of secret key, s_A, **A** generate the statements S_A and S_A^* for the blockchains $\mathcal{B}_A$ and $\mathcal{B}_B$ respectively. **B** generates a proof π_{s_a} that proves s_a is the witness to S_A and S_A^*. **B** also generates a proof π_{s_b} that proves that s_b is the witness of statement S_B. Both parties share $((\pi_{r_a}, R_A, R_A^*), (\pi_{s_a}, S_A, S_A^*), (\pi_{s_b}, S_B))$. The readers may refer the full version of the paper for details on generation of proof for each statement.

Pre-signing of Monero Refund transaction: **A** creates a Monero refund transaction $\mathtt{XMR_c}$, box (C) in Figure 2, where x_A coins locked in address pk is send to another address on $\mathcal{B}_A$ controlled by **A**.

$$\mathtt{XMR_c} : \mathsf{pk} \xrightarrow{x_A} \mathbf{A}$$

Later, **A** and **B** collaborate and pre-sign $\mathtt{XMR_c}$ based on the statement R_A. Both parties provide their share of private spend keys in the process of generating the adaptor signature without revealing it explicitly. This allows **A** to opt for a refund anytime she wants.

Exchanging signatures for the transactions on Bitcoin side: **B** shares his funding source, tx_{fund}, with **A**. The source has a balance of at least y_B coins. The transaction $\mathtt{BTC_1}$, box (i) in Figure 2, is created where **B** will lock his coins in a 2-of-2 multisig redeem script, $\mathsf{pk}^{lock}_{A,B}$. The output is denoted as $y_A \wedge y_B$.

$$\mathtt{BTC_1} : tx_{fund} \xrightarrow{y_A \wedge y_B} \mathsf{pk}^{lock}_{A,B}$$

The coins can either be redeemed by **A** or refunded by **B** after a certain timeperiod t_1. **A** can publish the transaction $\mathtt{BTC_r}$, box (ii) in Figure 2, spends the output of $\mathtt{BTC_1}$ and again locks into a 2-of-2 multisig redeem script, $\mathsf{pk}^{redeem}_{A,B}$.

$$\mathtt{BTC_r} : \mathsf{pk}^{lock}_{A,B} \xrightarrow{y_A \wedge y_B} \mathsf{pk}^{redeem}_{A,B}$$

The output of $\mathtt{BTC_r}$ can either be refunded to **B**, if there is an emergency, or it can be claimed by **A** after a certain timeperiod t_2. **A** creates the transaction $\mathtt{BTC_t}$, box (v) in Figure 2, which will allow her to spend the output of $\mathtt{BTC_r}$ after timeperiod t_2 and shares it with **B**.

$$\mathtt{BTC_t} : \mathsf{pk}^{redeem}_{A,B} \xrightarrow{y_A} \mathbf{A}$$

The latter signs $\mathtt{BTC_t}$ and sends it to **A**. Later **B** creates the transaction $\mathtt{BTC_c}$, represented in box (iii) in Figure 2. It allows him to refund the output $y_A \wedge y_B$ coins of $\mathtt{BTC_1}$.

$$\mathtt{BTC_c} : \mathsf{pk}^{lock}_{A,B} \xrightarrow{y_A} \mathbf{B}$$

B sends $\mathtt{BTC_c}$ to **A** for signature. **A** sends $\mathtt{BTC_r}$ to **B**. The latter verifies the transaction, pre-signs the transaction $\mathtt{BTC_r}$ based on the statement S^*_A and sends the partially signed transaction to **A**. Now **A** will sign the transaction $\mathtt{BTC_1}$ and send it to **B**.

Lock phase. **A** creates the transaction $\mathtt{XMR_1}$, box (A) in Figure 2 where she locks x_A coins into address pk.

$$\mathtt{XMR_1} : \mathbf{A} \xrightarrow{x_A} \mathsf{pk}$$

B, upon verification that **A** has locked the coins, proceeds with publishing $\mathtt{BTC_1}$ and locks his coins as well.

Redeem phase. **A** knows the witness s_A for the statement S^*_A and thus she generates a valid signature for $\mathtt{BTC_r}$. She publishes the transaction but cannot spend the output before a timperiod of t_2 has elapsed. Meanwhile, **B** extracts s_A from the signature on $\mathtt{BTC_r}$. He will create the transaction $\mathtt{XMR_r}$, box (B) in Figure 2 that will allow him to redeem the coins locked in address pk.

$$\mathtt{XMR_r} : \mathsf{pk} \xrightarrow{x_B} \mathbf{B}$$

By combining the secret keys s_A and s_B, he will be able to sign $\mathtt{XMR_r}$ and publish it on-chain.

Cancel swap. If **A** wants to cancel the swap, she will generate a valid signature for $\mathtt{XMR_c}$ using the witness r_A and publish it to claim her coins. Meanwhile, **B** can wait till t_1 has elapsed since $\mathtt{BTC_1}$ was published and **A** has not initiated the swap. He publishes $\mathtt{BTC_c}$ and unlocks his coins.

Emergency refund. Suppose **A** has initiated the swap by publishing $\mathtt{BTC_r}$ but she has unlocked her coins by publishing $\mathtt{XMR_c}$. Once $\mathtt{XMR_c}$ is published, **B** extracts r_A from the signature on $\mathtt{XMR_c}$. He will create transaction $\mathtt{BTC_e}$, box (iv) in Figure 2 and spend $y_A \wedge y_B$ coins locked in $\mathsf{pk}_{A,B}^{redeem}$.

$$\mathtt{BTC_e} : \mathsf{pk}_{A,B}^{redeem} \xrightarrow{y_B} \mathbf{B}$$

Now he will sign the transaction using r_A and publish the transaction on-chain before t_2 elapses.

From the above discussion on *emergency refund*, we emphasize the utility of not allowing **A** to redeem the coins locked by **B**. Instead, a waiting time of t_2 allows **B** to recover his coins, if **A** is malicious. On one hand, **A** can initiate a refund any time she wants but on the other hand, she cannot claim the bitcoins instantly.

Punish. If **B** has published $\mathtt{XMR_r}$ and claimed x_B coins, then **A** waits for t_2 timeperiod to elapse after publishing $\mathtt{BTC_r}$. She will publish $\mathtt{BTC_t}$ and claim y_A coins.

Now, consider that **B** has stopped responding and has neither claimed x_B coins nor initiated a refund. In that case, **A** can *punish* him for remaining inactive by publishing $\mathtt{BTC_t}$. Hence, this phase is called *punish* phase and **B** loses his bitcoins. A detailed description of the protocol can be found in the full version of our paper [2].

4.3 Security and privacy goals

- **Correctness**: If both parties are honest, with one party willing to exchange x units of coin for y units of coins of the other party, then the protocol terminates with each party obtaining the desired amount.
- **Soundness**: An honest party must not lose funds while executing the protocol with an adversary.
- **Unlinkability**: Any party not involved with the atomic swap must not be able to link two cross-chain transactions responsible for the atomic swap, except with negligible probability.
- **Fungibility**: An adversary must not be able to distinguish between a normal transaction and a transaction for atomic swap in Monero Blockchain, except with negligible probability.

We discuss how the security properties defined above holds for our proposed protocol:

– **Correctness**: If both parties **A** and **B** are honest, then the atomic swap protocol ensures that if party **A** is able to redeem y_A coins then party B can redeem x_B coins as well within a bounded timeperiod. This is possible since when **A** publishes $\texttt{BTC}_\texttt{r}$, **B** extracts the secret s_A from signature on $\texttt{BTC}_\texttt{r}$ and uses the same for signing transaction $\texttt{XMR}_\texttt{r}$.
– **Soundness**: If party **A** initiates the swap but publishes $\texttt{XMR}_\texttt{c}$ before **B** publishes $\texttt{XMR}_\texttt{r}$, then a relative locktime of t_2 on spending the output of $\texttt{BTC}_\texttt{r}$ allows **B** to opt for an emergency refund by publishing $\texttt{BTC}_\texttt{e}$ and refund his coins.
– **Linkability**: Since Monero transactions are confidential and signatures on transactions are generated from random values, any malicious party observing both the Monero and Bitcoin blockchains will be able to link a pair of Bitcoin and Monero transactions involved in the swap with negligible probability.
– **Fungibility**: There is no structural difference between a normal Monero transaction and a Monero transaction constructed for LightSwap. Any malicious party observing the Monero blockchain can distinguish between such a pair of transactions with negligible probability.

A detailed security analysis of LightSwap in the Global Universal Composability (GUC) [9] framework has been discussed in the full version of our paper [2].

5 Discussion

5.1 Building Monero transactions

Pre-signing transactions involve signing a transaction where the outputs that need to be spent as input in this transaction have not been added to the blockchain. Since the private spend key and private view key for spending the output of $\texttt{XMR}_\texttt{l}$ is generated using 2-of-2 secret sharing, it requires both parties to co-operate and generate a valid signature for spending this output. However, if Bob stops responding, Alice will never get back her coins. Pre-signing $\texttt{XMR}_\texttt{c}$ will allow her to go for refund anytime she wants prior to signing of $\texttt{XMR}_\texttt{l}$ [25]. Unfortunately, it is not possible to implement the pre-signing of Monero transaction in its present form. We specify the key components for building a Monero transaction - (i) a transaction has a ring signature per input to hide exactly which output is being spent, (ii) a unique key image for an input being spent to avoid double-spending, (iii) Pedersen commitments [32] for every input and output, retaining the confidentiality of the transaction, and lastly, (iv) to show that difference in input and output of a transaction is non-negative, bulletproofs [8] are used.

The input of a Monero transaction, denoted as `vin`, consists of the amount, key offsets, and key image. Since the amount is confidential, it is set to 0. The key offset allows verifiers to find ring member keys and commitments in the blockchain. It consists of the real output public key along with 10 other decoy outputs. The first offset value is the absolute height of the block where the first member is present. Rest are assigned values relative to the absolute value. For example, if the set of 11 public keys forming ring members have real offsets $\{h, h+4, h+6, h+10, h+20, h+33, h+45, h+50, h+67, h+77, h+98\}$, then it is recorded as $\{h, 4, 2, 4, 10, 13, 12, 5, 17, 10, 21\}$ where h is the height of the block where the first public key can be found and each subsequent offset is relative to the previous. This set is termed as *"ring"* and is stored in the transaction. To ensure that a particular output can only be used once as an input, Monero includes a key image of the output's public key. The key image is constructed using the public key of the output that will be spent. This avoids double-spending attacks in Monero blockchain. Next, we discuss how the input *"ring"* is used for constructing the ring signature CLSAG.

For computing the signature hash $c_{i+1}, \forall i \in \{0, 1, \ldots, 10\}$ where $c_{11} = c_0$, *"ring"* is taken as input along with other parameters and concatenated with L_i and R_i. To generate the signature, the offsets must be known. Offsets are not known until and unless all the outputs in set *ring* have been added to the blockchain. Lack of offsets violates the policy of pre-signing where the transaction must be signed before the output that needs to be spent gets added to the blockchain. To avoid this problem, instead of using the key offsets as input for generating a signature hash, the set of public keys can be used as input. However, this would require changing Monero's codebase but the change is necessary for realizing Layer 2 protocols in Monero blockchain.

5.2 Building Bitcoin transactions

We created the necessary Bitcoin transactions for LightSwap and deployed these transactions on the Bitcoin testnet. We observed and recorded the size of transactions in bytes, where $\texttt{BTC}_\texttt{l}$ and $\texttt{BTC}_\texttt{r}$ is 360 B each, $\texttt{BTC}_\texttt{c}$ is 230 B, $\texttt{BTC}_\texttt{e}$ is 231 B, and $\texttt{BTC}_\texttt{t}$ is 229 B. Our result demonstrates the compatibility of the protocol with the current Bitcoin network. The code is available in `https://anonymous.4open.science/r/btc_xmr_swap-A7B1`, forked from `https://github.com/generalized-channels/gc`.

6 Related work

There have been efforts to design time locks on Monero. DLSAG [29] mentions that Monero is locked in a 2-of-2 joint address comprising two different public keys. Any one of the public keys can be used to spend Monero from the address based on certain conditions, for example, pre-defined block height. However, Monero needs to undergo a hard fork to implement DLSAG. Thyagarajan et al. [38] proposed the first payment channel for Monero, PayMo, without requiring any system-wide modifications. Additionally, the authors have also proposed a secure atomic cross-chain swap using PayMo. The payment channel uses a new cryptographic primitive called *Verifiable Timed Linkable Ring Signature (VTLRS)*. The signature scheme uses the timed commitment of a linkable ring signature on a given Monero transaction. However, timed commitment requires a huge computation overhead, making it unsuitable for designing lightweight protocols.

Threshold ring multi-signature proposed by Goodell and Noether [15] was used for spender-ambiguous cross-chain atomic swaps. Their construction doesn't involve any timelock mechanism, it is based on sharing of secret keys - whenever one party goes on-chain for claiming the amount, the other party can reconstruct the secret key completely. However, the paper doesn't formally define the refund method in case one of the parties acts maliciously. Gugger [16] proposed atomic swaps between Monero and Bitcoin. However, as per the concept of the atomic swap, the party which initiates the swap must lock its money first in its native blockchain. However, Gugger's protocol requires the counterparty selling Bitcoin in exchange for Monero to move first. This is not desired as it puts the counterparty at risk. Since there is a timelock involved before which the Bitcoins can be refunded, the buyer of Bitcoin may resort to mounting draining attack [14] by not locking his Monero. We have provided a detailed comparison of Gugger's protocol and LightSwap in Section A of Appendix. Hoenisch and Pino [19] provide a high-level sketch of a protocol that mitigates the limitations of Gugger's protocol. However, it avoids any detailed description of the construction of the adaptor ring signature on Monero.

7 Conclusions

We propose LightSwap, a lightweight two-party atomic swap facilitating the exchange of Bitcoin and Monero. LightSwap does not require any type of timeout at one of the two blockchains, without additional trust assumptions. Our protocol is thus efficient, fungible, scalable, and can be used for any cryptocurrency whose script does not support timelock. Either the party can initiate a refund, even if the counterparty does not cooperate. We provide steps for implementing LightSwap that demonstrate the ability to seamlessly deploy the protocol if Monero's codebase is changed to enable Layer 2 protocols. In the future, we are interested to study if a protocol can be designed without using timelock even at the Bitcoin side and what additional trust assumptions would be needed.

Acknowledgments

This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement 771527-BROWSEC), by the Austrian Science Fund (FWF) through the projects PROFET (grant agreement P31621) and the project W1255-N23, by the Austrian Research Promotion Agency (FFG) through COMET K1 SBA and COMET K1 ABC, by the Vienna Business Agency through the project Vienna Cybersecurity and Pri- vacy Research Center (VISP), by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association through the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things

(CDL-BOT). This work has been partially supported by Madrid regional government as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union, by SCUM Project (RTI2018-102043-B-I00) MCIN/AEI/10.13039/501100011033/ERDF A way of making Europe, by grant `IJC2020-043391-I/MCIN/AEI/10.13039/501100011033` and European Union NextGenerationEU/PRTR, and by grant N00014-19-1-2292 from ONR.

References

1. Tiernolan. Technical report, https://github.com/TierNolan, 2013.
2. anonymous. Lightswap: An atomic swap does not require timeouts at both blockchains (full version), 2022. `https://anonymous.4open.science/r/LightSwap-7C07/Final-LongversionXMR_lock_then_BTC.pdf`.
3. Team Ark. Ark ecosystem whitepaper, 2019. `https://ark.io/Whitepaper.pdf`.
4. Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized bitcoin-compatible channels. *IACR Cryptol. ePrint Arch.*, 2020:476, 2020.
5. Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure multi-hop payments without two-phase commits. In *USENIX Security 21*, 2021.
6. Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *CCS '19, London, UK, November 11-15, 2019*, pages 1521–1538. ACM, 2019.
7. Michael Borkowski, Marten Sigwart, Philipp Frauenthaler, Taneli Hukkinen, and Stefan Schulte. Dextt: Deterministic cross-blockchain token transfers. *IEEE Access*, 7:111030–111042, 2019.
8. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.
9. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography Conference*, pages 61–85. Springer, 2007.
10. Peter Chvojka, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Versatile and sustainable timed-release encryption and sequential time-lock puzzles. In *ESORICS '21*, pages 64–85. Springer, 2021.
11. Amazon Elastic Compute Cloud. Amazon web services. *Retrieved November*, 9(2011):2011, 2011.
12. Bingrong Dai, Shengming Jiang, Menglu Zhu, Ming Lu, Dunwei Li, and Chao Li. Research and implementation of cross-chain transaction model based on improved hash-locking. In *International Conference on Blockchain and Trustworthy Systems*, pages 218–230. Springer, 2020.
13. Apoorvaa Deshpande and Maurice Herlihy. Privacy-preserving cross-chain atomic swaps. In *FC '20*, pages 540–549. Springer, 2020.
14. Thomas Eizinger, Philipp Hoenisch, and Lucas Soriano del Pino. Open problems in cross-chain protocols. *arXiv preprint arXiv:2101.12412*, 2021.
15. Brandon Goodell and Sarang Noether. Thring signatures and their applications to spender-ambiguous digital currencies. *IACR Cryptol. ePrint Arch.*, 2018:774, 2018.

16. Joël Gugger. Bitcoin-monero cross-chain atomic swap. Cryptology ePrint Archive, Report 2020/1126, 2020. `https://eprint.iacr.org/2020/1126`.
17. Runchao Han, Haoyu Lin, and Jiangshan Yu. On the optionality and fairness of atomic swaps. In *ACM AFT '19*, pages 62–75, 2019.
18. Maurice Herlihy. Atomic cross-chain swaps. In Calvin Newport and Idit Keidar, editors, *PODC '18, Egham, United Kingdom, July 23-27, 2018*, pages 245–254. ACM, 2018.
19. Philipp Hoenisch and Lucas Soriano del Pino. Atomic swaps between bitcoin and monero. *CoRR*, abs/2101.12332, 2021.
20. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification.
21. Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *FC '19*, pages 21–34. Springer, 2019.
22. Komodo. Komodo (advanced blockchain technology, focused on freedom), 2018. `https://cryptorating.eu/whitepapers/Komodo/2018-02-14-Komodo-White-Paper-Full.pdf`.
23. Jae Kwon and Ethan Buchman. Cosmos whitepaper. *A Netw. Distrib. Ledgers*, 2019.
24. Rongjian Lan, Ganesha Upadhyaya, Stephen Tse, and Mahdi Zamani. Horizon: A gas-efficient, trustless bridge for cross-chain transactions. *arXiv preprint arXiv:2101.06000*, 2021.
25. Lucas. How to build a monero transaction, 2021. `https://comit.network/blog/2021/05/19/monero-transaction/`.
26. Léonard Lys, Arthur Micoulet, and Maria Potop-Butucaru. *R-SWAP: Relay based atomic cross-chain swap protocol*. PhD thesis, Sorbonne Université, 2021.
27. Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.
28. Mahdi H Miraz and David C Donald. Atomic cross-chain swaps: development, trajectory and potential of non-monetary digital token swap facilities. *Annals of Emerging Technologies in Computing (AETiC) Vol*, 3, 2019.
29. Pedro Moreno-Sanchez, Arthur Blue, Duc Viet Le, Sarang Noether, Brandon Goodell, and Aniket Kate. DLSAG: non-interactive refund transactions for interoperable payment channels in monero. In Joseph Bonneau and Nadia Heninger, editors, *FC '20, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 325–345. Springer, 2020.
30. Krishnasuri Narayanam, Venkatraman Ramakrishna, Dhinakaran Vinayagamurthy, and Sandeep Nishad. Generalized htlc for cross-chain swapping of multiple assets with co-ownerships. *arXiv preprint arXiv:2202.12855*, 2022.
31. Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. `https://eprint.iacr.org/2015/1098`.
32. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.
33. Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, 1996.
34. Drew Stone. Trustless, privacy-preserving blockchain bridges. *arXiv preprint arXiv:2102.04660*, 2021.

35. Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A^{2}l: Anonymous atomic locks for scalability and interoperability in payment channel hubs. *IACR Cryptol. ePrint Arch.*, 2019:589, 2019.
36. Stefan Thomas and Evan Schwartz. A protocol for interledger payments. *URL https://interledger. org/interledger. pdf*, 2015.
37. Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. Verifiable timed signatures made practical. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20, USA, November 9-13, 2020*, pages 1733–1750. ACM, 2020.
38. Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Fritz Schmidt, and Dominique Schröder. Paymo: Payment channels for monero. *IACR Cryptol. ePrint Arch.*, 2020:1441, 2020.
39. Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sánchez. Universal atomic swaps: Secure exchange of coins across all blockchains. *Cryptology ePrint Archive*, 2021.
40. Hangyu Tian, Kaiping Xue, Xinyi Luo, Shaohua Li, Jie Xu, Jianqing Liu, Jun Zhao, and David SL Wei. Enabling cross-chain transactions: A decentralized cryptocurrency exchange protocol. *IEEE Transactions on Information Forensics and Security*, 16:3928–3941, 2021.
41. Gilbert Verdian, Paolo Tasca, Colin Paterson, and Gaetano Mondelli. Quant overledger whitepaper, 2018. `https://uploads-ssl.webflow.com/6006946fee85fda61f666256/60211c93f1cc59419c779c42_Quant_Overledger_Whitepaper_Sep_2019.pdf`.
42. Gang Wang. Sok: Exploring blockchains interoperability.
43. Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 21:2327–4662, 2016.
44. Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. Atomic commitment across blockchains. *arXiv preprint arXiv:1905.02847*, 2019.
45. Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William J. Knottenbelt. XCLAIM: trustless, interoperable, cryptocurrency-backed assets. In *IEEE S & P '19, San Francisco, CA, USA, May 19-23, 2019*, pages 193–210. IEEE, 2019.

A Detailed comparison with Gugger protocol

Gugger proposed a protocol for swapping **B**'s bitcoins for **A**'s monero without using timelocks at the Monero side[16]. **A** locks her monero in an address, whose one half of the private spend key is with **A** and other half with **B**. On the other hand, **B** locks bitcoin in a 2-of-2 multi-sig address having two outputs, one is redeemed and one is for refunding. The redeem script uses a hashlock where the preimage of the hash must be used for claiming Bitcoins. Initially **B** locks bitcoin and upon confirmation, **A** locks her monero. After **A** has verified that **B** has locked bitcoin, she sends the preimage of the hash defined in the redeem script. Using it, **B** publishes the redeem transaction and releases his part of the private spend key to **A**. The latter uses it to construct the private spend key and claim monero. **A** is at risk of losing her deposit forever if **B** refuses to collaborate while refunding. There is no way **A** can refund her coins without **B**'s secret. The schematic diagram of the protocol is shown in Figure 3.

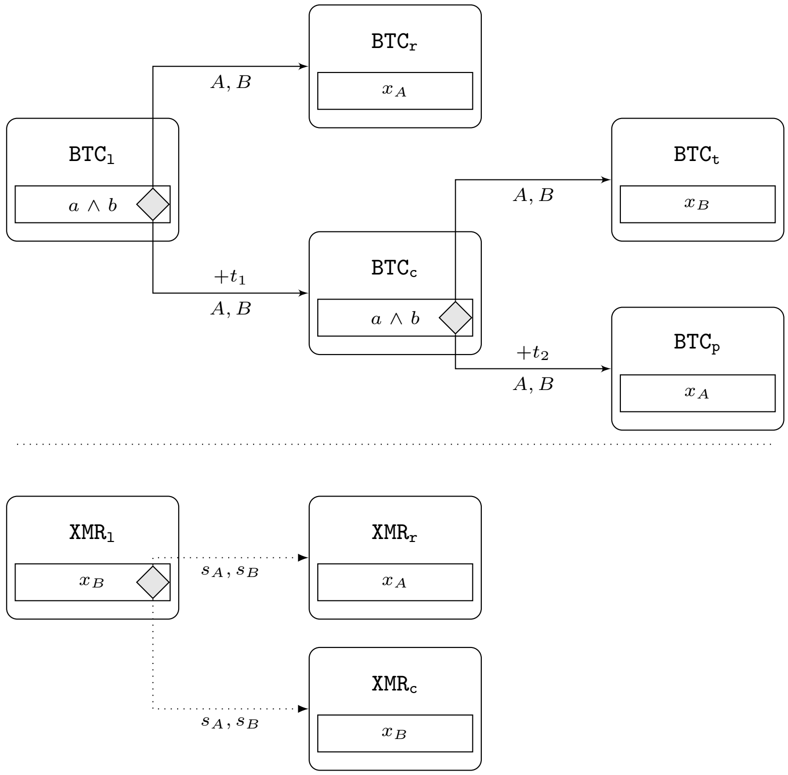

Fig. 3: Transaction schema for BTC to XMR atomic swaps from Gugger et al [16]. *Top*: Transaction schema for Bitcoin. *Bottom*: Transaction schema for Monero. *Note: Monero view keys are omitted for clarity.*

To address these problems, we propose a protocol that allows **A** to refund instead of depending on **B**. With this guarantee, she can always move first by locking XMR before **B** locks BTC. We use the adaptor ring signature for the refund transaction of Monero. But making this minor change in [16] won't help since providing freedom to **A** puts **B** at risk of losing money. It is quite possible that **A** publishes the refund transaction first and then claims bitcoins. To prevent such a situation, **A** will be allowed to claim bitcoins only after **B** has redeemed monero. Thus once **A** publishes the redeem transaction, the money cannot be spent immediately. A *contest period* is added before she can claim bitcoins.