Atomic swaps

Lorenzo Tucci

June 10, 2024

Contents

1 Introduction

1.1 Problem description

Consider two ledgers $\mathbb{B}_a$ and $\mathbb{B}_b$. Alice holds assets a in $\mathbb{B}_a$ and Bob holds assets beta in $\mathbb{B}_b$. We want to ensure that Alice transfers her assets a in $\mathbb{B}_a$ to Bob if and only if Bob transfers his asset b to Alice in $\mathbb{B}_b$.

There must be only two possible outcomes

1) Bob owns a in $\mathbb{B}_a$ and Alice owns b in $\mathbb{B}_b$ (swaps of assets)

2) Alice owns a in $\mathbb{B}_a$ and Bob owns b in $\mathbb{B}_b$ (assets refund)

Additionally, we want that the following properties hold:

Transparent: an observer should not distinguish a transfer executed as part of an atomic swap from a standard transfer in such ledger.
Trustless: there are no trust assumption on any party executing the protocol.
Scriptless: the protocol does not require any script capabilities on both of the ledgers.

2 Existing works

Type	Trustless	Transparent	Scriptless
TEE [BJZLZBDJ'17]		✓	✓
HTLC [Gugger '20]	✓		
Relays [LMP'21]	✓		
Universal [MTS'20]	✓	✓	✓

Table 1: Properties comparison of different atomic swaps protocol

2.1 TEE

2.2 Hashed Timelock Contract (HTLC)

Closest solution to a universal protocol for atomic swaps, implemented by large majority of trustless exchange protocols. Using HTLC as building block, an atomic swap can be constructed as follows: Alice chooses r, computes h = H(r), transfers a into an HTLC(a, h, t, A, B) in $\mathbb{B}_a$ and sends h, t to Bob.

Bob cannot claim the HTLC yet because r is only known by Alice. Bob finishes the setup of the exchange by choosing a time $t' < t$ and transferring his b assets into an HTLC(b, h, t', b, a) in $\mathbb{B}_b$. Thus either

(1) Alice claims HTLC in $\mathbb{B}_b$, effectively revealing r to Bob (and anyone observing $\mathbb{B}_b$) before t'. Bob can then use r to claim the HTLC in $\mathbb{B}_a$;

(2) Alice does not claim HTLC in $\mathbb{B}_b$ and Bob cannot claim the HTLC in $\mathbb{B}_a$. After the timeout they get the funds back.

These cross-chain swaps are deployed in practice but have high execution costs and transactions sizes, in addition to following problems.

(1) Both ledgers must support compatible hash functions within their scripting language. In fact both should support the same hash function and each ledger must use the same number of bits to represent it. An observer can use the same h value at two ledgers and link HTLC as part of the same swap. Monero, Mimblewimble, Ripple, Stellar and Zcash do not support the computation of the HTLC contract

(2) Presence of timelock. Adding timelock functionality is at odds with privacy because (i) if implemented naively, makes time-locked transfers easy to distinguish from normal ones (ii) including it can conflict other privacy operations available in the ledger (iii) adds a non trivial overhead to computation and storage of the ledger if implemented correctly

(3) swap is restricted to only two HTLC, one per ledger, and thus to exchange of a assets in $\mathbb{B}_a$ and b assets in $\mathbb{B}_b$ (no multi coin swap).

2.3 Relays

Another strategy to achieve atomic swaps relies on relays. Relays are abstractions (in general a smart contract or a script) hosted on some chain $\mathbb{B}_a$ that has light client like verification capabilities over chain $\mathbb{B}_b$. For each new block appended to chain $\mathbb{B}_a$, the block header is passed on to the relay on chain $\mathbb{B}_b$.
The relay itself implements the standard verification procedure of chain $\mathbb{B}_a$s consensus algorithm and can therefore verify the validity of the block. Once the proof of work has been verified, in the case of a Proof of Work (or PoW) blockchain, or the two-thirds of validators signatures, in the case of a Byzantine Fault Tolerant (or BFT) blockchain, it is possible to verify any transaction of chain $\mathbb{B}_a$ from chain $\mathbb{B}_b$. With light client like verification capabilities of chain $\mathbb{B}_a$ from chain $\mathbb{B}_b$, we can imagine the following scenario. Bob has X assets of chain $\mathbb{B}_b$. He is willing to exchange them for Y assets of chain $\mathbb{B}_a$.
Bob sets up a smart contract SC1 and locks his assets in it (1). This smart contract SC1 is set to release

the assets to anyone providing the proof that they made a payment of Y assets of chain $\mathbb{B}_a$ to Bobs address. Alice, who is interested in this trade, transfers Y assets to Bobs address (2). She retrieves the transaction hash tx and provide it to SC1 (3). SC1 calls the relay and asks for verification of transaction tx (4). The relay verifies that the transfer has taken place and if so, returns ok to SC1 (5). On receiving the answer from the relay, SC1 transfers the X assets of $\mathbb{B}_b$ to Alices address.

2.4 Verifiable Timed Signatures (VTS) / Universal Atomic Swaps

The central challenge that the protocol needs to address is in ensuring atomicity of the swap even in the presence of malicious parties, which is guaranteed by the HTLC-based protocols. Drawing inspiration from that approach, an immediate barrier is the absence of scripting language does not allow to set time-outs on transactions.

A timeout t for a transaction tx means that the transaction is accepted by the nodes in the network only after time t has expired. Typically this is implemented by expressing t in terms of a block number and leveraging a timelock script which is explicitly included in the transaction.

To simulate such functionality without using any on-chain script, the main leverage will be VTS, which lets a user generate a timed commitment C of a signature σ on a message m under a public key pk. The commitment C must hide the signature σ for time T. At the same time, the committer also generates a proof π that proves that the commitment C contains a valid signature σ. This guarantees that σ can be publicly recovered in time T by anyone who solves the computational puzzle.

To give an intuition, imagine Alice and Bob sharing an address $\mathsf{pk}(ab)$ where each party owns a share of the corresponding secret key. Before sending the funds to $\mathsf{pk}(ab)$, Alice and Bob sign a refund transaction $\mathsf{tx}(\mathsf{rfnd})$ that transfers all funds from $\mathsf{pk}(ab)$ back to Alice, in such a way that only Bob learns the signatures. Bob then generates a VTS on this refund signature and provides Alice the commitment C and proof π.

Note that the parties must make conservative estimates of each others computational power in force opening the VTD commitments. This is to prevent scenarios where Alice or Bob with a powerful machine force opens its VTD commitments earlier than expected in terms of real time. The party could potentially steal the coins of the other party during the swap lock or swap complete phase. In particular, the parties must ensure that Δ (such that $\mathrm{T0} = \mathrm{T1} + \Delta$) is large enough such that it tolerates the time differences to open the VTD commitments.

3 Protocol improvement proposal

Parties U_0 and U_1 hold assets a on blockchain $\mathbb{A}$ and assets b on chain $\mathbb{B}$ respectively.
We define with $\mathsf{amnt_a}$ and $\mathsf{amnt_b}$ the amount of the assets the parties agreed to swap before starting the protocol.
In the protocol definition, variables and functions that are blockchain-specific (unless clear from context) are denoted with a subscript, example for a publick key on chain $\mathbb{B}$ we denote $\mathsf{pk}_{(\mathbb{B})}$.

Informally, we want the atomicity security property: parties should either both end up with the original funds in a wallet they own (refund case) or they should own the agreed assets to swap on their respective target wallets.

We define the following oracles to interact with the blockchains.

- $\mathsf{PubTx}_{(\mathbb{A})}(\sigma_{\mathsf{tx}}, \mathsf{tx})$ publish the transaction tx with signature σ_{tx} on chain $\mathbb{A}$
- $\mathsf{InitTx}_{(\mathbb{A})}(\mathsf{pk_{tx}}, \mathsf{pk_{rx}}, \mathsf{amnt})$ create an unsigned transaction paying amnt from $\mathsf{pk_{tx}}$ to $\mathsf{pk_{rx}}$ on chain $\mathbb{A}$
- $\mathsf{WatchTx}_{(\mathbb{A})}(\mathsf{tx})$ wait for the transaction tx to be confirmed on chain $\mathbb{A}$
- $\mathsf{GetBal}_{(\mathbb{A})}(\mathsf{pk})$ get the balance of assets held by pk
- $\mathsf{GetSig}_{(\mathbb{A})}(\mathsf{pk})$ get the signature σ_{tx} of the latest transaction in pk's record on chain $\mathbb{A}$

U_1 starts counting the timeout from the moment they send the VTD commitment to U_0, and respectively U_0 starts counting down from the moment they receive it.

Global input $(\mathbb{G}, [1], q, T, \mathsf{amnt_a}, \mathsf{amnt_b}, \mathbb{A}, \mathbb{B})$

$(\mathsf{sk_{frz0}}, \mathsf{pk_{frz}}) \leftarrow$ **wait** $\Gamma_{\mathsf{KeyGen}_{(\mathbb{A})}}(\mathbb{G}, [1], q)$
$(C, \pi) \leftarrow$ **wait** $\mathsf{receive}(U_1)$
if $\Pi_{\mathsf{VTD}}.\mathsf{Verify}([\mathsf{pk_{frz}}] - [\mathsf{sk_{frz0}}], C, \pi) \neq 1$
  return $\perp$
$\mathsf{res} \leftarrow$ **select** {
  wait {
    $\Pi_{\mathsf{VTD}}.\mathsf{ForceOp}(C)$
  }
  wait {
    $(\mathsf{pk_{swp}}, \mathsf{sk_{swp}}) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KeyGen}_{(\mathbb{B})}(1^\lambda)$
    $\mathsf{tx_{frz}} \leftarrow \mathsf{InitTx}_{(\mathbb{A})}(\mathsf{pk_{init}}, \mathsf{pk_{frz}}, \mathsf{amnt_a})$
    $\sigma_{\mathsf{frz}} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}_{(\mathbb{A})}(\mathsf{sk_{init}}, \mathsf{tx_{frz}})$
    $\mathsf{PubTx}_{(\mathbb{A})}(\sigma_{\mathsf{frz}}, \mathsf{tx_{frz}})$
    $\mathsf{pk_{init(\mathbb{B})}} \leftarrow \mathsf{receive}(U_1)$
    $\mathsf{tx_{swp}} \leftarrow \mathsf{InitTx}_{(\mathbb{B})}(\mathsf{pk_{init}}, \mathsf{pk_{swp}}, \mathsf{amnt_b})$
    $\sigma_{\mathsf{swp}(\mathbb{B})} \leftarrow \Gamma_{\mathsf{Swap}}(\mathsf{sk_{frz0}}, \mathsf{tx_{swp}})$
    $\mathsf{PubTx}_{(\mathbb{B})}(\sigma_{\mathsf{swp}}, \mathsf{tx_{swp}})$
    $\mathsf{send}(U_1)$
  }
}
if $\mathsf{res} \neq 1$
  $\mathsf{sk_{frz}} := \mathsf{sk_{frz0}} + \mathsf{res}$
  $\mathsf{tx_{rfnd}} \leftarrow$ **wait** $\mathsf{InitTx}_{(\mathbb{A})}(\mathsf{pk_{frz}}, \mathsf{pk_{init}}, \mathsf{amnt_a})$
  $\sigma_{\mathsf{rfnd}} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}_{(\mathbb{A})}(\mathsf{sk_{frz}}, \mathsf{tx_{rfnd}})$
  wait $\mathsf{PubTx}_{(\mathbb{A})}(\sigma_{\mathsf{rfnd}}, \mathsf{tx_{rfnd}}, \mathbb{A})$

Global input $(\mathbb{G}, G, q, T, \mathsf{amnt_a}, \mathsf{amnt_b}, \mathbb{A}, \mathbb{B})$

$(\mathsf{sk_{frz1}}, \mathsf{pk_{frz}}) \leftarrow$ **wait** $\Gamma_{\mathsf{KeyGen}_{(\mathbb{A})}}(\mathbb{G}, [1], q)$
$(C, \pi) \leftarrow \Pi_{\mathsf{VTD}}.\mathsf{Commit}(\mathsf{sk_{frz1}}, T)$
$\mathsf{send}(U_0, (C, \pi))$
$\mathsf{res} \leftarrow$ **select** {
  wait {
    $\mathsf{timeout}(T/2)$
  }
  wait {
    do $\mathsf{bal} \leftarrow \mathsf{GetBal}_{(\mathbb{A})}(\mathsf{pk_{frz}})$
    while $\mathsf{bal} \neq \mathsf{amnt_a}$
    $\mathsf{send}(U_1, \mathsf{pk_{init}})$
    $lk \leftarrow \Gamma_{\mathsf{Swap}}(\mathsf{sk_{frz1}}, \mathsf{sk_{init(\mathbb{B})}})$
    $\mathsf{receive}(U_0)$
    $\sigma_{\mathsf{lk}} \leftarrow \mathsf{GetSig}_{(\mathbb{B})}(\mathsf{pk_{init}})$
    $\mathsf{sk_{frz}} \leftarrow (lk \oplus \sigma_{lk}) + \mathsf{sk_{frz1}}$
    $\mathsf{tx_{swp}} \leftarrow \mathsf{InitTx}_{(\mathbb{A})}(\mathsf{pk_{frz}}, \mathsf{pk_{swp}}, \mathsf{amnt_a})$
    $\sigma_{\mathsf{swp}} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}_{(\mathbb{A})}(\mathsf{sk_{frz}}, \mathsf{tx_{rfnd}})$
    $\mathsf{PubTx}_{(\mathbb{A})}(\sigma_{\mathsf{swp}}, \mathsf{tx_{swp}})$
  }
}
if $\mathsf{res} \neq 1 \wedge lk \neq \perp$
  $(\mathsf{pk_{rfnd}}, \mathsf{sk_{rfnd}}) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KeyGen}_{(\mathbb{B})}(1^\lambda)$
  $\mathsf{tx_{rfnd}} \leftarrow$ **wait** $\mathsf{InitTx}_{(\mathbb{B})}(\mathsf{pk_{init}}, \mathsf{pk_{rfnd}}, \mathsf{amnt_b})$
  $\sigma_{\mathsf{rfnd}} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}_{(\mathbb{B})}(\mathsf{sk_{init}}, \mathsf{tx_{rfnd}})$
  wait $\mathsf{PubTx}(\mathbb{B})(\sigma_{\mathsf{rfnd}}, \mathsf{tx_{rfnd}})$

Figure 1: Full protocol execution for U_0 and U_1, respectively left and right (alternative syntax)

4 Proof sketch

4.1 Party U_0

Informally, we want that the atomic property holds: after the protocol run either U_0 ends up with $\mathsf{amnt_b}$ on $\mathsf{pk}_{\mathsf{swp}(\mathbb{B})}$ in case of a successful swap or with $\mathsf{amnt_a}$ on $\mathsf{pk}_{\mathsf{init}(\mathbb{A})}$, in case the swap was aborted or refunded.
We consider an active adversary over the communication channel with U_1 that can also corrupt U_1. We assume liveness and correctness for the blockchains.

By general 2PC's privacy property, $\mathsf{sk_{frz1}}$ is only known to U_0.
By the $\Pi_{\mathsf{VTD}}.\mathsf{Verify}$ algorithm we have that $\Pi_{\mathsf{VTD}}.\mathsf{Verify}(\mathsf{pk_{frz}} - [\mathsf{sk_{frz0}}], C, \pi) = 1$ if and only if the value embedded x in the commitmemt C satisfies $[x] = \mathsf{pk_{frz}} - [\mathsf{sk_{frz0}}]$.
Assuming a group based Π_{DS} with $\mathsf{pk} := [\mathsf{sk}]$, we have that $\mathsf{sk_{frz}} := \mathsf{sk_{frz0}} + \mathsf{sk_{frz1}}$ and thus $[\mathsf{sk_{frz0}} + \mathsf{sk_{frz1}}] - [\mathsf{sk_{frz0}}] = [\mathsf{sk_{frz1}}] = [x]$. Hence U_0 proceeds to swap the assets if and only if the value committed in C is $\mathsf{sk_{frz1}}$ and $\mathsf{pk_{frz}} = [\mathsf{sk_{frz0}} + \mathsf{sk_{frz1}}]$.
Note that by the VTD's soundness property the $\mathsf{ForceOp}$ algorithm will produce the committed dlog value x in time $\mathbf{T}$, thus U_0 will be able to retrieve $\mathsf{sk_{frz1}}$ after time T and sign a refund transaction.
Now U_0 transfer the funds to $\mathsf{pk_{frz}}$ and proceeds to generate a new keypair $(\mathsf{pk_{swp}}, \mathsf{sk_{swp}})$ secret to the outside world.
When calling the 2PC protocol $\Gamma_{\mathsf{Swap}}(\mathsf{sk_{frz0}}, \mathsf{tx_{swp}})$, note that the inputs are again secret by 2PC's privacy property.
If $\sigma_{\mathsf{swp}(\mathbb{B})}$ is invalid, $\mathsf{PubTx}_{(\mathbb{B})}(\sigma_{\mathsf{swp}}, \mathsf{tx_{swp}})$ will fail and U_0 will wait until $\mathsf{ForceOp}$ completes to compute $\mathsf{sk_{frz}}$ and sign the refund transaction with it, ending up with $\mathsf{amnt_a}$ on $\mathsf{pk}_{\mathsf{init}(\mathbb{A})}$.
Otherwise, the swap will complete successfully, and U_0 ends up with $\mathsf{amnt_b}$ on $\mathsf{pk}_{\mathsf{swp}(\mathbb{B})}$.
An adversary has no information about $\mathsf{sk_{frz0}}$ and $(\mathsf{pk_{swp}}, \mathsf{sk_{swp}})$, thus it cannot sign transaction from $\mathsf{pk_{frz}}$ or compute a valid signature for $\mathsf{pk_{swp}}$, and is thus unable to retrieve information about $\mathsf{sk_{frz0}}$ from lk.

4.2 Party U_1

After the protocol run either U_1 ends up with $\mathsf{amnt_a}$ on $\mathsf{pk}_{\mathsf{swp}(\mathbb{A})}$ in case of a successful swap, with $\mathsf{amnt_b}$ on $\mathsf{pk}_{\mathsf{rfnd}(\mathbb{B})}$ in case the swap was refunded or $\mathsf{amnt_b}$ on $\mathsf{pk}_{\mathsf{init}(\mathbb{B})}$ if the swap was aborted.
We consider an active adversary over the communication channel with U_0 that can also corrupt U_0. We assume liveness and correctness for the blockchains.

Note that before calling $\Gamma_{\mathsf{Swap}}(\mathsf{sk_{frz1}}, \mathsf{sk}_{\mathsf{init}(\mathbb{B})})$, U_1 is in control of their assets on $\mathsf{pk}_{\mathsf{init}(\mathbb{B})}$ if it were to abort, and by general 2PC's privacy property both inputs are private.
Also note that U_1 waits until the funds $\mathsf{amnt_a}$ have been transferred to $\mathsf{pk_{frz}}$ before proceeding with Γ_{Swap}.
An adversary can only get the signature $\sigma_{swp(\mathbb{B})}$ if and only if it provided the correct $\mathsf{sk_{frz0}}$ and thus U_1 has received by correctness of 2PC $lk := \sigma_{swp(\mathbb{B})} \oplus \mathsf{sk_{frz0}}$.
If U_1 is unable to retrieve $\sigma_{swp(\mathbb{B})}$ before $T/2$, it proceeds to move the funds from $\mathsf{pk_{init}}$ to a newly generated $\mathsf{pk_{rfnd}}$, and thus an adversary holding $\sigma_{swp(\mathbb{B})}$ will be unable to get a transaction accepted by the correctness property of the blockchain (otherwise we occur in a double spending), so we end up with $\mathsf{amnt_b}$ on $\mathsf{pk}_{\mathsf{rfnd}(\mathbb{B})}$.

If the transaction with signature $\sigma_{swp(\mathbb{B})}$ gets posted on $\mathbb{B}$, then U_1 will be able to retrieve $\mathsf{sk_{frz0}}$ by the above argument, and thus compute $\mathsf{sk_{frz}}$ and sign the swap transaction with it, ending up with $\mathsf{amnt_a}$ on $\mathsf{pk}_{\mathsf{swp}(\mathbb{A})}$.

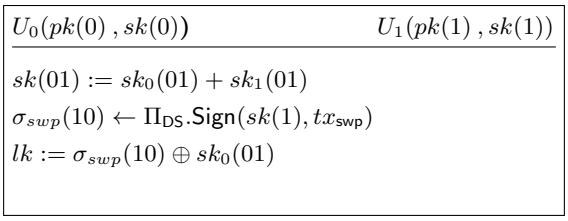

Figure 2: Protocol definition of 2PC Γ_{Swap}