Scriptless, atomic coinswap using cut-and-choose

version 0.1

1 Intro

The following is a method of achieving coinswap that aims to be scriptless, atomic and unlinkable both in the case of a successful swap or the use of backouts, extending the similar construction of the half-scriptless swap[1]. We assume that the blockchains in use by participants of the swap support timelocking mechanism for transactions; For Bitcoin and most Bitcoin-derived altcoins the `nLocktime` field of the transaction will be used by locking backout transactions to an absolute block height. Described hereafter is the protocol where the public keys are elliptic curve points. The signature algorithm used for signing and validation of transaction signatures is irrelevant to the protocol and can be different between the two chains.

As with any coinswap protocol, one party initiates the process and a second party might accept her offer for a swap. Timelocks are used to produce signed **Backout** transactions for both parties from their own funding for the swap in cases when the swap halts. As such, the two parties will know each other's funding UTXOs and backout pubkeys at the setup stage of the protocol. Ways to achieve those properly exist (using a discrete log equivalence proof for example), but are out of scope of this document. To achieve security in the protocol, a cut-and-choose [2] game, much like the one taking place in the Tumblebit [3] protocol will be taking place afterwhich both parties will have sufficient guarantee that cheating is extremely unlikely.

For notation, we will use lowercase letters for discrete logarithms and plain integers, uppercase for curve points and signatures, and `teletype` for scriptPubKeys, transactions and UTXOs. Subscript is used to convey relationships between parameters, or as their numerical index in a set. We will assume that Alice initates the process and Bob accepts her offer to swap.

2 Setup

The protocol begins with a setup stage where the parties exchange public keys for two multisig scriptpubkeys, UTXOs, their agreed locktimes and backout public keys.

1. Exchange UTXOs
   - Alice : UA
   - Bob : UB
2. Exchange public keys
   - Alice : $A1, A2$
   - Bob : $B1, B2$
3. Exchange backout scriptPubKeys
   - Alice : A_{back}
   - Bob : B_{back}
4. Agree on backout locktimes
   - Alice : L2
   - Bob : L1

Both parties now compute a transaction TX1 which is Bob's funding for the swap. They begin creating a 2-of-2 multisig scriptpubkey, and Bob's UTXO UB will be its funding.

- `scr1` : `2 A1 B1 2 CHECKMULTISIG`
- TX1 : UB $\rightarrow$ scr1

Alice now sends to Bob a signature by her key $A1$ for his backout from TX1, timelocked to L1.

- $Sig_{A1}(\mathsf{TX1} \rightarrow B_{back})$

Bob checks Alice's signature, and if it's valid, the parties continue to the next phase.

3 Game

In this stage, the parties play a cut-and-choose game. At its end Alice will have knowledge of a signature by Bob's $B1$, for a transaction that is spending $\mathsf{TX1}$ and pays her to a scriptPubKey A_{swap}, while Bob will have knowledge of a public key T and a guarantee that if his signature is used in a transaction, he will then learn the discrete log for that pubkey.

During Bob's signing phase, he is only given a **sighash** $h1$ to sign for. It's important for Alice's security that Bob is not able to find the preimage for for her $h1$ except by her using his signature in the transaction. She must select her swap pubkey at random as it acts as the only information Bob is missing to know the preimage.

Bob's security with regards to "blindly" signing a random hash comes from his choice of $B1$, which is a pubkey not used by him in any other UTXOs, as well as the fact that the funding for Alice's payment, $\mathsf{TX1}$ itself, has not been yet been signed for by him and relayed to the network.

The purpose of disclosing T's discrete log (DL) to Bob is to allow him to redeem Alice's funding of the swap. T in turn is used in a multisig scriptPubKey funded by Alice's UTXO. Knowledge of T's DL will allow him to redeem the output for his own swap scriptPubKey B_{swap}.

Atomicity in the protocol is achieved by conditioning Alice's use of Bob's signature in a transaction that pays her from $\mathsf{TX1}$, and by choosing a component of T's DL to be some **non-sighash** hash $h0$ of that transaction, to enable Bob the spending of her swap funds by providing a signature for T.

For Bob to be the sole owner of the complete DL for T, as Alice herself must not be able to sign for it, T is defined to be a tweak by $h0$ of a yet undetermined ephemeral public key $B3$ of Bob's.

Technically, for a signature by $B1$ that Alice wishes to receive from Bob for a transaction ($\mathsf{TX1} \rightarrow A_{swap}$), she will provide him with a pair of $(h1, T)$. Here *H0()* is a hash function **different** from the sighash function, and `*` is ecmul :

- $h0 = \mathit{H0}(\mathsf{TX1} \rightarrow A_{swap})$
- $T = h0 * G + B3$
- $h1 = \mathit{sighash}(TX1 \rightarrow A_{swap})$

By itself, this exchange is insecure towards Bob. He has no way of knowing whether Alice correctly tweaked $B3$ by $h0$ to produce T, and if $h0$ is anything else but a secret hash of her ($\mathsf{TX1} \rightarrow A_{swap}$), Bob learns nothing of value from the the transaction being used on the network.

By overlaying cut-and-choose here we enable Bob to test Alice's correct execution of the protocol many times over, enough to guarantee that cheating is

highly improbable. Bob will feed Alice a mixed set made of n of his pubkeys, and m NUMS pubkeys which are a hash of some random data chosen by Bob, and *coerced to point*.
At first, Alice can't tell which of the keys in the set are Bob's real keys and which are the NUMS points. She will generate $(n+m)$ different A_{swap} public keys and run her step of the protocol against each of the pubkeys provided by Bob, returning a different $(h1_i, T_i)$ pair for each.

Bob starts by computing a set of n ephemeral keys for himself :

```
for $i$ in $n$:
    $b3_i = random()$
    $B3_i = b3_i * G$
```

And a set of m NUMS public keys. Here *hash()* is a cryptographic hash function and | is concatenation :

```
for $i$ in $m$:
    $f_i = random()$
    $F_i = coerce\text{-}to\text{-}point(f_i)$
```

He then shuffles all $(n+m)$ pubkeys and creates a commitment c to the ordering of the shuffled set by taking :

- $\{B3'_1..B3'_{n+m}\} = shuffle(\{B3_1..B3_n\}, \{F..1, F_m\})$
- $w = random()$
- $c = hash(w \mid \text{shuffled-set-order})$

And sends the set of $B3'_i$ pubkeys along with the commitment c to Alice.

Alice first prepares a set of $(n+m)$ of her own public keys, to be used as her A_{swap} scriptPubKeys. For each of her pubkeys, she then creates a transaction redeeming TX1 (Bob's funding to the swap), and its sighash. Each of Bob's provided pubkeys will then be tweaked by this transaction's secret hash, creating $(n+m)$ different values of a public point T :

```
for $i$ in $(n+m)$:
    $h0_i = H0(TX1- > A_{swap_i})$
    $T_i = h0_i * G + B3'_i$
    $h1_i = sighash(\mathsf{TX1} \rightarrow A_{swap_i})$
```

She then sends the set of $(n+m)$ pairs $(h1_i, T_i)$ to Bob.

Bob tests Alice for proper execution of the protocol. He sends her his previously secret value w, all random f_i values, and reveals the order of his shuffled set of real and NUMS pubkeys. Alice is then able to check that the commitment c is in fact a commitment to the shuffled $B3'_i$ pubkeys :

```
$c' = hash(w \mid \text{shuffled-set-order})$
$c' \overset{?}{=} c$
for $i$ in $m$:
    $F'_i = coerce\text{-}to\text{-}point(f'_i)$
    $F'_i \overset{?}{=} F_i$
```

And if this check passes, she sends back to Bob a set of her secret ($\mathsf{TX1} \to A_{swap_i}$) for all m NUMS pubkey indexes. Bob is able check that Alice executed the protcol faithfuly for these pubkeys by comparing his results of the same steps against the set provided by her :

```
for $i$ in $m$:
    $h0'_i = H0(\mathsf{TX1} \to A_{swap_i})$
    $T'_i = h0'_i * G + F_i$
    $T'_i \overset{?}{=} T_i$
    $h1'_i = sighash(\mathsf{TX1} \to A_{swap_i})$
    $h1'_i \overset{?}{=} h1_i$
```

If all checks pass, Bob can be reasonably sure that Alice has not cheated and that the remaining n pairs $(h1_i, T_i)$ from her set are properly computed sighashes and tweaked $B3_i$ pubkeys. Bob now picks an index j from these n pairs at random, $(h1_j, T_j)$, tells Alice his choice of j, and they continue to the next stage which is the swap itself.

We see that by relying on cut-and-choose strategy for security, Alice's chance of successfully cheating is at the order of $1/\binom{n+m}{n}$.

- As this is done on a bitcoin-like blockchain, *sighash* is defined as *SHA256d*
- We define *H0()* to be *SHA256*. Coincidentally, this means that the result of *H0()* is a midstate of *sighash*.
- By aiming for a security parameter of 2^{128}, we choose $(n == m == 65)$

4 Swap

Both parties compute Alice's funding for the swap, TX2. They begin by creating a 2-of-3 multisig scriptPubKey, where the signers can be either two of $A2$, $B2$ and T_j. Alice will be paying into this script from her UTXO UA.

- $\texttt{scr2}$: `2 A2 B2` $\texttt{T}_j$ `3 CHECKMULTISIG`
- $\mathsf{TX2}$: $\mathsf{UA} \rightarrow \texttt{scr2}$

Bob now sends two signatures to Alice. The first is a signature by his key $B2$ for her backout from $\mathsf{TX2}$, timelocked to $\mathsf{L2}$, and the second signature is by his key $B1$ for the sighash $h1_j$.

- $Sig_{B2}(\mathsf{TX2} \rightarrow A_{back})$
- $Sig_{B1}(h1)$

Alice validates Bob's signatures, and if both are valid, she broadcasts $\mathsf{TX2}$ to the network. Bob waits for $\mathsf{TX2}$ to be mined, and then broadcasts $\mathsf{TX1}$. Alice waits for $\mathsf{TX1}$ to be mined and buried under a few more blocks.
Finally she can use Bob's $Sig_{B1}(h1)$ to sign a spend of $(\mathsf{TX1} \rightarrow A_{swap_j})$, redeeming Bob's funding to the swap.
As Bob sees his funding being spent, he learns the transaction $(\mathsf{TX1} \rightarrow A_{swap_j})$ itself and can compute the DL of T.

- $h0_j = \mathit{H0}(\mathsf{TX1} \rightarrow A_{swap_j})$
- $t_j = h0_j + b3_j$

He is then able to sign a spend of $\mathsf{TX2} \rightarrow B_{swap}$ using two of the three pubkeys authorized to do so, $B2$ and T_j, and redeem Alice's funding to the swap.

5 Summary

Presented here is a method of achieving atomic, scriptless coinswap. Compared to to other atomic coinswap protocols like the TierNolan swap [4], ours is unlinkable to observers. No hashes and preimages are revealed on chain which can connect the two transactions in the swap. Even if backouts are used by the participants, unlike in the CoinSwapCS [5] protocol which *does* remain unlinkable to observers in case of a succesful swap but not when it halts.
Privacy in case of backouts is kept by making use of the transaction's `nLocktime` field to timelock the backouts. This field is set by default in transactions created by wallet software such as Bitcoin Core, which means that a backout transaction from a halted swap attempt can be indistinguishable from a successful one if it is used exactly at its unlocking block height.

With regards to adaptor signatures based, scriptless coinswaps [6], this cut-and-choose based swap works around the specifics of signature algorithm selection by conditioning the disclosure of a discrete logarithm to a hash preimage. By making use of a transaction output's scriptPubKey and hiding it as part of the sighash itself, we also work around the need for revealing link-able hash values in Script. A signature used with a sighash on chain explicitly reveals the transaction itself, including the outputs.

As for more advanced systems such as Tumblebit [3] and blind scriptless swap [7], an open question remains: whether this protocol can be extended for use in a client-tumbler model where the tumbler itself can't link coins swapped between participants, and as a result lower the interactivity required in the protocol.

References

[1] waxwing,
The half-scriptless swap,
`https://joinmarket.me/blog/blog/the-half-scriptless-swap/`

[2] Yehuda Lindell,
Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries,
`https://eprint.iacr.org/2013/079.pdf`

[3] Ethan Heilman and Leen Alshenibr and Foteini Baldimtsi and Alessandra Scafuro and Sharon Goldberg,
TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub,
`https://eprint.iacr.org/2016/575.pdf`

[4] TierNolan,
Alt chains and atomic transfers
`https://en.bitcoin.it/wiki/Atomic_cross-chain_trading`

[5] waxwing,
CoinSwapCS,
`https://github.com/AdamISZ/CoinSwapCS/`

[6] Andrew Poelstra,
Adaptor Signatures and Atomic Swaps from Scriptless Scripts,
`https://github.com/apoelstra/scriptless-scripts/blob/master/md/atomic-swap.md`

[7] Jonas Nick,
Partially Blind Atomic Swap Using Adaptor Signatures,
textttthttps://github.com/jonasnick/scriptless-scripts/blob/46eb506dbfa51295853bc285ce667eeb47fe35b9/md/partially-blind-swap.md