Taprootized Atomic Swaps

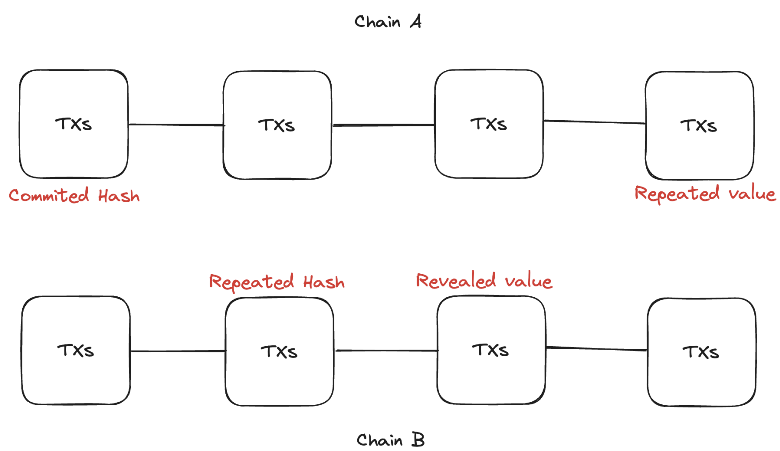
Cross-chain, Untraceable, Trustless

Distributed Lab, Jan 2024
Version 1.1

Abstract. Taprootized Atomic Swaps (TAS) is an extension for Atomic Swaps that presumes the untraceability of transactions related to a particular swap. Based on Schnorr signatures, Taproot technology, and zero-knowledge proofs, the taprootized atomic swaps hide swap transactions under regular payments.

Intro

Atomic swap is an incredible approach to cross-chain exchanges without mediators. However, one of the disadvantages of its implementation in the classical form is the "digital trail" — any party can make a matching between transactions in the blockchains in which the exchange took place and find out both the participants in the exchange and the proportion in which assets were exchanged.

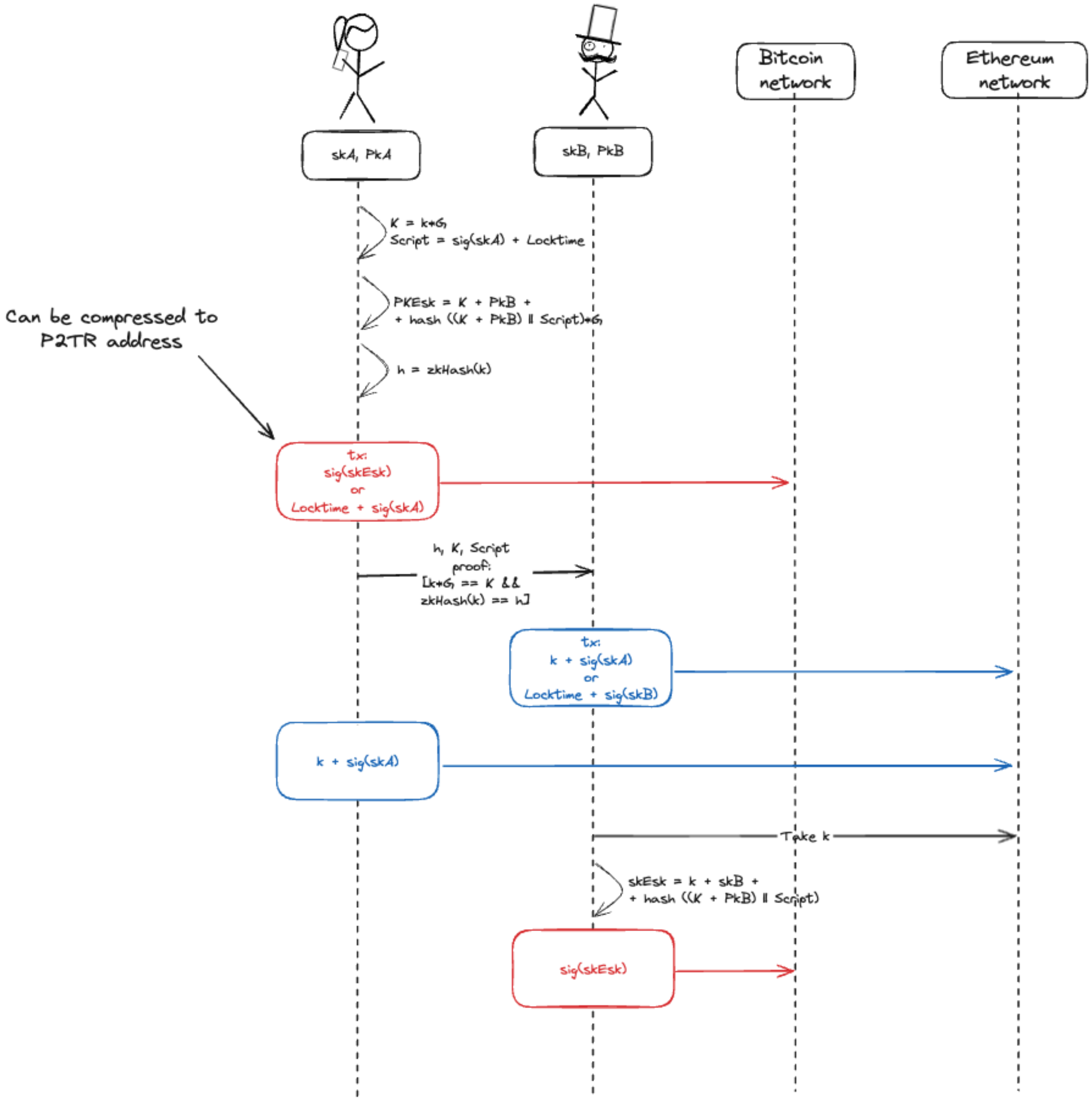


On the other hand, atomic swaps is a technology that initially assumed the involvement of only two parties and a "mathematical contract" between them directly. That is, an ideal exchange presupposes 2 conditions:

1) Only counterparties participate in the exchange (works by default)
2) Only counterparties know about the fact of the exchange (it would be nice to ensure)

This paper will provide a concept of taprootized atomic swaps that allow hiding the swap's very fact. To an external auditor, transactions to initiate and execute atomic swaps will be indistinguishable from regular Bitcoin payments. In the other accounting system involved in the transfer, more information is disclosed (the fact of exchange can be traced). Still, it is impossible to link this to the corresponding Bitcoin transactions (without additional context from the involved parties).

Protocol



The protocol includes the following steps:

1. Alice (sk_A, PK_A) and Bob (sk_B, PK_B) have their keypairs and know each other's public keys.
2. Alice generates a random $\boldsymbol{k}$ and calculates the public value $\boldsymbol{K} = \boldsymbol{k} * \boldsymbol{G}$
3. Alice forms the alternative spending path $\boldsymbol{Script} = \boldsymbol{sig(sk_A)} + \boldsymbol{Locktime}$ in the form of Bitcoin Script
4. Alice calculates an escrow public key as $\boldsymbol{PK_{Esc}} = \boldsymbol{K} + \boldsymbol{PK_B} + \boldsymbol{hash((K + PK_B) \mid\mid Script) * G}$ (here, escrow is just a public key, formed using Taproot technology)
   a. The signature $\boldsymbol{sig(sk_{Esc})}$, verified by the $\boldsymbol{PK_{Esc}}$, can be generated only with the knowledge of $\boldsymbol{k}$, $\boldsymbol{sk_B}$, and $\boldsymbol{Script}$
5. Alice calculates the $\boldsymbol{h}$ as a hash value of $\boldsymbol{k}$ (zk-friendly hash function is recommended to use)
6. Alice forms the funding transactions with the following conditions of how it can be spent:

    a. Signature of $\boldsymbol{sk_{Esc}}$: Bob, with knowledge of $\boldsymbol{k}$ and $\boldsymbol{sk_B}$ can spend the output
    b. Signature of $\boldsymbol{sk_A}$ + Locktime: Alice, with knowledge of $\boldsymbol{sk_A}$ can spend the output, but only after some point in time $\boldsymbol{t_1}$ (it's the ***Script*** itself)
7. Alice sends the transaction to the Bitcoin network
8. Alice generates the zero-knowledge ***proof*** that includes (for the same $\boldsymbol{k}$):
    a. The proof of knowledge of $\boldsymbol{k}$ that satisfies $\boldsymbol{k*G == K}$
    b. The proof of knowledge of $\boldsymbol{k}$ that satisfies $\boldsymbol{zkHash(k) == h}$
9. Alice provides the set of data to Bob:
    a. $\boldsymbol{h}$
    b. $\boldsymbol{K}$
    c. ***Script***
    d. ***proof***
10. Bob calculates $\boldsymbol{PK_{Esc}}$ as $\boldsymbol{K + PK_B + hash((K + PK_B) \,||\, Script) * G}$ and finds the transaction locked BTC (verifies it exists). Then Bob performs the following verifications:
    a. Verifies that Alice knows $\boldsymbol{k}$ that satisfies $\boldsymbol{k*G == K}$ and $\boldsymbol{zkHash(k) == h}$, it means that Bob can access the output $\boldsymbol{PK_{Esc}}$ if he receives $\boldsymbol{k}$
    b. Verifies that the ***Script*** is correct and includes only the required alternative path.
11. If verifications are passed, Bob forms the transaction that locks his funds on the following conditions:
    a. Publishing of $\boldsymbol{k}$ and the signature of $\boldsymbol{sk_A}$: only Alice can spend it if she reveals $\boldsymbol{k}$ (hash preimage)
    b. Signature of $\boldsymbol{sk_B}$ + Locktime: Bob, with knowledge of $\boldsymbol{sk_B}$, can spend the output, but only after some point in time $\boldsymbol{t_2}$
12. Bob sends the transaction to the Ethereum network (or any other that supports ***zkHash()***)
13. Alice sees the locking conditions defined by Bob and publishes the $\boldsymbol{k}$ together with the signature generated by her $\boldsymbol{sk_A}$. As a result — Alice spent funds locked by Bob.
    a. If Alice doesn't publish the relevant $\boldsymbol{k}$, Bob can return funds after $\boldsymbol{t_2}$ is reached
14. If Alice publishes a transaction with $\boldsymbol{k}$, Bob can recognize it and extract the $\boldsymbol{k}$ value
15. Bob calculates the needed $\boldsymbol{sk_{Esc}}$ as $\boldsymbol{sk_{Esc} = k + sk_B + hash((K + PK_B) \,||\, Script)}$
16. Bob sends the transaction with the signature generated by the $\boldsymbol{sk_{Esc}}$ and spends funds locked by Alice

Implementation notes

1. As an approach for escrow public key forming, the usage of MuSig aggregation mechanism is preferable [1].
2. All conditions described in step 5 (Protocol section) can be put into a P2TR address. The formed address will not differ from the regular Bitcoin address (single or multisig) formed using the P2TR method [2].
3. As a zk-friendly hash function, we can use Poseidon [3].
4. For zk operations with EC points, we can use the 0xPARC library [4].

Links

[1] https://bitcoinops.org/en/topics/musig/

[2] https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki

[3] https://github.com/iden3/circomlib/blob/master/circuits/poseidon.circom

[4] https://github.com/0xPARC/circom-ecdsa/blob/master/circuits/secp256k1.circom