Getting closer to multi-chain smart contracts

Multi-chain Taprootized Atomic Swap protocol

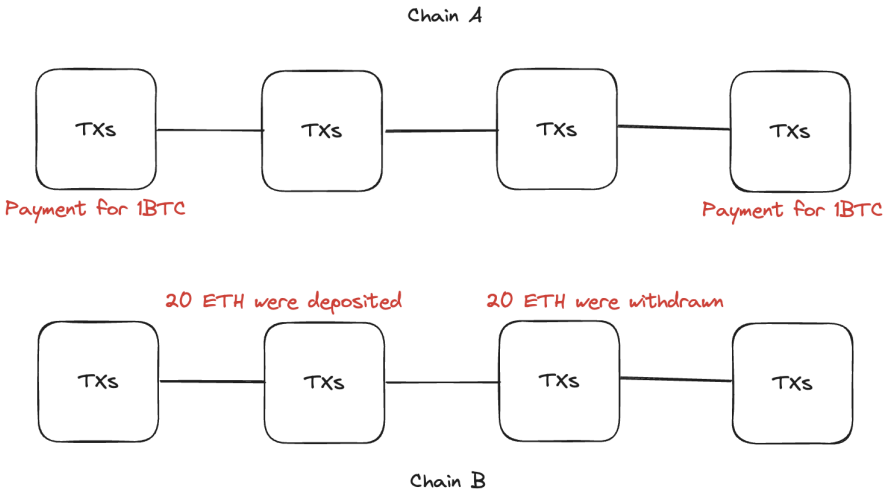
Distributed Lab, Jan 2024

Version 1.0

Abstract. We proposed Taprootized Atomic Swaps to add untraceability to atomic swaps. Our approach was still susceptible to brute force search of swapped amounts. Trying to solve that problem, we developed a new protocol that splits the receiving amount in an untraceable way. But the ultimate goal is to develop technology for multi-chain swaps (i.e., smart contracts that operate on multiple chains). This paper describes how a swap between a smart-contract-enabled chain and multiple Taproot-enabled chains can work.

Intro

Taprootized Atomic Swaps (TAS) allow hiding the swap transaction under the regular payment on the Bitcoin side. But simultaneously, the mentioned transaction has the "sibling" on the counterparty network. The relation between these transactions is the ratio of assets that were swapped. So it means that the external auditor can see how many coins/tokens were withdrawn from the appropriate contract and try to find the transaction in the Bitcoin networks that pays BTC (or assets issued in the Bitcoin system) in the corresponding ratio (based on market prices).



If the payment wasn't instant — the auditor can assume the time range in which the swap was performed (locktime in the contract can provide more info about it) and select all suitable transactions. Potentially, its number can be big, but anyway, it simplifies building the graph of transactions for auditors with specialized equipment.

This paper will provide a concept of breaking the amount of BTC that must be transferred into several transactions that will be processed via atomic way (not simultaneously, but within the timelock interval). It increases the difficulty of matching swap transactions because, in this case, the external auditor needs to solve a sudoku puzzle with a much larger number of combinations than direct swap transactions.

This solution can be applied to the swap between a smart contract platform and several chains that support Taproot technology. This way, the untraceability of swaps will be significantly increased, but the user experience will be more complicated.

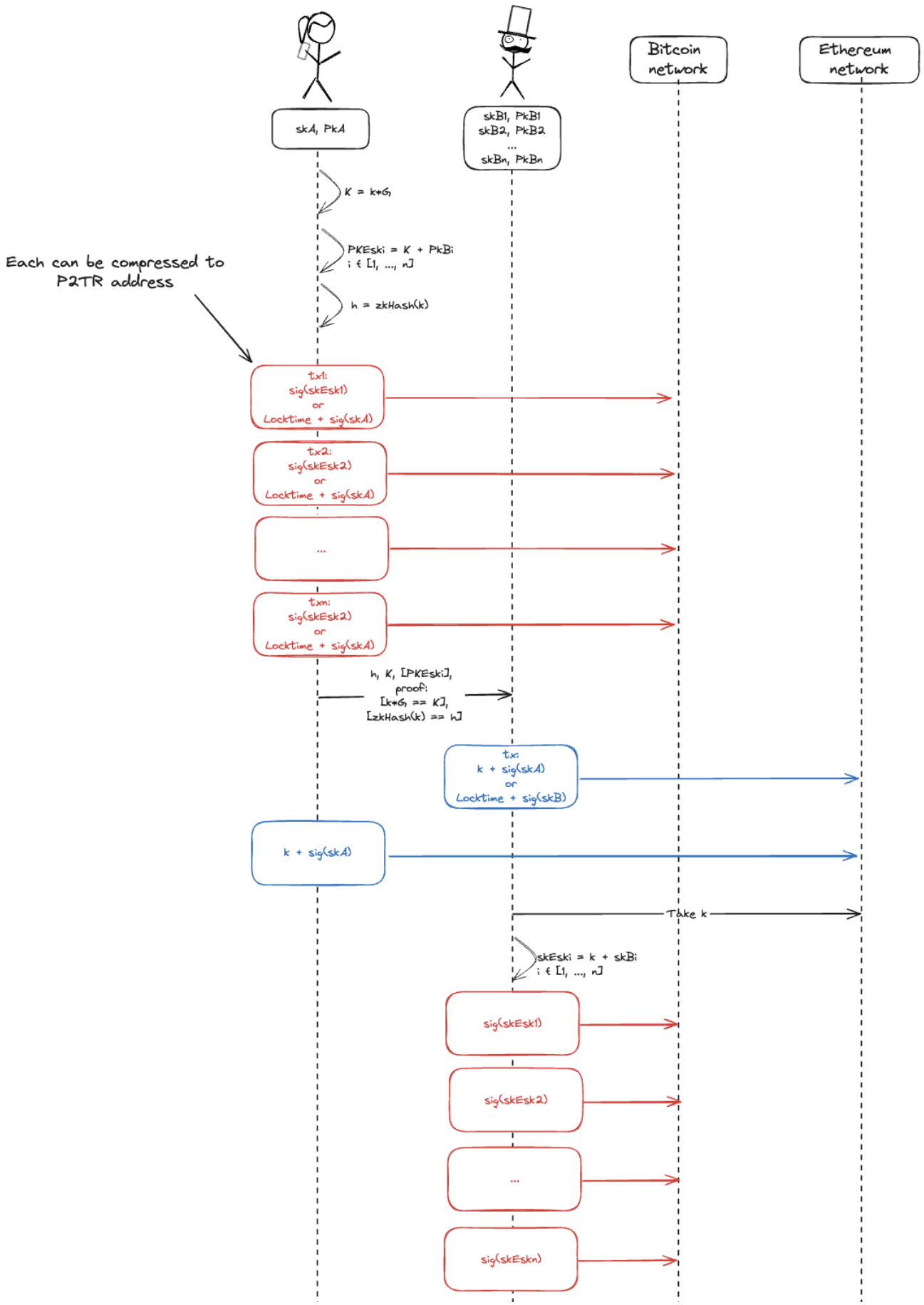
Splitting the amount between several transactions

The extension is the following — instead of forming a single PK_B by Bob*, he can generate the array of $[PK_{B\_i}]$ according to the number of transactions Alice wants to spend. So, the transformed protocol works the following way:

1. Alice has 1 BTC but on separate outputs (e.g., 0.5, 0.32, and 0.18 BTC). The number of keys needed by Bob in this case equals $n = 3$ (some of the outputs can be added into one transaction Alice wants).
2. Alice generates random k, and $K = k*G$
3. Alice calculates the array of $PK_{Esc\_i} = K + PK_{B\_i}$, where $i \in [1, n]$ and $h = hash(k)$
4. Alice sends array of transactions each of which pays to $\text{“}(PK_{Esc\_i})\ or\ (PK_{A\_i} + Locktime_i)\text{”}$
5. Alice sends **K**, $[PK_{Esc\_i}]$, **h** and $proof\ (hash(k) == h\ \&\&\ k*G == K)$
6. Bob verifies proofs and locks his 20 ETH to the smart contract with conditions $\text{“}(publishing\ k)\ or\ (PK_B + Locktime2)\text{”}$
7. Alice withdraws 20 ETH by publishing k
8. Bob takes k and generates the list of signatures for $[PK_{Esc\_i}]$ (using their sk_{B_i}).

So, as you can see, we use the same K for all Bob's public keys. As a result, after publishing the k value, Bob can unlock all transactions by calculating appropriate private keys for $[PK_{Esc\_i}]$.

This approach increases the cost of paying fees several times (depending on the n parameter). Still, it significantly affects the complexity of matching amounts over chains (the bigger combinations must be processed and accounted for in the graph).


Bitcoin network
Ethereum network
skA, PkA
skB1, PkB1
skB2, PkB2
...
skBn, PkBn
K = k*G,
PKEski = K + PkBi
i ∈ [1, ..., n]
h = zkHash(k)
Each can be compressed to P2TR address
tx1:
sig(skEsk1)
or
Locktime + sig(skA)
tx2:
sig(skEsk2)
or
Locktime + sig(skA)
...
txn:
sig(skEsk2)
or
Locktime + sig(skA)
h, K, [PKEski],
proof:
[k*G, == K],
[zkHash(k) == h]
tx:
k + sig(skA)
or
Locktime + sig(skB)
k + sig(skA)
Take k
skEski = k + skBi
i ∈ [1, ..., n]
sig(skEsk1)
sig(skEsk2)
...
sig(skEskn)

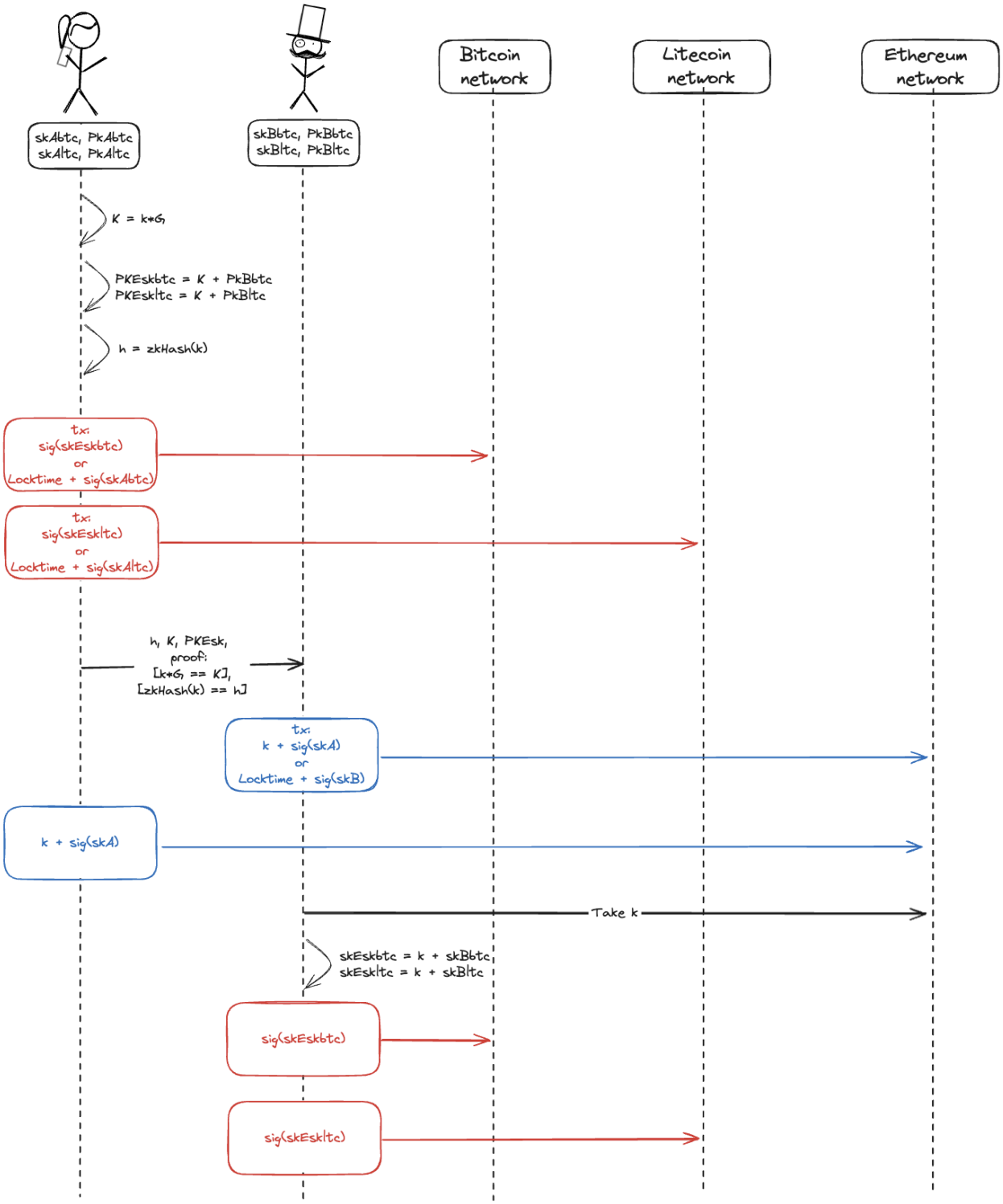
Multi-chain TAS protocol

The same approach can be applied to several networks that support Taproot technology. Imagine that Alice and Bob agreed to change 20 ETH to 0.8 BTC and 3 LTC. With Taptootized Atomic Swaps, they can do that via atomic way:

1. Alice has 0.8 BTC and 3 LTC. Bob needs to generate two keypairs (***n = 2***), the first for the payment on Bitcoin networks and the second for the payment on Litecoin.
2. Alice generates random ***k,*** and ***K = k*G***
3. Alice calculates $PK_{Esc\_BTC} = K + PK_{B\_BTC}$, and $PK_{Esc\_LTC} = K + PK_{B\_LTC}$
4. Alice broadcasts transactions to Bitcoin and Litecoin networks (locktimes could be different)
    a. ***"(PK_{Esc_BTC}) or (PK_{A_BTC} + Locktime)"***
    b. ***"(PK_{Esc_LTC}) or (PK_{A_LTC} + Locktime)"***
5. Alice sends **K**, PK_{Esc_BTC}, PK_{Esc_LTC}, **h** and ***proof (hash(k) == h && k*G == K)*** to Bob
6. Bob verifies proofs and locks his 20 ETH to the smart contract with conditions ***"(publishing k) or (PK_B + Locktime2)"***
7. Alice withdraws 20 ETH by publishing ***k***
9. Bob takes ***k*** and generates the list of signatures for PK_{Esc_BTC} and PK_{Esc_LTC} (using their sk_{B_BTC} and sk_{B_LTC}).

* The original protocol

1. Alice generates random ***k,*** and ***K = k*G***
2. Alice calculates $PK_{Esc} = K + PK_B$ and ***h = hash(k)***
3. Alice sends tx that pays 1 BTC to ***"(PK_{Esc}) or (PK_A + Locktime1)"***
4. Alice sends ***K***, PK_{Esc}, ***h*** and ***proof (hash(k) == h && k*G == K)***
5. Bob verifies proofs and locks his 20 ETH to the smart contract with conditions ***"(publishing k) or (PK_B + Locktime2)"***
6. Alice withdraws 20 ETH by publishing ***k***
7. Bob takes ***k*** and generates the signature for PK_{Esc} (using their sk_B).


skAbtc, PkAbtc
skAltc, PkAltc
skBbtc, PkBbtc
skBltc, PkBltc
Bitcoin network
Litecoin network
Ethereum network
K = k*G
PKEskbtc = K + PkBbtc
PKEskltc = K + PkBltc
h = zkHash(k)
tx:
sig(skEskbtc)
or
Locktime + sig(skAbtc)
tx:
sig(skEskltc)
or
Locktime + sig(skAltc)
h, K, PKEsk,
proof:
[k*G == K],
[zkHash(k) == h]
tx:
k + sig(skA)
or
Locktime + sig(skB)
k + sig(skA)
Take k
skEskbtc = k + skBbtc
skEskltc = k + skBltc
sig(skEskbtc)
sig(skEskltc)

Summary

The following list of properties is achieved with multichain TAS protocol:

1. Auditors can't match swap transactions based on **committed hash values** and appropriate **secrets** like in classic atomic swaps.
2. Auditors can't assume if the particular Bitcoin transaction participates in the swap — it's masked as a **regular payment transaction**. The locktime condition is hidden in the Taproot and revealed only if the swap wasn't performed.
3. Auditors **can't match amounts in chains** directly if the split mechanism is used. However, sudoku analysis can be applied to make some assumptions.
4. The protocol is **trustless**. The protocol guarantees that only the publishing of secret k can unlock money from the contract. At the same time, publishing ***k*** leads to the ability to form the correct key and produce the signature for BTC unlock.
5. **No mediator** is required. Users can exchange the needed information for the swap directly, using existing protocols for secure message transfer.
6. The protocol works **not only for the native currencies** but also supports tokenized assets, non-fungible tokens, etc. It can be a basic protocol for bridges, stablecoins, marketplaces, etc.

Links

[1] https://github.com/distributed-lab/taprootized-atomic-swaps