Secure cross-chain decentralized exchange

Lorenzo Tucci

June 24, 2025

Contents

Current Status and meeting notes

Main DEX Flow

Performing exchanges between client C who wants to exchange tokens from ledger L_a held at their address Addr_C^a with tokens from pool who holds tokens in ledger L_b at address Addr_{pool}^b (AMMs peer-to-pool method)
This is a recap of our discussions so far, based on what Ive been able to gather. Please feel free to correct any inaccuracies or missing thoughts. The approach is generic and currently follows the same blueprint as the ACNS'21 paper (just as a starting point for now)

1. **Burner address setup:** The servers set up pool addresses Addr_{pool}^a and Addr_{pool}^b on each ledger L_a and L_b from which they can transfer assets only if they all cooperate in threshold signing.
2. **order placement:** C transfer their tokens to pool addresses on ledger Addr_{pool}^a and send to the servers the addresses Addr_C^b on L_b where they will receive exchanged token.
3. **Confirmation:** If the servers have correctly received the deposits from the client, they proceed. Otherwise, they generate refund transactions transferring tokens from Addr_{pool}^a back to client address Addr_C^a. C retrieve these transactions and post them to L_a.
4. **Pay out:** Servers publish signed transactions for the final exchange operations to addresses Addr_C^b on L_b. If the servers are charging a fee for the exchange, they can similarly post transactions matching their fees from Addr_C^a to their accounts on L_a.

While the solution is generic, to ensure compatibility with Monero, this work proposes the following technical contribution:

Threshold RingCT The approach is to thresholdize the tag in RingCT. I am now reading Russell's thesis to understand the technical details on this.

- Construct tag from a **distributed PRF**.
- Generate aggregated proof for proving that the partial PRFs and partial key pairs are consistent.

Discussions and remarks

1. We consider private AMM and mempool privacy out of scope.
2. Any number of ledgers can be involved, as long as all parties can use a standard smart contract (e.g., Ethereum) on a ledger L_{Ex}.
3. In this protocol, there is no timelock assumption, in contrast to atomic swap protocols, which additionally require a timelock script.
4. Smart contract only gets in the loop if there is malicious activity from one of the committee members (servers). This is also the claim in the P2DEX paper, but there is a discussion referring to this P2DEX paper in Universal Atomic Swaps which I quote here: "Migration protocols like cryptocurrency-backed assets or side-chains, mimic the atomic swap functionality by requiring that Alice and Bob migrate their assets from (perhaps language-restricted) ledgers $\mathbb{B}_A$ and $\mathbb{B}_B$ into $\mathbb{B}_C$ with a more expressive scripting language (like Ethereum) [10]. Once the funds are in $\mathbb{B}_C$, they are swapped and sent back to the initial ledgers $\mathbb{B}_A$ and $\mathbb{B}_B$. Such an approach has the following drawbacks: (i) the swap protocol imposes transaction and cost overhead not only to $\mathbb{B}_A$ and $\mathbb{B}_B$ but also $\mathbb{B}_C$; and (ii) it is not an universal solution as it potentially requires a different asset migration as well as atomic swap protocols for each ledger that plays the role of $\mathbb{B}_C$ in the aforementioned example. Even in the unlikely case that everybody agrees to use $\mathbb{B}_C$ for their atomic swaps, we need to enhance every blockchain with the capability of migrating assets to/from $\mathbb{B}_C$." *This is contrary to what is claimed in P2DEX, and by looking into the paper, I couldn't confirm the validity of the criticism above. In case it is valid, our approach might have this additional advantage over P2DEX.*
5. Question/thinking out loudly: Are **accountable threshold signatures** beneficial in this setting, in the sense that the servers contributing to the final signature can be traced, and malicious behavior can be more easily identified?

1 Introduction

1.1 Our Contributions

1.2 Related Work

2 Preliminaries

2.1 Basic primitives

commitments, ZKP, ...

2.2 Non-Interactive Zero Knowledge Proofs

Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be a NP-witness-relation with corresponding NP-language $\mathcal{L} := \{x : \exists w \text{ s.t. } R(x,w) = 1\}$

A non-interactive zero-knowledge proof (NIZK) system for R consist of the following algorithms:

- $\mathsf{cr} \leftarrow \mathsf{ZK}_{\mathcal{L}}.\mathsf{Setup}(1^\lambda)$ takes on input the security parameter, outputs a common reference string crs
- $\pi \leftarrow \mathsf{ZK}_{\mathcal{L}}.\mathsf{Pr}(\mathsf{crs}, x, w)$ takes on input the reference string crs, a statement x and a witness w, outputs a proof π
- $0/1 \leftarrow \mathsf{ZK}_{\mathcal{L}}.\mathsf{Vr}(\mathsf{crs}, x, \pi)$ takes on input the reference string crs, a statement x and a proof π. Outputs 1 if w is a witness for the statment x, 0 otherwise.

We require a NIZK system to be *zero-knowledge*, where the verifier does not learn more than the validity of the statement x, and *simulation sound* where it is hard for any prover to convince a verifier of an invalid statement (chosen by the prover) even after having access to polynomially many simulated proofs for statements of his choosing.

2.3 Secure 2-Party Computation

A secure 2-party computa-tion (2PC) protocol allows two participating users P_0 and P_1 to securely compute some function f over their private inputs x_0 and x_1 respectively.

We require the standard *privacy* property, which states that the only information learned by the parties in the computation is that specified by the function output. We also require the standard security with aborts, where the adversary can decide whether the honest party will receive the output of the computation or not, and thus there are no assumptions on fairness or guaranteed output delivery.

2.4 Computational Assumptions

General setting of group-based crypto, implicit notation, assumptions that we need e.g. DLOG

3 Decentralized exchange

3.1 Syntax

3.2 Correctness

3.3 Security

3.4 Construction

4 Threshold RingCT transactions

4.1 Syntax

4.2 Correctness

4.3 Security

4.4 Construction

5 Instantiation and Performance Evaluation

5.1 Instantiation

Commitment

Tagging Scheme

2PC

Zero-Knowledge Proofs

5.2 Performance Evaluation