Swaptor Whitepaper

Current version: 1.0

Marko Ivanković[1]

[1]Berry Block, marko@berryblock.io

Abstract

Peer-to-peer (P2P) swaps on blockchain have the potential to greatly benefit users by eliminating the need for intermediaries in the exchange of assets. However, the use of P2P swaps on blockchain also presents several challenges, including trust issues that must be addressed in order to ensure their success. Since there is no intermediary to oversee the exchange of assets, users must rely on the trustworthiness of the other party to the transaction. In the absence of a trusted third party, it is difficult to verify the authenticity and quality of the assets being exchanged, which can lead to disputes and losses for users.

Furthermore, P2P swaps on blockchain are subject to potential security risks, such as hacking and fraud. Since the transactions are conducted directly between users, there is a greater risk of malicious actors attempting to exploit vulnerabilities in the system. This risk is exacerbated by the fact that blockchain transactions are irreversible, meaning that users have no recourse if their assets are stolen or lost.

In this paper we describe Swaptor, a decentralized P2P exchange dapp which aims to eliminate problems mentioned above.

Contents

Current version: 1.0

1 Architectural Overview

Current version: 1.0

As most modern dapps, Swaptor's architecture is a mix of on-chain and off-chain components:

- **On-chain**: Smart contracts
- **Off-chain**: Backend, Frontend and Database

1.1 On-chain Architecture

Smart contracts are written in Solidity, and their architecture is designed to be minimalistic. Specifically, the Swaptor contract is a singular entity whose sole purpose is to verify digital signatures and execute swaps upon successful verification. Digital signatures encode the specific terms under which a given swap is considered valid.

Elements encoded in a signature:

- **id**: Swap ID
- **swapType**: Swap type
- **seller**: Seller address
- **buyer**: Buyer address
- **offeredToken**: Address of the offered token
- **offeredTokenData**: Data corresponding to the offered token type
- **wantedToken**: Address of the wanted token
- **wantedTokenData**: Data corresponding to the wanted token type
- **chainId**: Id of the chain on which the swap will be executed
- **expirationTime**: Time until the swap is valid

In order to verify the signature, the smart contract must possess both the original elements and the signature itself. Prior to being passed as an argument to one of the *acceptSwap* functions, these elements are encoded using the *abi.encode* function. The signature is generated by creating a *keccak256* hash of the encoded arguments and signing it with the private key belonging to the seller.

1.2 Off-chain Architecture

2 Security

...

3 Tokenomics

...

4 Roadmap

Current version: 1.0

...

References

[1] Author(s), “title,” journal/proceedings info, page numbers, month & year.