# Whirlwind Swap: Tax-Compliant Private Swaps

Max Wu and Bao Mai

May 8, 2023

## 1 Introduction

In traditional finance, participants have temporary privacy until they are required to disclose trades: quarterly, semi-annually, or annually depending on the institution.

In decentralized finance, the conventional wisdom believes in a dichotomy between completely private solutions like Tornado Cash or completely public solutions like DEXes. However, with proper usage of ZK and smart contract design, we can achieve the temporary privacy desirable for institutions and high net worth individuals managing large amounts of money, while remaining fully tax-compliant.

Here we introduce Whirlwind, a protocol that achieves trades that are both temporarily private and tax-compliant.

### 1.1 Attracting Institutions to AMMs

The ability to reduce price impact from big trades is especially important for AMM-based DEXes, as bigger trades will have *exponentially* higher price impact. Without privacy, institutions would not be able to break big trades into smaller ones without front-running speculation. Temporary privacy also lets professional participants capture arbitrage opportunities without fear of immediate competition.

## 2 Design

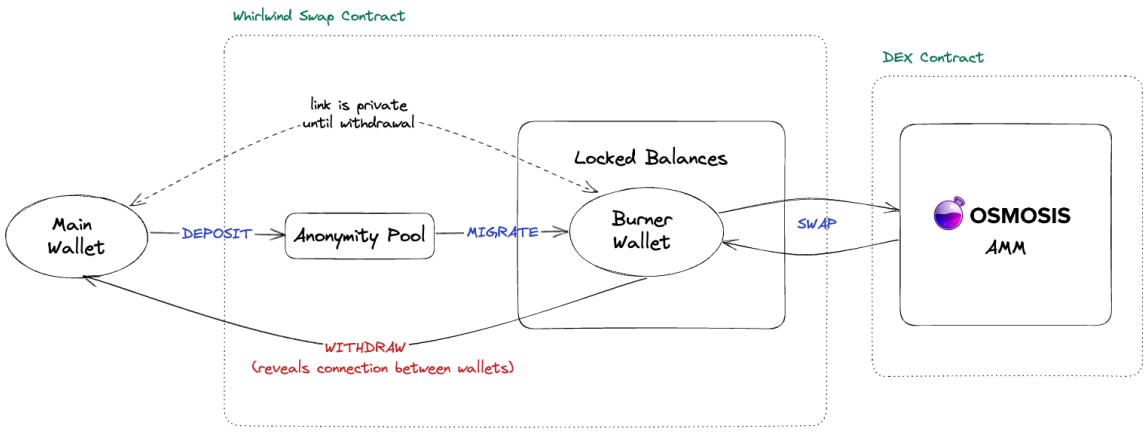


Figure 1: How Whirlwind Swap works

Our ZK logic is implemented as a CosmWasm contract that can be deployed through Osmosis's permissioned CW-module.

After depositing funds via an anonymity pool into Whirlwind, users are able to trade anonymously. However, they may only withdraw to the original source contract specified in the original deposit. This retains temporary privacy until withdrawal. While user funds are in the privacy pool, they are only

allowed to swap *exclusively with a contract-specified list of Osmosis pools*. All other Cosmos messages, like IBC, Bank, or Transfer, and CosmWasm messages are not allowed.

# 3 Implementation

We used Rust and Arkworks to implement the smart contract. We used Circom to implement the circuits, and SnarkJS to create the proofs and keys. We used code from webb.tools and the WasmJuicer contract found at gitopia.com/Juicer/juicy-10000.

The code can be found at github.com/legitmaxwu/whirlwind. We share implementation details below.

# 4 Contract State

- $A$, $D$: Amount, denomination of all initial deposit (e.g. 100 USDC). Fixing $A$ and $D$ allows us to create an anonymity pool.
- $T_{A,D}$: Deposit Merkle tree root. Merkle tree is initialized to all 0's.
- $S(N)$: Nullifier set keeping track of all spent nullifiers.
- Map($w_{\text{burner}} \rightarrow$ balances): Hashmap storing all locked balances in contract.
- Map($w_{\text{burner}} \rightarrow N_{\text{prev}}$): Hashmap storing the latest nullifier used by each burner address.

# 5 Contract Functions

A few notes:

- Function descriptions may be read in order.
- $H$ is a SNARK-friendly hash function (e.g. Poseidon).

## 5.1 Deposit

The user deposits amount $A$ of denomination $D$ from their main wallet into the anonymity pool.

### 5.1.1 Steps

1. Verify that amount $A$ of denomination $D$ are included in the transaction
2. Without revealing $s$, verify SNARK claiming that:
   - $s$ is any secret
   - $w$ is the wallet address of the owner of the funds
   - $C = H(w \,\|\, s)$
3. Add $C$ to Merkle tree $T_{A,D}$.

## 5.2 Migrate

Using a different burner wallet $w_{\text{burner}}$, the user creates a ZK proof of a previous deposit, authorizing them to claim amount $A$ of denomination $D$ into their smart contract balance.

#### 5.2.1 Steps

1. Check that $N$ does not exist in $S(N)$. (Otherwise, it's being double-spent.)
2. Without revealing $w$, $s$, or $s_{\text{prev}}$, verify SNARK claiming that:
   - $H(w \parallel s) \in T_{A,D}$
     - This means that there is a deposit corresponding to $w$ and $s$ in the anonymity pool.
   - $N = H(w \parallel s \parallel 1)$.
     - $N$ will be used to prevent double-spending $C$.
   - $N_{\text{prev}} = H(w \parallel s_{\text{prev}} \parallel 1)$ **OR** $N_{\text{prev}} = 0$.
     - This ensures that the previous migration by the caller claimed a credential owned by the same $w$ as this time. This ensures that all deposits claimed by this burner address are only redeemable by a single owner $w$, preventing mixing.
3. Grant amount $A$ of denomination $D$ to caller's locked balance.
4. Add $N$ to the spent nullifiers set $S(N)$.
5. Record $N$ as the new $N_{\text{prev}}$ for the caller for future migrations.

### 5.3 Swap

Once funds are migrated, the owner of $w_{\text{burner}}$ can freely execute swaps using their locked funds. Whirlwind acts as a proxy.

### 5.4 Withdraw All Funds

The owner of $w$ is able to withdraw all funds at any time, by proving they know the preimage of $N_{\text{prev}}$.

#### 5.4.1 Steps

1. Without revealing $s_{\text{prev}}$, verify SNARK claiming that:
   - $N_{\text{prev}} = H(w \parallel s_{\text{prev}} \parallel 1)$
   - $w$ is the caller.
2. Return all locked balances to the owner.