This is a crosspost from the flahsbots forum

DeFi today is designed to operate in a completely transparent state. E.g. for an AMM, traders or liquidity providers want to know existing reserves and volume in a pool. Indeed, an FHE (Fully Homomorphic Encryption) based AMM may be possible, but it comes at the cost of losing analytics and speed.

So how would you build a privacy-focused DEX or a DEX on a privacy network like Aztec (an L2 that has composability across private<>public state)?

Table of Contents

- Naive Constant Product AMM slapped with privacy - what are the minimal modifications we can make to uniswap v2 to make it private?
- Splitting/Batching flow on AMM - obfuscating the swap amount
- Onchain OTC Settlement
- Private RFQs

Privacy DEXs:

1. Naive Constant Product AMM slapped with privacy

Users can start their private transaction of swapping or providing liquidity privately (or from a “private” function (or off-chain), but then they must interact with the liquidity pool in public (so people can track the volume and liquidity). So, in each case, you would need to call a public function that updates the pool with the new amounts of tokens. So this design would leak the amounts of tokens swapped but not the account that performed the swap or who redeemed the output tokens

Privacy Assessment:

We can hide the user address but tokens to swap and input/output amount will be public. You may think of this as “Privacy for the user, transparency for the protocol”.

2. Splitting/Batching flow on AMMs

If we can’t hide the amount in Design 1, can we somehow obfuscate it

Broadly, there are two ways:

1. Splitting flow - split value into random sub-amounts and call the AMM for the small amounts.
2. Batching flow - You batch multiple users’ trades and execute them in one go. And later users can claim their pro rata share.

Flow 1 has worse UX as it requires users to be aware of splitting their amount and doing multiple calls.

A combination of both flows helps alleviate some of that. You could deposit your private tokens into a contract and then the contract divides it into smaller amounts and a 3rd party (solver) executes transactions for you. It can also aggregate various users’ intents and swap them all simultaneously. Penumbra’s ZSwaps deserves a shoutout for this design. Ref here. The user encrypts their amount and the solver adds all the encrypted amount to find out the final amount.

Here you encounter the paradigm of a “shared private state”. Typically this requires additive holomorphic encryption which is simpler than FHE.

Privacy Assessment:

This leaks the total amount and tokens to swap but not the individual user’s intent.

3. Onchain OTC Settlements

Instead of using an AMM (a shared pool), you enter into 1-1 private negotiations between parties off-chain. They select their preferred party at an agreed-upon swap rate. The user deploys a private contract with the agreed-upon parameters (swap rate, tokens). Both the user and the party enter their funds into this escrow and settle/swap onchain. For people to send notes to a contract and still keep them private, the contract must be deployed with a key that it can use to encrypt/transfer out the notes. This key can be a shared key generated by the user and the 3rd party.

Privacy Assessment:

The party sees everything but from the perspective of the network participants, everything is private! But of course, you need to find a party to trade with you off-chain. Therefore this is quite simple and synergistic with privacy

4. Private RFQs

RFQs like Cowswap provide superior UX due to their gasless nature and more MEV Protection guarantees than using an AMM.

Solvers could create a batch of user requests off-chain and the solver can settle publicly but individual users can redeem the output tokens privately (read Appendix on how to do this)

Privacy Assessment:

Individual trades are privacy-protected from the rest of the network but at the cost of latency. Solvers would see individual trade requests (but not necessarily the user’s address, like with the naive AMM implementation).

Appendix

Most of the above schemes spoke about how users may be able to redeem their output tokens in private even if their swaps were public. How is this possible?

In general, when giving your swap intent, the user specifies a hash of a secret. A solver creates a public note (UTXO) and encrypts it with a secret hash. A user can decrypt this note by specifying their preimage and therefore claim their note.

An Aztec-specific way to do this is detailed[here]