Private Decentralized Exchange

Mehmet Ugurbil

April 2, 2023

Abstract

We propose a private decentralized exchange that has Uniswap like invariant and uses secure multi-party computation (MPC) to retain privacy.

1 Introduction

Decentralized exchanges are used increasingly, however, they lack privacy. This leads to user data being publicly available as well as vulnerabilities such as front running. We survey the technologies required and the levels of privacy that may be provided for a private decentralized exchange using secure multi-party computation [DPS].

2 MPC Base

2.1 Field

We can choose to work in a large prime field or a ring 2^k [DEF]. Since the nodes will be distributed globally as in a blockchain, we need to accomodate a large number of nodes and the networking cost is expected to trump cpu cost, therefore we choose to work in a prime field over the cpu friendly ring. The prime number will be denoted p.

We will be working with linear secret sharing (LSS), where shares of a secret s are denoted generically as $[s]$.

2.2 Basic Protocols

While some operations can be run locally, MPC relies on network communication to run some. These will be denoted using capital letters such as SHARE for secret sharing to the network and REVEAL for revealing a secret.

Random elements can be generated in the field by all the nodes choosing their own random elements, sharing these and either summing up all the randomness or using better tricks such as hyper-invertible matrices. This protocol will be denoted RAN.

While additions and multiplications of public elements and additions of secrets can be done locally, multiplication requires a network protocol. The multiplication protocol is denoted by MULT.

We will build our system agnostic of the multiplication protocol used. Beaver triples or BGW multiplication can be used for example [AL].

If we want to reveal the result of the multiplication $[a] \cdot [b]$, we can do so by rather than sharing, we add a sharing of zero with the order of the product, $[0]$, then revealing the result, therefore skipping the communication step. This protocol will be denoted PUBMULT. In cases where the underlying multiplication protocol we chose does not allow this, we can run PUBMULT using MULT then a REVEAL.

2.3 Inverse

The inverse of an element x in the field is y such that $x \cdot y = 1 \mod p$. We can find this number via the multiplying and revealing by a random element and then inverting the public element we get:

$$
\begin{aligned}
[r] &= \texttt{RAN}() \\
y &= \texttt{PUBMULT}([x], [r]) \\
[x^{-1}] &= [r] \cdot y.inv()
\end{aligned}
$$

We denote this protocol INV.

2.4 Mod

We can use bit tricks [DFK] to accomplish more complicated tricks such as comparison [CH], exponentiation and modulo reduction [NX]. We note here that mod 2^k, MOD2k, is simpler than mod any number, MOD. This influences us to choose our fee in the form $2^{-8} \approx 0.004$ for example.

2.5 Division

Division is really expensive in boolean arithmetic, IEEE float division taking around 85k AND gates [AAS]. Instead we use integer division, DIV, which can be achieved by using MOD and INV, noting that if the numerator is divisible by the denominator, field division is equivalent to integer division.

$$
\begin{aligned}
[r] &= \texttt{MOD}([n], [d]) \\
[d^{-1}] &= \texttt{INV}([d]) \\
[z] &= \texttt{MULT}([n] - [r], [d^{-1}])
\end{aligned}
$$

2.6 Negatives

We represent negatives using the second half of the field such as $-x = p - x$. This is useful for comparison protocols and underflows.

2.7 Rationals

We want to represent token amounts either using rationals, so dollar amount of \$9.51 would be 951 with quotient 100 or avoiding this quotient by working in cents. We note however that either way, we need to keep track of multiplications and size of elements to avoid overflows.

3 Exchange

3.1 Swap

We base our invariant after Uniswap. We can leave out the fees in the initial version to make it simpler. Let the treasuries of two tokens be denoted as t_a and t_b and the invariant by k. Hence the exchange aims to keep the invariant:

$$
t_a \cdot t_b = k
$$

We will store all these privately as $[t_a], [t_b], [k]$.

For an incoming transaction amount a, we update the variables, where the new treasury amounts are t'_a and t'_b, the fee is denotred by f:

$$
\begin{aligned}
[f] &= \texttt{MOD2k}([a]) \\
[t'_a] &= [t_a] + [a] \\
[d] &= [t_a] + [a] + [f] \\
[t'_b] &= \texttt{DIV}([k], [d])
\end{aligned}
$$

Note here that $t'_a = t_a + a$ and $t'_b = \frac{k}{t_a+a+f}$ as in Uniswap. Since k is a multiplication of two rationals, we don't need to do rational correction for t'_b since it is k divided by a rational d, keeping the same quotient.

3.2 Semi-Private

Some information about the exchange might need to be public, like the approximate exchange rate of the tokens even if there is a small privacy component in the final result. We can achieve this by separating the reserve amounts into two parts $t_a = r_a + [s_a]$, where $[s_a]$ can be checked to be private using a comparison to check that it is within some bound.