# Abstract

Negotiations are a big part of our world. It can be as simple as negotiating the price of trading coin A for coin B. It can be as complex as international peace negotiations, or company acquisitions. In many negotiations, trust is a problem. People don't always want to reveal what they want. There is an ever present danger that sharing information about one's wants will be used against you. And you may be forced to accept unfavorable terms.

We propose a way to enforce private binding negotiations. In such negotiations, revealing the details of the trade finalizes them. The trade details are only disclosed to the matched party. We leverage this to build a decentralized dark pool. This system will let us make a bigger anonymity set and offer guaranteed execution so traders can propose trades without worrying that they will move the market.

We also discuss other applications of such negotiations, including corporate mergers and acquisitions.

# Introduction

Trading coin A for coin B is an important feature of blockchain. To trade coin A for coin B, the person with coin A needs to find the person with coin B to make the trade. We call this order matching. After the match, the parties must work together to execute the transaction. Front running means when someone else steals your match and then insists that you pay a higher fee to them to execute your trade.

Privacy of execution does not solve the front running problem because the traders still need to find each other. As soon as you reveal to another trader what you want to trade, they can front run you. The primary solution now is that trades are executed as soon as they are revealed. This limits the types of trades that are possible. For example, it is impossible to have an If Else clause around revealed trades. You can't say if my trade is possible to execute, reveal and execute. The other main issue with this approach is that all orders get shown, so privacy is lost.

The anonymity set is the set of people who could have taken an action. If someone withdraws one eth in tornado cash, only a party who deposited one eth could withdraw it. At the moment, the anonymity set of Tornado cash 1 eth pool is 38229. The anonymity set of the 1000 DAI pool is 832. Trying to maximize the anonymity set is an important goal.

It would be nice if we could have a private order book where trades get matched, executed, and never revealed. If we had that, we would be able to merge the anonymity set of DAI and ETH pools into one mega anonymity set. Because if a user withdraws from the DAI pool, you won't know if they got their funds via deposit or atomic swap between pools. It would also bring more utility to being in the anonymity set, increasing its size.

Previous work

A previous attempt to implement private trades used an MPC (multi-party computation) to try and find someone who wanted to trade with us. The issue here is that execution is not guaranteed. This gives both parties a chance to cancel the trade, which means you can still get front run.

To avoid this, we need to add this concept of guaranteed execution such that after the trade is matched, both parties can unilaterally execute it.

Problem setting

Let's say we have Alice, Bob, and Carol who want to make the following trades

Party

Ask

Get

Alice

1 eth

1 dai

Bob

2 dai

1 eth

Carol

1 dai

1 eth

Alice and Carol can execute a trade. But they don’t want to advertise their trades, and they want to be 100% sure that as soon as the other knows the trade matches, it can be executed unilaterally.

Guaranteed execution

To guarantee execution Alice, Bob and Carol

- Commit to their trade
- Prove they have enough funds in an on-chain private pool to execute the trade they committed to
- Lock these funds for the duration of the protocol so they are available to trade and nothing else.

Alice , Bob and Carol decide in which order they want to try and match with each other. They each create their prioritized list of matching partners. Then their lists get merged by the smart contract to make the global list seen below.

Iteration

Party 1

Party 2

0

Alice

Bob

1

Carol

Bob

2

Alice

Carol

Multi-Party Computation

For each round of the MPC, each party provides a ZKP that they have not already matched previously. They then take their commitment from above and use that in the MPC to compare orders.

The MPC returns 0 if the trades don’t match. If the trades match, it returns the witness data required to execute that trade. Therefore, after a trade gets matched, both users can unilaterally execute the trade.

To match trades has to be the same. We don’t have support for partial fills now but could consider adding it in the future.

After each round of the MPC, they make a ZKP that proves all previous rounds have failed. This is used to guarantee that the funds have not been traded already.

Comparison and Execution

Iteration

Party 1

Party 2

Match

0

Alice

Bob

No

1

Carol

Bob

No

2

Alice

Carol

Yes

Alice and Carol match, Alice and Carol get enough info to make a ZKP to execute the trade. After this match, Alice and Carol can no longer try to match with others because their orders have been matched. This is enforced because other members will not trade with Alice or Carol unless they can prove they did not match already. To not block the other searches, Alice and Carol can give a skip message to other participants. The skip message is a ZKP that they already matched.

After matching, both Alice and Carol can unilaterally execute their trades. They have until the end of the epoch to do this. They take the witness data that the MPC generated and use this to create a ZKP that executes a trade. This ZKP also includes a proof that they failed to match with every previous iteration of the search.

Who knows what

After this protocol:

If match

- Alice and Carol know each other's trades,
- All future MPC participants with Alice and Carlo know they matched with someone previously.

Else:

- Alice and Carol know their trades don't match
- All future MPC participants with Alice and Carlo know they didn't match with each other.

Hiding Success of Matching

We can also conceal a successful match by allowing our matched parties to continue trying to match with others, provided they change the trade they are advertising to be impossible to match. For example, we could set the tokenID to equal the hash of their private key. That way, it will never match with anyone as it's an invalid token ID.

Liveness

The fact that everyone needs to be online for the protocol to progress introduces some liveness issues. Where a single unresponsive peer can result in both parties being unable to continue.

To mitigate this we allow each peer to select the order in which they try and match with their peers. This allows peers to try and match with trusted peers first and leave trying to match with untrusted ones until the end.

If any peer fails to complete the matching protocol we wait until the end of the epoch to start searching again. We also never try to match with that peer again.

Future Work

Private Binding Negotiations can be expanded into more complicated scenarios. We don't have to have guaranteed execution via smart contracts, we could also have guaranteed execution via legal contracts. This would allow these techniques to be used for other big problems where people need to agree on something but don't want to reveal their positions. A future post will explore applications of this to legacy world examples, peace negotiations and/or complex business negotiations.

Conclusion

Here, we introduced a mechanism of private binding negotiations. We used this to force trades to be executed after matching. This will allow for a decentralized dark pool but requires advanced MPC and ZKP tooling to build. Most of the MPC and ZKP are off-chain. On-chain we need to be able to define a specialized ZKP that confirms the MPC was run

correctly and no peer was skipped, we also need to have token locks for a certain amount of time.

There is also considerable future work in applying these techniques to more complicated negotiations inside and outside the blockchain.