Multi-token Batch Auctions with Uniform Clearing Prices on Plasma

-

Specification For Implementation

Gnosis

July 9, 2018

This document outlines a formal specification of a batch auction exchange of ERC20 tokens using the plasma technologies. The batch auction mechanism will enable the execution of ring trades between different tokens. Plasma technologies will provide the same security for users as if the tokens were held in the ethereum blockchain. But gas costs will be significantly reduced due to the off-chain nature of the exchange mechanism. It is recommended for the reader to familiarize themselves with Plasma[3], Plasma MVP[4], snarks[6] and batch auctions[9] before reading this spec.

Contents

1 Introduction

Multi-batch auctions with uniform clearing prices This specification outlines a trading mechanism between several ERC20 tokens. Each batch accepts orders to buy any ERC20 token with any other ERC20 token for a maximal specified limit price. All orders are collected over some time interval and then an *uniform clearing price* over all token pairs is calculated.

For a list of k trading tokens denoted by T with clearing prices $\{p_{ij}\}$, we call the set of prices $\{p_{ij}|\tau_i, \tau_j \in \mathrm{T}\}$ an uniform clearing price, if

$$p_{ij} \cdot p_{jk} = p_{ik} \tag{1}$$
$$p_{ij} = p_{ji}^{-1} \tag{2}$$

Finding the uniform clearing prices, which also maximize trading volume or other predefined metrics is a complicated task. The paper [2] describes a mechanism in detail. Uniform clearing prices are beneficial for any trader since they are arbitrage-free and ensure good liquidity via built-in ring trades. For the purpose of this document, it is assumed that these uniform clearing prices for a multi-batch auction can be efficiently determined.
The multi-token batch mechanism is secured against front-running [1] by allowing users to submit encrypted orders into the batch auction. Only after the batch has closed the orders be revealed and then the prices can be computed.

Plasma minimal viable product This paper describes the specification for porting this auction mechanism onto a plasma chain. Plasma chains are a second layer scaling solution for blockchains which allow processing a much higher volume of transactions as the underlying blockchain (a.k.a. *root-chain*) with the same security guarantees. This is the main motivation for implementing such mechanism on a plasma chain as it will result in significantly less costs per trade. Unfortunately, plasma chains have also a disadvantages: They come with burdens in terms of complexity and sometimes usability. We are planning to use Ethereum as the root-chain.

The trade-flow of a batch auction The batch auction exchange is built on a plasma chain operated by a single entity on top of a root-chain. A single batch auction consists of several phases:

(i) Encryption Key generation,

(ii) Order Collection,

[1]Front-running is when a broker/operator enters into an equity/asset trade with foreknowledge of a block transaction which will influence the price

(iii) Double Signing,

(iv) Batch Price Calculation,

(v) Batch Challenging and

(vi) Order Executions.

In brief, a batch will proceed as follows: Users will post encrypted orders to the plasma operator, who will then bundle these orders and commit them by posting a single order-block on the plasma chain. Each client will have to validate the orders of this order-block. Only once they have validated the orders, they may double sign their orders to participate in the current batch. The participation acknowledgment will be compactly represented using a crypto-economic signature aggregation schema. With this construction many data-unavailability problems will be avoided because the orders can be represented on the root-chain using only a bitmap. Once the bitmap is seen by many parties on the plasma chain, the private key for decrypting the orders will be generated using distributed key generation (DKG). Finally, all the data is available to calculate correct prices from the orders and the batch can be settled.

2 Detailed specification overview

2.1 Setup description

2.1.1 Definitions

In our batch auctions all trading will occur between a fixed set, $\mathrm{T} = \{\tau_i\}_{i=1}^{n}$, of n ERC20 tokens. The collection of orders for which token i is being sold in exchange for token j will be denoted by $\mathcal{O}_{i \to j}$.

After an auction has been closed and uniform clearing prices have been determined, the price matrix can be represented as an $n \times n$ a matrix

$$P = \{p_{ij} | \tau_i, \tau_j \in \mathrm{T}\}.$$

For any order $\sigma \in \mathcal{O}_{i \to j}$ we define $\nu_p(\sigma)$ as the volume of an order denominated in token τ_i at clearing prices p.

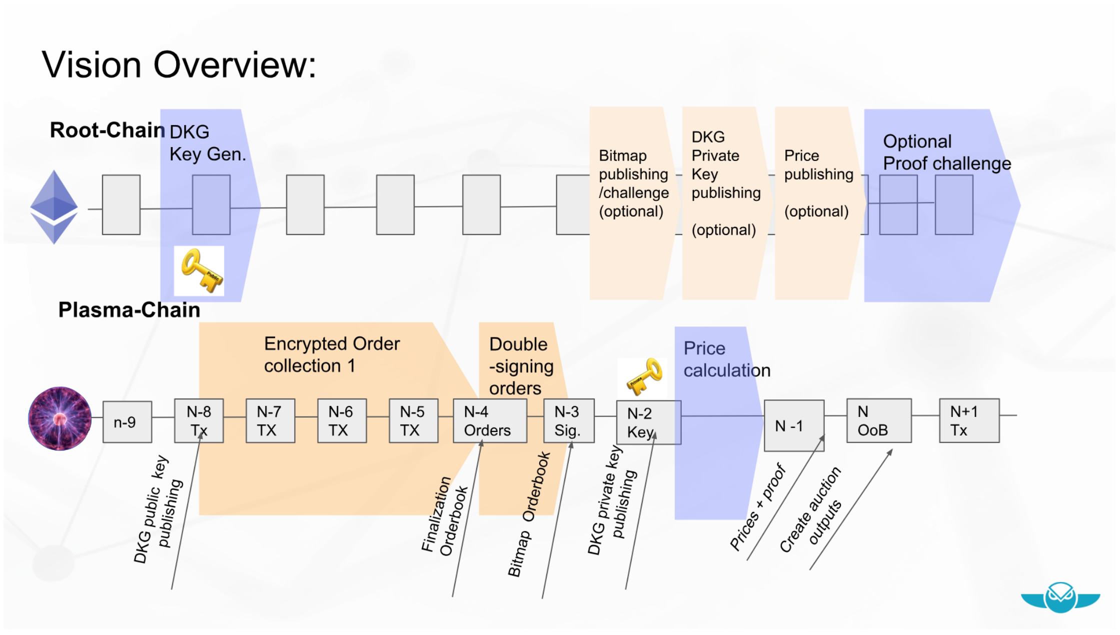
2.1.2 Plasma chain setup

We are following the description of the Plasma Minimal Viable Product[4]. This means all assets are represented in the plasma chain as unspent transaction outputs (utxo). These utxos can be transferred to a new owner by making a transaction, waiting for its inclusion into the plasma chain, validating the chain, waiting for the plasma block to be submitted to the root-chain and then, sending a double-sign message to the receiver.

Any utxo may be exited at any time, by providing a Merkle proof of the receiving transaction. However, if the exit is not valid, then it will be canceled via a challenge process. Note that the plasma implementation will require users to be online and validate the chain's state at least once a week.

2.2 The trade flow

In the following section we will describe the general tradeflow through all its phases: Encryption Key generation, Order Collection, Double signing, DKG Private Key Generation, Price submission, Price Proofs and Order Executions.



2.2.1 Encryption Key Generation

All orders will be encrypted with a public key generated by the Distributed Key Generation (DKG) algorithm [1]. A predefined number of bonded participants will be part of the DKG algorithm. By exchanging special messages and withholding others, they will generate a new public key for which the private decryption key is not known to any single participant. Only later, after the auction closes, the DKG participants are requested to publish their private messages for the generation of the private key for the decryption of the orders. If a predefined threshold of participants publish it, this will allow anyone to generate the private key. If the participants publish their messages previous to the closing of the batch and its documentation on the root-chain, the protocol will allow to slashed them. With this approach, we have encrypted orders, unless a large portion of bonded DKG participants are malicious.

The DKG algorithm generates a public key represented as an elliptic point aG, for a generator G of a elliptic group and a secret $a \in \mathbb{N}$ (cp. [1]). This public key can then be used to encrypt messages via the ECIES [7]. Each trader calculates bG for a random natural number b and publishes the point bG in his order. Then he calculates: baG and determines from this number the symmetric key pair for encrypting his order. A good symmetric encryption mechanism might be the Knudsen-Nyberg cipher[8], as this one would be efficient to calculate and verify using snarks in the price correctness proof2.2.6. Later, once a is public, we can decrypt the orders by calculating the symmetric key pair from abG, which equals baG.

2.2.2 Order Collection

Once the public key for the encryption process is available, every trader can submit orders. In the plasma chain, all assets of the ERC20 tokens are represented by unspent transaction outputs (utxo). An order $\mathcal{O}_{i\to j}$ will refer to a utxo of an ERC20 token τ_i and this utxo will be exchanged for the specified target token τ_j for a price below the given, encrypted limit price.

```
Order[  Utxo,
     Token  $\tau_j$,
     Limit price,
     Encryption salt,
     Signatures]
```

The encryption salt is the elliptic point used for encryption: bG (cp. 2.2.1). All orders will be collected and board-casted in one block per auction, the *order-block*. All orders included in this order-block will be able to participate in the next auction. Theoretically, the chain operator can censor orders, but since they are encrypted, he does not know what he is censoring. Sometimes the operator needs to censor some orders since, in a single batch, there shouldn't be more than 2^{16} orders, due to a constraint of space. If a user wants to cancel their order, they should spend their utxo referred in their order or simple do not double sign the order.

2.2.3 Double Signing

Once the block with the orders is published and the Merkleroot hash is submitted in the root-chain, every participant in the batch auction needs to validate this new block and the complete chain. Once they commit that their order should be included in the next batch, they will be able to withdraw their order funds only with the priority[2] of this order-block - this will be explained in detail here: 2.3.
If participants validated the whole chain, they can send the operator the double signing message to signal that they want to participate.

```
DoubleSign [OrderHash,
            OrderbookBlockHash,
            BatchIndex,
            Signature]
```

The operator collects all these DoubleSign messages and constructs a bitmap, in order to generate a crypto-economic signature aggregation (cp.[5]). If the operator gets a DoubleSign

[2]The priority-queue is an important concept in the plasma MVP

message for the order at the i-th place in the block, then the operator puts a 1 into the bitmap at the i-th place. Otherwise, he puts a 0 in there. Next, the operator publishes the bitmap and all DoubleSign messages in a next plasma block. If the bitmap is not available or incorrect, anyone can ask the operator to publish it on the root-chain and then each bit of the map could be challenged by asking the anyone to show the DoubleSign message and its Merkle-proof.

Placing an order, which will not be executed by the limit price, is for free. If people miss-use this free option to spam the plasma chain, they will be censored by the plasma chain operator. The operator could keep on censoring their data, which would force a spammer to eventually withdraw their funds from the exchange.

This double signing process will hinder a very smooth user experience. Hence, we would allow users outsource this process of double signing to another service, if they prefer to. This involves some trust, but the trust is limited, as this service can not change an order. It can just double sign the order.

2.2.4 DKG Private Key Generation

Once the plasma block, containing the bitmap is published and submitted to the root-chain, the DKG participants will publish their hidden messages used for generating the DKG-public key. They do it by sending their inputs to the chain operator. Once the threshold is reached and there are enough messages available to generate the private key, the operator will do it and publish it in a next plasma block.

2.2.5 Price Submission

Once the previous step is finished, the chain operator can calculate the optimal prices between all ERC20 tokens with the current batch orders. If $\{p_{ij} | \tau_i, \tau_j \in \mathrm{T}\}$ are the uniform clearing prices, they are satisfying the following equation

$$p_{ij} \cdot p_{jk} = p_{ik} \tag{3}$$

$$p_{ij} = p_{ji}^{-1} \tag{4}$$

Due to these equations, it is enough for the operator to publish only the $n = |\mathrm{T}|$ prices $\{p_{ij} | \tau_j \in \mathrm{T}\}$ for a $\tau_i \in \mathrm{T}$ and all other prices are implied.

The operator would additionally publish another block, which contains the information about which order was matched and by which volume. The block would have the following structure:

$$[[\text{order}_1, \text{volume}_1], [\text{order}_2, \text{volume}_2], ...].$$

We want to call this block the *volume-block*. We need to specify the trading volume for each order since not every order below the limit price will be completely executed. This is non-intuitive but this is one of the things we have to deal with in a batch auction.

Optional mechanism: Price submission by a third party Usually the price would be submitted by the chain operator only. But if the chain operator does not publish the prices, we could accept price submission root-chain by any person. The smart contract on the root-chain would select the price submission as the final price, which is valid, has a snark valid proof of correctness and fairness and has the biggest trading volume measured in Ether.

Note that in this case, the price submitter would also need to provide the volume-block (cp. 2.2.5) or at least assist traders with their exits games, by providing the Merkle proofs for their trading volumes (cp. 2.3).

2.2.6 Proof of Correctness and Fairness

Once the prices are calculated by the chain operator, he needs to prove that this set of prices is actually valid and fair. A price set and a volume-block (VB) 2.2.5 is valid and fair, if

- the price set satisfies 3
- Orders are only touched, if the limit price is below the calculated price:

$$\forall \tau_i, \tau_j \in \mathrm{T} \quad \forall o \in \mathcal{O}_{i \to j}, \quad \nu_{p_{ij}}(o) > 0 \Rightarrow p_{ij} \geq \mathsf{p}(o) \tag{5}$$

- the amount of sell volume for a token equals its buy volume:

$$\forall \tau_i \in \mathrm{T} \sum_{\tau_j \in \mathrm{T}} \sum_{o \in \mathcal{O}_{(\tau_i \to \tau_j)}} \nu_{p_{ij}}(o) == \sum_{\tau_j \in \mathrm{T}} \sum_{o \in \mathcal{O}_{(\tau_j \to \tau_i)}} \frac{\nu_{p_{ji}}(o)}{p_{ji}} \tag{6}$$

- If an $\mathcal{O}_{(\tau_i \to \tau_j)}$ with a limit price p has a positive trading volume, then every $\mathcal{O}_{(\tau_i \to \tau_j)}$ with a lower limit price as p should be traded completely.

In order to prove this, we will use snarks (Succinct non-interactive arguments of knowledge) [6]. They are a powerful tool, which enables us to encode the above mentioned checking logic into an algebraical formulation. This allows us to generate some verification keys, which compactify this checking logic. Then for each set of prices and any volume-block, we can generate quite quickly a proof that they satisfy the constraints using the verification keys. The proof size is very small (8 bytes64 variables) and can easily be provided to the main chain. Also, the execution of the proof is not too costly: it is around 1.6 million gas on the root-chain. The library Zokrates demonstrates these methods very well for the ethereum eco-system.

Now, we will describe the snark mechanism in more detail: In the following we denote the block containing all orders OB 2.2.2, the volume-block VB 2.2.5, the bitmap B and $M_R(OB)$, $M_R(VB)$, $M_R(B)$ their Merkleroot hashes, respectively. Let us denote the decrypted orders from the block OB as list [orders]= [order$_1$, order$_2$,...] and the trading volume of each order by the list [volumes] = [volume$_1$, volume$_2$...], where the volume$_i$ corresponds to the order$_i$.

Then the program P(i,w) should prove the above mentioned facts. The public input into P is declared by the variable i and w will be the witness. In details it will look like:

$$\text{P(i=}M_R\text{(OB)},M_R\text{(VB)},M_R\text{(B)},\text{key}_p), \text{w=([orders],[volumes],[prices],[bitmap]))} \quad (7)$$

And P would check that:

- M_R(OB) $==$ M_R(Encrypted[orders])
- M_R(VB) $==$ M_R([orders] $\cup$ [volumes]),
  where [orders] $\cup$ [volumes] $=$ [order$_1$, volume$_1$, ...]
- M_R(B) $==$ M_R([bitmap])
- $\forall o \in$ [orders], vol$(o) > 0 \implies$ bitmap$(o) = 1$,
- $$\forall \tau_i, \tau_j \in \mathrm{T} \quad \forall o \in \mathcal{O}_{i \to j}, \quad \nu_{p_{ij}}(o) > 0 \Rightarrow p_{ij} \geq \mathsf{p}(o) \quad (8)$$
- $$\forall \tau_i \in \mathrm{T} \quad \sum_{\tau_j \in \mathrm{T}} \sum_{o \in \mathcal{O}_{(\tau_i \to \tau_j)}} \nu_{p_{ij}}(o) == \sum_{\tau_j \in \mathrm{T}} \sum_{o \in \mathcal{O}_{(\tau_j \to \tau_i)}} \frac{\nu_{p_{ji}}(o)}{p_{ji}} \quad (9)$$
- For each trading pair, P loops through all orders and memorizes the highest limit price of an order of this pair, which still has a positive trading volume. Then it loops through them again and checks that no other order with a lower limit price has not been filled.

Once the proof is calculated, the chain operator will publish it on the plasma chain.

2.2.7 Enforcing Publishment Key Variables

In case the operator goes off-line and we get a data-unavailability, we still want to make sure that the last auction can be completed by all participants or it will be cleanly un-rolled. Since data-availability on the plasma chain is subjective, we need to make sure that all necessary information will be loaded into the root-chain, in the worst case. The necessary data is the bitmap (2^{16} bits), the private key for decrypting the orders (2x 32 bytes), the prices (n x 32 bytes) and the proof for correct volumes (10 x 32 bytes).

bitmap It is essential for everyone to know, what the bitmap of an auction is, in order to know which orders are in the batch. If the bitmap has not been sent out by the chain operator, we have to make it available on the root chain. Everyone can request - in exchange for a small fee - that the bitmap gets published on the root chain. Then anyone can publish the bitmap along with a plasma containment proof of this bitmap and get the fee. If no one publishes it, then the last batch will be rolled back completely. This means that only utxos, which were submitted before the last order book hash submission, will be accepted for any withdraw.

decryption keys Also, the decryption key of an auction is essential to know, in order to see the limit prices of the encrypted orders. If the decryption key is not publicly available, we can apply the same process as for the bitmap to make it public.

prices The prices are also essential to know for each participant. If they are not public, again they can be made public with the same mechanism. We note that the gas costs for publishing the prices increase linearly with the number of tokens.

price verification The proof of the correctness is initially only stored in the plasma chain. But anyone can request to publish the proof also on the root-chain. We use again the same mechanism as before.

2.2.8 Order-output Transactions

After the prices and a price verification proof are published, the operator will publish another block with all order-output transactions. An order-output transaction for an order within $\mathcal{O}_{i \to j}$ will have the following format:

```
OrderOutput [ Order txHash,
    Order output denominated in Token $\tau_j$ ($oo_j$),
    newOwner,
    Order output denominated in Token $\tau_i$ ($oo_i$),
    orderCreator]
```

where the order-output denominated in Token τ_i should be zero is the most cases, as orders will be most likely be fully filled. The order-output (oo_j) denominated in Token τ_j will be calculated by the price

$$oo_j = p_{ij} \times \text{OrderInputAmount} \times \nu(\text{order}) \times (1 - \text{fee}_{\text{ratio}}) \tag{10}$$

where ν(order) is the volume declared in the volume-block. Whereas, the order-output (oo_i) denominated in Token τ_i is simply the left over, if the trade could not be executed completely:

$$oo_i = \text{OrderInputAmount} - \nu(\text{order}) \tag{11}$$

These order-outputs can then be spent again via a transaction, but cannot be used as an input for another order. If any of these order-outputs is not correctly created by the chain operator, people have to leave the plasma chain. Then the last batch will be settled on the root-chain or be reverted, depending whether all trading volumes are available.

2.2.9 Claiming Fees

After each batch auction, the plasma-contract on the root-chain will hold some tokens, which are not covered by transaction outputs in the plasma chain, due to fee mechanism specified in 10. The chain operator is allowed to create a new utxo with the number of uncovered tokens, in order to claim the fees.

2.2.10 Auction scheduling process

An auction can have a variable timing. The operator can decide on his own when to close the batch and create the order-block 2.2.2. Once this block is posted, it will take some time until the orders are processed. But the operator will ensure a smooth flow by accepting new orders right away. But of course, these orders and all transfers will not be executed if the chain will halt. Hence batch auctions can be scheduled pretty fast and smooth one after the other. But the finality of an auction is only guaranteed after the client sees all the data available for a batch auction, including the bitmap, decryption key, prices, and price verification data.

2.3 Exit rules of the plasma chain

Transfer transactions have quite the same exit rules, as in the usual plasma mvp. But there are two differences.

- We would maintain a variable *highestExitPriority*. This highestExitPriority would be infinity in normal cases. But if the plasma chain is halted, because a data-unavailability request 2.2.7 was not answered in a certain time frame, then this variable will play an important role. It will set the highest exit priority to the priority of the last block, before the order-block submission. Any utxo generated after highestExitPriority will not be exitable and any exit request cannot be challenged with a utxo spend after this highestExitPriority. Effectively, we are resetting the plasma state to a previous state by setting highestExitPriority.
- Any utxo can be challenged as spent and not exit-able, by showing an order, a double signing of the order and that the trading volume of this order is positive. Of course the Merkle proofs for this data needs to be provided.

Order transactions can also be referenced for an exit request. The request are put into the same priority queue as the normal exits. They need to prove that the order transaction was also double signed and had a positive trading volume by providing data by providing Merkle proofs for it. For a "touched" order in $\mathcal{O}_{i \to j}$ only the target token τ_j can be withdrawn, but not the input token τ_i. Hence, trades are enforced and cannot be reverted arbitrarily. If a data-unavailability appears right after the closing the batch, the order-output transaction might be already in a plasma block. These order-outputs cannot be used to challenge an order exit. Only if the order-output is already spent and double signed, then the order exit process can be challenged successfully. It might happen that a trader does not know the trading volume of his order in the last batch auction because the order-output block might be unavailable for him. Then, he would not be able to provide the Merkle proof to attest his positive trading volume. In this case, we would ask on anyone to complete his exit, by providing the Merkle proof with the trading volume. If neither the chain operator nor anyone else does answer this exit request within 7 days, then the auction will be stopped and all trades will be reverted if the auction was conducted in the last 7 days. If the auction was not conducted in the last 7 days, then the exit requester should have the data available.

3 Features

3.1 Trade enforcement

Trades on this exchange will usually be enforced. That is, if an order was submitted, double signed and was touched by auction price, then the actor can only withdraw the trade-output nominated in the target token. But the trader will not back his trade input.

Unfortunately, some actors have an advantage. If many DKG key holders do not reveal the decryption key, they can stop the chain. The DKG algorithm works with a predefined threshold. Only if more DKG participants as specified by the algorithm do not reveal their private input, then the batch cannot be executed. But in this case, we could also slash these DGK participants, which are misbehaving.

3.2 Safety in case of a data-unavailability

For token transfers, we have the same data-unavailability fallbacks, as the MVP product. Hence a double sign mechanism protects people from losing funds. It is only getting a little bit more complex, as users should not double sign a transaction, which happens right after the closing of a batch, before they validated the WHOLE process of the batch closing, including the publishing of valid prices, snark proofs of correctness and a correct order-output block.
Also for traders, the safety is guaranteed by a double signing mechanism. If there is anything incorrect, before the double signing step, traders will just not double sign their batch. If any network attack happens right after the order block, the traders withdrawal priority will still be higher than any withdrawal request of this attack. Hence traders will always get their funds back or they can settle the last batch via the on chain exit. If there is any data not available for settling the trade with an exit request after the double signing step, then this data can be requested to be uploaded to the root chain. The exchange is constructed in such a way with snarks and bitmaps, that it is easy and relatively cheap to publish this data on the root-chain. If the data is published, any trader can get his trade settled on chain. If the data is not published, the batch is reverted. For a detailed description of the data, which needs to be made available, see here: 2.2.7. The proof that this data is enough to settle the trade either on the plasma chain or on the root-chain is left to the reader.

3.3 Non-custody of funds

From the above analysis, we can conclude that the plasma chain operator does never have the ability to steal tokens if all users come online regularly and behave correctly. This is one of the key features of this construction.

3.4 Grieving vectors and tragedy of the common

Unfortunately, our specification had to take some trade-offs. The main problem is that data-unavailability is subjective. Hence, people can always claim that data is missing. Then someone is required to publish these data on the root-chain including a Merkle proof attesting that the published data is the same as on the plasma chain. This is a problem, since providing data on the root-chain can be quite costly. Our solution is that the requester needs to pay some decent fee to the data-provider, in order to prevent grieving attacks. But this leaves the construction with a tragedy of the commons, as no one, in particular, would have the incentive to ask for the data in case of a real data-unavailability case. But still, this action would be required from someone.

Since we estimate the costs of this data-request on the root chain to be definitively lower than 10 Dollar, the tragedy of the commons is very small. There should always be a party, which request the data for everyone else. Hence, it is a reasonable trade-off.

Since the plasma chain operator will never intentionally generate these situations, as it will hurt his reputation, the incentives are well set.

3.5 Performance

There are some bottlenecks, which are considered for evaluating the performance:

- Price calculation. Finding these uniform clearing prices is a non-linear optimization problem. The solving time to find these prices increases exponentially with the number of tokens involved. For concrete timings, take a look at this paper [2]
- Generating the public verification keys for the snark proofs is a very challenging task having its own limitations. But we think that we can generate these verification keys for trade blocks with 2^{16} transactions. The calculation of the actual proof should then be feasible within a minute.
- Gas costs for bitmap and other data, which needs to be made available on the root-chain. Uploading a bitmap with 9000 entries costs about 100k gas.
- Plasma exists. Plasma exits are controversially discussed in the community. If there are too many utxo, which want to exit all at the same time, - maybe triggered by a data-unavailability -, the root chain will be under heavy load and gas prices might spike.

We think that the price calculation is the biggest bottleneck. But still, we are confident that we will be able to schedule auctions every 3 minutes with about 2^{16} orders.

3.6 Complexity

Unfortunately, the described mechanisms are quite complex. Shipping such a complex construction without any bugs will require a huge engineering effort. Hence, Gnosis is planning to cut out some features and add them only in the next versions of its product.

The following pieces can reduce the complexity:

1. DKG can be replaced with one single entity, which is very trustworthy. This single entity would only generate the public keys and later reveal the private keys for it. This does not put customers funds at risk at any point in time.

2. The described mechanism 2.2.5 making it possible to everyone to submit the best prices can be left out.

4 Open research questions

4.1 Future iterations

We hope that in future iterations, we can outsource more and more logic to the snarks proofs. This has the potential to improve the user experience a lot since verification times will be reduced and double signing processes can be omitted.

4.2 Plasma rollovers

In order to reduce the verification time for clients, a potential solution is a chain rollover. That is, all transaction outputs needs to be spend all 3 weeks. If people do not send them to themselves within 3 weeks, they will not longer be exit-able from the plasma chain. In a plasma chain, where transactions are for free, this is a viable model. It does not really come with any new burdens for the users, as the user needs to be online and check the validity of the chain anyways.

But it would have the big benefit that not the whole chain needs to be stored and processed, but only the blocks of the last 3 weeks.

References

[1] Caimu Tang *ECDKG: A Distributed Key Generation Protocol Based on Elliptic Curve Discrete Logarithm.* https://pdfs.semanticscholar.org/3c52/35523be1d305de6dbf3433965c99d9fe4aea.pdf

[2] Gnosis *Multi-token Batch auctions with uniform clearning prices* https://github.com/gnosis/dex-research/tree/master/BatchAuctionOptimization

[3] Vitalik Buterin and Joseph Poon *Plasma*. https://plasma.io/plasma.pdf

[4] Vitalik Buterin *Plasma minimal viable product*. https://ethresear.ch/t/minimal-viable-plasma/426

[5] Crypto aggregated signatures
https://ethresear.ch/t/cryptoeconomic-signature-aggregation/1659

[6] Snarks using the library Zokrates
`snarks`

[7] ECIES
`ECIES` https://crypto.stackexchange.com/questions/31602/how-does-encryption-work-in-elliptic-curve-cryptography

[8] Knudsen-Nyberg cipher
`Knudsen-Nyberg cipher` https://eprint.iacr.org/2016/492.pdf

[9] Batch Trading
https://www.investopedia.com/terms/b/batchtrading.asp