Rollups on a data-sharded Ethereum 2: linking the data availability with the execution

Thanks to Nicolas Liochon and Thomas Piellard for the review

On Ethereum 2, rollups write the transactions on the data shards, and then update the rollup state on the execution shard. This is different from Ethereum 1, where the data and the execution happens on the same shard. As a consequence, the rollup's smart contract must perform an extra validation on Ethereum 2: it must prove that the transactions put on the data shards are the same as the one used in the execution shard, without providing these transactions to the smart contract.

References

Throughout the paper we will use results from the following publications:

- Easy proof of equivalence between multiple polynomial commitment schemes to the same data
- Pointproofs: Aggregating Proofs for Multiple Vector Commitments
- Using polynomial commitment to replace state root

Previous work

This post comes as a follow up onerror correcting code to optimize zk-rollup verifier, aiming at reducing the public input size (and thus the gas cost) of a zk-rollup state update. We use different methods than this earlier proposal and do not rely on error-correcting codes anymore. Instead, we rely on sub-polynomial-commitment extraction and proof of equivalence. Our proposal covers several settings that we describe in the assumptions.

Assumptions and problem

The datasharded blockchain consists of 64 data-shards. Only one executes transactions (i.e.: verify the zk-rollup proof and updates the rollup's root hash). We call it the "main shard". The 63 others are the "data shards" and are used to store "calldata" only, and they can be used to store the public inputs of the rollup verifier (as they take a lot of space). We furthermore assume that the sharded blockchain guarantees the availability of the calldata of any block at any point in time.

We consider zk-rollups using pairing based arguments (Groth16, Marlin, Plonk…) as long as the verifier only needs to process the public inputs into one or more multi-exponentiation (what we call generically the "public input part of the verifier" throughout this post). This step is typically driving the quasi-totality of the verifier computation for rollups.

- For Plonk/TurboPlonk, we compute the public inputs part as several Lagrange Basis polynomial commitment $\Pi_{public} = \prod_{i \in \text{PubIndices}}(g^{L_{\rho^{\sigma(i)}}(\tau)})^{x_i}$

where ρ

is a root of unity. $\sigma: [[0;n]] \rightarrow [[0;N]]$

is a function that maps the the public inputs to their respective positions in a polynomial commitment.

- For Groth16, the multi-exponentiation is performed on a circuit dependant verifier key and has an opaque structure. $\Pi_{public} = \prod_{i=0}^{n}{g_i}^{x_i}$

The problem is that in order to perform this step, the verifier must read the entire public input string. And since the verifier has to run on the "main shard" (as the other shards don't run a VM), that would mean storing the public inputs in the main shard block. Our goal, in this post is to investigate how to do it by taking advantage of the data-shards.

In Eth2, the main shard checks the data-shard's blocks inclusion through an accumulator. We cover two cases:

The accumulator is a polynomial commitment and encode the data in monomial

basis (as in Pointproofs

). $acc(x_0, x_1, .., x_n) = \prod_{i=0}^{n}(g^{\tau^{i}})^{x_i}$

where $(g^{\tau^{i}})_{i\in[[0;n]]}$

are part of a public setup but not τ

- The accumulator is a polynomial commitment and encodes the data in Lagrange basis as in Vitalik’s and Justin’s proposal to use Kate commitments.

$acc(x_0, x_1, .., x_n) = \prod_{i=0}^{n}(g^{L_{\rho^{i}}(\tau)})^{x_i}$

where ρ

is a root of unity and the $L_\rho(X)$

are the Lagrange polynomials on a set of root of unity. $(g^{L_{\rho^i}(\tau)})_{i\in[[0;n]]}$

is part of a public setup but not τ

.

We suppose that all schemes rely on the same curve

and leave the case where they don’t as an open problem. However, each schemes relies on its own setup (which can intersect).

Subcommitment extraction

We are given a global commitment of the form $\prod_{i=0}^{N}{g_i}^{x_i}$

which corresponds to the data-shard block commitment. And we want to extract a subcommitment $\prod_{i=0}^{n}{g_{i}}^{x_{i+k}}$

. It corresponds to a commitment to a subvector $x[k:k+n]$

of x

. This method aims at converting a commitment to a full chunk of blockdata into a commitment to the public inputs of the rollup transactions.

For coefficient based polynomials (PointProof style)

Given $C = Com(x)$

and $C' = Com(y)$

the prover wants to prove with knowledge that $y = x[k: n+k]$

for $k + n < N = |x|$

The public parameters are:

$\text{pp} = (g_1^{\alpha}, ... , g_1^{\alpha ^ N}, g_1^{\alpha ^ {N + n + 2}}, ... , g_1^{\alpha^{2N}}, g_2^{\alpha}, ... , g_2^{\alpha^N}, g_1^{\beta\alpha}, ... , g_1^{\beta\alpha^n}, g_2^{\beta^\alpha})$

The commitments C

and C'

are defined as $C = \prod_{i=1}^{N}{g_1}^{x_i\alpha^i}$

and $C' = \prod_{i=1}^{n}{g_1}^{x_{i+k}\alpha^i}$

The prover computes the proof $\pi = (\pi_1, \pi_2) = (\prod_{i \notin [[k; k+n]]} (g_1^{\alpha^{N-k+i}})^{x_i}, \prod_{i = 0}^n (g_1^{\beta\alpha^i})^{x_{k+i}})$

The verifier checks $e(C-C', g_2^{\alpha^{N-k}}) = e(\pi_1, g_2^\alpha)$

and $e(C', g_2^{\beta\alpha}) = (\pi_2, g_2^{\alpha})$

. The second check ensures that the degree of C'

is bound by n

.

An advantage of this scheme (if we can prove its security with good assumptions) is that the setup is updatable. However, we can’t directly reuse a general public powers of tau setup for this protocol.

If we say it does not matter if C'

is built as $C' = \prod_{i=1}^{n}{g_1}^{x_{i+k}\beta\alpha^i}$

. Ie: the base group elements are not the same as the one for C

then we can optimize the protocol to reduce the verifier work to be: $e(C, g_2^{\alpha^{N-k}}) = e(\pi_1, g_2^\alpha).e(C', g_2^{\beta^{-1}\alpha^{N-k}})$

. In order to do so, we must also update the public parameters to be $\text{pp} = (g_1^{\alpha}, ... , g_1^{\alpha ^ N}, g_1^{\alpha ^ {N + n + 2}}, ... , g_1^{\alpha^{2N}}, g_2^{\alpha}, ... , g_2^{\alpha^N}, g_1^{\beta\alpha}, ... , g_1^{\beta\alpha^n}, g_2^{\beta^{-1}\alpha}, ... g_2^{\beta^{-1}\alpha^{n}})$

For Lagrange Basis Polynomials

For Lagrange basis, we use a method based on evaluation at random points. We note P

the polynomial whose values encode the full block data. And P'

encodes the subslices. P and P’ are defined over different Lagrange basis and that’s because they use different domains. As a consequence, we can avoid degree bounds checks.

The intuition of the protocol is that we want to check that: $P(X) - P'(X-k)$

is divisible by a vanishing polynomial $Z_n(X-k) = \prod_{i=0}^{n} (X-i-k)$

. Essentially we want to prove that there exists a polynomial $H(X)$

such that $P(X) - P'(X-k) = H(X)Z_n(X-k)$

.

- As public parameters, we have the public parameters of the polynomial commitments of P and P’.
- In an offline phase, the prover and the verifier computes a commitment to a vanishing polynomial $Z_n = \prod_{i=0}^n (X-i)$

using the parameters of the scheme of P.

- The prover sends a commitment to $Q(X)$

using the same setup as $P(X)$

.

- The verifier responds with a random challenges point γ

.

- The prover computes $\zeta = Z_n(\gamma - k)$

. And she sends back a polynomial opening for ζ

, an openings for the polynomials $P(X) - \zeta H(X)$

at γ

and $P'(X)$

at $\gamma - k$

along with a claim that they take the same value ϕ

- The verifier checks the three openings.

We can then compile this protocol into a non-interactive one using Fiat-Shamir. A cool feature of this protocol is that it works even if P and P’ are committed using different commitment schemes. The only restriction is that we need them be defined over the same field.

Proof of equivalence between vector commitments

Now that we know how to prove the extraction of an accumulator of the public inputs of the zk-rollup proof from an accumulator of block data. It remains for us to convert it into a public part.

Given two vector commitments $C = \prod_{i=0}^{N}{g_i}^{x_i}$

and $C' = \prod_{i=0}^{N}{h_i}^{x_i}$

in the same group, but without further restrictions on g_i

and h_i

, one can prove the equality of those committed values if he can prove knowledge of the exponents of $C + C'$

for the base groups elements $(g_ih_i)_{i = 0..n}$

. This can be done with the following protocol.

- Public parameters $\text{pp} = (g_0, g_1, ..., g_n, h_0, h_1, .., h_n, (g_0h_0)^\rho, (g_1h_1)^\rho, ..,(g_nh_n)^\rho, g_2, g_2^\rho)$
- The prover computes $\pi = \prod_{i=0}^{N}(g_ih_i)^{\rho x_i}$
- The verifier checks $e(C + C', g_2^\rho)$

$= e(\pi, g_2)$

Putting everything together

- The rollup operator sends a batch of transactions to a data shard
- Then it sends to the main shard
- A subvector commitment to the public inputs along with a proof of subcommitment.
- The public input part of the verifier along with a proof of equivalence
- The zk-rollup proof
- The update rollup state accumulator
- A subvector commitment to the public inputs along with a proof of subcommitment.
- The public input part of the verifier along with a proof of equivalence
- The zk-rollup proof
- The update rollup state accumulator

The main shard verifies the correctness of the subcommitment and its equivalence with the claimed “public input part”. Then it completes the public input part with the previous rollup state accumulator and the new one. Then it runs the verifier algorithm normally.

It should be possible to build a similar protocols for optimistic rollups fraudproof as well.