TL;DR:

In this post we discuss a low-degreeness proof for data-availability checks based of the KZG10 (short Kate) [5] polynomial commitment scheme. This is an alternative to the STARK proof suggested by @vbuterin [1]. Compared to the STARK construction, our construction shows the potential to reduce the computational requirements on full-nodes constructing Fraud Proofs. We have no dependency on STARK-friendly hash functions. As a downside we do require a trusted setup.

Background

I wrote a primer on data availability checks which should help to understand the basic problem and principles.

Data availability checks – motivation

Data availability checks are an attempt to remedy the data availability problem. The data availability problem occurs whenever a node is a light client on a chain (which, for the purpose of the sharded Ethereum 2.0 protocol, almost all nodes are on most shards). A light client verifies consensus and does not verify state – it does not even download the state. Instead, it relies on fraud proofs to be certain of correctness: Full nodes observing a chain with an incorrect state transition can broadcast a succinct message showing that fraud has occurred, alerting all light clients of the fraudulent chain.

We find fraud proofs an elegant solution for reducing the trust requirements in full nodes, but there is a catch: if the full node is not given access to the information inside the block then they cannot construct a fraud proof. We call the problem where a node cannot prove that they have not been provided with the necessary information the data availability problem

. Data availability thus has to be guaranteed by another means – data availability checks.

Data availability checks

The way data availability checks do this is by using an erasure code with a rate r

, that stretches the original data of length n

to length n/r

chunks in a way such that any qn/r

chunks are sufficient to reconstruct all blocks, where q

is the stopping rate. A light client wanting to check that the data is available will use local entropy to sample k

chunks at random and check that it can download these chunks. If an attacker published a fraction less than q

of the erasure coded data, the probability of all k

randomly sampled chunks being available is less than q^k

. The light client thus knows that if the check passes, it is very likely that anyone could reconstruct the full data by downloading qn/r

blocks.

We consider the use of Reed-Solomon code where one stretches the data to m

blocks using a one-dimensional polynomial of degree n

. Then $q=r=n/m$

, which is optimal for our purposes.To use this solution we additionally require some assurance to the validity of these codes. If this is not the case (the producer cheated), a light client still knows that qn/r

blocks can very likely be downloaded by doing the checks, but these blocks may not allow reconstruction of the data.

Fraud proofs can be used to prove correct encoding, but in the naive Reed-Solomon scheme, $n+1$

blocks need to be provided as a fraud proof, which is quite big and inefficient. Research on data availability has thus recently centered on decreasing the size of these fraud proofs or removing their need entirely. Among these are:

1. The approach by al-Bassam et al. [2], which encodes the data on a two-dimensional instead of on a one-dimensional polynomial. Any violation of the low-degreeness can thus be proved by providing a single row or column of evaluations of that polynomial. However, in this case, $q=\sqrt{r}$

, making the checks less efficient than in the optimal one-dimensional case

1. The Yu et al Coded Merkle Tree approach [3]. Instead of Reed-Solomon codes, it uses linear codes that are optimized for small fraud proofs. However, the coding is also quite inefficient, achieving a stopping rate of $q=0.875$

at $r=0.25$

.

1. Have the creator of the code provide Zero-Knowledge proof of the low-degreeness of the (univariate) polynomial, and the correct computation of the Merkle root [1], using a STARK construction. Computing this STARK is extremely expensive, even when using STARK-friendly hash functions for the Merkle root (STARK-friendly hash functions are currently considered to be experimental).

2. Have the creator of the code provide a FRI (the polynomial commitment scheme behind STARKs) attesting to the low-degreeness of the polynomial, but do not prove the full computation of the Merkle root. Then additionally rely on fraud proofs to guarantee full correctness. This fraud proof can itself be a FRI [8].

The advantage of implementing #3

are obvious: If the correctness of the encoding can be proved, it is not necessary to implement fraud proofs, simplifying the protocol and getting rid of the synchrony assumption, at least where the data availability check is concerned. However, the difficulty with this has turned out to be that it is currently prohibitively expensive to compute a Merkle tree inside a ZKP using SHA256 or any other well-established hash functions. The construction suggested by Vitalik usies MiMC, a hash function based on a symmetric key primitive that is only about three years old, which is relatively little time for cryptanalysis. Indeed, it seems currently quite dangerous to use any of the STARK-friendly constructions as the basis for all of Ethereum 2.0. So likely this will result in two roots having to be included:

1. The data availability root, which is guaranteed to be computed from a valid low-degree encoding and

2. A separate data root, using a traditional hash function like SHA256. All consensus and application layer Merkle proofs will want to use this root, as otherwise an attack on the hash function used in the data availability root would be catastrophic for the whole protocol (and potentially unfixable, if the attack can be used to generate arbitrary second preimages).

The compatibility of the two roots can unfortunately not be guaranteed by the ZKP, as this would entail computing the ZKP of the Merkle tree of SHA256, which we wanted to avoid. So instead, we will have to rely on fraud proofs. Effectively, unfortunately, this means that this construction will still require fraud proofs for the low degree proof, however the fraud proofs can be very small (only two Merkle branches).

In this post, I will present a construction based on the Kate polynomial commitment scheme that achieves the same property, but without the requirement for a STARK-friendly hash function.

Kate commitment scheme with multi-reveal

The Kate polynomial commitment scheme [5] is a way to commit to a polynomial $f(x)=\sum_{i=0}^n f_i x^i$

and decommitting a value $y=f(r)$

(i.e. giving y

and a witness that proves that it is the evaluation of the previously committed polynomial at r

). The witness is a single group element. I will here present a slight variation that allows decommitting multiple evaluations using just one group element.

Let $e: G_1 \times G_2 \rightarrow G_t$

be a pairing of two elliptic curves G_1

and G_2

with generators $G \in G_1$

and $H \in G_2$

(I use additive notation for group operations). We need a trusted setup that provides $s^i G$

and $s^i H$

for $i=0,\ldots, n$

where d

is the degree of the polynomial to be committed to and s

is a random value not known to anyone (called the structured reference string or SRS

).

As a matter of notation, we will denote by f//g

the whole part of the polynomial division of f

by g

and by f \% g

the remainder. So

\displaystyle f(x) = f // g g(x) + f \% g

An interesting fact is that if g(x) = \prod_{z\in S} (x-z)

, then f \% g

is completely determined by the values that f

takes on S

. The reason for this is that when computing \mod g

, all polynomials that take the same values on S

are congruent. We thus get the unique representative with degree <#S

by Lagrange interpolation on the set S

(which can easily be computed by anyone who knows the values of f

on S

).

Commitment:

The Kate commitment to a polynomial is

\displaystyle C = \sum_{i=0}^n f_i s^i G \text{ .}

Let's say we want to prove evaluations of this polynomial at y_0 = f(x_0), \ldots , y_{m-1} = f(x_{m-1})

. Let S = {x_0, ..., x_{m-1}}

and Z_S(x) = \prod_{z\in S}(x -z)

. To prove that the polynomial was correctly evaluated, the prover provides the group element \pi = f // Z_S G

, which can be computed from the SRS. The verifier will check that the values are correct by computing the value f \% Z_S G

, which he can do by only knowing the values of f

on S

using Lagrange interpolation.

This can then can be checked using the pairing equation

\displaystyle e(\pi, Z_S(s) H) e(f \% Z_S G, H) = e(C, H) \text{ .}

Naïve Kate-based scheme

One obvious way in which we can apply Kate is to compute a full polynomial commitment for a polynomial of degree n

on all m

chunks, where n

is the amount of data that needs to be encoded. For simplicity, let's assume that a chunk is 32 bytes of data, although that will never exactly align with field elements – some additional engineering will be required at the end to make the data fit, but we'll ignore that at the moment.

Now let's assume that clients will download the data in blocks of b

chunks. The producer will create a Kate proof for each such block, and include that in the Merkle tree together with the block. Each block will thus come with its own proof of lying on the polynomial $f(x)$

committed to by the commitment C

and can be checked in isolation. Fraud proofs are very easy as any block can be individually checked – if the proof π

does not match the data this is enough to prove fraud.

A second kind of fraud proof is needed where some data that was incorrectly computed is not available. In this case, the fraud prover would have downloaded enough data blocks to compute the missing data, and can give the complete data, up to the Merkle root where it is missing, including the Kate proofs for that data. This is enough to convince any other node that this is the correct data. (The fraud prover can compute Kate proofs in this case because they have enough data to reconstruct the polynomial).

Efficiency

This construction is probably almost optimal from the verifier perspective. The data overhead is low (only one group element for the commitment and one group element per block is required). The verifier will need to compute a few pairings to check this (one per downloaded block), but this can be reduced using some tricks. They will also need to do a small number of group operations to compute the remainder polynomial.

The main difficulty lies in the work that the producer has to do. Let's add some numbers using the current phase 1 rework [6]:

Degree of the polynomial to be evaluated: Average: $1.5 \cdot 64 \cdot 2^{18}/32 = 786{,}432$

, Max $12\cdot4\cdot64\cdot2^{18}/32 = 25{,}165{,}824$

This is important because it tells us the number of group multiplications required to compute one commitment. Incidentally, a Kate proof requires the same number of operations (the proof is a polynomial commitment with only a slightly lower degree). Using the estimate of $0.25 \mathrm{ms}$

for a group multiplication in BLS12-381 (G_1

), and Pippenger's algorithm which can compute n

multiplications (when only their sum is required) at the cost of $n/\log_2(n)$

multiplications, we get:

Commit/reveal cost average case: ca. 10 s

Commit/reveal cost worst case: ca. 256 s

Number of places where the polynomial will be evaluated ($r=q=50\%$

): Average: $2\cdot1.5\cdot64\cdot2^{18}/32 = 1{,}572{,}864$

, Max $2\cdot12\cdot4\cdot64\cdot2^{18}/32 = 50{,}331{,}648$

This divided by the block size b

is the number of proofs that will have to be computed. For example choosing $b=32$

(a reasonable tradeoff making block sizes and their Merkle proofs similar in sizes), we get a total computation time

Commit/reveal cost average case: ca. 134 h

Commit/reveal cost worst case: ca. 111,807 h

This can be slightly alleviated by choosing a larger block size. By for example choosing $b=1024$

, we can reduce the prover cost at the expense of the verifiers having to download larger blocks:

Commit/reveal cost average case: ca. 4.3 h

Commit/reveal cost worst case: ca. 3494 h

While it can be parallelized, the total cost of this computation, which would have to be performed once per beacon chain block, is quite steep. This leads us to look for another solution to this problem.

Kate domain decomposition

The main problem with the schemes above is the cost of revealing in the Kate polynomial scheme. Here I will present a new scheme that almost completely avoids having to compute any Kate proofs.

Let’s assume we want to reveal a polynomial $f(x)$

on a domain $\mathbb{D}$

with $\deg f + 1 = \# \mathbb{D} = n$

. In this case, the verifier can easily check the commitment C

by just computing it from the evaluations on $\mathbb{D}$

– the easiest way is if we include the Lagrange polynomials for $\mathbb{D}$

in the SRS:

$\displaystyle L_{z, \mathbb{D}} = \ell_{z, \mathbb{D}} (s) G$

where

$\displaystyle \ell_{z, \mathbb{D}} (x) \prod_{w \in \mathbb{D}\ z\ne w} \frac{x - w}{z - w}$

are the Lagrange interpolation polynomials on $\mathbb{D}$

. Then $C = \sum_{x \in \mathbb{D}} f(x) L_{x, \mathbb{D}}$

.

Evaluating on several domains

If we want to open the polynomial on two domains $\mathbb{D}^{(1)}$

and $\mathbb{D}^{(2)}$

of size n

, we can do this by having two sets of Lagrange interpolations prepared and checking that $\sum_{x \in \mathbb{D}^{(1)}} f(x) L_{x,\mathbb{D}^{(1)}} = C = \sum_{x \in \mathbb{D}^{(2)}} f(x) L_{x, \mathbb{D}^{(2)}}$

. This is essentially a simple way to check that evaluations on two domains of size n

were of the same polynomial.

Evaluating on a domain decomposition

Now let’s assume that $\mathbb{D} = \bigcup_{i} \mathbb{D}_i$

is a disjoint union. Then we can compute the commitment of f

on each subdomain $\mathbb{D}_i$

via Lagrange interpolation as $C_i = \sum_{x \in \mathbb{D_i}} f(x) L_{x, \mathbb{D}}$

. This commitment is for a polynomial that evaluates to f

on $\mathbb{D}_i$

, and to 0 on $\mathbb{D} \setminus \mathbb{D}_i$

. Also $\sum_i C_i = C$

. Now we can actually check that C_i

is indeed a commitment that evaluates to zero outside $\mathbb D_i$

– by showing that it is a multiple of $Z_{\mathbb{D} \setminus \mathbb{D}_i}$

, the polynomial that is zero on all points of $\mathbb{D} \setminus \mathbb{D}_i$

. We do this by the producer additionally providing the proof $P_i = \sum_{x \in \mathbb{D_i}} f(x) L_{x, \mathbb{D_i}}$

.

The way this helps us is by allowing is to introduce a polynomial decommitment layer with much lower cost to the prover than computing Kate decommitments: Let's say our data availability Merkle tree is κ

layers deep (the average case will be $\kappa=21$

, worst case $\kappa=26$

). Let's decompose our evaluation domain at Merkle layer $\lambda = \lceil \kappa / 2\rceil$

, so that there are $f=2^\lambda$

domains, $\mathbb{D}^{(1)}_0, \ldots, \mathbb{D}^{(1)}_{f/2 - 1}, \mathbb{D}^{(2)}_0, \ldots, \mathbb{D}^{(2)}_{f/2 - 1}$

(assuming $r=q=0.5$

). Then we let the producer provide the commitments $C_i^{(*)}$

and proofs $P_i^{(*)}$

that show that they are only local interpolations.

Efficiency

In this way, we can gain a huge amount of prover efficiency – the cost of computing this will be only about three times that of computing one Kate commitment.

However, the verifier will obviously have to do more work. First, they will have to download all the commitments C_i

and proofs P_i

for the domain. This will be $2*48*2^\lambda$

bytes, so 192 kbytes in the average case and 768 kbytes in the worst case.

The data blocks for availability testing in the average case will be $32*2^{\kappa - \lambda}$

bytes, so 32 kbytes in the average case and 256 kbytes in the worst case. We probably want to improve this, as downloading several mbytes of samples would be required for the data availability check in the worst case.

Note that normal clients only need to do a minimum amount of group operations as all of these can be covered by small fraud proofs. We would probably want them to check $\sum_i C_i^{(*)} = C$

and do the pairing checks only for the blocks that they are downloading for data availability checks, where they will also have to compute the corresponding $C_i^{(*)}$

to check the polynomial commitment.

Improvement – two layer domain decomposition

We can improve on this scheme by doing the decomposition at two Merkle layers $\lambda_1 = \lceil \kappa/3 \rceil$

and $\lambda_2 = \lfloor 2\kappa/3 \rfloor$

. All clients will download the full decomposition at layer λ_1

, but they will only download the parts of λ_2

where they do data availability samples. More precisely, if the decomposition at λ_1

is \mathbb{D}^{(1)}*0, \ldots, \mathbb{D}^{(1)}*{f/2 - 1}, \mathbb{D}^{(2)}*0, \ldots, \mathbb{D}^{(2)}*{f/2 - 1}

, we will further decompose each \mathbb{D}^{(*)}_i

into \mathbb{E}^{(*)}_{i,j}

for j=1,\ldots,g-1

. A client will sample the blocks at the \mathbb{E}^{(*)}_{i,j}

level but will also download all the siblings \mathbb{E}^{(*)}_{i,j}

for j=1,\ldots,g-1

to make sure they are available.

The producer will only need to do a small amount of extra work at a cost which is now roughly five times that of one Kate commitment. However, the cost for the verifier goes down significantly in terms of the data that is required to download:

1. 2*48*2^{\lambda_1}

bytes for the first layer, which means 48 kbytes in the worst case

1. Per data availability sampling, 2*48*2^{\lambda_2 - \lambda_1} + 32 * 2^{\kappa - \lambda_2 - \lambda_1}

, which is 40 kbytes in the worst case

In addition to this, there is some overhead for Merkle commitments to the data and the \mathbb{E}^{(*)}_{i,j}

commitments.

These seem fairly acceptable. All possible fraud cases can be covered by small fraud proofs of at most 10s of kbytes (the worst case being fraudulent data missing on an \mathbb{D}^{(*)}_i

or \mathbb{E}^{(*)}_{i,j}

domain).

SRS size

To do all the computations efficiently, we need to have an expanded SRS. Here are some estimates using BLS12-381 (G_1

element size 48 bytes, G_2

element size 96 bytes) and a degree of 25{,}165{,}824

at a rate of 0.5:

1. G_1

and G_2

monomials: 3.375 GB

1. G_1

Lagrange basis plus interval proofs at layers \lambda_1

and \lambda_2

: 9 GB

1. G_1

Lagrange polynomials at the bottom layer: 2.25 GB

1. Interval checking polynomials in G_2

at \lambda_1

and \lambda_2

: 12 MB

However, not everyone needs the full SRS for their work:

- Updaters need 1-4 (The SRS is updatable as 2-4 can all be computed from the monomials in 1)
- Producers and fraud provers need 2 and 3
- Verifiers need 3 and 4

Comparison to other schemes

Advantages over STARK scheme [1]

- No dependency on STARK-friendly hash function
- Producer computation is more manageable

Advantages over FRI scheme [8]

- Does not require any "big" fraud proofs – even where a large chunk of data is missing, this can be done using a small fraud proof

Disadvantages

- Needs trusted setup (updatable)
- Large SRS due to Lagrange interpolated precomputes

[1] STARK-proving low-degree-ness of a data availability root: some analysis

[2] https://arxiv.org/abs/1809.09044

[3] https://arxiv.org/abs/1910.01247

[4] https://eprint.iacr.org/2016/492

[5] https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf

[6] https://github.com/ethereum/eth2.0-specs/pull/1504

[7] https://eprint.iacr.org/2019/1177.pdf

[8] FRI as erasure code fraud proof