An introduction to hash-based proof systems

A short explanation of how they work

Diego Kingston
Co-founder/Head of Research, Aligned

Introduction

Why are hash-based proof systems important?

Why are hash-based proof systems important?

Hash-based proof systems are used by many projects nowadays, helping in the development of performant zkvm -> this makes developing zkapps easier ans faster.

What are the properties that hash-based proof systems offer?

1. Can work over small finite fields.
2. Don't need trusted setup.
3. Minimal security assumptions -> security depends on hash functions.
4. Easier to generate recursive proofs (no field emulation or EC operations)

Ingredients for hash-based proof systems

The following are the main ingredients for a hash-based proof system:

a. Linear codes (for example, Reed-Solomon codes)
b. Collision resistant hash function
c. An arithmetization procedure (for example, AIR)

The choice of the concrete linear code, the CRH and the arithmetization affect the efficiency of the proof system, determine proof size, ease of recursion, prover time, verifier time.

Examples of hash-based proof systems

These are some of the popular hash-based proof systems:

1. STARKs/FRI (FFT-friendly field)
2. Circle STARKs (Mersenne primes)
3. Ligero
4. Brakedown (Field agnostic)
5. Binius (Binary fields)
6. FRI-Binius (Binary fields)

Application

Using Reed-Solomon codes to build a PCS

A commitment scheme from Reed-Solomon codes

We can compile our program to a set of polynomial equations that have to be fulfilled (arithmetization).

We can boil down the correctness of the execution of the program to showing that a polynomial p(x) has zeros over some set, or evaluates to some value at z.

How can we commit to p(x)? Choose a domain D (much larger than the degree of the polynomial) and compute the evaluations of p(x) over D (Reed-Solomon encoding).

To show we are not cheating, we use a Merkle tree to commit to these evaluations.

Prove evaluation

Say that the verifier wants us to give him the evaluation of $p(x)$ at z and be convinced that we evaluated $p(x)$ correctly.

We can show that $p(z) = v$ if and only if $(p(x) - v)/(x - z) = q(x)$ is a polynomial

We can commit to the polynomial $q(x)$ repeating the same idea as before, but how do we know that the list of values corresponds to the evaluations of a polynomial?

One naïve way would be to send all the coefficients of $q(x)$, but this makes the proof very large! Solution: FRI

Fast Reed-Solomon IOPP (FRI)

We can solve the problem of showing that $q(x)$ is a polynomial by showing that it is close to a low-degree polynomial, doing statistical sampling.

We can use a decomposition into odd and even powers,
$q(x) = q_e(x^2) + xq_o(x^2)$

We let the verifier choose random a_0 and we compute a new polynomial,
$q_1(y = x^2) = q_e(y) + a_0 q_o(y)$
The degree of $q_1(y)$ is half the degree of $q(x)$! So we could test proximity easier.
We can do the same trick recursively until $q_n(x)$ is constant

FRI - continued

Two phases:

1. Commitment: provide the root to the Merkle tree of evaluations of each $q_i(x)$
2. Decommitment/query phase: the verifier chooses points x_j in D, and the prover responds with the evaluations of $q_i(x_j^i)$ and $q_i(-x_j^i)$, together with their authentication paths.

The proof is polylogarithmic in the degree of the polynomial ($O(\log^2(n))$), and the prover time is $O(n\log(n))$

Unlocking faster arithmetic

Mersenne primes have the fastest known finite field arithmetic. However, we cannot use them in regular STARKs due to lack of nice domains to support FFT algorithm.

Solution: Move to the circle, and unlock a special FFT.

High-level the protocol looks the same, but has more complex mathematical functions.

Thank you!

Diego Kingston

Co-Founder/Head of Research, Aligned

@zkdiegokingston