Zpoken, OU.
Harju maakond, Tallinn,
Kesklinna linnaosa, Sakala tn 7-2, Estonia, 10141

ZKP BASED LIGHT CLIENT RESEARCH

By Zpoken team (Anton Yezhov Andrii Tikhonov **)**

Monthly progress (June & July 2023)

Different proof systems have a vast list of features and limitations. The Plonky2 proof system was initially chosen because of its ability to create recursive proofs. This makes it possible to create compact proofs of large computations and easily accommodate various private and public inputs. However, during testing, it turned out that on-chain verification of such proofs requires too much gas in the Ethereum network. It is closely corresponding with other research and previous calculations.

Therefore, we spent a time researching other proof systems that have a cheaper way of verifying on the blockchain. Of these systems, only Circom was affordable and had the ability to integrate another proof system into it. The gas price could be explained by the small proof size, types of underlying computations, and number of constraints. That is why we decided to use Circom.

To implement the transition between systems, you need to create a circuit in Circom that verifies computation of proof verification of Plonky2. The exact approach of nested verification is the following:

1. Create Plonky2 proof of the desired computation:
    a. Build circuit.
    b. Generate proof.
    c. Convert circuit and proof into JSON for Circom input.
2. Create Circom proof of the Plonky2 verification:
    a. Generate trusted setup. Groth16 requires a separate trusted setup for each circuit. This generates a common reference string (CRS), which is stored in a .zkey file.
    b. Verify final zkey.
    c. Export verification key. The verifier does not need the full zkey to verify a Groth16 proof. They only require a shorter verification key.
    d. Generate witness for Plonky2 data (1.c).
    e. Generate proof with witness and verification key.
3. Write a solidity contract for Circom verification.

Zpoken, OU.
Harju maakond, Tallinn,
Kesklinna linnaosa, Sakala tn 7-2, Estonia, 10141

4. Verify on-chain Circom proof that verifies validity of Plonky2 circuit with corresponding data.

Circom implementation of Plonky2 verification requires following the exact algorithm. The best way to accomplish it is to break down verification to separate circuits that copy behavior of different modules or even functions used in the Plonky2 library. For instance, All FRI completed inside one Circom circuit. Almost every library function from that module is implemented as a template (circuit function analogy) in Circom. The main problem of Circom implementation is the abundance of signals to provide all data movement and operations. This leads to a large size of the final circuit and corresponds to an extremely long ZKEY generation – 3+ hours on a production server with 128 GB of RAM.

Furthermore, to implement the transition between the systems, it was necessary to implement basic primitives to reproduce the verification calculations. These include Goldilocks, Poseidon, Quadratic extension, and various types of specific gates. We have projected and tested each scheme. Here is an example of how the operation is implemented within Circom's DLS. Here example of reimplementation of the reduce function for the field operations:

Plonky2 reduce (Rust)	Circom reduce

fn reduce128(x: u128) -> GoldilocksField
{
let (x_lo, x_hi) = split(x);
let x_hi_hi = x_hi >> 32;
let x_hi_lo = x_hi & EPSILON;
 	
let (mut t0, borrow) = x_lo.overflowing_sub(x_hi_hi);
if borrow {
branch_hint();
t0 -= EPSILON;
}
let t1 = x_hi_lo * EPSILON;
let t2 = unsafe { add_no_canonicalize_trashing_input(t0, t1) };
GoldilocksField(t2)
}
	template GlReduce(N) {
	signal input x;
	signal output out;
	
	var r = x % Order();
	var d = (x - r) \ Order();
	out <-- r;
	signal tmp0 <-- d;
	tmp0 * Order() + out === x;
	
	component c0 = LessNBits(N);
	c0.x <== tmp0;
	component c1 = LessNBits(64);
	c1.x <== out;
	}

Zpoken, OU.
Harju maakond, Tallinn,
Kesklinna linnaosa, Sakala tn 7-2, Estonia, 10141

Plonky2 GL multiplication	Circom GL multiplication

fn mul(self, rhs: Self) -> Self {
reduce128((self.0 as u128) *
(rhs.0 as u128))
}
	template GlAdd() {
	signal input a;
	signal input b;
	signal output out;
	
	component cr = GlReduce(1);
	cr.x <== a + b;
	out <== cr.out;
	}

Plonky2 GL extension multiplication	**Circom GL extension multiplication**

fn mul(self, rhs: Self) -> Self {
let Self([a0, a1, a2, a3]) = self;
let Self([b0, b1, b2, b3]) = rhs;
let c = ext4_mul([a0.0, a1.0, a2.0, a3.0], [b0.0, b1.0, b2.0, b3.0]);
Self(c)
}
	template GlExtMul() {
	signal input a[2];
	signal input b[2];
	signal output out[2];
	
	component a0_mul_b0 = GlMul();
	component a1_mul_W = GlMul();
	component a1_mul_W_mul_b1 = GlMul();
	component a0_mul_b1 = GlMul();
	component a1_mul_b0 = GlMul();
	component cadd0 = GlAdd();
	component cadd1 = GlAdd();
	
	a0_mul_b0.a <== a[0];
	a0_mul_b0.b <== b[0];
	a1_mul_W.a <== a[1];
	a1_mul_W.b <== W();
	a1_mul_W_mul_b1.a <== a1_mul_W.out;
	a1_mul_W_mul_b1.b <== b[1];
	a0_mul_b1.a <== a[0];
	a0_mul_b1.b <== b[1];
	a1_mul_b0.a <== a[1];
	a1_mul_b0.b <== b[0];
	
	cadd0.a <== a0_mul_b0.out;
	cadd0.b <== a1_mul_W_mul_b1.out;

	cadd1.a <== a0_mul_b1.out;
	cadd1.b <== a1_mul_b0.out;
	
	out[0] <== cadd0.out;
	out[1] <== cadd1.out;
	}

As you can see, some examples are more compact and some aren't. Overall Circom tends to grow in size faster with increasing complexity than regular Rust code.

Zpoken, OU.
Harju maakond, Tallinn,
Kesklinna linnaosa, Sakala tn 7-2, Estonia, 10141

Here are overall system diagrams. The next one describes aggregation of data by Plonky2.

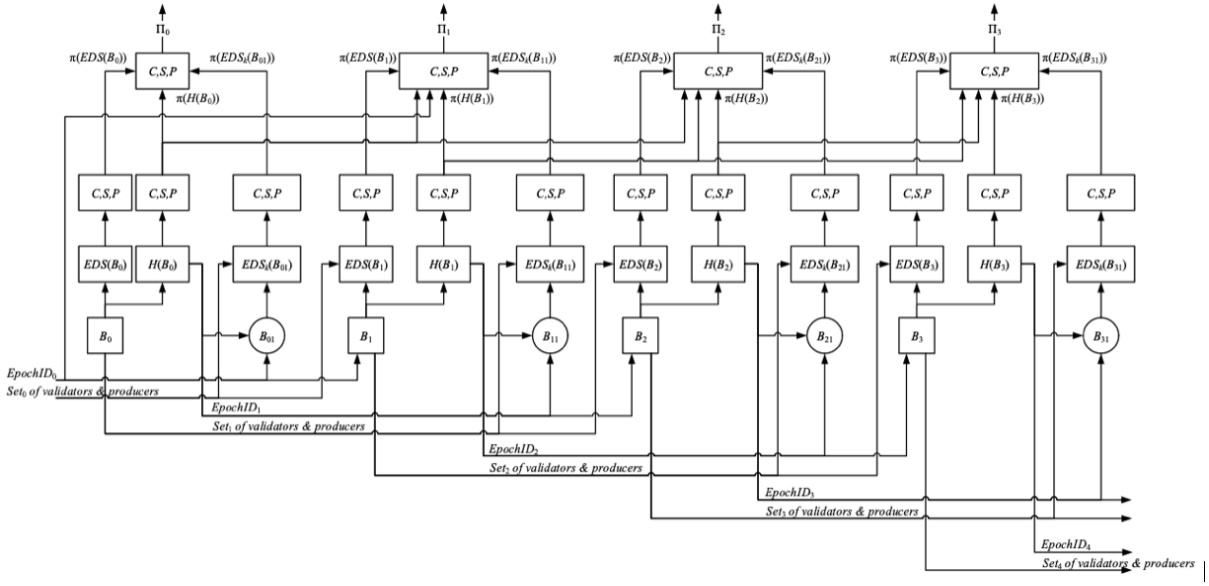
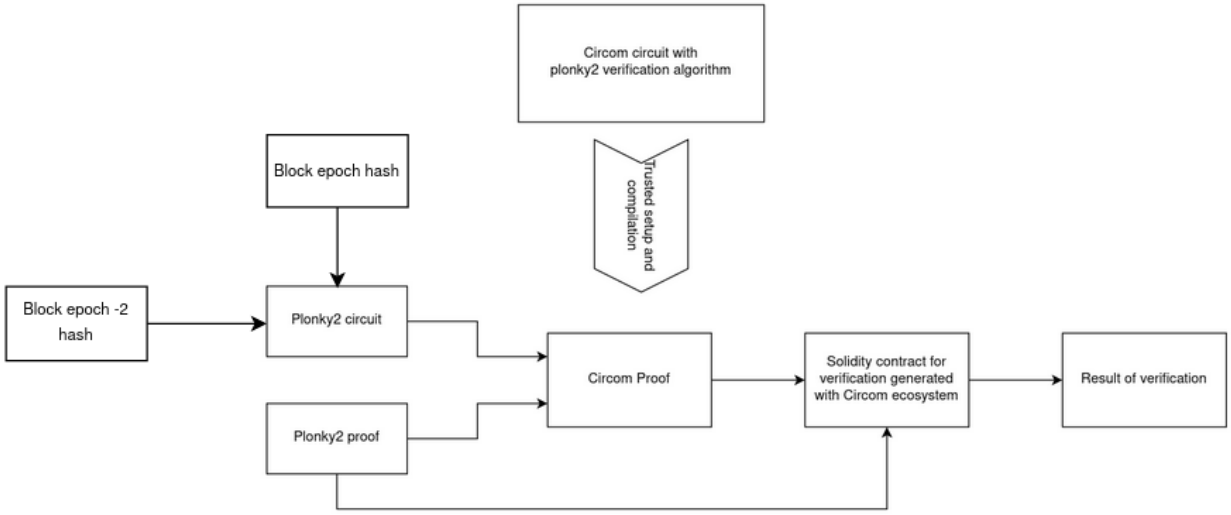


Fig. 1 – Cryptographic chain of epochal blocks (from previous reports)

This one describes the usage of Circom part



Summary

We have performed a study of the capabilities of heterogeneous proof systems. During the work, we managed to successfully conduct e2e testing of the entire system. Thus, we supplemented the previous development with an additional stage of proof in another system and completed the development of the overall architecture. Based on the results, we can consider the study successful and the system effective. With the new approach, we have obtained a significant reduction in the amount of gas required to verify a proof of stake in the Ethereum blockchain.

Zpoken, OU.
Harju maakond, Tallinn,
Kesklinna linnaosa, Sakala tn 7-2, Estonia, 10141

Appendix A — Comparison table

Proof system	Circom	Plonky2
Arithmetic	R1CS	Plonk
Algorithm	Groth16	FRI
Field	BN254	Goldiloks
Available config		Bits size of proof Sizes of wires/queries/routed wires
Feature	Sublinear verification time	Recursion
Num of constraints (sha 256 circuit), millions	32	8
Proof time generation	Of the same order, depends on config	
Setup	Trusted, long computation	No trusted, fast
Proof size	4K	132KB
Gas verification cost	790K	1100K