Plonk unroller for Ethereum

2020

Abstract

1 Main argument unrolled

1.1 Transcript implementation

To make verification non-interactive one has to use FS transformation. For this purpose we instantiate a transcript using keccak256 hash function that is a PRF hash function. We keep a two 32 byte variables $state_0$ and $state_1$, single $uint32$ variable $counter$, and create constants (domain separation tags) $DST_0 = uint32(0)$, $DST_1 = uint32(1)$, $DST_{chall} = uint32(2)$ and instantiate the following five functions ($uint32$ variables are encoded as BigEndian when used as byte strings):

1. $Commit(data)$: takes 32 bytes of $data$ and updates a state as

   $$tmp := state_0$$

   ,

   $$state_0 := keccak256(DTS_0||tmp||state_1||data)$$

   ,

   $$state_1 := keccak256(DTS_1||tmp||state_1||data)$$

   . Initially $state_0 = bytes32(0)$ and $state_1 = bytes32(0)$.

2. Commit $uint256$ scalar s: scalar is $uint256$ type that is in turn encoded as Big Endian 32 bytes. Thus commitment updates a state as

   $$Commit(EncodeBE32(s))$$

3. Commit scalar field element Fr: On BN254 curve used in Ethereum scalar field element is encoded as $uint256$ type that is in turn encoded as Big Endian 32 bytes. Thus commitment updates a state as

   $$Commit(EncodeBE32(Fr))$$

4. Commit curve point P: On BN254 curve used in Ethereum base field element is encoded as $uint256$ type that is in turn encoded as Big Endian 32 bytes. Commitment of the curve point (in affine form with coordinates x and y) is a sequence of steps

$$Commit(EncodeBE32(P.x))$$

,

$$Commit(EncodeBE32(P.y))$$

5. $GetChallenge$: draws a random challenge to perform aggregation over values of the scalar field Fr. To ensure uniform distribution of the challenge we do not use modular reduction. Bit length of the scalar field modulus is 254 bits on BN254 curve, so we take

$$challenge = uint256(keccak256(DST_{chall}||state_0||state_1||counter))$$

$$counter+ = 1$$

and take lowest 253 bits are of received $challenge$

1.2 Protocol

Before the verification there is a setup phase when selector and permutations polynomials are defined and the corresponding commitments are output for the polynomials $q_a(x)$, $q_b(x)$, $q_c(x)$, $q_d(x)$, $q_m(x)$, $q_{const}(x)$, $q_{d_{next}}(x)$, $\sigma_0(x)$, $\sigma_1(x)$, $\sigma_2(x)$, $\sigma_3(x)$, as well as quadratic non-residues $k_{1,2,3}$ are agreed on.

One can refer to the separate document for expanded equations that make the main argument. Here we only unroll the sequence of steps to perform the verification. We use a bar symbol like $\bar{a}$ to define a commitment value to the polynomial $a(x)$ using Kate commitment.

Terms "transcript is updated" and "prover sends ... to verifier" are intrinsically synonims as this is unrolled non-interactive protocol. Explicit instantiation of the transcript described above is just a particular choice. In non-interactive version prover would send the same values to the verifier and verifier would send truly random challenges to the prover.

1. Prover outputs commitments to witness polynomials $a(x)$, $b(x)$, $c(x)$, $d(x)$ using Kate commitment

2. Transcript is updated by committing curve points $\bar{a}$, $\bar{b}$, $\bar{c}$, $\bar{d}$

3. Challenges β, γ are drawn from the transcript using $GetChallenge$

4. Prover constructs permutation argument polynomial $z(x)$ using β and γ and commits to it

5. Transcript is updated by committing curve point $\bar{z}$

6. Challenge α is drawn from the transcript using *GetChallenge* to construct $T(x)$ polynomial
7. Prover constructs $T(x)$ polynomial of degree $4n$ and splits it into four polynomials of degree n as $t_{0,1,2,3}(x)$ and commits to them
8. Transcript is updated by committing curve points $\bar{t_0}$, $\bar{t_1}$, $\bar{t_2}$, $\bar{t_3}$
9. Random point z is drawn from the transcript using *GetChallenge* to check for relationships between polynomials
10. Prover constructs linearization polynomial $r(x)$ and claims values $a(z)$, $b(z)$, $c(z)$, $d(z)$, $d(z\omega)$, $\sigma_0(z)$, $\sigma_1(z)$, $\sigma_2(z)$, $r(z)$ and $T(z)$
11. Transcript is updated by committing claimed values (scalar field elements) in the same order
12. Challenge v is drawn from the transcript using *GetChallenge*
13. Prover performs aggregated multiopening of the polynomial by constructing

$$W(x) = \frac{1}{x-z}[$$

$$t_3(x)z^3n + t_2(x)z^2n + t_1(x)z^n + t_0(x) - t(z)$$
$$v(r(x) - r(z))$$
$$v^2(a(x) - a(z))$$
$$v^3(b(x) - b(z))$$
$$v^4(c(x) - c(z))$$
$$v^5(d(x) - d(z))$$
$$v^6(\sigma_0(x) - \sigma_0(z))$$
$$v^7(\sigma_1(x) - \sigma_1(z))$$
$$v^8(\sigma_2(x) - \sigma_2(z))$$
$$]$$

and

$$W'(x) = \frac{1}{x-z\omega}[$$

$$v^9(z(x) - z(z\omega))$$
$$v^{10}(d(x) - d(z\omega))$$
$$]$$

that are quotient polynomials for opening of Kate commitment (effectively proofs of opening) and output commitments to them $\bar{W}$ and $\bar{W}'$.

14. Transcript is updated by committing curve points $\bar{W}$ and $\bar{W}'$.

15. Challenge u is drawn from the transcript using $GetChallenge$ (only verifier actually performs this step as described below).

16. All the values that were committed to the transcript are part of the proof, so verifier trivially reconstructs the transcript and pulls all the values from there (e.g. α, β, ...) to perform the checks below.

17. Verifier checks an equation

$$\begin{aligned} T(z) = \frac{1}{Z_H(z)}[r(z) + PI(z) \\ - \alpha z(z\omega)(a(z) + \beta\sigma_0(z) + \gamma)(b(z) + \beta\sigma_1(z) + \gamma) \\ (c(z) + \beta\sigma_2(z) + \gamma)(d(z) + \gamma) \\ - \alpha^2 L_0(z)] \end{aligned} \tag{1}$$

where $PI(z)$ is a value of the public inputs polynomial as described in the original paper. If this check fails verifier returns "false".

18. Verifier keeps a track of claimed values and reconstructs $\bar{r}$ himself

$$\begin{aligned} \bar{r} = \bar{q_a}a(z) + \bar{q_b}b(z) + \bar{q_c}c(z) + \bar{q_d}d(z) \\ + \bar{q_m}a(z)b(z) + \bar{q}_{const} + \bar{q}_{d_{next}}d(z\omega) \\ + \alpha[\bar{z}(a(z) + \beta z + \gamma)(b(z) + \beta k_1 z + \gamma)(c(z) + \beta k_2 z + \gamma)(d(z) + \beta k_3 z + \gamma) \\ - z(z\omega)(a(z) + \beta\sigma_0(z) + \gamma)(b(z) + \beta\sigma_1(z) + \gamma)(c(z) + \beta\sigma_2(z) + \gamma)\beta\bar{\sigma_3}] \\ + \alpha^2[\bar{z}L_0(z)] \end{aligned}$$

19. Verifier aggregates pairing checks for opening equations into one using u as a random challenge by reconstructing a right hand side of equations for $W(x)$ and $W'(x)$. Reconstruction happens "in the exponent", so we use a symbol [1] to define multiplication by the generator of G1 group of BN254

$\bar{E} = [$

$$\bar{t_3} z^3 n + \bar{t_2} z^2 n + \bar{t_1} z^n + \bar{t_0} - t(z)[1]$$
$$v(\bar{r} - r(z)[1])$$
$$v^2(\bar{a} - a(z)[1])$$
$$v^3(\bar{b} - b(z)[1])$$
$$v^4(\bar{c} - c(z)[1])$$
$$v^5(\bar{d} - d(z)[1])$$
$$v^6(\bar{\sigma_0} - \sigma_0(z)[1])$$
$$v^7(\bar{\sigma_1} - \sigma_1(z)[1])$$
$$v^8(\bar{\sigma_2} - \sigma_2(z)[1])$$

$]$

$\bar{F} = [$

$$v^9(\bar{z} - z(z\omega)[1])$$
$$v^{10}(\bar{d} - d(z\omega)[1])$$

$]$

20. Verifier performs actual aggregated pairing check. We use [2] and $[s]$ to define generator of G2 and corresponding value $s[2]$ from the trusted setup.

$$e(E + uF + z\bar{W} + uz\omega\bar{W}', [2])e(-\bar{W} - u\bar{W}', [s]) == 1$$

If this check succeeds verifier returns "true" and "false" otherwise.

We should note that aggregation of all the parts in a form $y[1]$ can be done by first aggregating scalars and then performing a single multiplication by the generator of G1.