CRS Implementation (zei-library)

August 2021

In our implementation the CRS is based in KZG10 polynomial commitments over BLS12-381 elliptic curv, all this to get a PlonK inplementation.

"Due to real-world deployments of zk-SNARKs, it has become of significant interest to have the structured reference string (SRS) be constructible in a "universal and updat- able" fashion. Meaning that the same SRS can be used for statements about all circuits of a certain bounded size; and that at any point in time the SRS can be updated by a new party, such that the honesty of only one party from all updaters up to that point is required for soundness. For brevity, let us call a zk-SNARK with such a setup process universal..."[1]

If Fp = F is a field of prime order, we can denote by $F_{<d}[X]$ the set of univariate polynomials over F of degree smaller than d. The implementation of KZG polynomial commitment scheme based on `https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf` it relies on a bilinear map $e : G_1 \times G_2 \longrightarrow G_t$, where G_1, G_2, G_t are cyclic groups of prime order p, and g_1 is a generator fro G_1 and g_2 is a generator for G_2.

The operations of the scheme are as follows:

- SRS generation(n: max polynomial degree)
  - Pick a random scalar s in F_p
  - Compute public_parameter_group_1:= $(g_1, g_1^s, g_1^{s^2}, ..., g_1^{s^n})$
  - Compute public_parameter_group_2:= (g_2, g_2^s)

[1] Taken from the PlonK paper's introduction.

- return (public_parameter_group_1,public_parameter_group_2)

This is the function in our implementations that takes care of this.
`https://github.com/FindoraNetwork/zei/blob/9951a0364f5f4228d3fa8ac1cabf679b0a881a55/poly-iops/src/commitments/kzg_poly_com.rs#L149`

- Commit(P: polynomial)
  - let $P(x) = a0 + a1X + a2X^2 + ... + a_nX^n$
  - let $C := g1^{P(s)} = \pi_{i=0}^{n}(g_i^{s^i})^{a_i}$
  - return C
- Prove eval(P:polynomial, x: evaluation point)
  - Let $y = P(x)$
  - Compute Q(X) = (P(X)-P(x))/(X-x) if indeed $y == P(x)$ then $(X - x)|P(X) - y$
  - return $g1^{Q(s)}$
- Verify eval(C: commitment, x: evaluation point, y: evaluation of P on x, proof: proof of evaluation). The goal of this verification procedure is to check that indeed $P(X) - y = Q(X)(X - x)$ using pairings. Check that $e(C/g1^y, g2) == e(proof, g2^s/g2^x)$

With help of this we could prove that $f(7) = 57$, for a polynomial without revel who exactly is the polynomial

The max degree of the polynomial that can be committed is computed according to the number of constraints in the system by this function "build_multi_xfr_cs"

```
/// A prover can provide honest `secret_inputs` and obtain
///the cs witness by calling `cs.get_and_clear_witness()`.

/// One provides an empty secret_inputs to get the constraint
///system `cs` for verification only.

/// Returns the constraint system (and associated number of
///constraints) for a multi-inputs/outputs transaction.
```

https://github.com/FindoraNetwork/zei/blob/9951a0364f5f4228d3fa8ac1cabf679b0a881a55/zei_api/src/anon_xfr/circuits.rs#L191

The process pad the outcome number to the minimum power of two greater or equal to the actual number of constraints (for example, if the outcome number of constraints is 250 it is set to $256 = 2^8$)

```
/// Pad the number of constraints to a power of two.
```

https://github.com/FindoraNetwork/zei/blob/9951a0364f5f4228d3fa8ac1cabf679b0a881a55/poly-iops/src/plonk/turbo_plonk_cs/mod.rs#L537

The CRS is part of the PublicParams structure who has the following inputs and outputs:

The "impl PublicParams" needs this parameters

```
impl UserParams {
    pub fn new(
        n_payers: usize,
        n_payees: usize,
        tree_depth: Option<usize>,
        bp_num_gens: usize,
    ) -> UserParams {
```

And returns a structure like this

```
pub struct UserParams {
    pub bp_params: PublicParams,
    pub pcs: KZGCommitmentSchemeBLS,
    pub cs: TurboPlonkCS,
    pub prover_params: ProverParams<KZGCommitmentSchemeBLS>,
}
```

https://github.com/FindoraNetwork/zei/blob/9951a0364f5f4228d3fa8ac1cabf679b0a881a55/zei_api/src/setup.rs#L121

Where the field pub pcs is the substructure that contains the CRS to perform the KZG10 polynomial commitments based on the BLS12-381 elliptic curve.

If the public params were generated before they are stored as binaries files in path "zei/zei_api/data", so the first step is verify if pior public parameters exists in affirmative case they are load from the file, otherwise they are generated and stored in the path for future instances.