Developing and using a modular folding schemes library

Pierre D.-M. & Arnau Cube

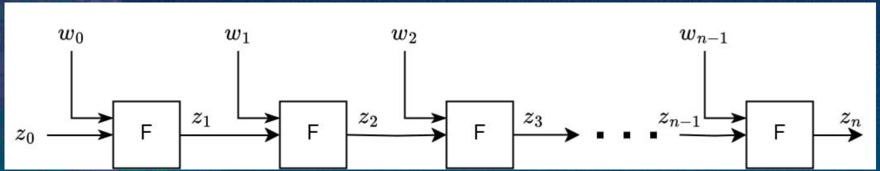
PSE 0xPARC

Introduction to IVC

Incrementally verifiable computations (IVC) are repetitive

$$s_n = F^{(n)}(s_0)$$

1. A model which infer on thousands of different data points
2. A VM with a fixed set of opcodes
3. A rollups which batch-verify signatures
4. A blockchain consensus



Benefits of folding based IVC schemes

Why folding schemes based IVC are cool:

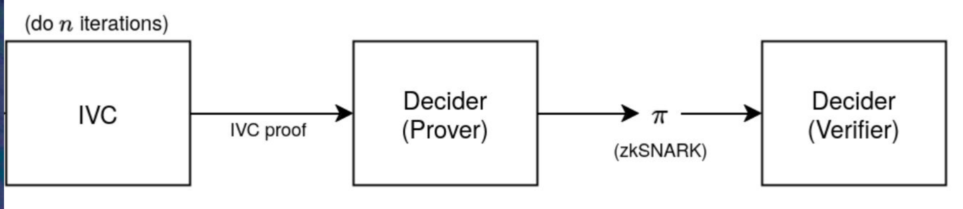
1. No need to specify the number of steps beforehand
2. O(|F|) prover memory
3. Verifier time does not depend on n

$$W \leftarrow W_1 + r \cdot W_2$$

Using folding schemes for recursion

Coined within the Nova paper (Kothapalli et al., 2021)

Intuition: if you have two witnesses satisfying a particular R1CS, a random linear combination of those two witnesses results in another satisfying unique witness.



Decider: the final step

Compressing the IVC proof is required since it is not succinct.

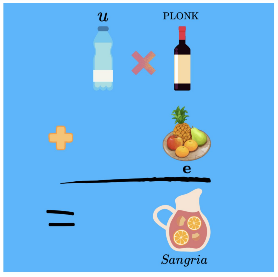
You can verify compressed IVC-proof onchain!

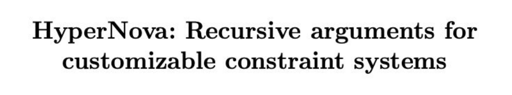
Folding schemes

Folding schemes zoo

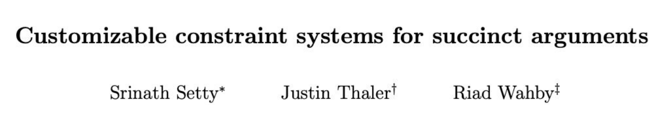
Many papers popped:

- Folding for plonkish circuits (e.g. sangria)
- Folding-adapted arithmetizations
- Many-to-1 folding
- Lattice based folding



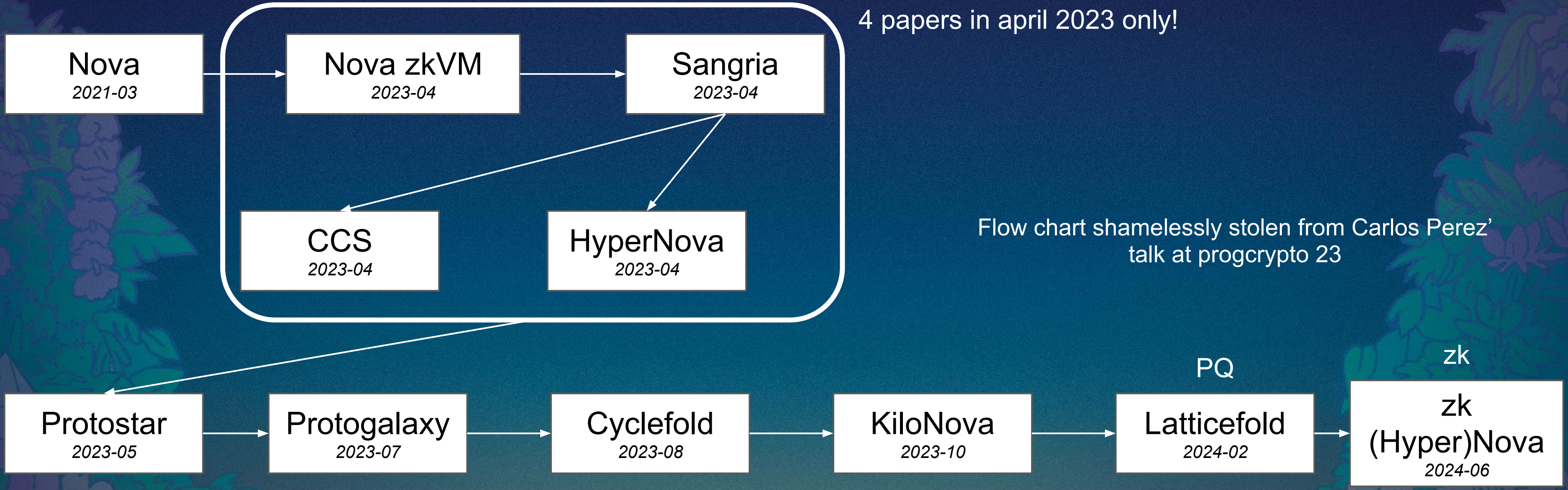


HyperNova: Recursive arguments for customizable constraint systems



Customizable constraint systems for succinct arguments

Srinath Setty* Justin Thaler† Riad Wahby‡



Nova
2021-03

Nova zkVM
2023-04

Sangria
2023-04

CCS
2023-04

HyperNova
2023-04

4 papers in april 2023 only!

Flow chart shamelessly stolen from Carlos Perez' talk at progcrypto 23

Protostar
2023-05

Protogalaxy
2023-07

Cyclefold
2023-08

KiloNova
2023-10

PQ

Latticefold
2024-02

zk

zk (Hyper)Nova
2024-06

Applications

Bitcoin light clients

Proving that the current block hash is the block hash obtained after N blocks.

Verify the POW for 100k blocks.

Cost (servers + tx): less than 30$ on Optimism

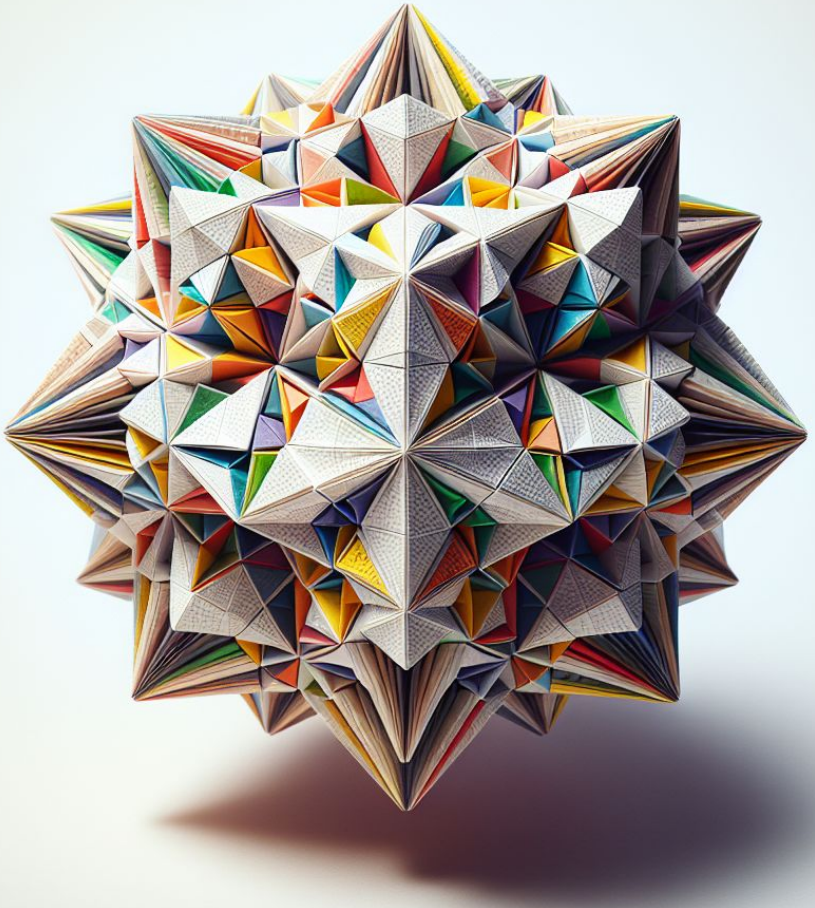
https://github.com/dmpierre/sonobe-btc

zkVMs

Nexus

Jolt (for memory checking)

Sonobe



Sonobe

Experimental folding schemes library implemented jointly by 0xPARC and PSE
https://github.com/privacy-scaling-explorations/sonobe

Modular library,

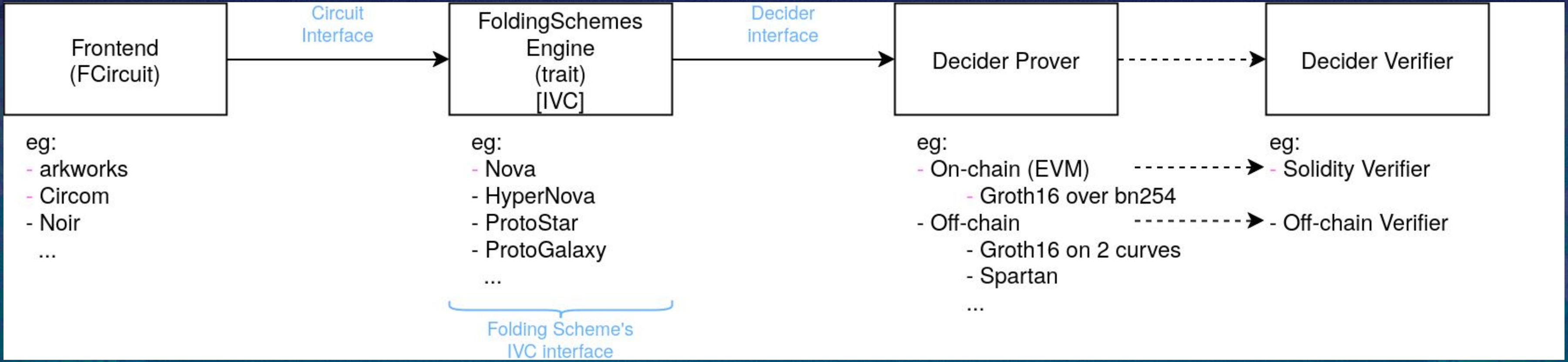
- Be able to
  - Add and test new folding schemes
  - Compare schemes ‘apples-to-apples’
  - Researchers can easily add their own schemes (eg. Mova paper)
- Make it easy for devs to use folding
  - Minimal code to fold your circuits (‘plug-and-fold’)
  - Easy to switch between folding schemes and curves
  - Support multiple zk-circuit languages
- Achieve Onchain Verification on Ethereum

Remark: experimental & research library, unoptimized.

Sonobe - Dev experience

Dev flow:

1. Define a circuit to be folded
2. Set which folding scheme to be used (eg. Nova with CycleFold)
3. Set a final decider to generate the final compressed proof (eg. Groth16 over BN254 curve)
4. Generate the Solidity decider verifier (EVM Solidity contract)



Frontend (FCircuit)

Circuit Interface

FoldingSchemes Engine (trait) [IVC]

Decider interface

Decider Prover

Decider Verifier

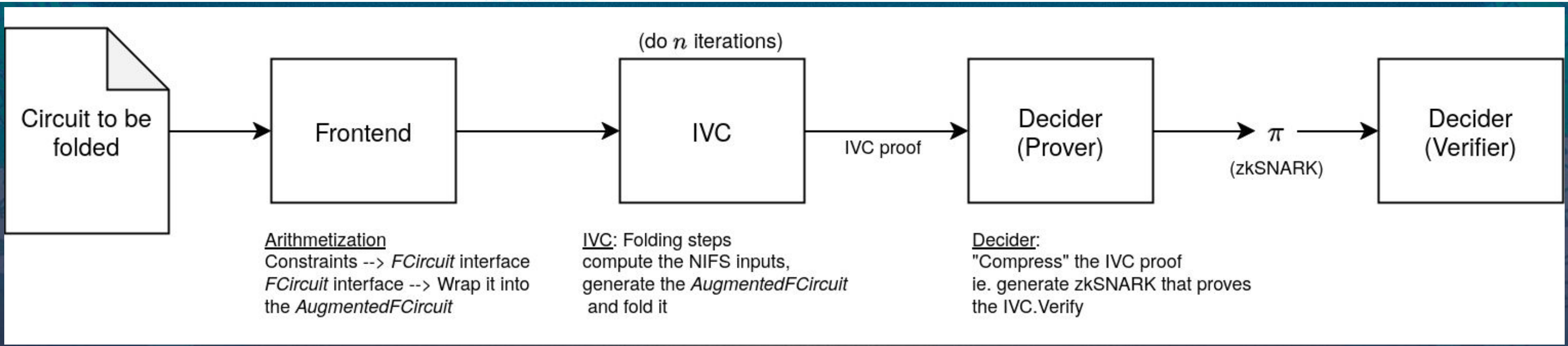
eg:
- arkworks
- Circom
- Noir
...

eg:
- Nova
- HyperNova
- ProtoStar
- ProtoGalaxy
...

Folding Scheme's IVC interface

eg:
- On-chain (EVM)
  - Groth16 over bn254
- Off-chain
  - Groth16 on 2 curves
  - Spartan
  ...

eg:
- Solidity Verifier
- Off-chain Verifier



Circuit to be folded

Frontend

(do n iterations)

IVC

IVC proof

Decider (Prover)

π (zkSNARK)

Decider (Verifier)

Arithmetization
Constraints --> *FCircuit* interface
FCircuit interface --> Wrap it into the *AugmentedFCircuit*

IVC: Folding steps
compute the NIFS inputs,
generate the *AugmentedFCircuit*
and fold it

Decider:
"Compress" the IVC proof
ie. generate zkSNARK that proves the IVC.Verify

Status of Sonobe - schemes implemented

Implemented (fully implemented):

- **Nova**: Recursive Zero-Knowledge Arguments from Folding Schemes, Abhiram Kothapalli, Srinath Setty, Ioanna Tzialla. 2021. https://eprint.iacr.org/2021/370.pdf
- **CycleFold**: Folding-scheme-based recursive arguments over a cycle of elliptic curves, Abhiram Kothapalli, Srinath Setty. 2023. https://eprint.iacr.org/2023/1192.pdf
- **HyperNova**: Recursive arguments for customizable constraint systems, Abhiram Kothapalli, Srinath Setty. 2023. https://eprint.iacr.org/2023/573.pdf
- **ProtoGalaxy**: Efficient ProtoStar-style folding of multiple instances, Liam Eagen, Ariel Gabizon. 2023. https://eprint.iacr.org/2023/1106.pdf

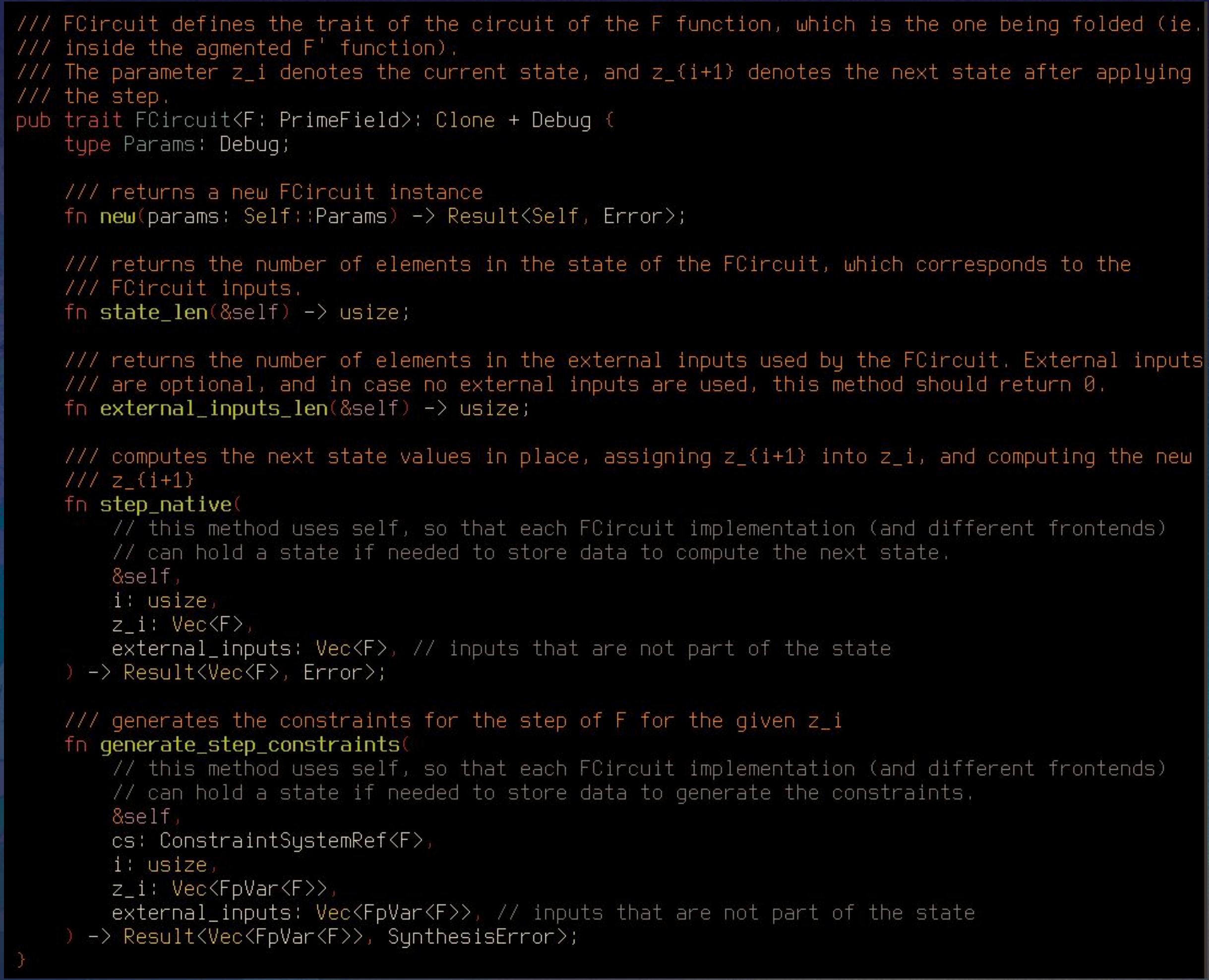
Started (NIFS implemented, next: folding circuit, IVC, Decider, etc):

- **Mova**: Nova folding without committing to error terms, Nikolaos Dimitriou, Albert Garreta, Ignacio Manzur, Ilia Vlasov. 2024. https://eprint.iacr.org/2024/1220.pdf
- **Ova**: Reduce the accumulation verifier in Nova from 2 to just 1 group operation, Benedikt Bünz. 2024. https://eprint.iacr.org/2024/1220.pdf

Frontends - how can the dev define a circuit to be folded

- Arkworks https://github.com/arkworks-rs
- experimental: Circom, Noir, Noname.

FCircuit interface

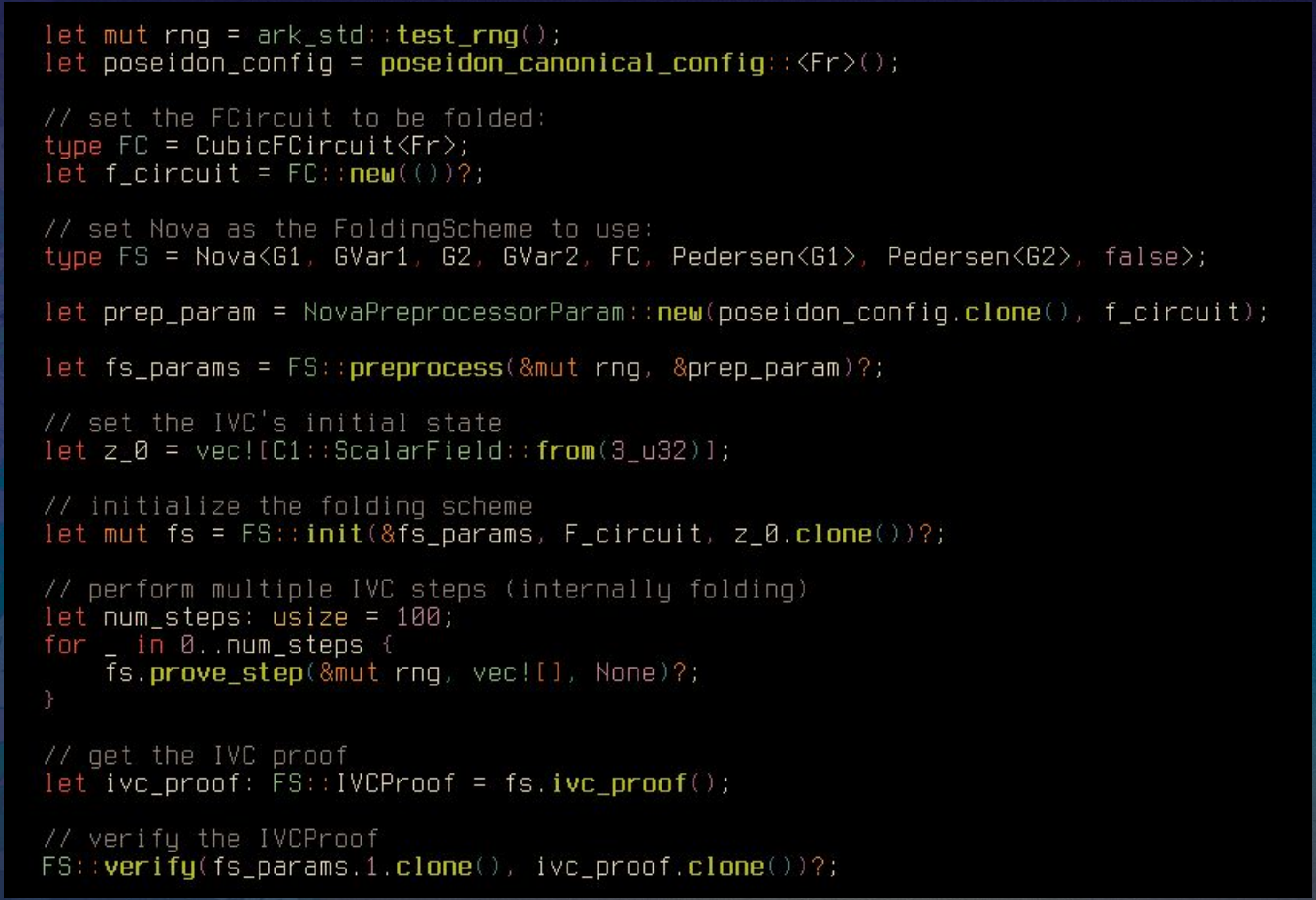


```
/// FCircuit defines the trait of the circuit of the F function, which is the one being folded (ie.
/// inside the agmented F' function).
/// The parameter z_i denotes the current state, and z_{i+1} denotes the next state after applying
/// the step.
pub trait FCircuit<F: PrimeField>: Clone + Debug {
    type Params: Debug;

    /// returns a new FCircuit instance
    fn new(params: Self::Params) -> Result<Self, Error>;

    /// returns the number of elements in the state of the FCircuit, which corresponds to the
    /// FCircuit inputs.
    fn state_len(&self) -> usize;

    /// returns the number of elements in the external inputs used by the FCircuit. External inputs
    /// are optional, and in case no external inputs are used, this method should return 0.
    fn external_inputs_len(&self) -> usize;

    /// computes the next state values in place, assigning z_{i+1} into z_i, and computing the new
    /// z_{i+1}
    fn step_native(
        // this method uses self, so that each FCircuit implementation (and different frontends)
        // can hold a state if needed to store data to compute the next state.
        &self,
        i: usize,
        z_i: Vec<F>,
        external_inputs: Vec<F>, // inputs that are not part of the state
    ) -> Result<Vec<F>, Error>;

    /// generates the constraints for the step of F for the given z_i
    fn generate_step_constraints(
        // this method uses self, so that each FCircuit implementation (and different frontends)
        // can hold a state if needed to store data to generate the constraints.
        &self,
        cs: ConstraintSystemRef<F>,
        i: usize,
        z_i: Vec<FpVar<F>>,
        external_inputs: Vec<FpVar<F>>, // inputs that are not part of the state
    ) -> Result<Vec<FpVar<F>>, SynthesisError>;
}
```

To fold a circuit, it just needs to implement the *FCircuit* trait.

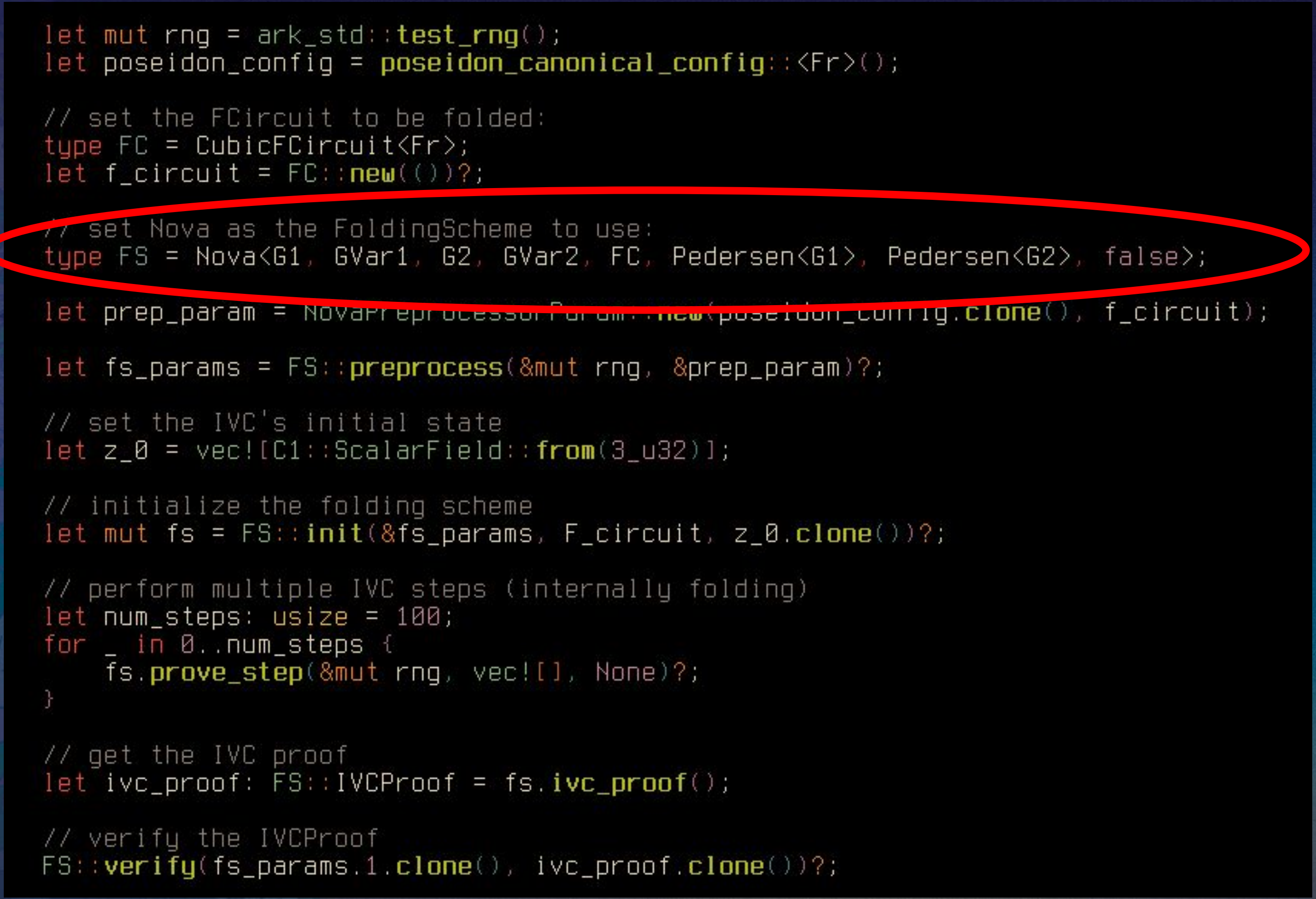
That's also an easy way to add new frontends for other zk-circuit languages.

Folding the circuit



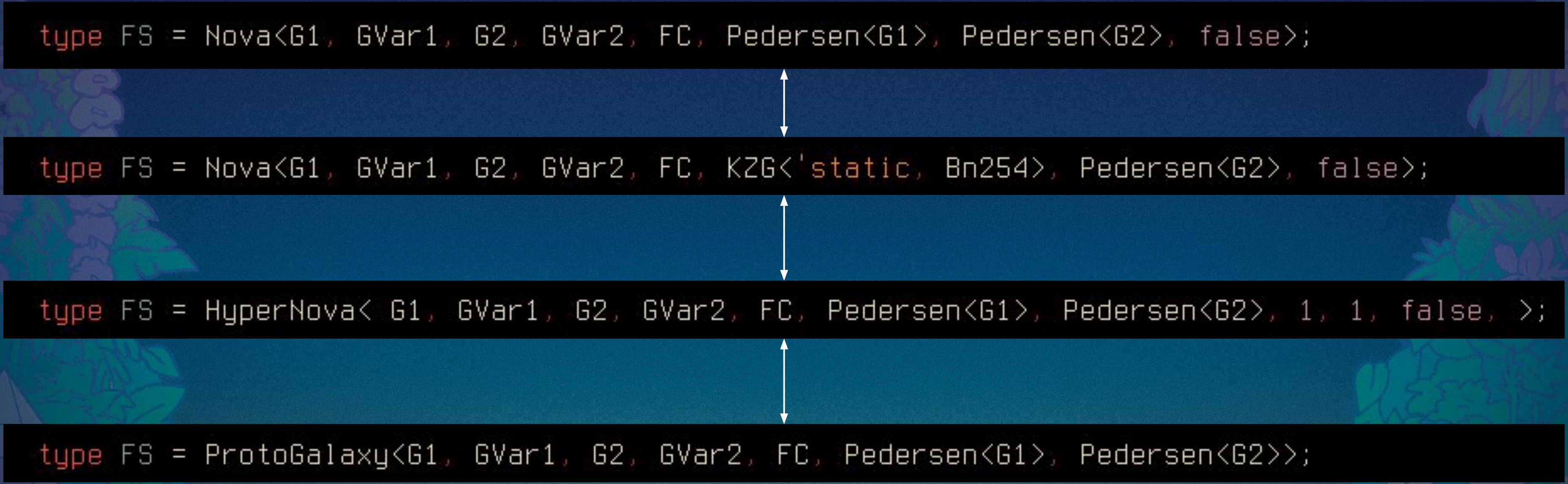
```rust
let mut rng = ark_std::test_rng();
let poseidon_config = poseidon_canonical_config::<Fr>();

// set the FCircuit to be folded:
type FC = CubicFCircuit<Fr>;
let f_circuit = FC::new(())?;

// set Nova as the FoldingScheme to use:
type FS = Nova<G1, GVar1, G2, GVar2, FC, Pedersen<G1>, Pedersen<G2>, false>;

let prep_param = NovaPreprocessorParam::new(poseidon_config.clone(), f_circuit);

let fs_params = FS::preprocess(&mut rng, &prep_param)?;

// set the IVC's initial state
let z_0 = vec![C1::ScalarField::from(3_u32)];

// initialize the folding scheme
let mut fs = FS::init(&fs_params, F_circuit, z_0.clone())?;

// perform multiple IVC steps (internally folding)
let num_steps: usize = 100;
for _ in 0..num_steps {
    fs.prove_step(&mut rng, vec![], None)?;
}

// get the IVC proof
let ivc_proof: FS::IVCProof = fs.ivc_proof();

// verify the IVCProof
FS::verify(fs_params.1.clone(), ivc_proof.clone())?;
```

Folding the circuit



```rust
let mut rng = ark_std::test_rng();
let poseidon_config = poseidon_canonical_config::<Fr>();

// set the FCircuit to be folded:
type FC = CubicFCircuit<Fr>;
let f_circuit = FC::new(())?;

// set Nova as the FoldingScheme to use:
type FS = Nova<G1, GVar1, G2, GVar2, FC, Pedersen<G1>, Pedersen<G2>, false>;

let prep_param = NovaPreprocessorParam::new(poseidon_config.clone(), f_circuit);

let fs_params = FS::preprocess(&mut rng, &prep_param)?;

// set the IVC's initial state
let z_0 = vec![C1::ScalarField::from(3_u32)];

// initialize the folding scheme
let mut fs = FS::init(&fs_params, F_circuit, z_0.clone())?;

// perform multiple IVC steps (internally folding)
let num_steps: usize = 100;
for _ in 0..num_steps {
    fs.prove_step(&mut rng, vec![], None)?;
}

// get the IVC proof
let ivc_proof: FS::IVCProof = fs.ivc_proof();

// verify the IVCProof
FS::verify(fs_params.1.clone(), ivc_proof.clone())?;
```

Switching between Folding Schemes & curves



```
type FS = Nova<G1, GVar1, G2, GVar2, FC, Pedersen<G1>, Pedersen<G2>, false>;
```

```
type FS = Nova<G1, GVar1, G2, GVar2, FC, KZG<'static, Bn254>, Pedersen<G2>, false>;
```

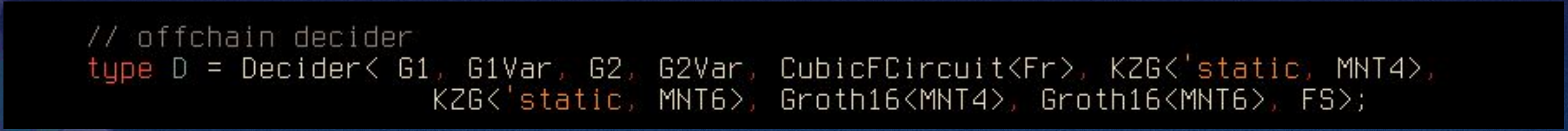
```
type FS = HyperNova< G1, GVar1, G2, GVar2, FC, Pedersen<G1>, Pedersen<G2>, 1, 1, false, >;
```

```
type FS = ProtoGalaxy<G1, GVar1, G2, GVar2, FC, Pedersen<G1>, Pedersen<G2>>;
```

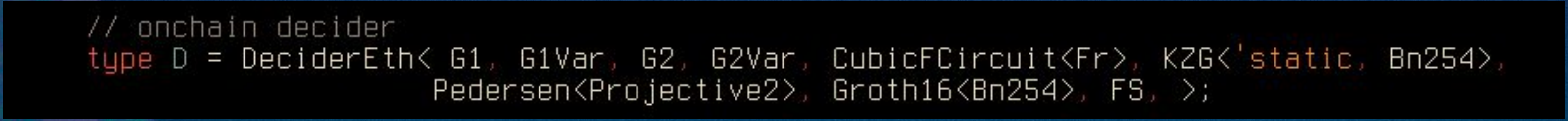
G1 & G2 could be any cycle of curves available in arkworks.

Decider: compress the IVC Proof with a zkSNARK

Setting which decider to use:

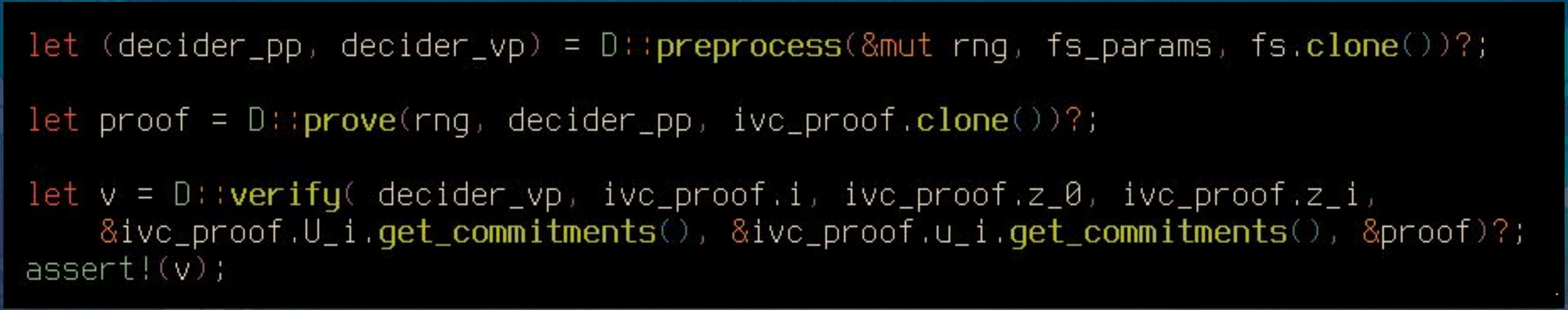


```
// offchain decider
type D = Decider< G1, G1Var, G2, G2Var, CubicFCircuit<Fr>, KZG<'static, MNT4>,
                    KZG<'static, MNT6>, Groth16<MNT4>, Groth16<MNT6>, FS>;
```



```
// onchain decider
type D = DeciderEth< G1, G1Var, G2, G2Var, CubicFCircuit<Fr>, KZG<'static, Bn254>,
                    Pedersen<Projective2>, Groth16<Bn254>, FS, >;
```

Using the Decider:



```
let (decider_pp, decider_vp) = D::preprocess(&mut rng, fs_params, fs.clone())?;

let proof = D::prove(rng, decider_pp, ivc_proof.clone())?;

let v = D::verify( decider_vp, ivc_proof.i, ivc_proof.z_0, ivc_proof.z_i,
    &ivc_proof.U_i.get_commitments(), &ivc_proof.u_i.get_commitments(), &proof)?;
assert!(v);
```

Onchain verification

- At the end of n folding steps, we have the last *IVC state* and the *IVC Proof*
- We *compress* it through a zkSNARK (Decider)
- One of Sonobe's goals: verify it onchain in Ethereum

- Original Nova: wrapp the Decider checks in 2 Spartan (zkSNARK) proofs (one over each curve of the cycle of curves).
  → 2 Spartan proofs, one on each curve
- In our case we were interested into verifying the proofs in Ethereum's EVM.

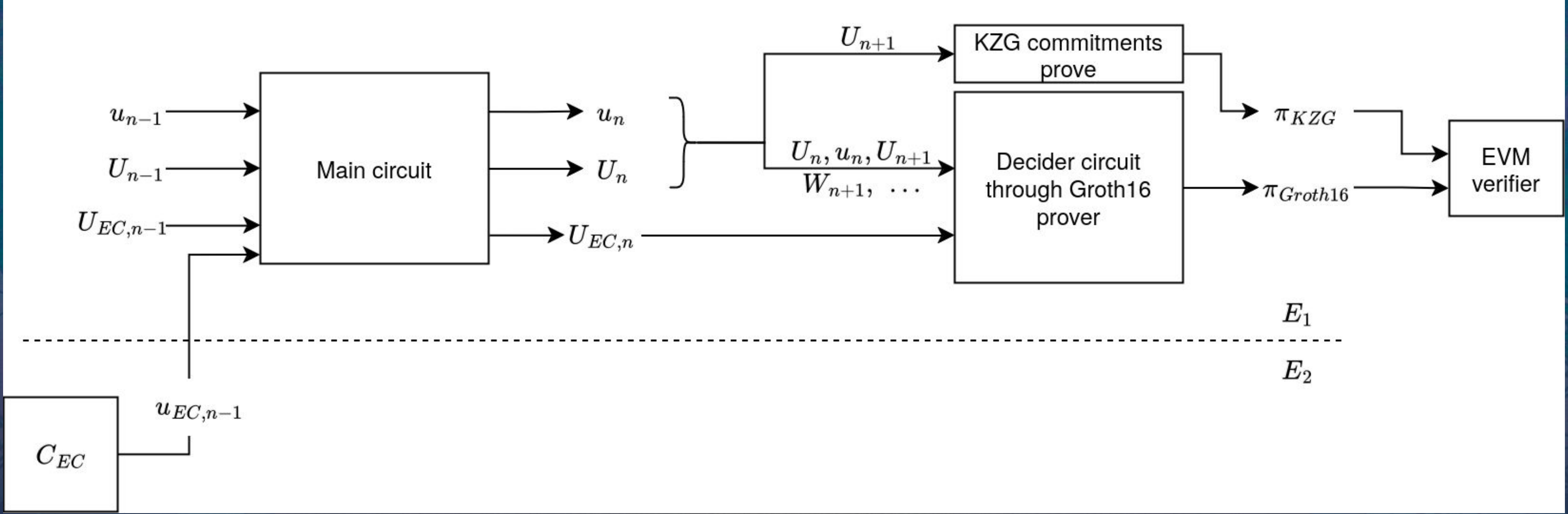
Need to do a bit of gymnastics to verify the folding proofs in Ethereum,
EVM limitations:

- limited to BN254 curve
- constrained by gas costs

Onchain verification

Sonobe's Onchain verification:

- Generate a Decider proof that can be verified in Solidity
- Offer methods to generate the Solidity smart contract code that verifies the Sonobe proof



Some preliminary numbers

- Till now we've been focusing on implementing the various schemes
  - Nova, HyperNova, ProtoGalaxy, CycleFold
- Without focusing on optimization/efficiency
- So far the numbers we got look promising

Next steps: we're going to start profiling, optimizing, adding benchmarks, etc. Getting to a ‘first release’.

Some preliminary numbers

- Till now we've been focusing on implementing the various schemes
  - Nova, HyperNova, ProtoGalaxy, CycleFold
- Without focusing on optimization/efficiency
- So far the numbers we got look promising

Next steps: we're going to start profiling, optimizing, adding benchmarks, etc. Getting to a 'first release'.

Recall, this proof is proving that applying *n* times the function *F* (the circuit that we're folding) to an initial state *z_0* results in the state *z_n* .

Unoptimized preliminary numbers:

- folding step (the recursive iteration): **~ 300ms**
  - Folding circuit (Nova+CycleFold): ~ 50k R1CS constraints
- offchain Decider prove: < 1 min
- onchain Decider:
  - Circuit: ~ 10M R1CS constraints
    - < 3 minutes in a 32GB RAM 16 core laptop
  - gas costs (Decider proof verification): ~ 800k gas
    - mostly from G16, KZG10, public inputs processing
    - will be reduced by hashing the public inputs & batching the pairings check
  - expect to get it down to **< 500k gas**.

Repo: https://github.com/privacy-scaling-explorations/sonobe

Docs: https://privacy-scaling-explorations.github.io/sonobe-docs/

Wrap up

- Folding Schemes are not a tool that fits in all use cases, but in those where it fits it can provide significantly speed & memory improvements.
- Sonobe: experimental research modular folding schemes library
- Schemes available: Nova, HyperNova, ProtoGalaxy (all with CycleFold)
- Onchain verification available
- Preliminary benchmarks look promising
- Next steps: optimizations & first release

(QR code contains a link to the repo)

PSE & 0xPARC joint work