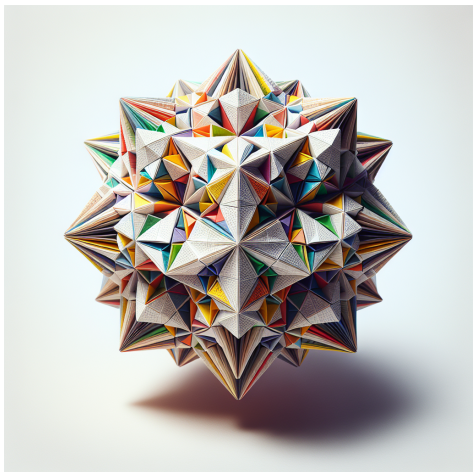
An overview on folding schemes, and an introduction to Sonobe



2024-08-08

San Francisco 0xPARC Summer

Polynomials and SNARKs

- define the 'program' that we want to be able to prove as a set of constraints
- encode the constraints as polynomials
  eg. R1CS: $Az \circ Bz - Cz == 0$
  $$A(X) \cdot B(X) - C(X) == 0$$
- and then use some scheme to prove that those polynomials satisfy the relation. eg. Groth16, Spartan, etc

tl;dr: want to prove polynomial relations

Why folding

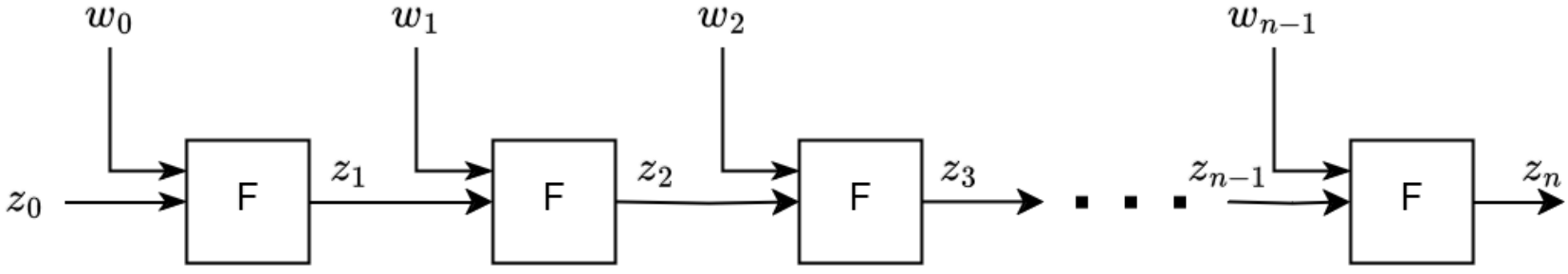
- Repetitive computations take big circuits $\longrightarrow$ large proving time
  - and in some cases takes too much memory and can not be even computed
  - eg. prove a chain of 10k sha256 hashes (¿600M R1CS constraints, not feasible with most traditional SNARK proving systems)
- Traditional recursion: verify (in-circuit) a proof of the correct execution of the same circuit for the previous input
  - issue: in-circuit proof verification is expensive (constraints)
    - ie. verify a Groth16 proof inside a R1CS circuit

IVC - Incremental Verifiable Computation

Valiant'08

Folding schemes efficitently achieve IVC, where the prover recursively proves the correct execution of the incremental computations.

Prove that applying n times the F function (the circuit being folded) to the initial state (z_0) results in the final state (z_n).



In other words, it allows to prove efficiently that $z_n = F(... \ F(F(F(F(z_0, w_0), w_1), w_2), ...), w_{n-1})$.

Folding idea

Random linear combination of homomorphic commitments

We rely on homomorphic commitments, eg. Pedersen commitments
Let $g \in \mathbb{G}^n,\ v \in \mathbb{F}_r^n$,

$$Com(v) = \langle g, v \rangle = g_1 \cdot v_1 + g_2 \cdot v_2 + \ldots + g_n \cdot v_n \in \mathbb{G}$$

RLC:
Let $v, w \in \mathbb{F}_r^n$,
set $cm_v = Com(v),\ cm_w = Com(w) \in \mathbb{G}$.
then,

$$y = v + r \cdot w$$
$$cm_y = cm_v + r \cdot cm_w$$

so that

$$cm_y = Com(y)$$

Relaxed R1CS

R1CS instance: $(\{A, B, C\} \in \mathbb{F}^{n \times n},\ n,\ l)$, such that for $z = (1, io \in \mathbb{F}^l, w \in \mathbb{F}^{n-l-1}) \in \mathbb{F}^n$,

$$Az \circ Bz = Cz$$

Relaxed R1CS:

$$Az \circ Bz = uCz + E$$

for $u \in \mathbb{F}$, $E \in \mathbb{F}^n$.

Committed Relaxed R1CS instance: $CI = (\overline{E}, u, \overline{W}, x)$
Witness of the instance: $WI = (E, W)$

Relaxed R1CS

$u = u_1 + ru_2, \quad z = z_1 + rz_2, \quad x = x_1 + rx_2$
$E = E_1 + r(Az_1 \circ Bz_2 + Az_2 \circ Bz_1 - u_1 Cz_2 - u_2 Cz_1) + r^2 E_2$
Relaxed R1CS: $Az \circ Bz = uCz + E, \quad with\ z = (u,\ x,\ W)$

$$
\begin{aligned}
Az \circ Bz &= A(z_1 + r \cdot z_2) \circ B(z_1 + r \cdot z_2) \\
&= Az_1 \circ Bz_1 + r(Az_1 \circ Bz_2 + Az_2 \circ Bz_1) + r^2(Az_2 \circ Bz_2) \\
&= (u_1 Cz_1 + E_1) + r(Az_1 \circ Bz_2 + Az_2 \circ Bz_1) + r^2(u_2 Cz_2 + E_2) \\
&= u_1 Cz_1 + \underbrace{E_1 + r(Az_1 \circ Bz_2 + Az_2 \circ Bz_1) + r^2 E_2}_{\mathsf{E}} + r^2 u_2 Cz_2 \\
&= u_1 Cz_1 + r^2 u_2 Cz_2 + E \\
&= (u_1 + ru_2) \cdot C \cdot (z_1 + rz_2) + E \\
&= uCz + E
\end{aligned}
$$

For R1CS matrices $(A,\ B,\ C)$, the folded witness W is a satisfying witness for the folded instance $(E,\ u,\ x)$, following the Relaxed R1CS relation:
$Az \circ Bz - uCz - E == 0$.
Since we don't want that the Verifier learning about the witness, we commit to it, and the Verifier will run the RLC on the commitment (not on the witness), obtaining the 'folded' commitment.

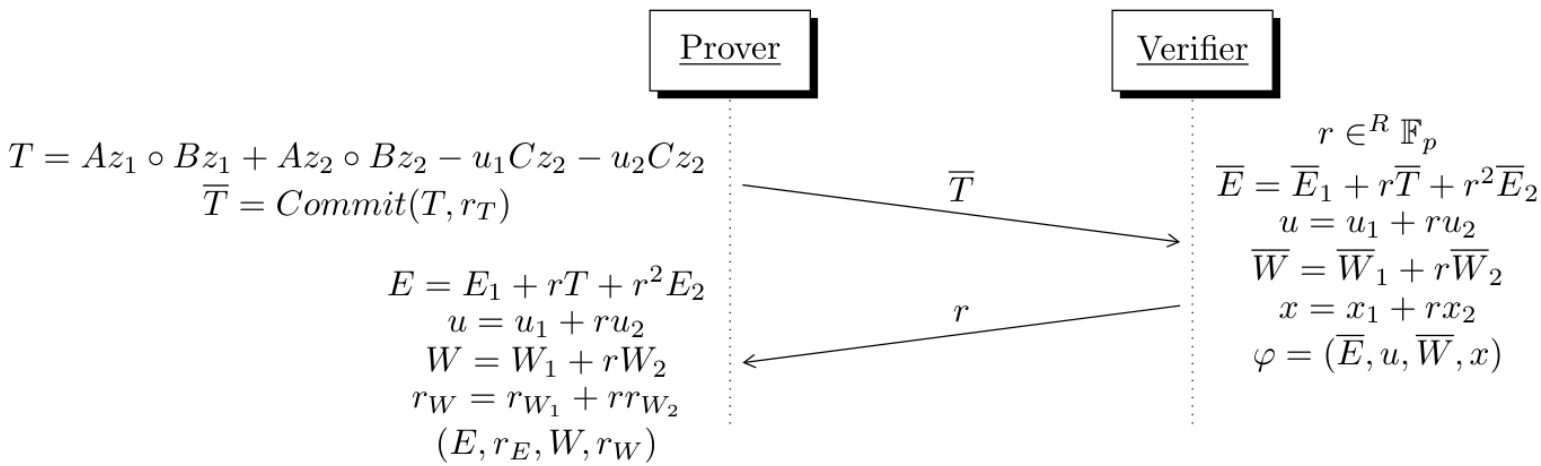
Full details at Nova's paper, pages 13-15 ("first attempt, second attempt, third attempt")

NIFS - Non Interactive Folding Scheme (in Nova)

Main idea:

- interactive protocol between P and V
- where V obtains the 'folded' commitment that corresponds to the 'folded' witness that P computes
- so V does not know the witness

We make it non-interactive with Fiat-Shamir.



Relation check:

$$z = (1, x, W)$$

$$\begin{cases} Az \circ Bz - uCz - E \stackrel{?}{=} 0 \\ \overline{W} \stackrel{?}{=} Com(W) \\ \overline{E} \stackrel{?}{=} Com(E) \end{cases}$$

HyperNova NIMFS

3. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol $c \leftarrow \langle \mathcal{P}, \mathcal{V}(r'_x) \rangle (g, s, d+1, T)$, where:

$$g(x) \coloneqq \left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1) \cdot t + j} \cdot L_{j,k}(x) \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot Q_k(x) \right)$$

$$L_{j,k}(x) \coloneqq \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{1,k}(y) \right)$$

$$Q_k(x) \coloneqq \widetilde{eq}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right) \right)$$

$$T \coloneqq \sum_{j \in [t], k \in [\mu]} \gamma^{(k-1) \cdot t + j} \cdot \mathcal{L}_k.\phi.v_j$$

4. $\mathcal{P} \rightarrow \mathcal{V}$: $\{\sigma_{j,k}\}$, where for all $j \in [t]$, $k \in [\mu]$:

$$\sigma_{j,k} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}_{1,k}(y)$$

Similarly, $\{\theta_{j,k}\}$, where for all $j \in [t]$ and $k \in [\nu]$:

$$\theta_{j,k} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}_{2,k}(y)$$

5. $\mathcal{V}$: Compute $e_1 \leftarrow \widetilde{eq}(r_x, r'_x)$ and $e_2 \leftarrow \widetilde{eq}(\beta, r'_x)$, and check that

$$c = \left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1) \cdot t + j} \cdot e_1 \cdot \sigma_{j,k} \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot e_2 \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \theta_{j,k} \right) \right)$$

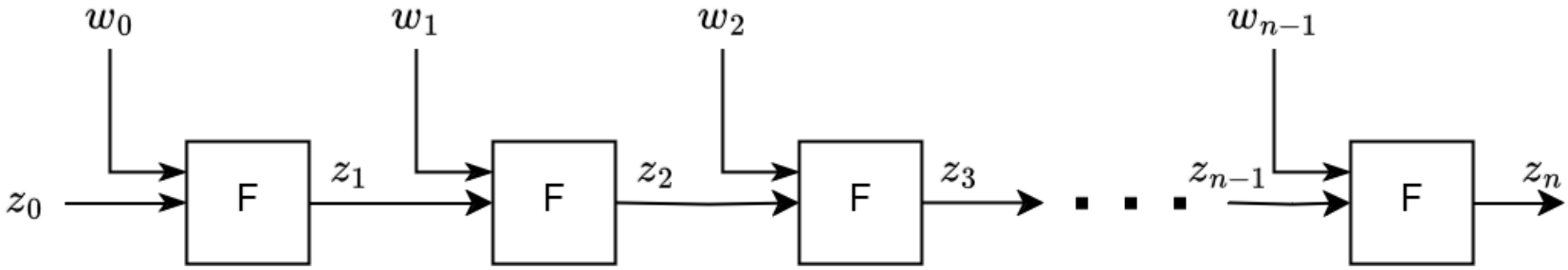
ProtoGalaxy Folding

5. **P** sends the non-constant coefficients $F_1, \ldots, F_t$ of $F(X)$ to **V**.
6. **V** sends a random challenge $\alpha \in \mathbb{F}$.
7. **P** and **V** compute $F(\alpha) = e + \sum_{i \in [t]} F_i \alpha^i$.
8. **P** and **V** compute $\boldsymbol{\beta}^* \in \mathbb{F}^t$ where $\boldsymbol{\beta}^*_i := \boldsymbol{\beta}_i + \alpha \cdot \boldsymbol{\delta}_i$.
9. **P** computes the polynomial
$$G(X) := \sum_{i \in [n]} \mathsf{pow}_i(\boldsymbol{\beta}^*) f_i(L_0(X)\omega + \sum_{j \in [k]} L_j(X)\omega_j).$$
10. **P** computes polynomial $K(X)$ such that
$$G(X) = F(\alpha)L_0(X) + Z(X)K(X).$$
11. **P** sends the coefficients of $K(X)$.
12. **V** sends a random challenge $\gamma \in \mathbb{F}$.
13. **P** and **V** compute
$$e^* := F(\alpha)L_0(\gamma) + Z(\gamma)K(\gamma).$$

At the end of the protocol

- **V** outputs the instance $\Phi^* = (\phi^*, \boldsymbol{\beta}^*, e^*)$, where
$$\phi^* := L_0(\gamma)\phi + \sum_{i \in [k]} L_i(\gamma)\phi_i.$$
- **P** outputs the witness $\omega^* := L_0(\gamma)\omega + \sum_{i \in [k]} L_i(\gamma)\omega_i$.

IVC



- We have our folding protocol, in which P & V 'fold' the instances (witness & commitments)
- Will use it in the IVC setting
- at each IVC step, need to ensure that the 'folding' of the previous step was done correctly
- we 'augment' the circuit with extra checks that compute the folding Verifier

IVC - Nova example

U_i (*running instance*): committed instance for the correct execution of invocations $1, \ldots, i-1$ of F'
u_i (*incoming instance*): committed instance for the correct execution of invocation i of F'

F':
i) execute a step of the incremental computation, $z_{i+1} = F(z_i)$
ii) invoke the NIFS.V to fold U_i, u_i into U_{i+1}
iii) other checks to ensure that the IVC is done properly

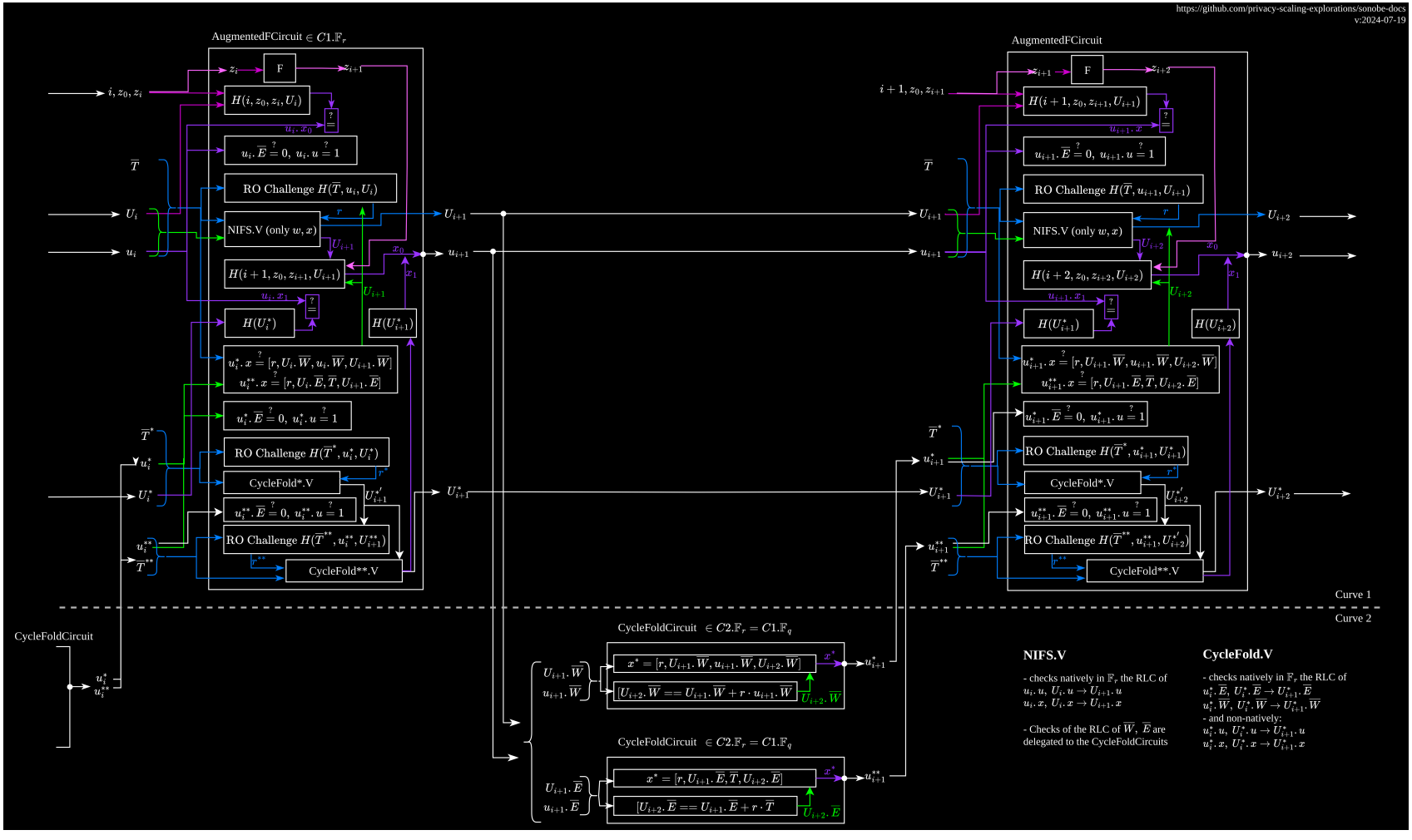
Cycle of curves

NIFS.V involves $\mathbb{G}$ point operation, which are not native over $\mathbb{F}_r$ of $\mathbb{G}$.
$\longrightarrow$ delegate them into a circuit over a 2nd curve We use:

- $\mathbb{G}_1.\mathbb{F}_r = \mathbb{G}_2.\mathbb{F}_q$
- $\mathbb{G}_1.\mathbb{F}_q = \mathbb{G}_2.\mathbb{F}_r$
- eg. for Ethereum compatibility:
  $\mathbb{G}_1$: BN254, $\mathbb{G}_2$: Grumpkin.

We 'mirror' the main F' circuit into the 2nd curve
each circuit computes natively the point operations of the other curve

Augmented F Circuit + CycleFold Circuit

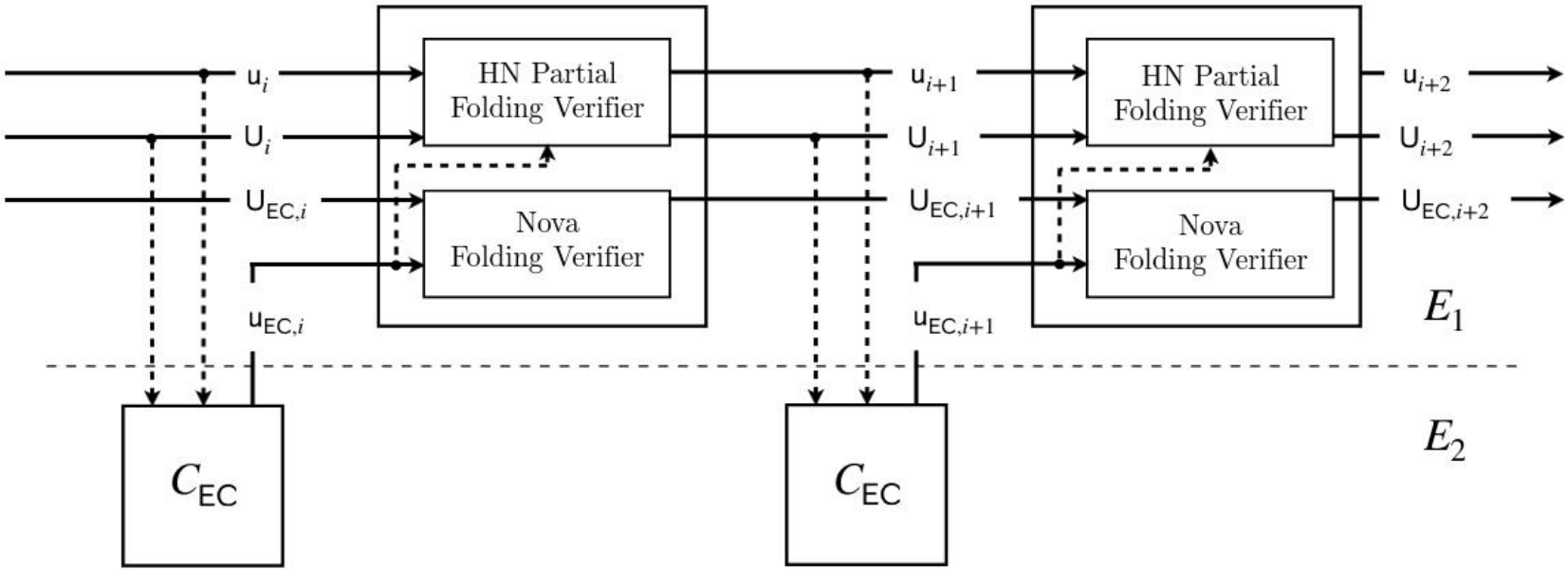


explain: circuit overhead

Adding zk to the IVC

- fold the original witness with a randomized instance
- then we can delegate the rest of the computation to a third party server

Decider (Final compressed proof)



With Prover knowing the respective witnesses for $U_n, u_n, U_{EC,n}$

Issue: IVC proof is not succinct

Decider

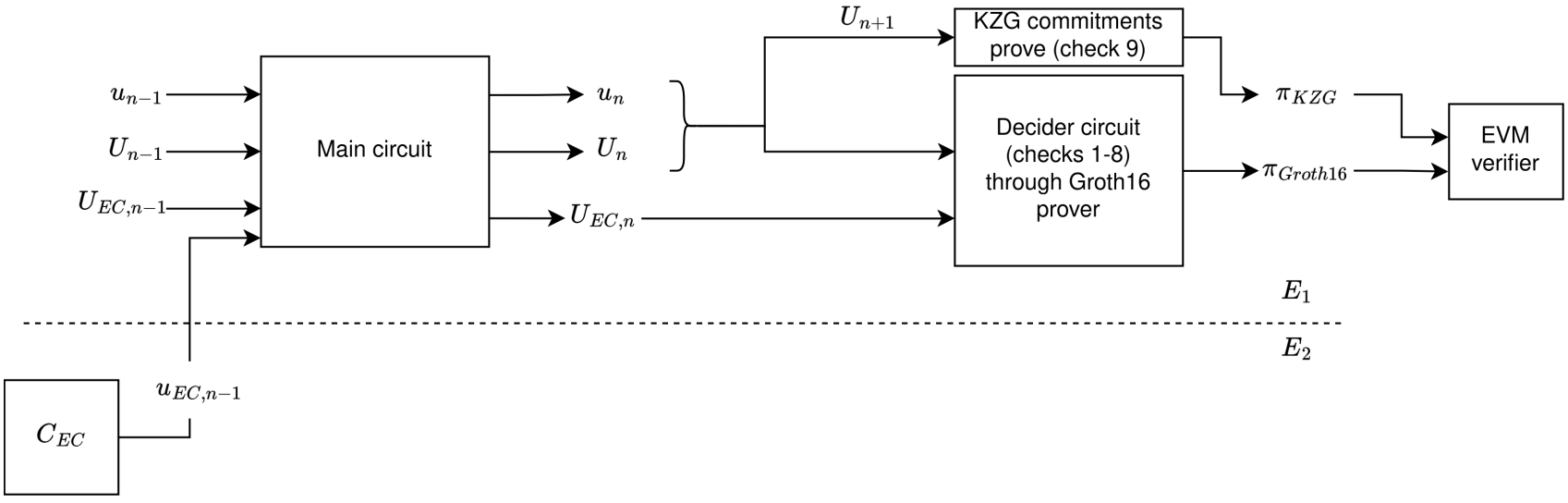
Original Nova: generate a zkSNARK proof with Spartan for $U_n, u_n, U_{EC,n}$
$\longrightarrow$ 2 Spartan proofs, one on each curve
(not EVM-friendly)

Decider

checks (simplified)

1. (U_{n+1}, W_{n+1}) satisfy Relaxed R1CS relation of AugmentedFCircuit
2. verify commitments of $U_{n+1}.\{\overline{E}, \overline{W}\}$ w.r.t. $W_{n+1}.\{E, W\}$
3. $(U_{EC,n}, W_{EC,n})$ satisfy Relaxed R1CS relation of CycleFoldCircuit
4. verify commitments of $U_{EC,n}.\{\overline{E}, \overline{W}\}$ w.r.t. $W_{EC,n}.\{E, W\}$
5. $u_n.E == 0,\ u_n.u == 1$, ie. u_n is a fresh not-relaxed instance
6. $u_n.x_0 == H(n, z_0, z_n, U_n)$
   $u_n.x_1 == H(U_{EC,n})$
7. $NIFS.V(U_n, u_n) == U_{n+1}$

Decider


u_{n-1}
U_{n-1}
$U_{EC,n-1}$
Main circuit
u_n
U_n
$U_{EC,n}$
U_{n+1}
KZG commitments prove (check 9)
Decider circuit (checks 1-8) through Groth16 prover
π_{KZG}
$\pi_{Groth16}$
EVM verifier
E_1
E_2
C_{EC}
$u_{EC,n-1}$

Sonobe

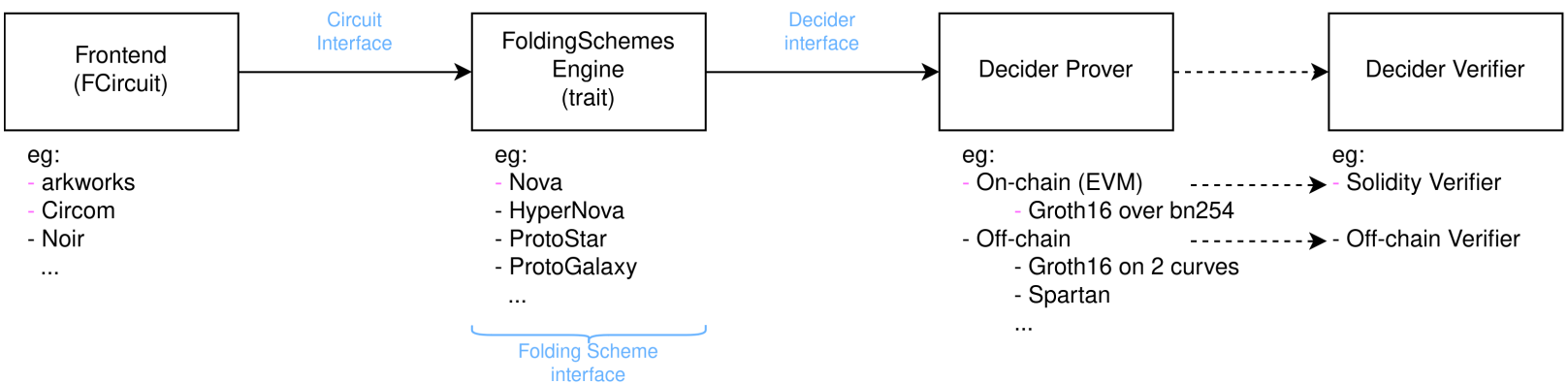
Experimental folding schemes library implemented jointly by 0xPARC and PSE.
https://github.com/privacy-scaling-explorations/sonobe
Modular library,

- 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 of multiple zk-circuit languages

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 proof (eg. Groth16 over BN254 curve)
4. Generate the the decider verifier (EVM Solidity contract)



Status of Sonobe - dev experience

- Verify in Ethereum
  - solidity verifier contract generator
- Frontends - how can the dev define a circuit to be folded
  - Arkworks https://github.com/arkworks-rs/
  - Circom https://github.com/iden3/circom
  - Noir https://noir-lang.org/
  - Noname https://github.com/zksecurity/noname

Status of Sonobe - schemes implemented

Implemented:

- **Nova**: Recursive Zero-Knowledge Arguments from Folding Schemes https://eprint.iacr.org/2021/370.pdf, Abhiram Kothapalli, Srinath Setty, Ioanna Tzialla. 2021
- **CycleFold**: Folding-scheme-based recursive arguments over a cycle of elliptic curves https://eprint.iacr.org/2023/1192.pdf, Abhiram Kothapalli, Srinath Setty. 2023
- **HyperNova**: Recursive arguments for customizable constraint systems https://eprint.iacr.org/2023/573.pdf, Abhiram Kothapalli, Srinath Setty. 2023

Almost finished:

- **ProtoGalaxy**: Efficient ProtoStar-style folding of multiple instances https://eprint.iacr.org/2023/1106.pdf, Liam Eagen, Ariel Gabizon. 2023

Soon:

- **Mova**: Nova folding without committing to error terms https://eprint.iacr.org/2024/1220.pdf, Nikolaos Dimitriou, Albert Garreta, Ignacio Manzur, Ilia Vlasov. 2024

Temptative:

- **LatticeFold**: A Lattice-based Folding Scheme and its Applications to Succinct Proof Systems https://eprint.iacr.org/2024/257.pdf, Dan Boneh, Binyi Chen. 2024
- Parallel folding

Code example

[show code with a live demo]

Code example

Some numbers (still optimizations pending):

- AugmentedFCircuit (Nova): $\sim 50k$ R1CS constraints
- DeciderEthCircuit: $\sim 10M$ R1CS constraints
  - < 3 minutes in a 32GB RAM 16 core laptop
- gas costs (DeciderEthCircuit proof): $\sim 800k$ gas
  - mostly from G16, KZG10, public inputs processing
  - will be reduced by hashing the public inputs
  - expect to get it down to $< 500k$ gas.

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.

In Srinath Setty words, you can prove practically unbounded computation onchain by 800k gas (and soon $< 500k$).

Wrappup

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

2024-08-08

0xPARC & PSE.