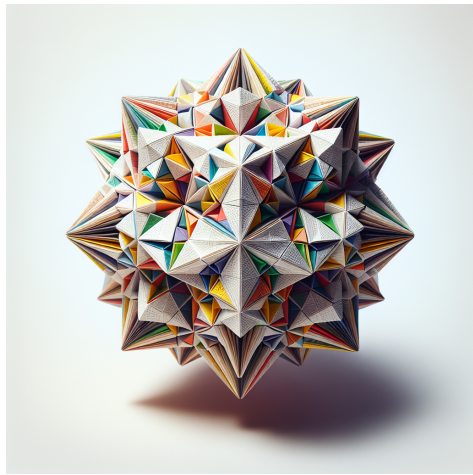
Sonobe, a modular folding schemes library



2024-10-08
0xPARC & PSE

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

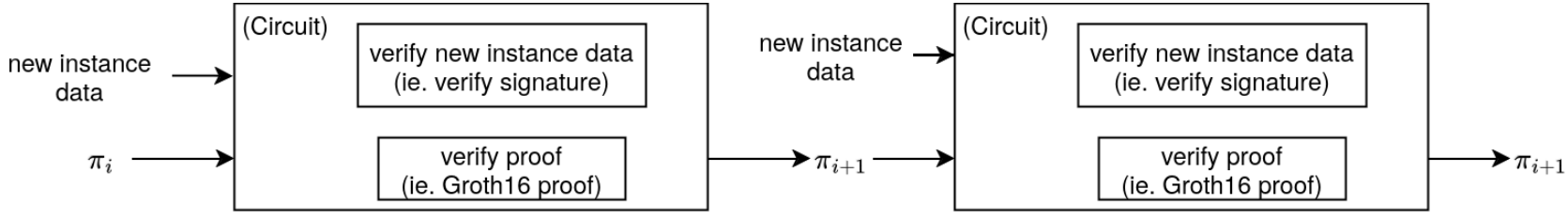
TL;DR: want to prove polynomial relations

Why folding

- Repetitive computations take big circuits $\longrightarrow$ large proving time
  - 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)
  - eg. zkVM opcodes

Why folding

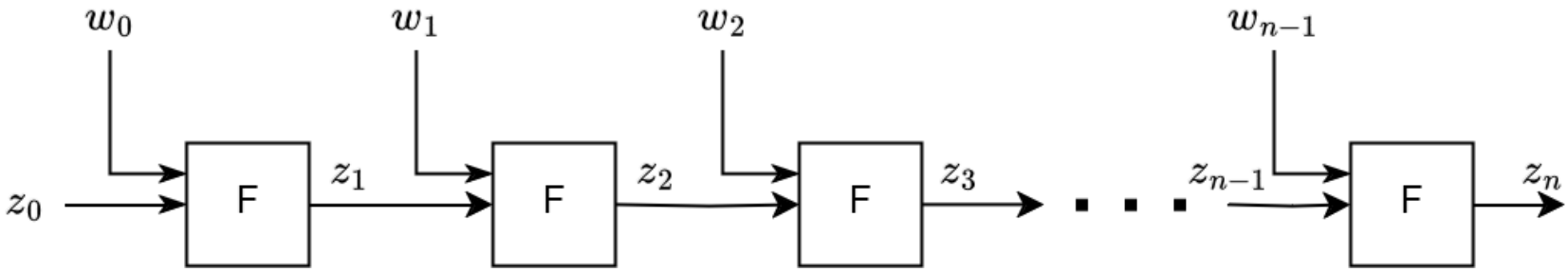
- Repetitive computations take big circuits $\longrightarrow$ large proving time
  - in some cases takes too much memory and can not be even computed
  - eg. prove a chain of 10k sha256 hashes (¿600M R1CS constraints, not feasible with most traditional SNARK proving systems)
  - eg. zkVM opcodes
- Traditional 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

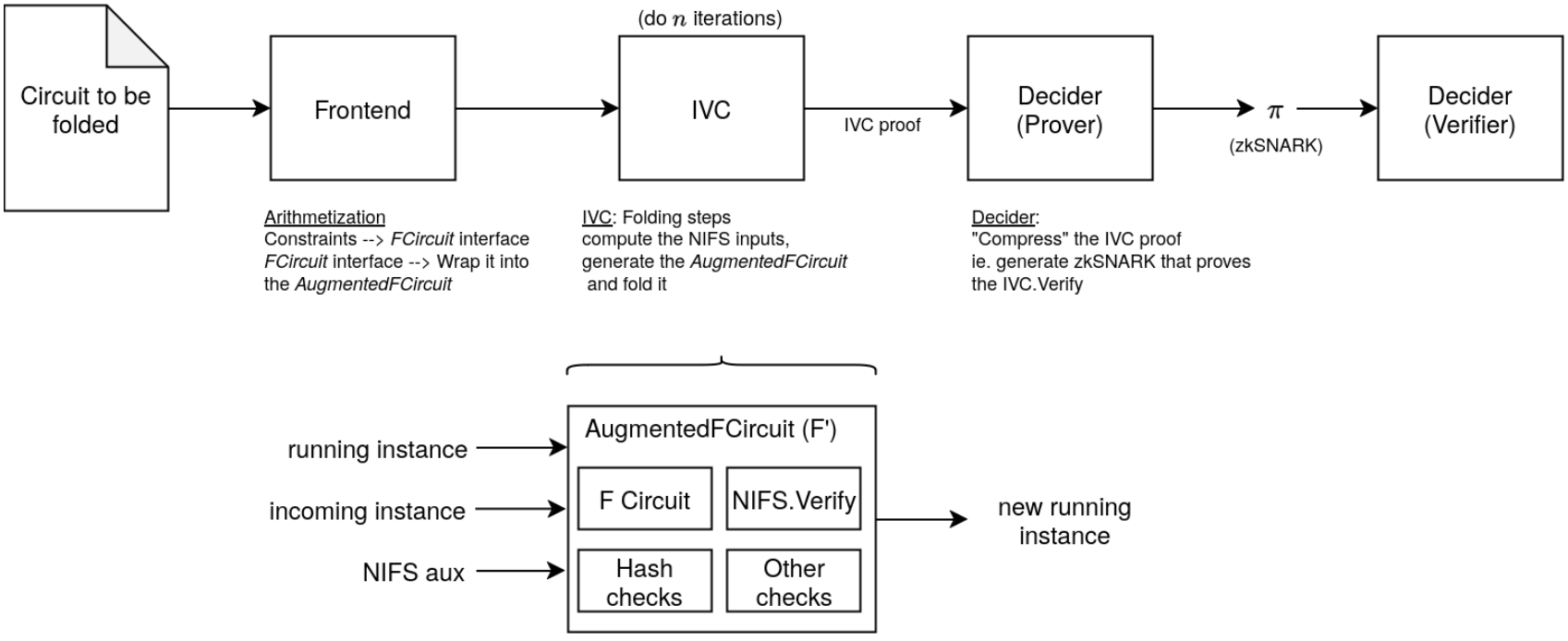
Prove that applying n times the F function (the circuit being folded) to the initial state (z_0) results in the final state (z_n).



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 scheme pipeline



The IVC folds the AugmentedFCircuit, which ensures that the NIFS.Verify checks out.

RLC of homomorphic commitments

In the initial folding schemes, 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

$$cm_y = Com(y)$$

Random linearly combining 2 R1CS instances

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

$$Az \circ Bz = Cz$$

If we try to do a RLC with two instances ($z = z_1 + r\dot{z}_2$):

$$\begin{aligned} Az \circ Bz &= A(z_1 + rz_2) \circ B(z_1 + rz_2) \\ &= (Az_1 \circ Bz_1) + r \cdot (Az_1 \circ Bz_2 + Az_2 \circ Bz_1) + r^2 \cdot (Az_2 \circ Bz_2) \\ &\neq Cz = C(z_1 + rz_2) = Cz_1 + rCz_2 = (Az_1 \circ Bz_1) + r \cdot (Az_2 \circ Bz_2) \end{aligned}$$

we end up with some inequality.

Random linearly combining 2 R1CS instances

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

$$Az \circ Bz = Cz$$

If we try to do a RLC with two instances ($z = z_1 + r\dot{z}_2$):

$$\begin{aligned} Az \circ Bz &= A(z_1 + rz_2) \circ B(z_1 + rz_2) \\ &= (Az_1 \circ Bz_1) + r \cdot (Az_1 \circ Bz_2 + Az_2 \circ Bz_1) + r^2 \cdot (Az_2 \circ Bz_2) \\ &\neq Cz = C(z_1 + rz_2) = Cz_1 + rCz_2 = (Az_1 \circ Bz_1) + r \cdot (Az_2 \circ Bz_2) \end{aligned}$$

we end up with some inequality.
Nova's solution, Relaxed R1CS:

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

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

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

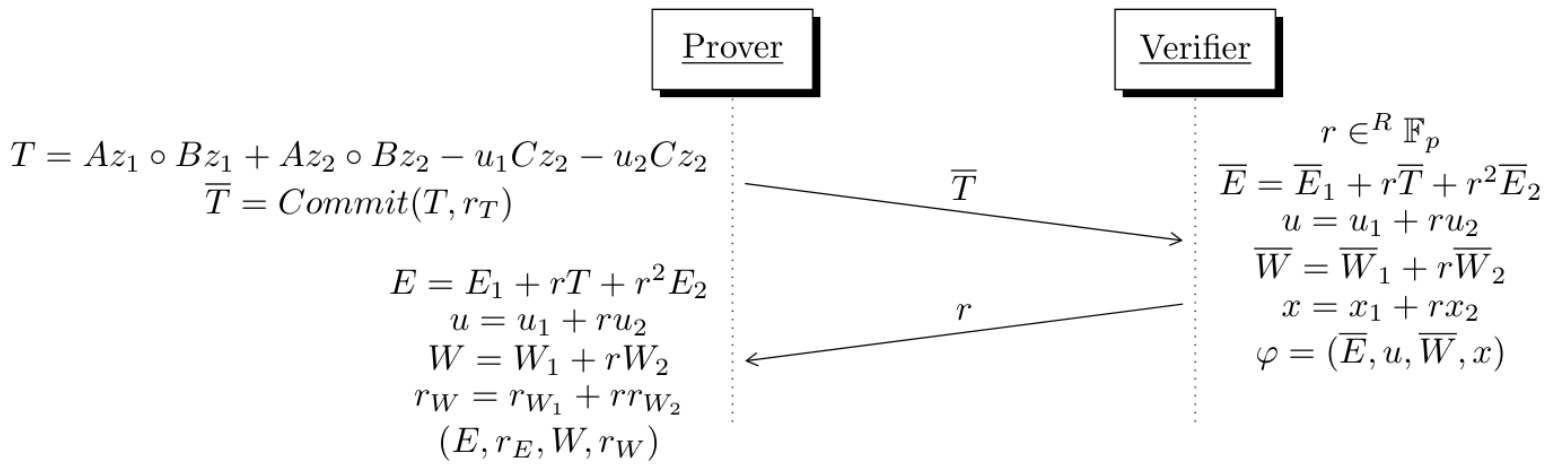
Full details at Nova's paper, pages 13-15 ("first attempt, second attempt, third attempt")

NIFS: Non Interactive Folding Scheme (in Nova)

Main idea:

- protocol between P (Prover) and V (Verifier)
- where P and V obtain the 'folded' committed instance that corresponds to the 'folded' witness instance that P computes
- so V does not know the witness

RelaxedR1CS relation: $Az \circ Bz = uCz + E$, for $u \in \mathbb{F}, \quad E \in \mathbb{F}^n$.
Witness of the instance: $WI = (E, W)$
Committed Relaxed R1CS instance: $CI = (\overline{E}, u, \overline{W}, x)$



Relation check:

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

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

NIMFS

issue: the RLC with > 2 instances makes the cross-terms grow
ie. some vectors (T) would get bigger and their commitments ($\overline{T}$) more expensive

NIMFS

issue: the RLC with > 2 instances makes the cross-terms grow
ie. some vectors (T) would get bigger and their commitments ($\overline{T}$) more expensive

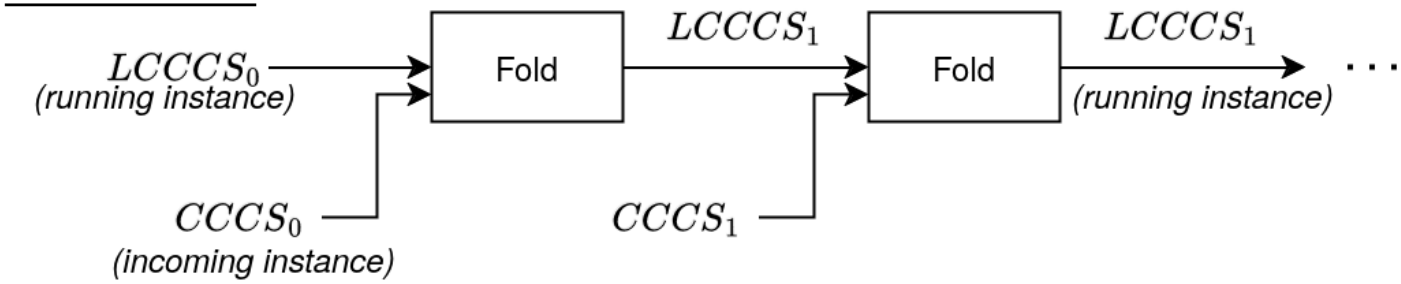
NIMFS in HyperNova, ProtoGalaxy, etc

- instead of 2 RLC between 2 instances
  - HyperNova: run SumCheck
  - ProtoGalaxy: protocol using Lagrange Basis (instead of random values) for the linear combinations and then the zero-check protocol
    alternatively using MLE and SumCheck

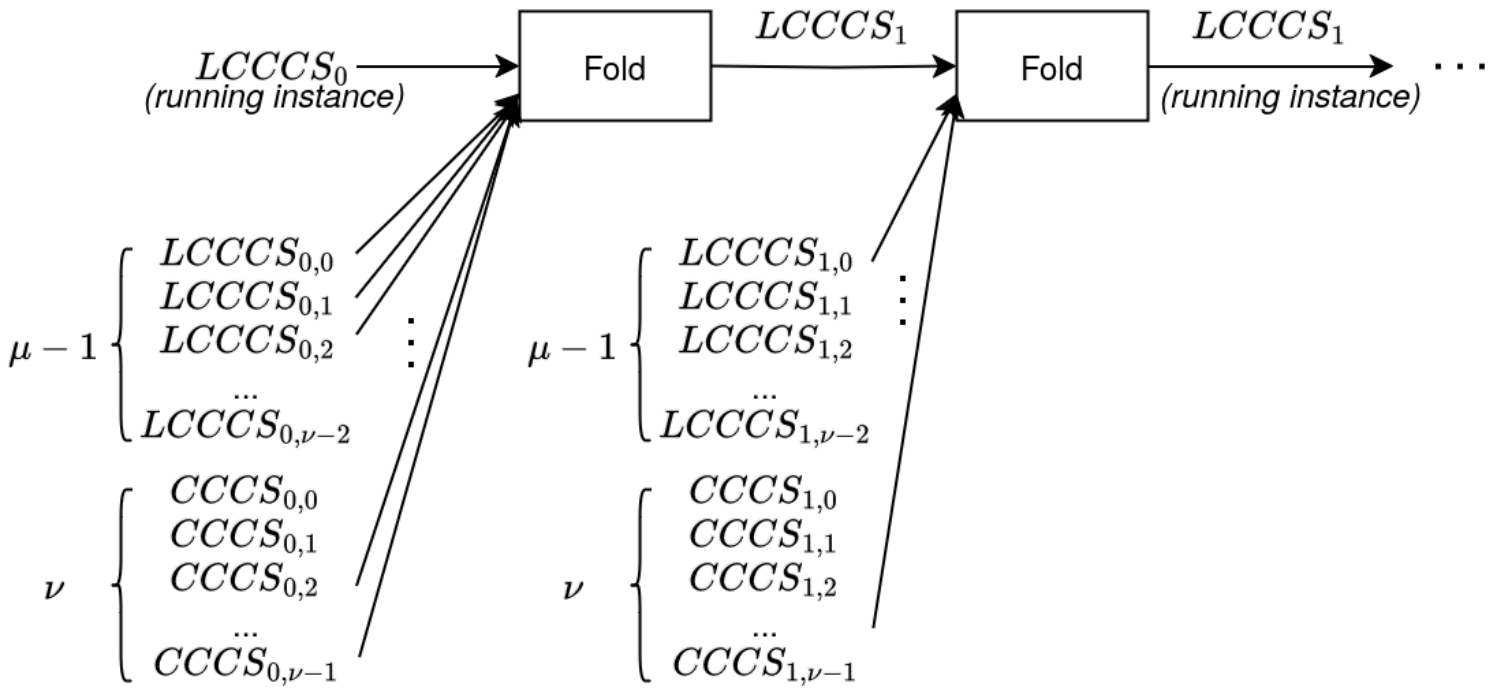
Allowing to fold > 2 instances at each folding step.

HyperNova NIMFS (Non-Interactive Multi Folding Scheme)

NIFS: 2-to-1 fold:



NIMFS: k-to-1 fold:



HyperNova NIMFS

src: https://eprint.iacr.org/2023/573.pdf (HyperNova paper)

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:

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

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

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

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

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

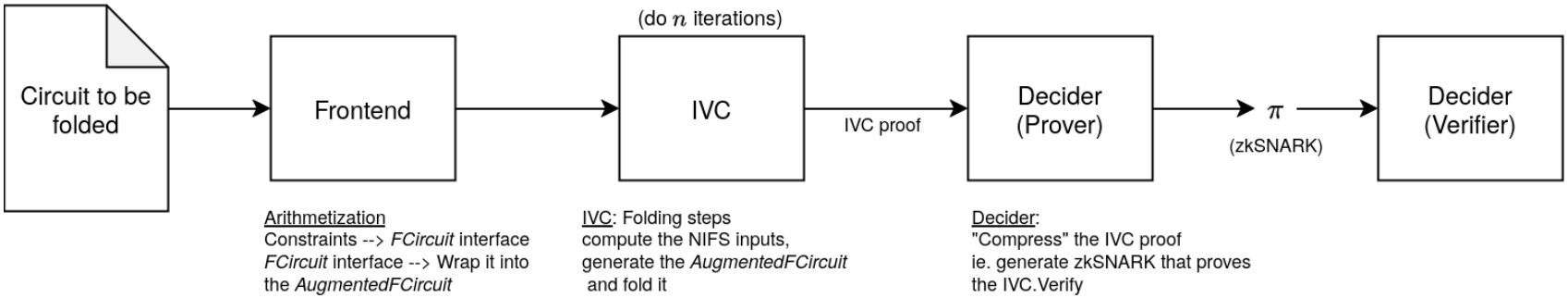
src: https://eprint.iacr.org/2023/1106.pdf (ProtoGalaxy paper)

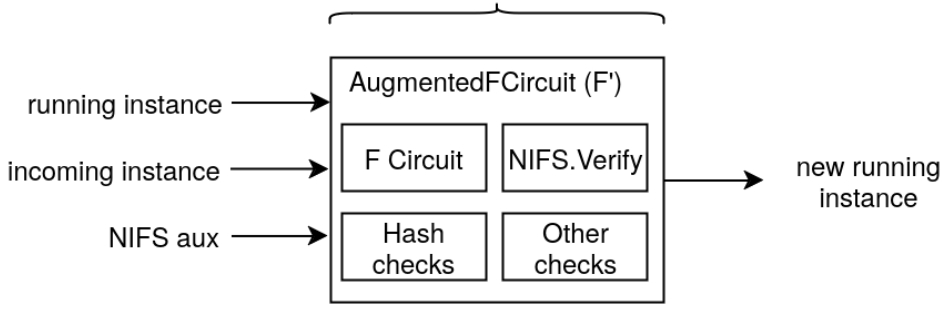
5. $\mathbf{P}$ sends the non-constant coefficients $F_1, \ldots, F_t$ of $F(X)$ to $\mathbf{V}$.
6. $\mathbf{V}$ sends a random challenge $\alpha \in \mathbb{F}$.
7. $\mathbf{P}$ and $\mathbf{V}$ compute $F(\alpha) = e + \sum_{i \in [t]} F_i \alpha^i$.
8. $\mathbf{P}$ and $\mathbf{V}$ compute $\boldsymbol{\beta}^*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

At the end of the protocol

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

(Recall) Folding scheme pipeline





The IVC folds the AugmentedFCircuit, which ensures that the NIFS.Verify checks out.

IVC - Incrementally Verifiable Computation

- need to ensure that at each fold (IVC step), the NIFS is correctly done
- want to have a construction where we fold the circuit that itself contains the checks of the Folding Scheme (NIFS.Verify)
- we encode the NIFS.Verify inside a circuit (*AugmentedFCircuit*), together with other checks that ensure the correct relations
- and is this *AugmentedFCircuit* (F') the one which we actually fold

Folding circuit - AugmentedFCircuit

wrapper over the circuit that we want to fold

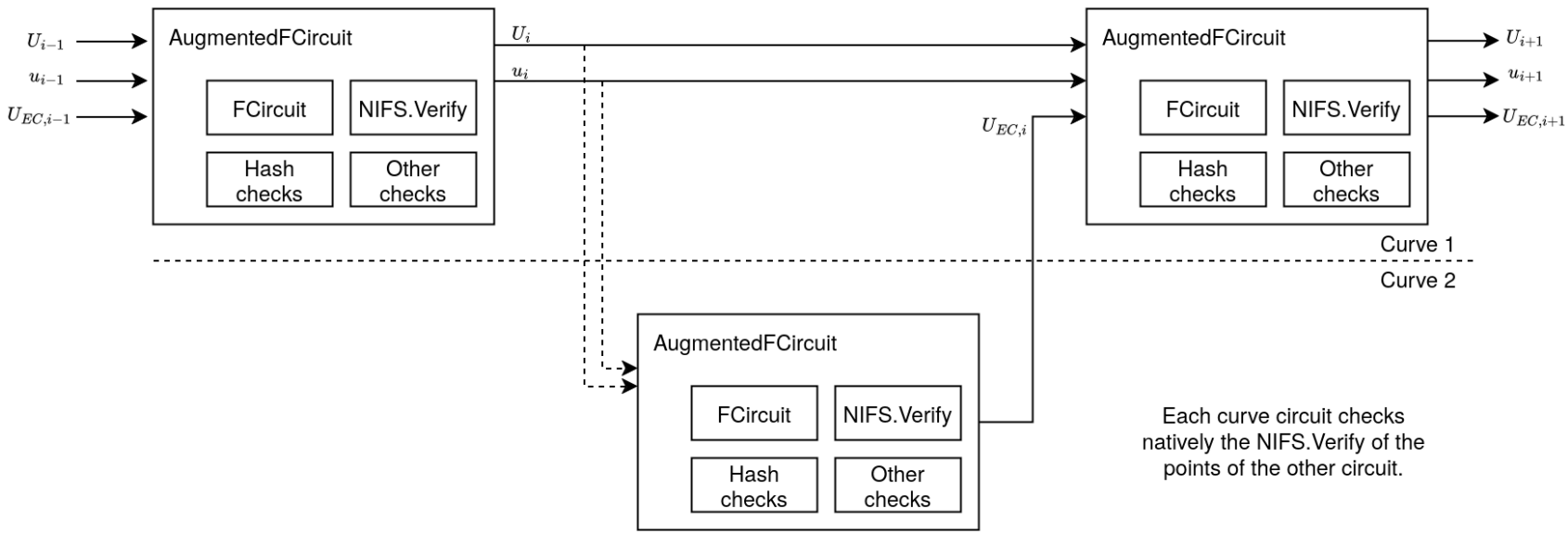
- contains the circuit that we want to fold
- plus extra checks ensuring that the fold is being done correctly
  - some hashing and values checks
  - run the NIFS.Verify in-circuit
    - Nova: RLC of $\mathbb{F}$ and $\mathbb{G}$ elements
    - HyperNova: RLC of $\mathbb{F}$ and $\mathbb{G}$ elements + SumCheck verifier
    - ProtoGalaxy: linear cominations of $\mathbb{F}$ and $\mathbb{G}$ elements with the Lagrange basis and some polynomial evaluations + ... (alternatively SumCheck)

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's circuit

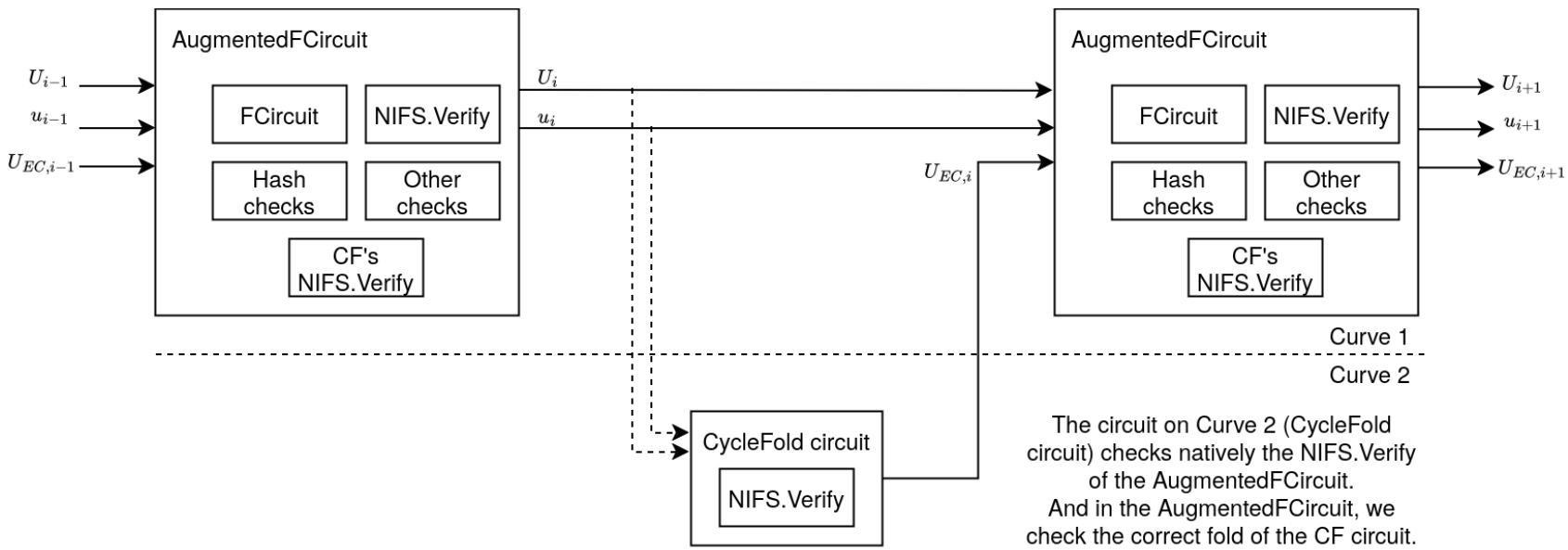


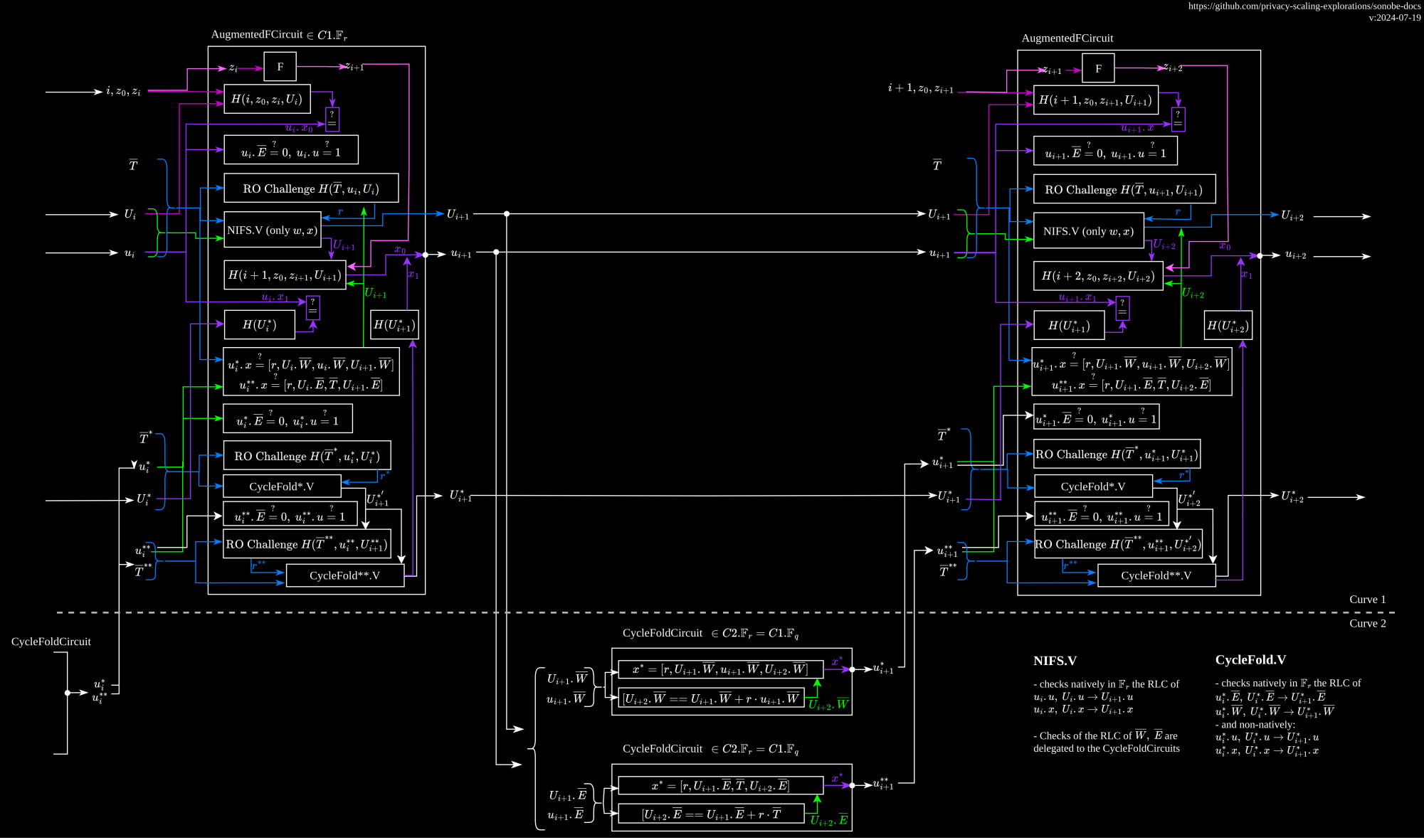
Cycle of curves

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's circuit

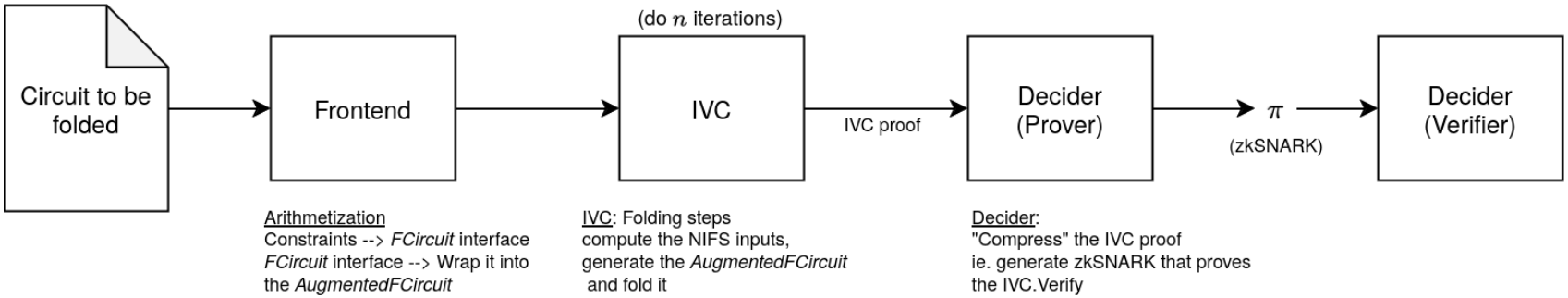


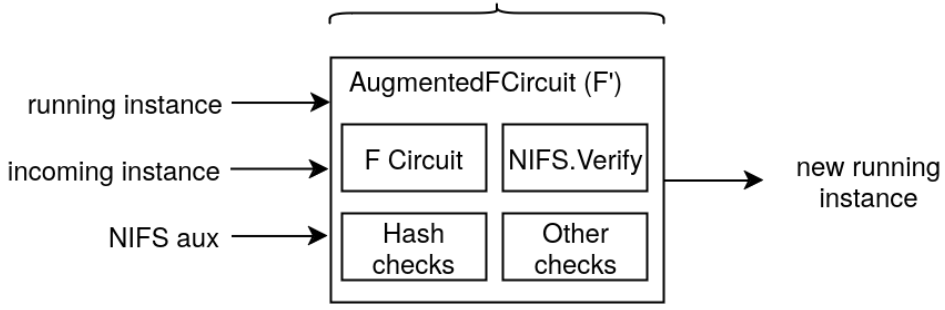


Nova's Folding Circuit (F') + CycleFold circuit

src: https://privacy-scaling-explorations.github.io/sonobe-docs/design/novacyclefold-circuit.html

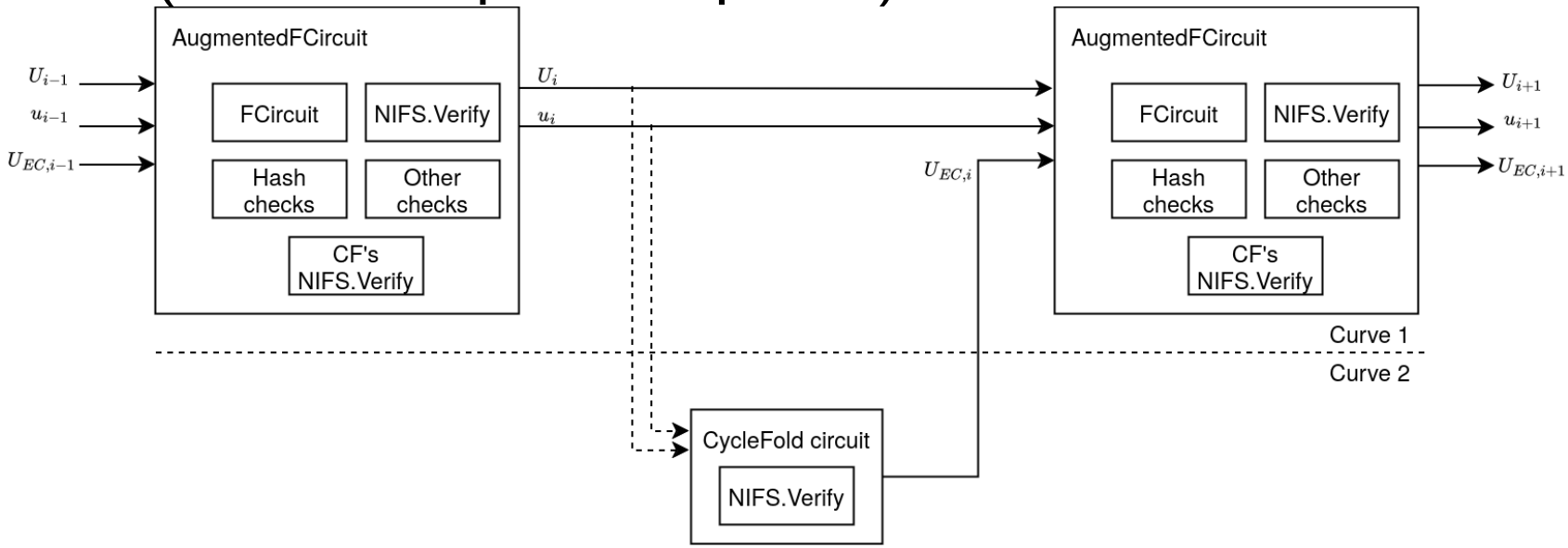
(Recall) Folding scheme pipeline





The IVC folds the AugmentedFCircuit, which ensures that the NIFS.Verify checks out.

Decider (Final compressed proof)



Issue: IVC proof is not succinct.

- Committed Instances of the AugmentedFCircuit U_n, u_n, where each one is $\{\overline{E} \in \mathbb{G}_1, \overline{W} \in \mathbb{G}_1, u \in \mathbb{F}_r, x \in \mathbb{F}_r^{|io|}\}$
- and their respective Witnesses W_n, w_n, each one being $\{E \in \mathbb{F}_r^n, W \in \mathbb{F}_r^n\}$
- Committed instance of the CycleFold circuit $U_{EC,n}$ contains: $\{\overline{E} \in \mathbb{G}_2, \overline{W} \in \mathbb{G}_2, u \in \mathbb{F}_q, x \in \mathbb{F}_q^{|io|}\}$
- and its Witnesses $W_{EC,n}$, being $\{E \in \mathbb{F}_r^n, W \in \mathbb{F}_r^n\}$

So the idea is to 'compress' the IVC proof into a zkSNARK proof, minimizing the size and the verifier computation.

Decider checks

1. check $NIFS.V(r, U_n, u_n, \overline{T}) \stackrel{?}{=} U_{n+1}$
2. check that $u_n.\overline{E} = 0$ and $u_n.u = 1$
3. check that $u_n.x_0 = H(n, z_0, z_n, U_n)$ and $u_n.x_1 = H(U_{EC,n})$
4. correct RelaxedR1CS relation of U_{n+1}, W_{n+1} of the AugmentedFCircuit
5. check commitments of $U_{n+1}.\{\overline{E}, \overline{W}\}$ with respect W_{n+1} (where $\overline{E}, \overline{W} \in E_1$)
6. check the correct RelaxedR1CS relation of $U_{EC,n}, W_{EC,n}$ of the CycleFoldCircuit
7. check commitments of $U_{EC,n}.\{\overline{E}, \overline{W}\}$ with respect $W_{EC,n}$ (where $\overline{E}, \overline{W} \in E_2$)

Decider checks

1. check $NIFS.V(r, U_n, u_n, \overline{T}) \stackrel{?}{=} U_{n+1}$
2. check that $u_n.\overline{E} = 0$ and $u_n.u = 1$
3. check that $u_n.x_0 = H(n, z_0, z_n, U_n)$ and $u_n.x_1 = H(U_{EC,n})$
4. correct RelaxedR1CS relation of U_{n+1}, W_{n+1} of the AugmentedFCircuit
5. check commitments of $U_{n+1}.\{\overline{E}, \overline{W}\}$ with respect W_{n+1} (where $\overline{E}, \overline{W} \in E_1$) non-native!
6. check the correct RelaxedR1CS relation of $U_{EC,n}, W_{EC,n}$ of the CycleFoldCircuit non-native!
7. check commitments of $U_{EC,n}.\{\overline{E}, \overline{W}\}$ with respect $W_{EC,n}$ (where $\overline{E}, \overline{W} \in E_2$)

Original Nova Decider

- Original Nova: wrapp these checks in 2 Spartan (zkSNARK) proofs (one over each curve of the cycle of curves).
  $\longrightarrow$ 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
- constrainted by gas costs

Onchain Decider

1.1: check that the given NIFS challenge r is indeed well computed.
This challenge is then used outside the circuit by the Verifier to compute NIFS.V obtaining U_{i+1}

2: check that $u_n.\overline{E} = 0$ and $u_n.u = 1$

3: check that $u_n.x_0 = H(n, z_0, z_n, U_n)$ and $u_n.x_1 = H(U_{EC,n})$

4: correct RelaxedR1CS relation of U_{n+1}, W_{n+1} of the AugmentedFCircuit

5.1: Check correct computation of the KZG challenges
$c_E = H(\overline{E}.\{x, y\}), \quad c_W = H(\overline{W}.\{x, y\})$
which we do through in-circuit Transcript.

5.2: check that the KZG evaluations are correct, $eval_W == p_W(c_W), \quad eval_E == p_E(c_E)$
where $p_W, p_E \in \mathbb{F}[X]$ are the interpolated polynomials from $W_{i+1}.W, \ W_{i+1}.E$ respectively.

6: check the correct RelaxedR1CS relation of $U_{EC,n}, W_{EC,n}$ of the CycleFoldCircuit
(this is non-native operations and with naive sparse matrix-vector product blows up the number of constraints)

7: Pedersen commitments verification of $U_{EC,n}.\{\overline{E}, \overline{W}\}$ with respect $W_{EC,n}$
(the witness of the committed instance)
(where $\overline{E}, \overline{W} \in E_2$, this check is native in F_r)

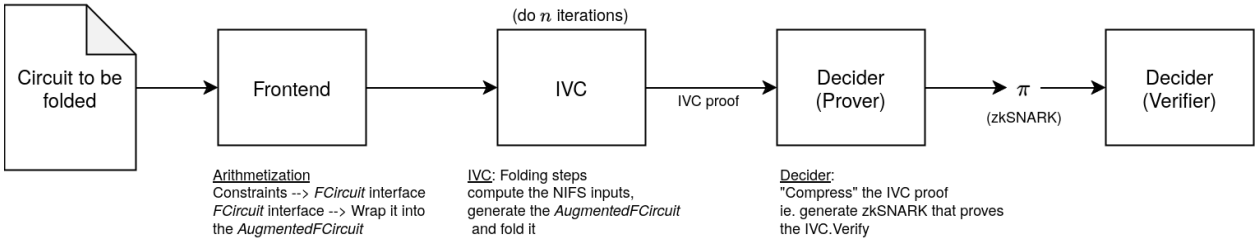
The following check is done non-natively (in F_r): Additionally we would have to check (outside of the circuit):

1.2: check $NIFS.V(r, U_n, u_n, \overline{T}) \stackrel{?}{=} U_{n+1}$

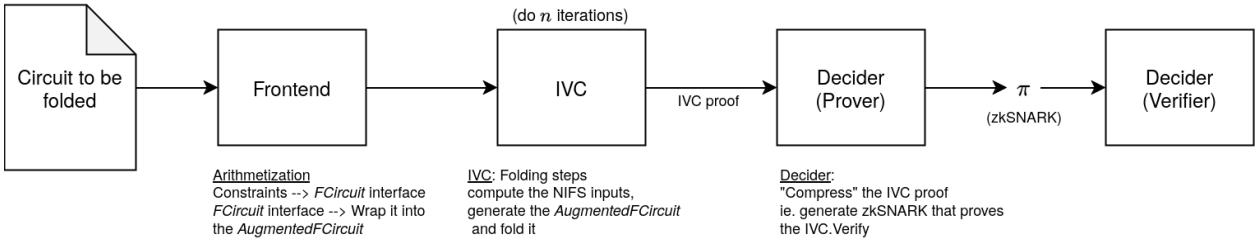
5.3: Commitments verification of $U_{n+1}.\{\overline{E}, \overline{W}\}$ with respect W_{n+1} (where $\overline{E}, \overline{W} \in E_1$)

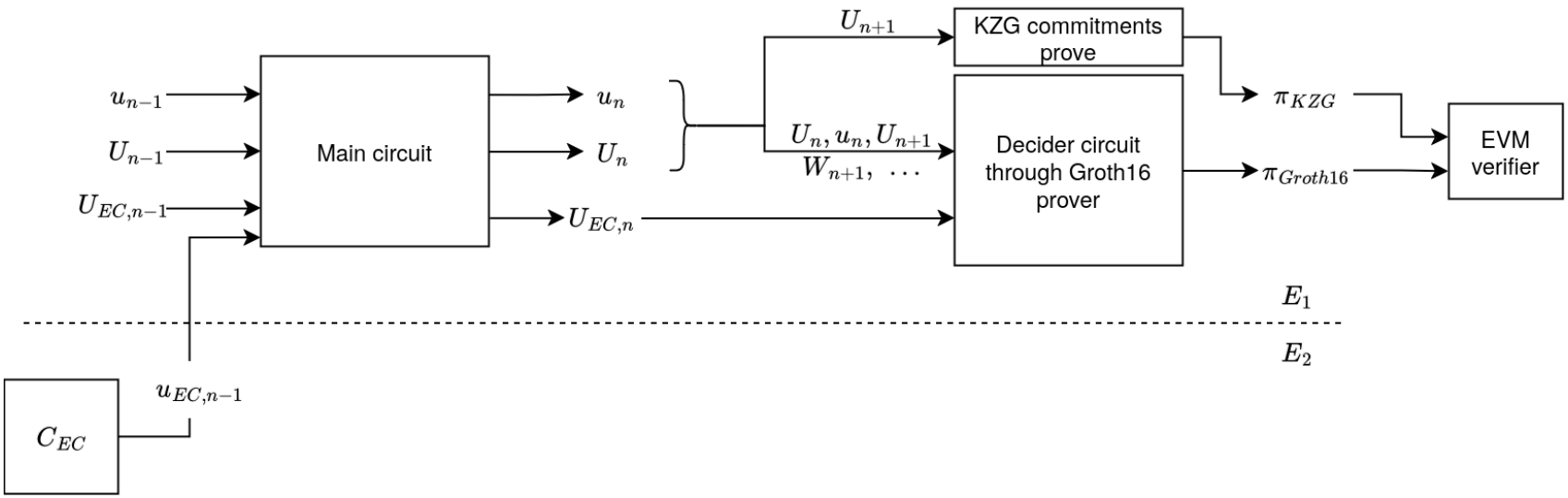
More details in Sonobe's docs: https://privacy-scaling-explorations.github.io/sonobe-docs/design/novacyclefold-circuit.html

Onchain Decider



Onchain Decider





Sonobe

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

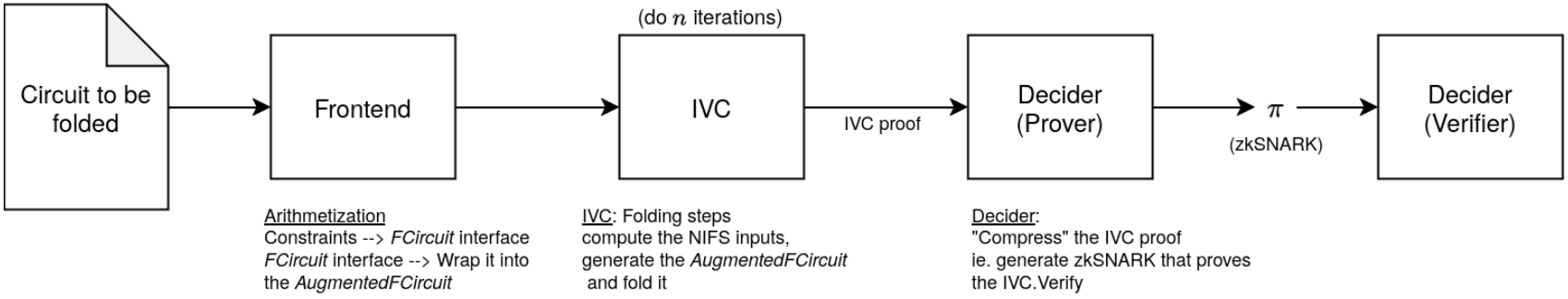
- Be able to
  - Add and test new folding schemes
  - Compare schemes 'apples-to-apples'
  - Researchers can easily add their own schemes (eg. Mova paper)
- Make it easy for devs to use folding
  - minimal code to fold your circuits ('plug-and-fold')
  - easy to switch between folding schemes and curves
  - support of multiple zk-circuit languages

Remark: experimental and unoptimized.

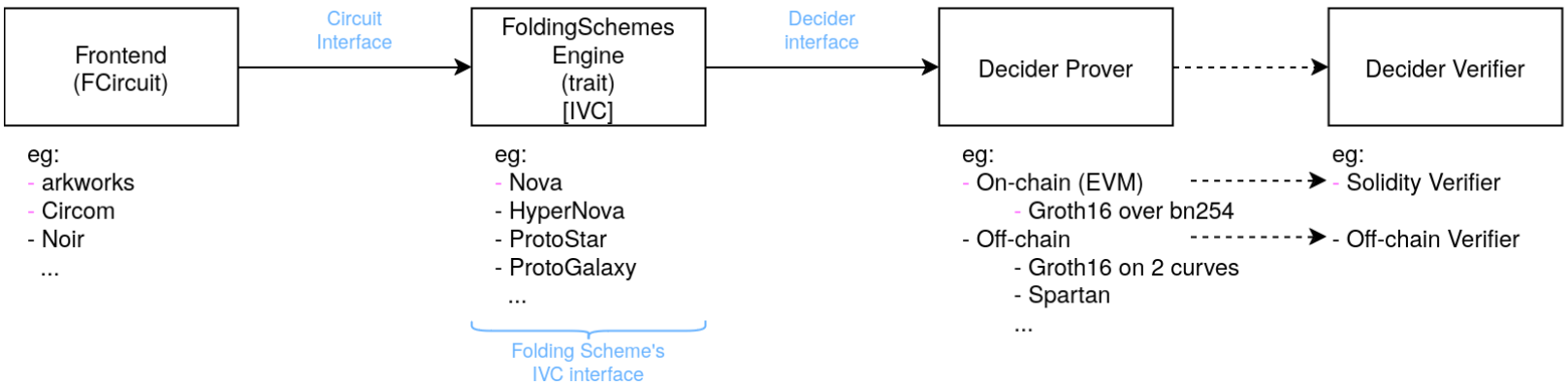
Sonobe - Dev experience

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



Sonobe lib pipeline:



Status of Sonobe - schemes implemented

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

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

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

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

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

Modularity

Big thanks to arkworks https://github.com/arkworks-rs

[Code example in the code editor]

Some numbers

(code highly unoptimized)

- folding step $\sim 300ms$
  - Folding circuit (Nova+CycleFold): $\sim 50k$ R1CS constraints (overhead)
- Offchain Decider prove: < 1 min
- Onchain Decider Verification:
  - 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
    - can be reduced by hashing the public inputs & batching the pairings check
    - 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.

Wrappup

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

2024-10-08

0xPARC & PSE.