Investigating IVC with Accumulation Schemes

Rasmus Kirk Jakobsen

Computer Science Aarhus

2025-02-04 - 14:21:32 UTC

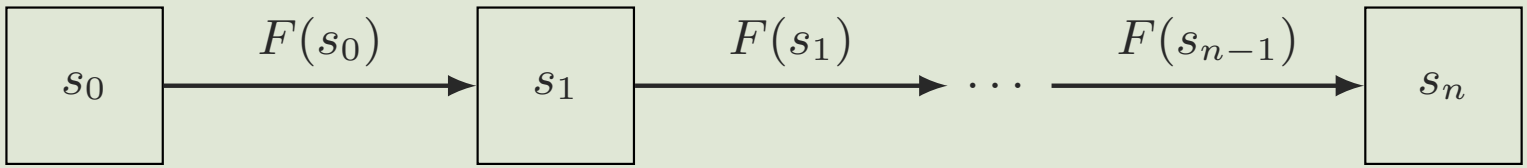
Simple IVC

Motivation

IVC is designed to solve the following problem:

If a computation runs for hundreds of years and ultimately outputs 42, how can we check its correctness without re-executing the entire process?

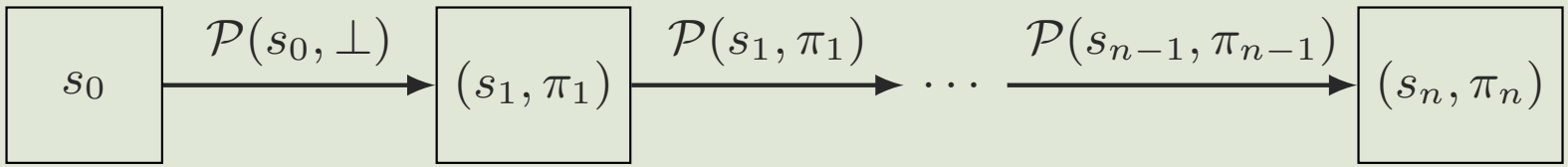
We define the transition function F run on an initial state s_0:



- *How can we verify $s_n = F^n(s_0)$ without re-executing the computation?*

IVC chain

We can use a SNARK to prove each computation step:



$\mathcal{P}(s_{i-1}, \pi_{i-1})$ represents:

- $s_i = F(s_{i-1})$
- $\pi_i = \mathsf{SNARK.PROVER}(R, x = \{s_0, s_i\}, w = \{s_{i-1}, \pi_{i-1}\})$
- $R :=$ I.K. $w = \{\pi_{i-1}, s_{i-1}\}$ s.t.
  - $s_i \overset{?}{=} F(s_{i-1}) \ \wedge \ (s_{i-1} \overset{?}{=} s_0 \vee \mathcal{V}(R, x = \{s_0, s_i\}, \pi_{i-1}))$

Proof

- R gives us a series of proofs of the claims:

$$\begin{aligned}
&\text{I.K. } w \text{ s.t. } && s_n = F(s_{n-1}) \wedge (s_{n-1} = s_0 \vee \mathcal{V}(R, x, \pi_{n-1}) = \top),\\
&\text{I.K. } w \text{ s.t. } && s_{n-1} = F(s_{n-2}) \wedge (s_{n-2} = s_0 \vee \mathcal{V}(R, x, \pi_{n-2}) = \top), \ \ldots\\
&\text{I.K. } w \text{ s.t. } && s_1 = F(s_0) \wedge (s_0 = s_0 \vee \mathcal{V}(R, x, \pi_0) = \top)
\end{aligned}$$

- Which, if all verify means that:

$$\begin{aligned}
&\textsf{SNARK.Verifier}(R, x, \pi_n) = \top \implies\\
&s_n = F(s_{n-1}) \wedge\\
&\textsf{SNARK.Verifier}(R, x, \pi_{n-1}) = \top \implies\\
&s_{n-1} = F(s_{n-2}) \wedge\\
&\textsf{SNARK.Verifier}(R, x, \pi_{n-2}) = \top \implies \ldots\\
&s_1 = F(s_0)
\end{aligned}$$

PCS based on DL

Polynomial Commitment Scheme

- $\mathsf{PC}_{\mathsf{DL}}.\textsc{Setup}(\lambda, D)^{\rho_0} \rightarrow \mathrm{pp}_{\mathsf{PC}}$: $(S, D, H, \boldsymbol{G})$
- $\mathsf{PC}_{\mathsf{DL}}.\textsc{Commit}(p : \mathbb{F}_q^{d'}[X], d : \mathbb{N}) \rightarrow \mathbb{E}(\mathbb{F}_q)$:

  Creates a Pedersen commit to $\boldsymbol{p}^{(\text{coeffs})}$ of degree $d' \leq d$.
- $\mathsf{PC}_{\mathsf{DL}}.\textsc{Open}^{\rho_0}(p : \mathbb{F}_q^{d'}[X], C : \mathbb{E}(\mathbb{F}_q), d : \mathbb{N}, z : \mathbb{F}_q) \rightarrow \mathbf{EvalProof}$:

  "I know degree $d' \leq d$ polynomial p with commit C s.t. $p(z) = v$"
- $\mathsf{PC}_{\mathsf{DL}}.\textsc{SuccinctCheck}^{\rho_0}(q : \mathbf{Instance}) \rightarrow \mathbf{Result}((\mathbb{F}_q^d[X], \mathbb{E}(\mathbb{F}_q)), \bot)$:

  Partially check π. The expensive part of the full check is deferred.
- $\mathsf{PC}_{\mathsf{DL}}.\textsc{Check}^{\rho_0}(q : \mathbf{Instance}) \rightarrow \mathbf{Result}(\top, \bot)$:

  The full check on π.

$$q : \mathbf{Instance} = (C : \mathbb{E}(\mathbb{F}_q), d : \mathbb{N}, z : \mathbb{F}_q, v : \mathbb{F}_q, \pi : \mathbf{EvalProof})$$

Notes on Checking Evaluation Proofs for $\mathsf{PC}_{\mathsf{DL}}$

Verifying checks:

1. $\pi = \boldsymbol{L}, \boldsymbol{R}, U, c$
2. $C_{lg(n)} \stackrel{?}{=} cU + ch(z)H' = c^{(0)}G^{(0)} + c^{(0)}z^{(0)}H'$
3. $U \stackrel{?}{=} \mathsf{CM.Commit}(\boldsymbol{G}, \boldsymbol{h}^{(\text{coeffs})}) \stackrel{?}{=} \langle \boldsymbol{G}, \boldsymbol{h}^{(\text{coeffs})} \rangle$

- $\mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}$ either rejects or accepts and returns:
  - U:
    - Represents $G^{(0)}$.
    - May be wrong!
  - $h(X) := \prod_{i=0}^{\lg(n)-1}(1 + \xi_{\lg(n)-i}X^{2^i})$:
    - Compression polynomial for $\boldsymbol{G} \to G^{(0)}, z \to z^{(0)}$.
    - Degree-d, $\mathcal{O}(\lg(d))$ evaluation time.

AS based on DL

Overview

Generally

- $\mathsf{AS.Setup}(\lambda) \to \mathrm{pp}_{\mathsf{AS}}$
- $\mathsf{AS.Prover}(\boldsymbol{q} : \mathbf{Instance}^m, acc_{i-1} : \mathbf{Option}(\mathbf{Acc})) \to \mathbf{Acc}$
- $\mathsf{AS.Verifier}(\boldsymbol{q} : \mathbf{Instance}^m, acc_{i-1} : \mathbf{Option}(\mathbf{Acc}), acc_i : \mathbf{Acc}) \to \mathbf{Result}(\top, \bot)$
- $\mathsf{AS.Decider}(acc_i : \mathbf{Acc}) \to \mathbf{Result}(\top, \bot)$

$\mathsf{AS}_{\mathsf{DL}}$: Based on Discrete Log

- $\mathsf{AS}_{\mathsf{DL}}.\mathsf{Setup}(1^\lambda, D) \to \mathrm{pp}_{\mathsf{AS}}$
- $\mathsf{AS}_{\mathsf{DL}}.\mathsf{Prover}(\boldsymbol{q} : \mathbf{Instance}^m) \to \mathbf{Result}(\mathbf{Acc}, \bot)$:
- $\mathsf{AS}_{\mathsf{DL}}.\mathsf{Verifier}(\boldsymbol{q} : \mathbf{Instance}^m, \mathrm{acc}_i : \mathbf{Acc}) \to \mathbf{Result}(\top, \bot)$:
- $\mathsf{AS}_{\mathsf{DL}}.\mathsf{Decider}(\mathrm{acc}_i : \mathbf{Acc}) \to \mathbf{Result}(\top, \bot)$:
- $\mathsf{AS}_{\mathsf{DL}}.\mathsf{CommonSubroutine}(\boldsymbol{q} : \mathbf{Instance}^m) \to \mathbf{Result}((\mathbb{E}(\mathbb{F}_q), \mathbb{N}, \mathbb{F}_q, \mathbb{F}_q^d[X]), \bot)$

Prover

Algorithm $\mathsf{AS}_{\mathsf{DL}}.\textsc{Prover}$

Inputs

$\boldsymbol{q} : \mathbf{Instance}^m$

Output

$\mathbf{Result}(\mathbf{Acc}, \perp)$

Require: $\forall d_i \in \boldsymbol{q}, \forall d_j \in \boldsymbol{q} : d_i = d_j \wedge d_i \leq D$

Require: $(d_i + 1) = 2^k$, where $k \in \mathbb{N}$

1: Compute the tuple $(\bar{C}, d, z, \bar{h}(X)) := \mathsf{AS}_{\mathsf{DL}}.\textsc{CommonSubroutine}(\boldsymbol{q})$.

2: Generate the evaluation proof $\pi := \mathsf{PC}_{\mathsf{DL}}.\textsc{Open}(\bar{h}(X), \bar{C}, d, z)$.

3: Finally, output the accumulator $\mathrm{acc}_i = (\bar{C}, d, z, v := \bar{h}(z), \pi)$.

Verifier

Algorithm $\mathsf{AS}_{\mathsf{DL}}.\textsc{Verifier}$

Inputs

$\boldsymbol{q} : \mathbf{Instance}^m$

$\mathrm{acc}_i : \mathbf{Acc}$

Output

$\mathbf{Result}(\top, \bot)$

Require: $(d+1) = 2^k$, where $k \in \mathbb{N}$

1: Parse acc_i as $(\bar{C}_{\mathrm{acc}}, d_{\mathrm{acc}}, z_{\mathrm{acc}}, v_{\mathrm{acc}}, \_)$

2: Compute $(\bar{C}, d, z, \bar{h}(X)) := \mathsf{AS}_{\mathsf{DL}}.\textsc{CommonSubroutine}(\boldsymbol{q})$

3: Then check that $C_{\mathrm{acc}} \stackrel{?}{=} \bar{C}, d_{\mathrm{acc}} \stackrel{?}{=} d, z_{\mathrm{acc}} \stackrel{?}{=} z$, and $v_{\mathrm{acc}} \stackrel{?}{=} \bar{h}(z)$.

Decider

Algorithm 1 $\mathsf{AS}_{\mathsf{DL}}.\textsc{Decider}$

Inputs

$\mathrm{acc}_i : \mathbf{Acc}$

Output

$\mathbf{Result}(\top, \bot)$

Require: $\mathrm{acc}_i.d \leq D$

Require: $(\mathrm{acc}_i.d + 1) = 2^k$, where $k \in \mathbb{N}$

1: Check $\top \stackrel{?}{=} \mathsf{PC}_{\mathsf{DL}}.\textsc{Check}(\mathrm{acc}_i)$

Common Subroutine

Algorithm $\mathsf{AS}_{\mathsf{DL}}.\textsc{CommonSubroutine}$

Inputs

$\boldsymbol{q} : \mathbf{Instance}^m$

Output

$\mathbf{Result}((\mathbb{E}(\mathbb{F}_q), \mathbb{N}, \mathbb{F}_q, \mathbb{F}_q^d[X]), \perp)$

Require: $(d+1) = 2^k$, where $k \in \mathbb{N}$

1: Parse d from q_1.
2: **for** $q_j \in \boldsymbol{q}$ **do**
3: Parse d_j from q_j.
4: Compute $(h_j(X), U_j) := \mathsf{PC}_{\mathsf{DL}}.\textsc{SuccinctCheck}^{\rho_0}(q_j)$.
5: Check that $d_j \stackrel{?}{=} d$
6: **end for**
7: Compute the challenge $\alpha := \rho_1(\boldsymbol{h}, \boldsymbol{U})$
8: Linearize h_j, U_j: $\bar{h}(X) := \sum_{j=1}^{m} \alpha^j h_j(X)$, $\bar{C} := \sum_{j=1}^{m} \alpha^j U_j$
9: Compute: $z := \rho_1(\bar{C}, \bar{h}(X))$
10: Output $(\bar{C}, d, z, \bar{h}(X))$.

AS Properties

AS Completeness

Assuming that $\mathsf{PC}_{\mathsf{DL}}$ is complete.

- $\mathsf{AS}_{\mathsf{DL}}.\textsc{Verifier}$:
  - Runs the same deterministic $\mathsf{AS}_{\mathsf{DL}}.\textsc{CommonSubroutine}$ with the same inputs.
  - Given that $\mathsf{AS}_{\mathsf{DL}}.\textsc{Prover}$ is honest, $\mathsf{AS}_{\mathsf{DL}}.\textsc{Verifier}$ will get the same outputs.
  - These are checked to be equal to the ones from acc_i.
  - Therefore $\mathsf{AS}_{\mathsf{DL}}.\textsc{Verifier}$ accepts with probability 1.
- $\mathsf{AS}_{\mathsf{DL}}.\textsc{Decider}$:
  - Runs $\mathsf{PC}_{\mathsf{DL}}.\textsc{Check}$ on the acc_i, representing an evaluation proof (instance).
  - The prover constructed the acc_i honestly.
  - Therefore this check will pass with probability 1.

AS Soundness

- $\mathsf{AS_{DL}}.\textsc{Verifier}$ shows that $\bar{h}(X), \bar{C}$ are linear combinations of $h_j(X), U_j$'s.
- $\mathsf{AS_{DL}}.\textsc{Decider}$ shows that $\bar{C} = \mathsf{PC_{DL}}.\textsc{Commit}(\bar{h}(X), d)$, checks all U_j's.
  - Let $\mathrm{acc}_i = (\bar{C}, d, z, v, \pi)$
  - $\mathsf{PC_{DL}}.\textsc{Check}$ shows that $\bar{C}$ is a commitment to $\bar{h}'(X)$ and $\bar{h}'(z) = v$.
  - $\mathsf{AS_{DL}}.\textsc{Verifier}$ shows that $\bar{C} = \sum_{j=1}^{m} \alpha^j U_j$, $\bar{h}(z) = v$.
  - Since $v = \bar{h}(z) = \bar{h}'(z)$ then $\bar{h}(X) = \bar{h}'(X)$ with $1 - \Pr[d/|\mathbb{F}_q|]$.
  - Define $\forall j \in [m] : B_j = \langle \boldsymbol{G}, \boldsymbol{h_j}^{(\text{coeffs})} \rangle$. If $\exists j \in [m]\ B_j \neq U_j$ then:
    - U_j is not a valid commitment to $h_j(X)$ and,
    - $\sum_{j=1}^{m} \alpha^j B_j \neq \sum_{j=1}^{m} \alpha^j U_j$

    As such $\bar{C}$ will not be a valid commitment to $\bar{h}(X)$. Unless,
  - $\alpha := \rho_1(\boldsymbol{h}, \boldsymbol{U})$ or $z = \rho_1(\bar{C}, \bar{h}(X))$ is constructed maliciously.
- Previous accumulators are instances, as such, they will also be checked.
- More formal argument in the report.

AS Efficiency

Analysis

- $\mathsf{AS}_{\mathsf{DL}}$.CommonSubroutine:
  - Step 4: m calls to $\mathsf{PC}_{\mathsf{DL}}$.SuccinctCheck, $\mathcal{O}(m \lg(d))$ scalar muls.
  - Step 8: $m \lg(d)$ field muls.
  - Step 8: m scalar muls.

  Step 4 dominates with $\mathcal{O}(m \lg(d))$ scalar muls.
- $\mathsf{AS}_{\mathsf{DL}}$.Prover:
  - Step 1: Call to $\mathsf{AS}_{\mathsf{DL}}$.CommonSubroutine, $\mathcal{O}(m \lg(d))$ scalar muls.
  - Step 2: Call to $\mathsf{PC}_{\mathsf{DL}}$.Open, $\mathcal{O}(d)$ scalar muls.
  - Step 3: Evaluation of $\bar{h}(X)$, $\mathcal{O}(m \lg(d))$ field muls.

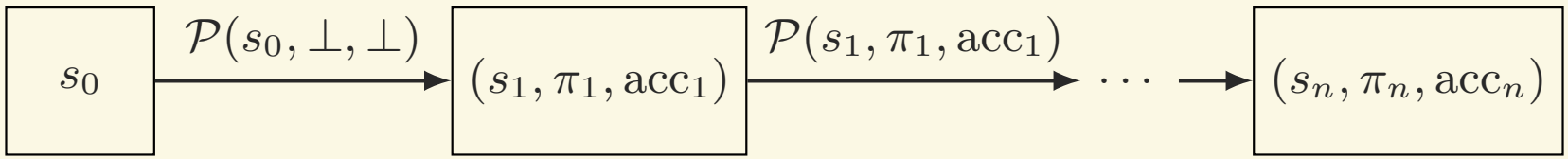
  Step 2 dominates with $\mathcal{O}(d)$ scalar muls.
- $\mathsf{AS}_{\mathsf{DL}}$.Verifier:
  - Step 2: Call to $\mathsf{AS}_{\mathsf{DL}}$.CommonSubroutine, $\mathcal{O}(m \lg(d))$ scalar muls.
- $\mathsf{AS}_{\mathsf{DL}}$.Decider:
  - Step 1: Call to $\mathsf{PC}_{\mathsf{DL}}$.Check, with $\mathcal{O}(d)$ scalar muls.

So $\mathsf{AS}_{\mathsf{DL}}$.Prover, $\mathsf{AS}_{\mathsf{DL}}$.Decider are linear and $\mathsf{AS}_{\mathsf{DL}}$.Verifier is sub-linear.

IVC based on AS

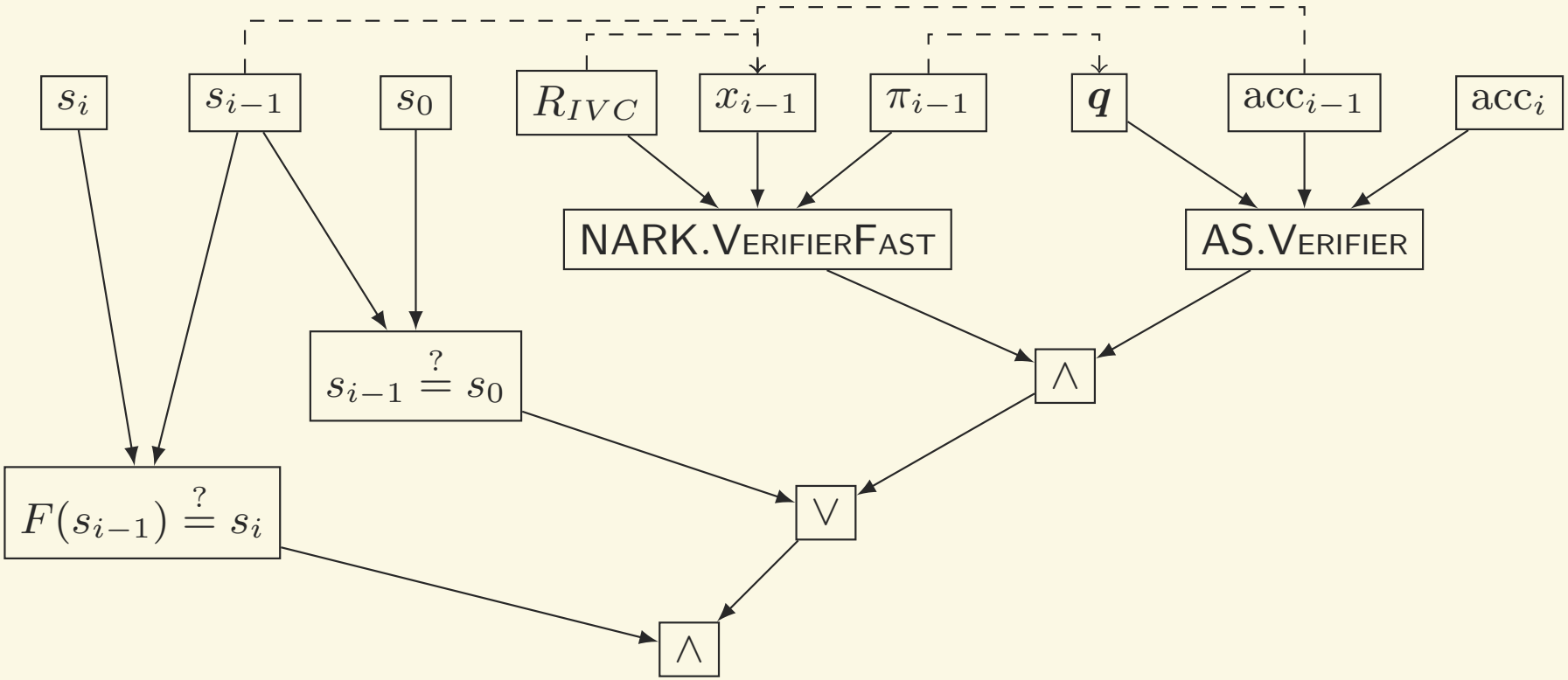
Assuming a Nark

- We assume we have an underlying NARK which proof consists of only instances $\pi \in \mathbf{Proof} = [\boldsymbol{q}]^m$. We assume this NARK has three algorithms:
  - NARK.Prover$(R : \mathbf{Circuit}, x : \mathbf{PublicInputs}, w : \mathbf{Witness}) \rightarrow \mathbf{Proof}$
  - NARK.Verifier$(R : \mathbf{Circuit}, x : \mathbf{PublicInputs}, \pi : \mathbf{Proof}) \rightarrow \mathbf{Result}(\top, \bot)$
  - NARK.VerifierFast$(R : \mathbf{Circuit}, x : \mathbf{PublicInputs}, \pi : \mathbf{Proof}) \rightarrow \mathbf{Result}(\top, \bot)$



The Circuit Visualized

- $x = \{R_{IVC}, s_0, s_i, \mathrm{acc}_i\}$
- $w = \{s_{i-1}, \pi_{i-1} = \boldsymbol{q}, \mathrm{acc}_{i-1}\}$



The IVC prover

Algorithm IVC.Prover

Inputs

$R_{IVC} : \mathbf{Circuit}$	The IVC circuit as defined above.
$x : \mathbf{PublicInputs}$	Public inputs for R_{IVC}.
$w : \mathbf{Option}(\mathbf{Witness})$	Private inputs for R_{IVC}.

Output

$(S, \mathbf{Proof}, \mathbf{Acc})$	The values for the next IVC iteration.

Require: $x = \{s_0\}, \quad w = \{s_{i-1}, \pi_{i-1}, \mathrm{acc}_{i-1}\} \vee w = \bot$

1: **If** (base-case) **Then** ... **Else**
2: Run the accumulation prover: $\mathrm{acc}_i = \text{AS.Prover}(\pi_{i-1} = \boldsymbol{q}, \mathrm{acc}_{i-1})$.
3: Compute the next value: $s_i = F(s_{i-1})$.
4: Define $x' = x \cup \{R_{IVC}, s_i, \mathrm{acc}_i\}$.
5: Generate a NARK proof using R_{IVC}: $\pi_i = \text{NARK.Prover}(R_{IVC}, x', w)$.
6: Output $(s_i, \pi_i, \mathrm{acc}_i)$

The IVC Verifier

Algorithm IVC.VERIFIER

Inputs

R_{IVC} : **Circuit**	The IVC circuit.
x : **PublicInputs**	Public inputs for R_{IVC}.

Output

Result$(\top, \bot)$	Returns $\top$ or $\bot$.

Require: $x = \{s_0, s_i, \mathrm{acc}_i\}$

1: Define $x' = x \cup \{R_{IVC}\}$.
2: Verify that the accumulation scheme decider accepts: $\top \stackrel{?}{=}$ AS.DECIDER(acc_i).
3: Verify the validity of the IVC proof: $\top \stackrel{?}{=}$ NARK.VERIFIER(R_{IVC}, x', π_i).
4: If the above two checks pass, then output $\top$, else output $\bot$.

Why Does it Work?

$$
\begin{aligned}
&\mathsf{IVC.Verifier}(R_{IVC}, x_n = \{s_0, s_n, \mathrm{acc}_i\}, \pi_n) = \top \implies \\
&\forall i \in [n], \forall q_j \in \pi_i = \boldsymbol{q} : \mathsf{PC_{DL}.Check}(q_j) = \top \quad \wedge \\
&F(s_{n-1}) = s_n \wedge (s_{n-1} = s_0 \vee (\mathcal{V}_1 \wedge \mathcal{V}_2)) \implies \\
&\mathsf{AS.Verifier}(\boldsymbol{q} = \pi_{n-1}, \mathrm{acc}_{n-1}, \mathrm{acc}_n) = \top \quad \wedge \\
&\mathsf{NARK.VerifierFast}(R_{IVC}, x_{n-1}, \pi_{n-1}) = \top \implies \ldots \\
&F(s_0) = s_1 \wedge (s_0 = s_0 \vee (\mathcal{V}_1 \wedge \mathcal{V}_2)) \implies \\
&F(s_0) = s_1 \implies
\end{aligned}
$$

1. $\forall i \in [2, n]$: $\mathsf{AS.Verifier}(\pi_{i-1}, \mathrm{acc}_{i-1}, \mathrm{acc}_i) = \top$, i.e, all accumulators are valid.
2. $\forall i \in [2, n]$: $\mathsf{NARK.VerifierFast}(R_{IVC}, x_{i-1}, \pi_{i-1})$, i.e, all the proofs are valid.

Efficiency Analysis

Assumptions

- NARK.Prover runtime scales linearly with d ($\mathcal{O}(d)$)
- NARK.Verifier runtime scales linearly with d ($\mathcal{O}(d)$)
- NARK.VerifierFast runtime scales sub-linearly with d ($\mathcal{O}(\lg(d))$)
- F runtime is less than $\mathcal{O}(d)$, since $|R_F| \lessapprox d$

Analysis

- IVC.Prover:
  - Step 2: The cost of running $\mathsf{AS}_{\mathsf{DL}}$.Prover, $\mathcal{O}(d)$.
  - Step 3: The cost of computing F, $\mathcal{O}(F(x))$.
  - Step 5: The cost of running NARK.Prover, $\mathcal{O}(d)$.

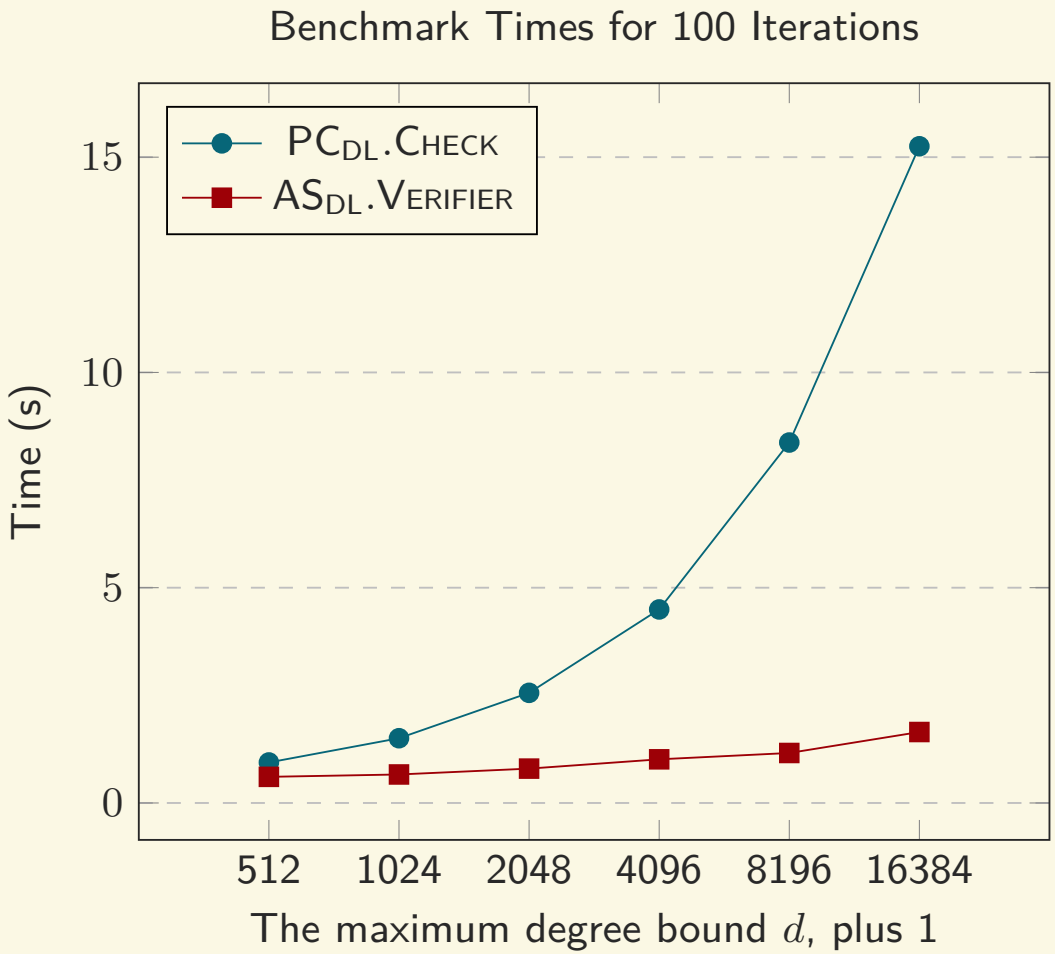
  Totalling $\mathcal{O}(d)$.
- IVC.Verifier:
  - Step 2: The cost of running $\mathsf{AS}_{\mathsf{DL}}$.Decider, $\mathcal{O}(d)$ scalar muls.
  - Step 3: The cost of running NARK.Verifier, $\mathcal{O}(d)$ scalar muls.

  Totalling $\mathcal{O}(d)$

Although the runtime of IVC.Verifier is linear, it scales with d, **not** n.

Benchmarks & Conclusion

Benchmarks


Benchmark Times for 100 Iterations
PC_DL.Check
AS_DL.Verifier
Time (s)
0
5
10
15
512
1024
2048
4096
8196
16384
The maximum degree bound d, plus 1

Conclusion

The project:

- Gained a deeper understanding of advanced cryptographic theory.
- Learned to better carry theory into practice.
- Implementing full IVC is *hard*.
- Benchmarks looks good, excited to see degree bound increase.
- Future work...