KZH-Fold: Accountable Voting from Sublinear Accumulation

George Kadianakis: EF (*Arbitrary Order)
Aranxta Zapico: EF
Hossein Hafezi: University of Cambridge
Benedikt Bunz: NYU

dd May 2026

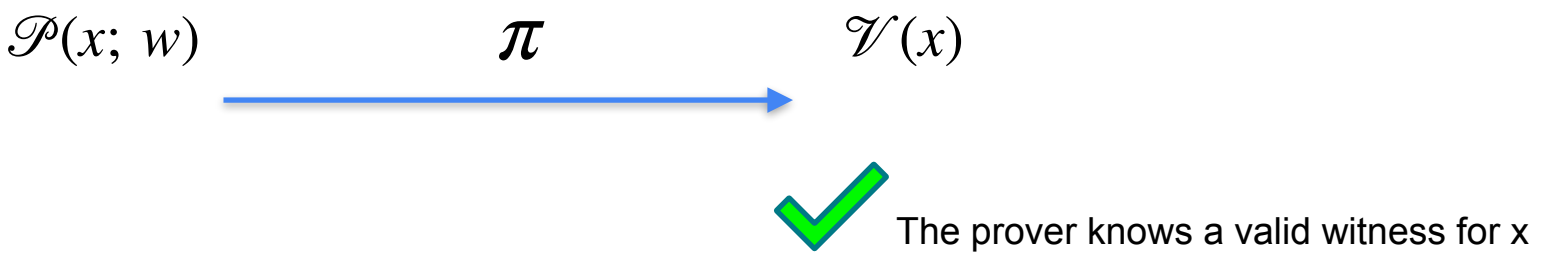
ZKPROOF
Paving the path to adoption

8th Workshop

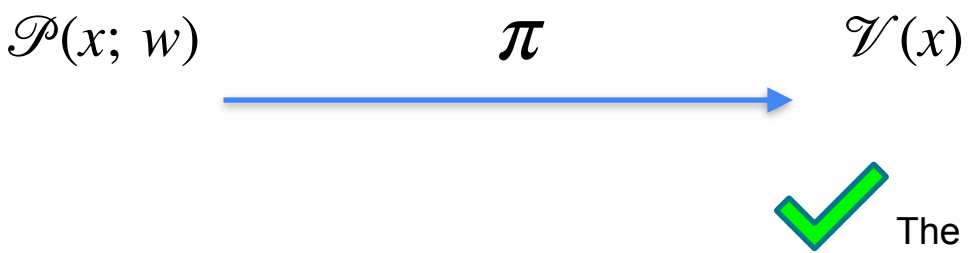
Rome, Italy
9-10 May 2026

NARK (Non-interactive Argument of Knowledge)

What is a NARK (Non-interactive Argument of Knowledge)?



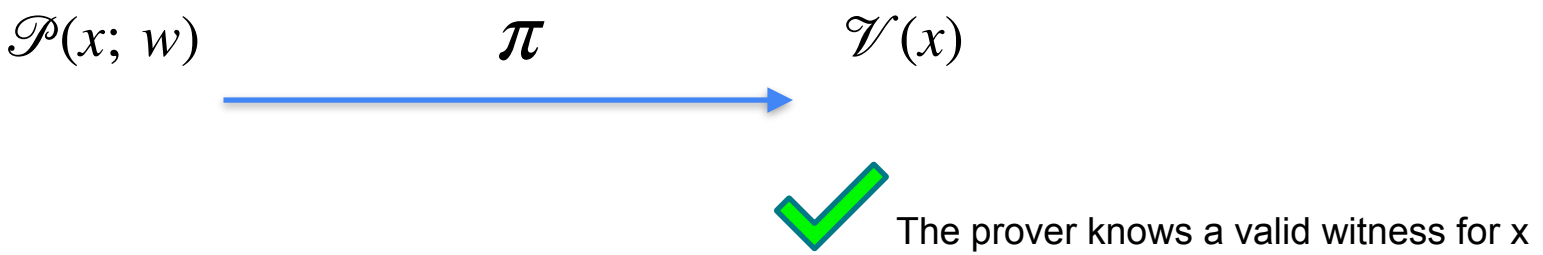
What is a NARK (Non-interactive Argument of Knowledge)?



The prover knows a valid witness for x

Completeness: If (x; w) is valid the verifier always accepts

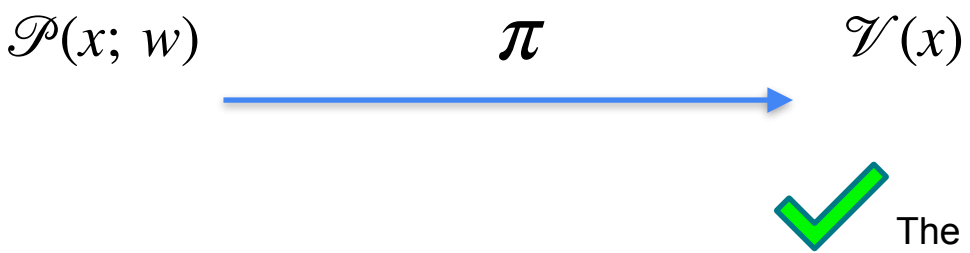
What is a NARK (Non-interactive Argument of Knowledge)?



Completeness: If (x; w) is valid the verifier always accepts

Knowledge-soundness: If the verifier accepts, then the prover must “know” a valid witness.

What is a NARK (Non-interactive Argument of Knowledge)?



The prover knows a valid witness for x

Completeness: If (x; w) is valid the verifier always accepts

Knowledge-soundness: If the verifier accepts, then the prover must “know” a valid witness.

Succinct: It’s called a SNARK if the size of the proof is sublinear in the size of w

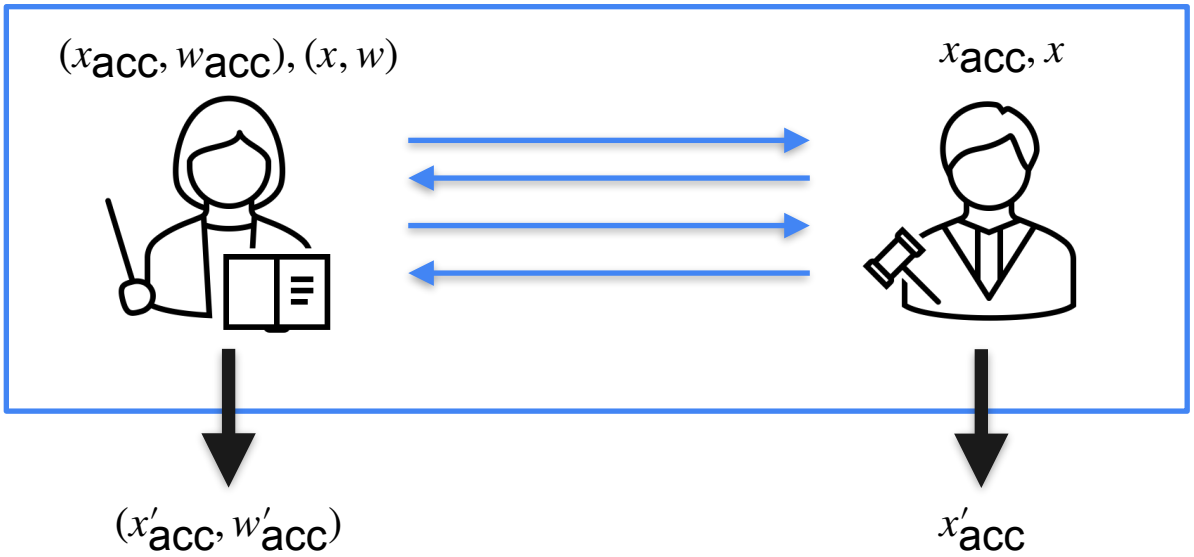
Accumulation Scheme

Accumulation Scheme [BCLMS21; KST21; KP22; BC23]

Let $\mathscr{L}$ and $\mathscr{L}_{\mathsf{acc}}$ be two NP languages. $(x_{\mathsf{acc}}, w_{\mathsf{acc}}) \in \mathscr{L}_{\mathsf{acc}}, (x, w) \in \mathscr{L}$

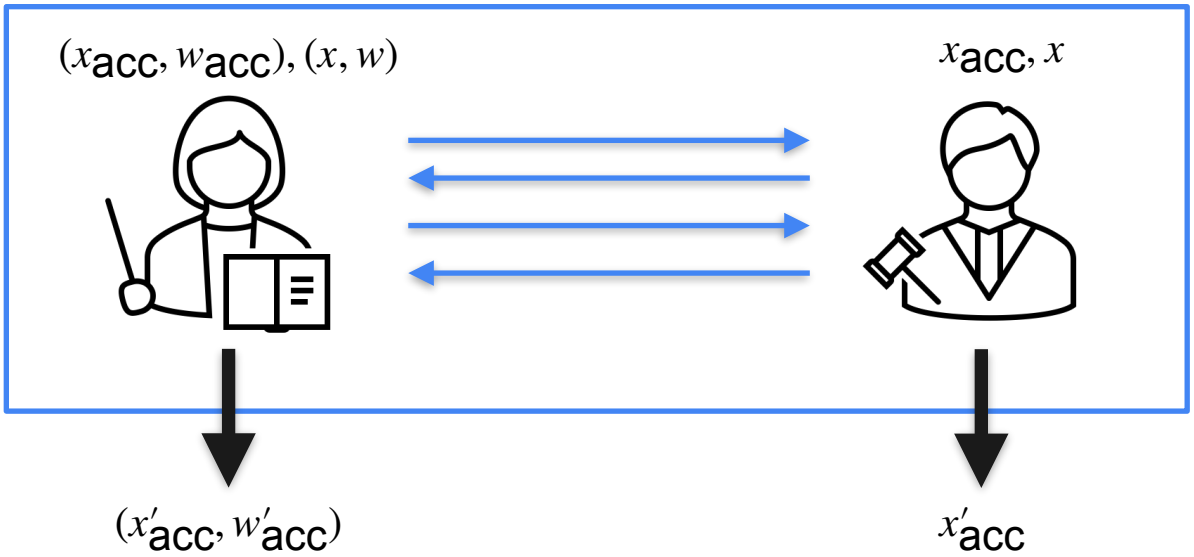
Accumulation Scheme [BCLMS21; KST21; KP22; BC23]

Let $\mathscr{L}$ and $\mathscr{L}_{\mathsf{acc}}$ be two NP languages. $(x_{\mathsf{acc}}, w_{\mathsf{acc}}) \in \mathscr{L}_{\mathsf{acc}}, (x, w) \in \mathscr{L}$



Accumulation Scheme [BCLMS21; KST21; KP22; BC23]

Let $\mathscr{L}$ and $\mathscr{L}_{\mathsf{acc}}$ be two NP languages. $(x_{\mathsf{acc}}, w_{\mathsf{acc}}) \in \mathscr{L}_{\mathsf{acc}}, (x, w) \in \mathscr{L}$

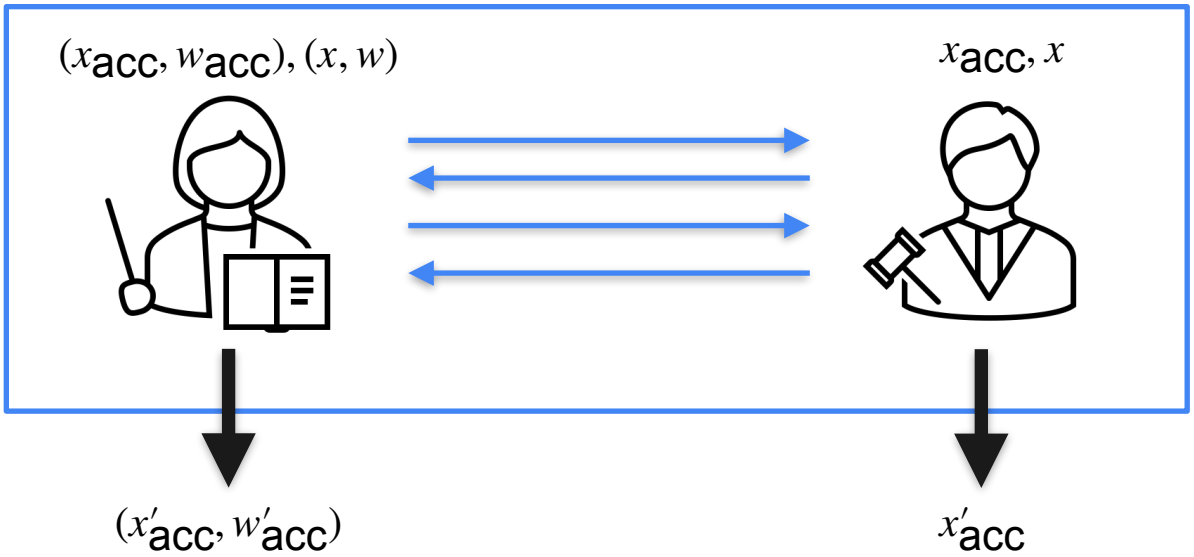


$(x_{\mathsf{acc}}, w_{\mathsf{acc}}) \in \mathscr{L}_{\mathsf{acc}}, (x, w) \in \mathscr{L}$

$\iff (x'_{\mathsf{acc}}, w'_{\mathsf{acc}}) \in \mathscr{L}_{\mathsf{acc}}$

Accumulation Scheme [BCLMS21; KST21; KP22; BC23]

Let $\mathscr{L}$ and $\mathscr{L}_{\mathsf{acc}}$ be two NP languages. $(x_{\mathsf{acc}}, w_{\mathsf{acc}}) \in \mathscr{L}_{\mathsf{acc}}, (x, w) \in \mathscr{L}$



$(x_{\mathsf{acc}}, w_{\mathsf{acc}}) \in \mathscr{L}_{\mathsf{acc}}, (x, w) \in \mathscr{L}$

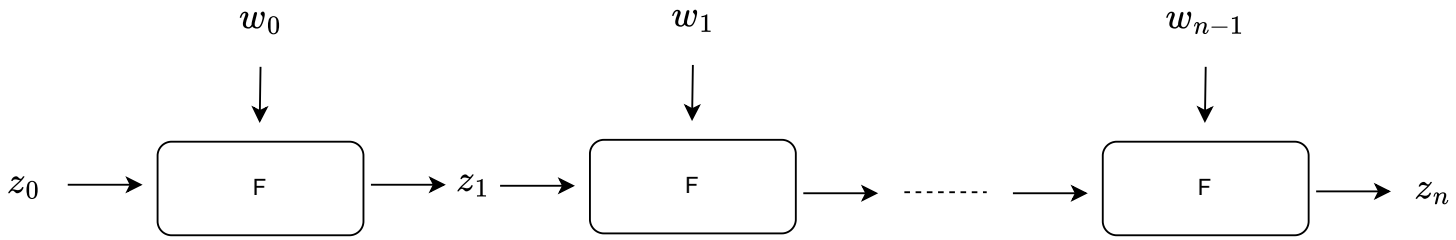
$\iff (x'_{\mathsf{acc}}, w'_{\mathsf{acc}}) \in \mathscr{L}_{\mathsf{acc}}$

It reduces check two NP instances into one

Incremental Verifiable Computation (IVC)

What is Incremental Verifiable Computation (IVC)?

IVC is used to prove correctness of iterative computations such as the following:



IVC has n steps and every step proves the evaluation of F at step i

Proof-Carrying Data (PCD)

Generalization of IVC from straight-line to DAGs (directed acyclic graphs).

PCD enables distributed and parallel computation.

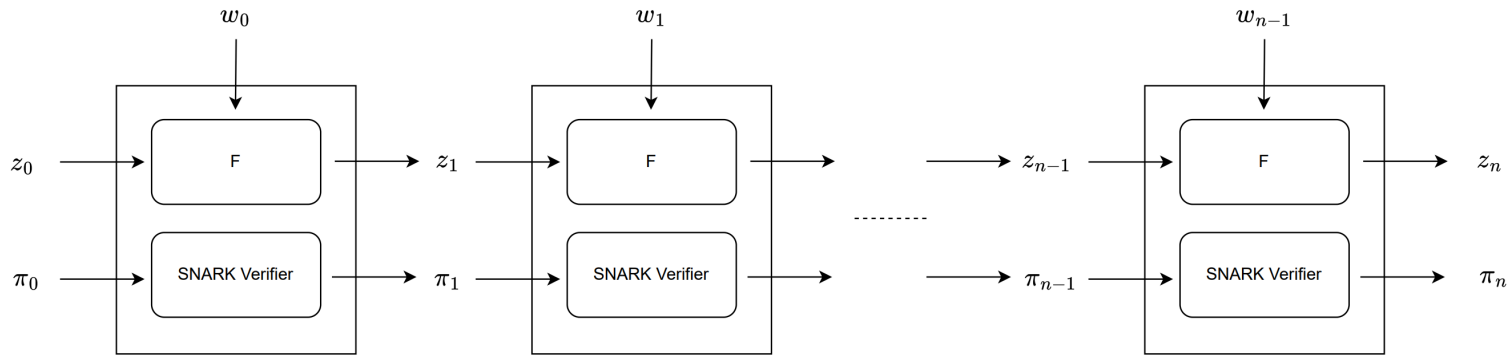
Used in distributed proving, distributed signature aggregation, secure aggregation, etc

How to build IVC/PCD? (Traditional approach :D)

Build an augmented circuit:

1) Subcircuit F

2) SNARK verifier subcircuit (recursive overhead)

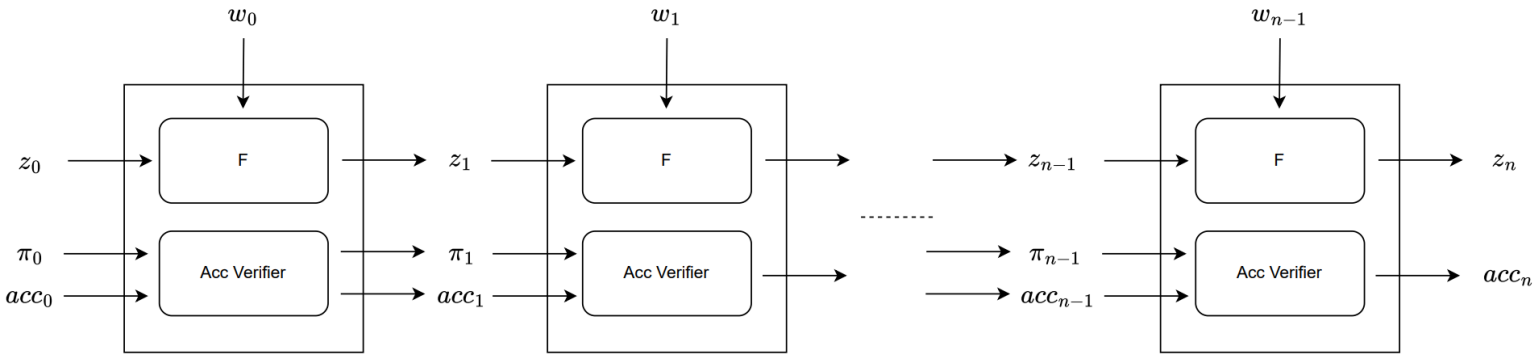


IVC/PCD Construction from Accumulation Schemes [Halo]

The SNARK verifier in the circuit is quite expensive!

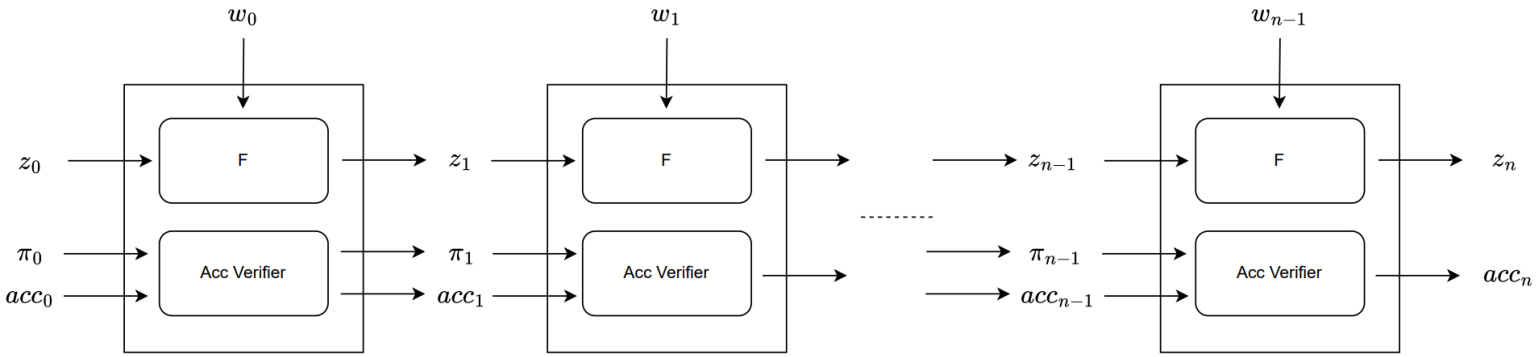
Observation: Defer checking the step proves to the last step via accumulation schemes.

How to build IVC/PCD from accumulation schemes?



$\pi_1, \pi_2, \ldots, \pi_n \implies$ Accumulate them into acc_n and check acc_n.

How to build IVC/PCD from accumulation schemes?



$\pi_1, \pi_2, \ldots, \pi_n \implies$ Accumulate them into acc_n and check acc_n.

Acc verifier is more efficient than the SNARK verifier in the circuit

BCLMS Compiler

BCLMS Compiler: NARK + NARK Verifier Accumulation $\implies$ IVC

- IVC proof = $|\text{acc}|$
- IVC verifier = $\text{Decider}_{\text{acc}}$
- IVC prover = $\text{Prover}_{\text{NARK}}$ (for augmented circuit) + $\text{Prover}_{\text{acc}}$

Long Line of Work On Accumulation Schemes

Proof-Carrying Data from Accumulation Schemes

Benedikt Bünz
benedikt@cs.stanford.edu
Stanford University

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Nicholas Spooner
nick.spooner@berkeley.edu
UC Berkeley

Proof-Carrying Data without Succinct Arguments

Benedikt Bünz
benedikt@cs.stanford.edu
Stanford University

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

William Lin
will.lin@berkeley.edu
UC Berkeley

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Nicholas Spooner
nspooner@bu.edu
Boston University

HyperNova: Recursive arguments for customizable constraint systems

Abhiram Kothapalli
Carnegie Mellon University

Srinath Setty
Microsoft Research

LatticeFold+: Faster, Simpler, Shorter Lattice-Based Folding for Succinct Proof Systems

Dan Boneh and Binyi Chen

Stanford University

LatticeFold: A Lattice-based Folding Scheme and its Applications to Succinct Proof Systems

Dan Boneh and Binyi Chen

Stanford University

Nova: Recursive Zero-Knowledge Arguments from Folding Schemes

Abhiram Kothapalli[†] Srinath Setty[⋆] Ioanna Tzialla[‡]

[†]Carnegie Mellon University [⋆]Microsoft Research [‡]New York University

PROTOSTAR: Generic Efficient Accumulation/Folding for Special-sound Protocols

Benedikt Bünz
Stanford University,
Espresso Systems

Binyi Chen
Espresso Systems

NeutronNova: Folding everything that reduces to zero-check

Abhiram Kothapalli
University of California, Berkeley

Srinath Setty
Microsoft Research

Accumulation without Homomorphism

Benedikt Bünz
bb@nyu.edu
New York University

Pratyush Mishra
prat@upenn.edu
University of Pennsylvania

Wilson Nguyen
wdnguyen@stanford.edu
Stanford University

William Wang
ww@priv.pub
New York University

Long Line of Work On Accumulation Schemes

Proof-Carrying Data from Accumulation Schemes

Benedikt Bünz
benedikt@cs.stanford.edu
Stanford University

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Nicholas Spooner
nick.spooner@berkeley.edu
UC Berkeley

Proof-Carrying Data without Succinct Arguments

Benedikt Bünz
benedikt@cs.stanford.edu
Stanford University

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

William Lin
will.lin@berkeley.edu
UC Berkeley

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Nicholas Spooner
nspooner@bu.edu
Boston University

HyperNova: Recursive arguments for customizable constraint systems

Abhiram Kothapalli
Carnegie Mellon University

Srinath Setty
Microsoft Research

The focus of all these papers is to reduce the recursive overhead, and build "prover" efficient IVC/PCD

LatticeFold+: Faster, Simpler, Shorter
for Succinct Proof Sys

Dan Boneh and Binyi Ch
Stanford University

Zero-Knowledge Arguments
Folding Schemes

Srinath Setty*
*Microsoft Research

Ioanna Tzialla‡
‡New York University

PROTOSTAR: Generic Efficient Accumulation/Folding for Special-sound Protocols

Benedikt Bünz
Stanford University,
Espresso Systems

Binyi Chen
Espresso Systems

NeutronNova: Folding everything that reduces to zero-check

Abhiram Kothapalli
University of California, Berkeley

Srinath Setty
Microsoft Research

Accumulation without Homomorphism

Benedikt Bünz
bb@nyu.edu
New York University

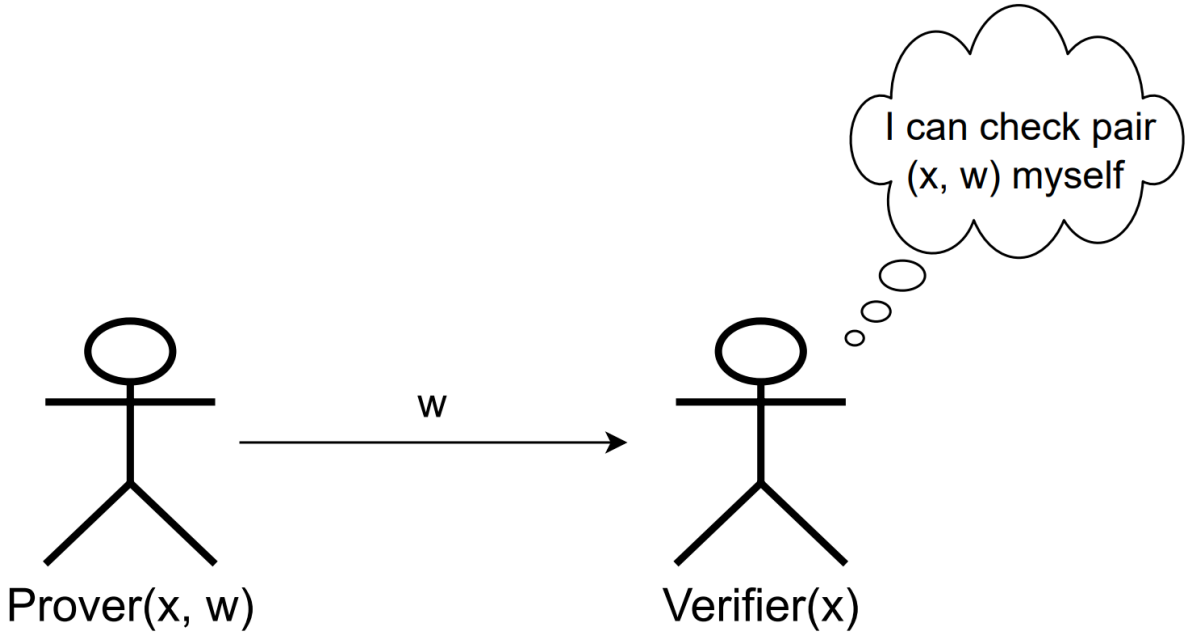
Pratyush Mishra
prat@upenn.edu
University of Pennsylvania

Wilson Nguyen
wdnguyen@stanford.edu
Stanford University

William Wang
ww@priv.pub
New York University

Nova: IVC/PCD with efficient prover

Example of Nova: What is a trivial NARK?



Nova: Accumulation Scheme for Trivial R1CS NARK

Nova accumulator for (relaxed) R1CS: (n =instance+witness size of R1CS)

- Accumulator size: $O(n)$
- Accumulator prover time: $O(n)$
- Accumulator decider time: $O(n)$
- Verifier circuit: $O(1)$, 2 or 3 scalar multiplications

Nova: Accumulation Scheme for Trivial R1CS NARK

Nova accumulator for (relaxed) R1CS: (n =instance+witness size of R1CS)

- Accumulator size: $O(n)$
- Accumulator prover time: $O(n)$
- Accumulator decider time: $O(n)$
- Verifier circuit: $O(1)$, 2 or 3 scalar multiplications

Nova IVC via BCLMS compiler:

- IVC proof size: $O(n)$
- IVC prover time: $O(n)$
- IVC decider time: $O(n)$

Nova: Accumulation Scheme for Trivial R1CS NARK

Nova accumulator for (relaxed) R1CS: (n =instance+witness size of R1CS)

- Accumulator size: $O(n)$
- Accumulator prover time: $O(n)$
- Accumulator decider time: $O(n)$
- Verifier circuit: $O(1)$, 2 or 3 scalar multiplications

Nova IVC via BCLMS compiler:

- IVC proof size: $O(n)$
- IVC prover time: $O(n)$
- IVC decider time: $O(n)$

For a circuit of 1 million constraints ==>
1) the accumulator size = IVC proof size = 80 MB
2) Decider time = IVC verifier time = 4s

Nova / MicroNova solution ==> run an additional zkSNARK at the end (24x prover overhead, no longer incremental)

IVC/PCD With Linear-Sized Proof and Verifier

- Accumulator size: $O(n)$
- Accumulator prover time: $O(n)$
- Accumulator decider time: $O(n)$

The asymptotic are the same across all the state-of-the-art accumulation schemes

IVC/PCD With Linear-Sized Proof and Verifier

- Accumulator size: $O(n)$
- Accumulator prover time: $O(n)$
- Accumulator decider time: $O(n)$

The asymptotic are the same across all the state-of-the-art accumulation schemes

- IVC proof size: $O(n)$
- IVC prover time: $O(n)$
- IVC decider time: $O(n)$

Similar asymptotic for the resulting IVC/PCD scheme

IVC/PCD With Linear-Sized Proof and Verifier

- Accumulator size: $O(n)$
- Accumulator prover time: $O(n)$
- Accumulator decider time: $O(n)$

The asymptotic are the same across all the state-of-the-art accumulation schemes

- IVC proof size: $O(n)$
- IVC prover time: $O(n)$
- IVC decider time: $O(n)$

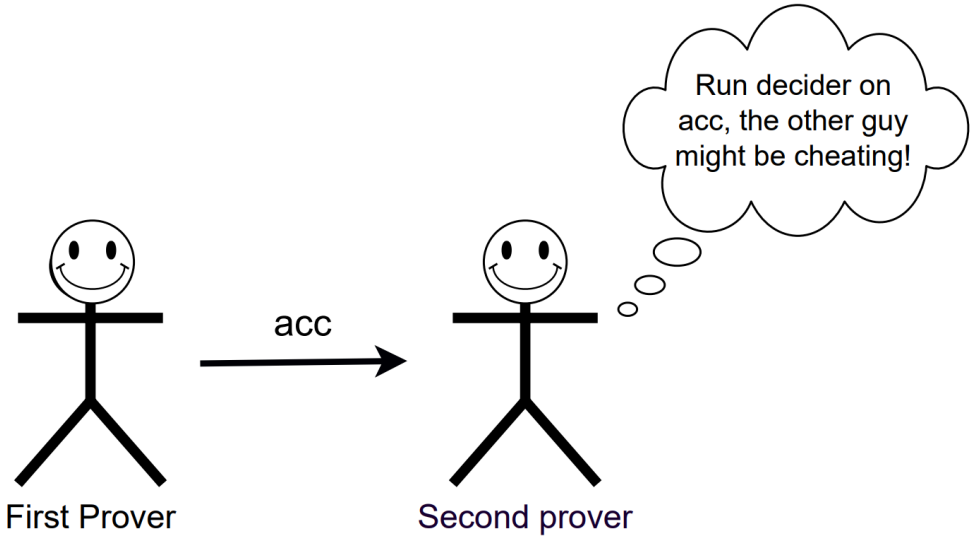
Similar asymptotic for the resulting IVC/PCD scheme

This holds for Nova, Hypernova, Protostar, NeutronNova, KiloNova, ProtoGalaxy, Lattice-Fold, Lattice-Fold+, hash-based such as Arc etc

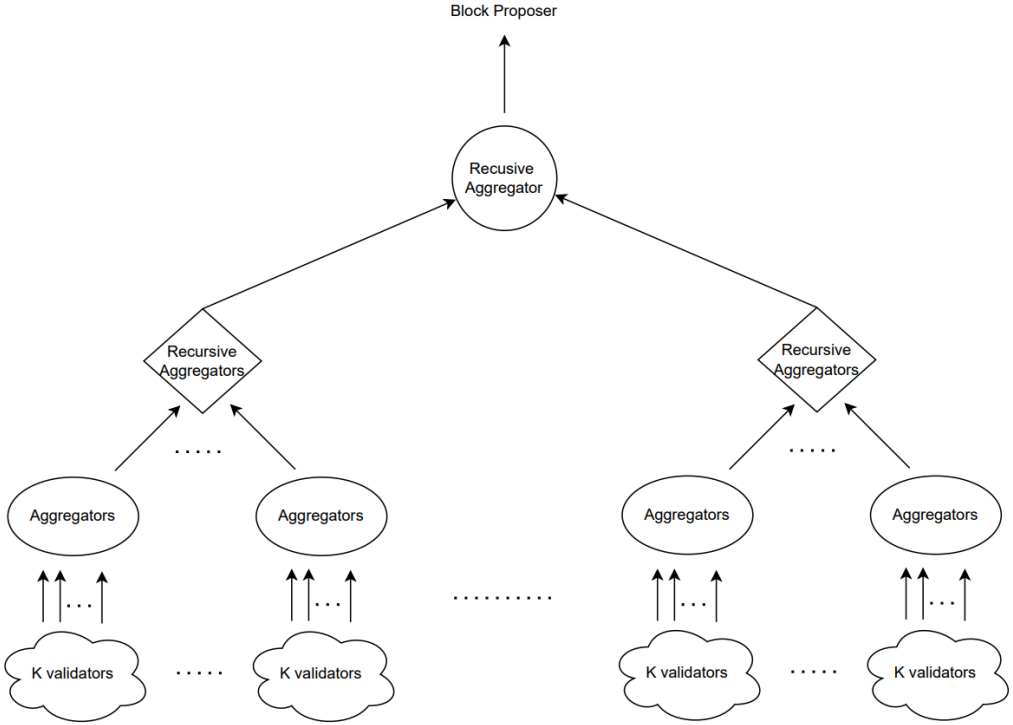
Main Motivation of KZH-Fold: IVC/PCD with Sublinear Proofs

Distributed Proving with IVC / PCD

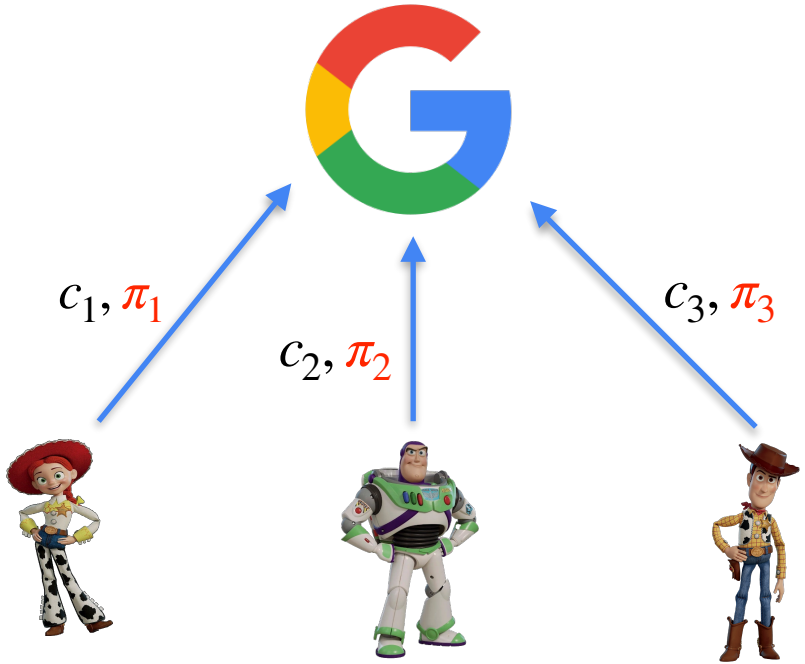
Different prover do the different proving steps.



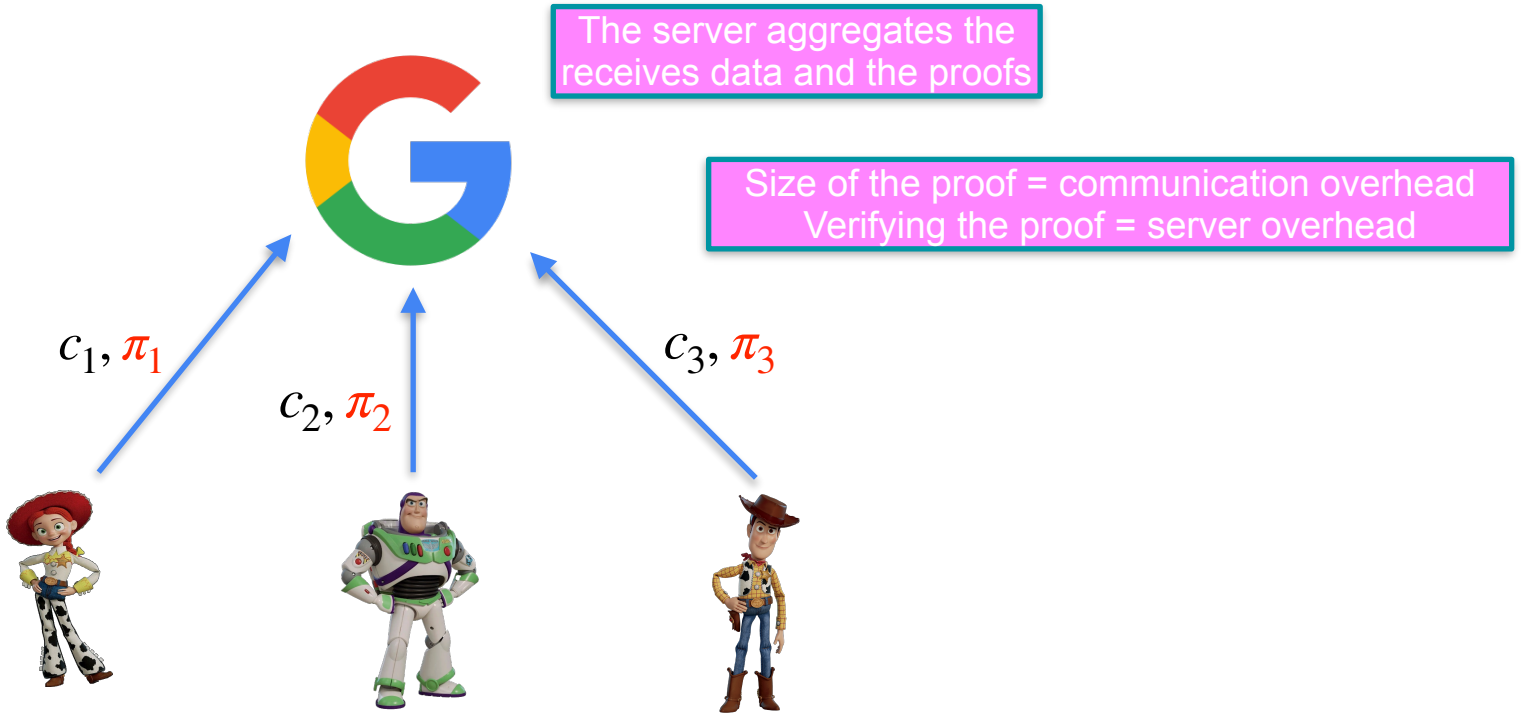
Applications of Distributed Proving (Signature Aggregation)



Application of Distributed Proving (Secure Data Aggregation)



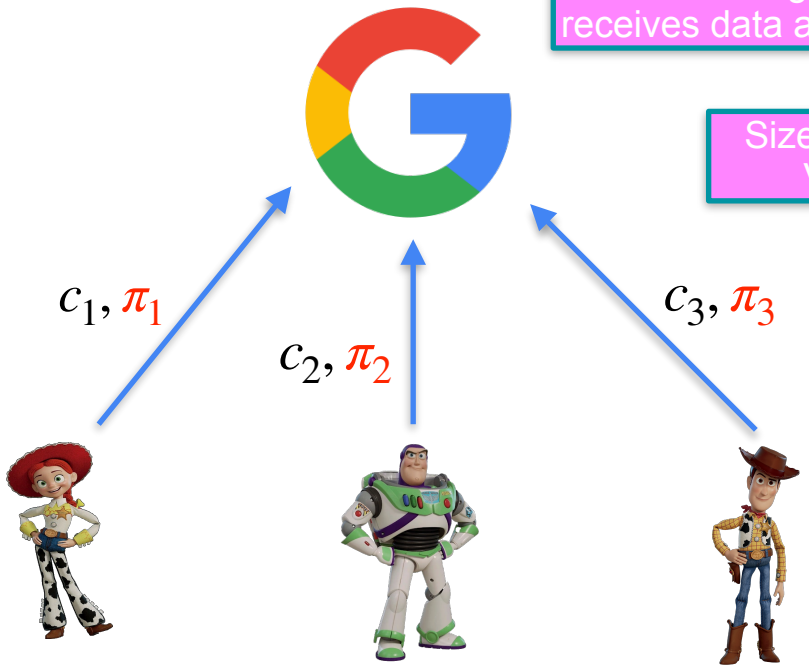
Application of Distributed Proving (Secure Data Aggregation)



Application of Distributed Proving (Secure Data Aggregation)

The server aggregates the receives data and the proofs

Size of the proof = communication overhead
Verifying the proof = server overhead



WillowFold: Secure Aggregation with a Lightweight Committee

Hossein Hafezi
NYU & KU Leuven
h.hafezi@nyu.edu

Kasra Abbaszadeh
University of Maryland
kasraz@umd.edu

Adrià Gascón
Google
adriag@google.com

Phillipp Schoppmann
Google
schoppmann@google.com

Mariana Raykova
Google
marianar@google.com

Benedikt Bünz
NYU
bb@nyu.edu

Sublinear Accumulation: Key to IVC/PCD with Sublinear Proofs

Sublinear Accumulation Scheme

Let “decider” be the NP verification function of $\mathscr{L}_{\text{acc}}$

We call an accumulation scheme sublinear if:

1) $|\mathscr{L}_{acc}| \in o(|\mathscr{L}|)$

2) $\text{decider}_{\mathscr{L}_{\text{acc}}} \in o(\text{verifier}_{\mathscr{L}})$

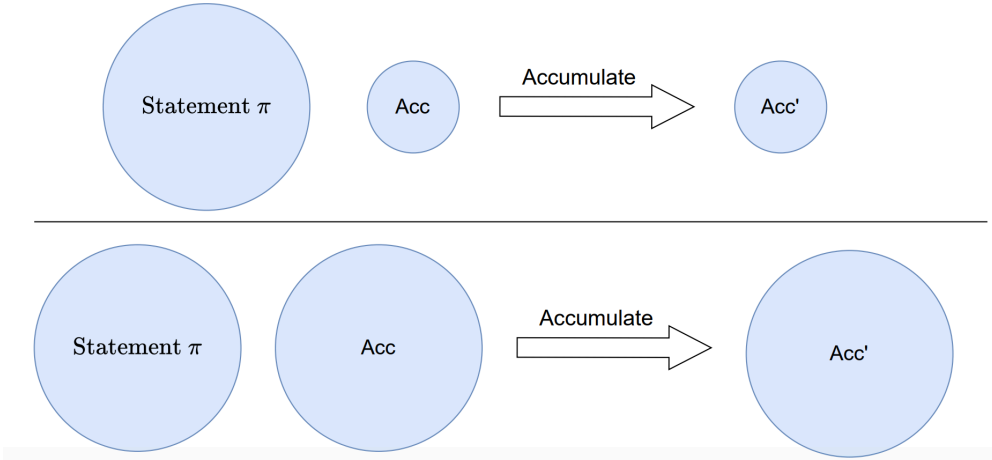
Sublinear Accumulation Scheme

Let “decider” be the NP verification function of $\mathscr{L}_{\text{acc}}$

We call an accumulation scheme sublinear if:

1) $|\mathscr{L}_{acc}| \in o(|\mathscr{L}|)$

2) $\text{decider}_{\mathscr{L}_{\text{acc}}} \in o(\text{verifier}_{\mathscr{L}})$



BCLMS Compiler

BCLMS Compiler: NARK + NARK Verifier Accumulation $\implies$ IVC

- IVC proof $= |\text{acc}|$
- IVC verifier $= \text{Decider}_{\text{acc}}$
- IVC prover $= \text{Prover}_{\text{NARK}}$ (for augmented circuit) $+ \text{Prover}_{\text{acc}}$

BCLMS Compiler

BCLMS Compiler: NARK + NARK Verifier Accumulation $\Longrightarrow$ IVC

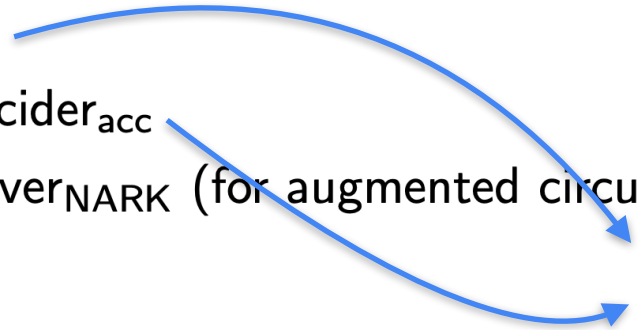
- IVC proof = $|\text{acc}|$
- IVC verifier = $\text{Decider}_{\text{acc}}$
- IVC prover = $\text{Prover}_{\text{NARK}}$ (for augmented circuit) + $\text{Prover}_{\text{acc}}$

Sublinear accumulator => sublinear IVC

BCLMS Compiler

BCLMS Compiler: NARK + NARK Verifier Accumulation $\Longrightarrow$ IVC

- IVC proof = $|\text{acc}|$
- IVC verifier = $\text{Decider}_{\text{acc}}$
- IVC prover = $\text{Prover}_{\text{NARK}}$ (for augmented circuit) + $\text{Prover}_{\text{acc}}$



If NARK is the original function F ==> these are sublinear in the size of the original function F

Sublinear accumulator => sublinear IVC

Contributions

1) Build KZH-Fold, i.e., a sublinear accumulation scheme

1) KZH-Fold is based on KZH, a PCS with sublinear opening

---

3) New approach to designing non-uniform IVC/PCD (first efficient N-PCD)

Our Goal

- IVC with sublinear proof/decider $\iff$ Sublinear accumulation.
- PCS with sublinear predicate $\implies$ Sublinear accumulation.

Starting Point: Hyrax

Reminder: Hyrax is a polynomial commitment scheme with $\sqrt{n}$ opening and commitment size.

Overview of Hyrax

Given a Pedersen setup $(g_1, g_2, \ldots, g_m)$, commit to rows of the coefficient matrix:

$$\begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ t_{21} & t_{22} & \cdots & t_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ t_{k1} & t_{k2} & \cdots & t_{km} \end{bmatrix} \begin{matrix} \Longrightarrow c_1 \\ \Longrightarrow c_2 \\ \vdots \\ \Longrightarrow c_k \end{matrix}$$

Commitment size is $O(\sqrt{n}) \Rightarrow$ homomorphism/acc verifier is $O(\sqrt{n})$

Starting Point: Hyrax

Reminder: Hyrax is a polynomial commitment scheme with $\sqrt{n}$ opening and commitment size.

Sublinear opening ==> sublinear accumulation scheme

Square-root size commitment ==> High recusive overhead (square root group operations in the circuit)

Starting Point: Hyrax

Reminder: Hyrax is a polynomial commitment scheme with $\sqrt{n}$ opening and commitment size.

Sublinear opening ==> sublinear accumulation scheme

Square-root size commitment ==> High recusive overhead (square root group operations in the circuit)

Our goal: make the commitment one group element

How to Build KZH? KZH = KZG + Hyrax

KZH2: Overview

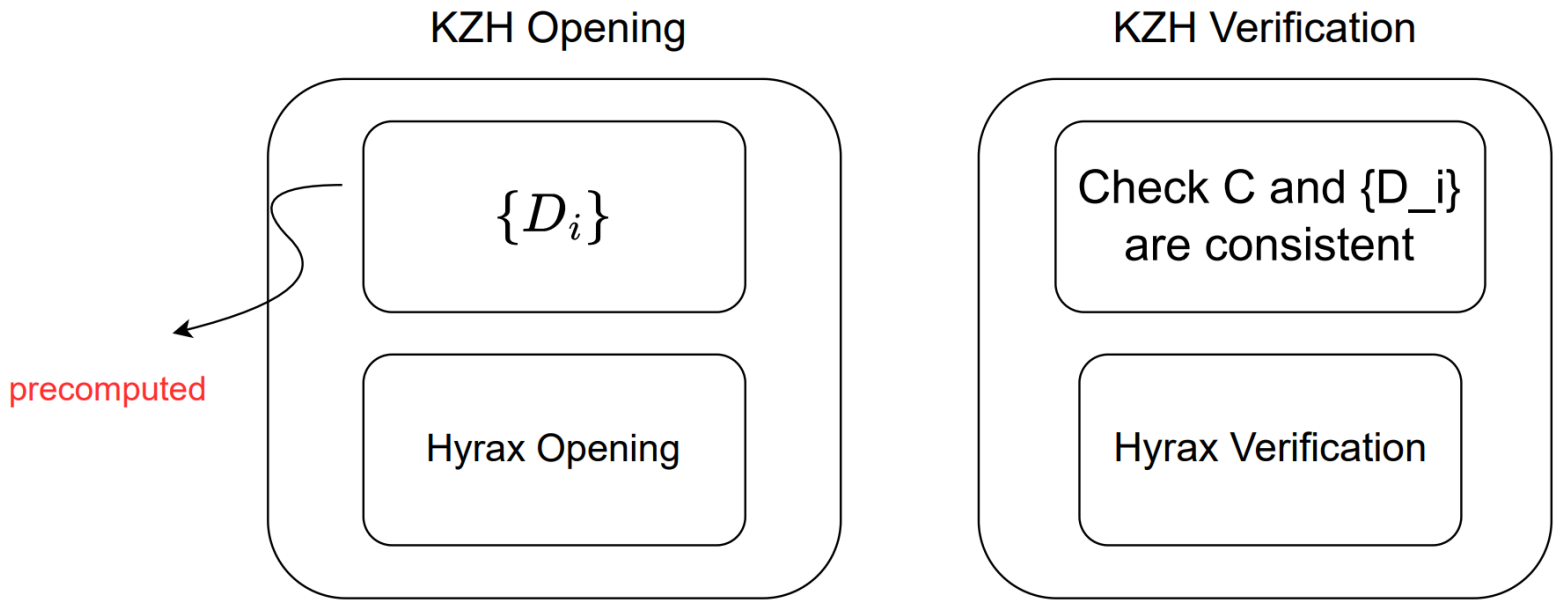
Commitment phase of KZH: (with SRS)

1. C: commit to the whole matrix in $1\mathbb{G}$ element, using a universal-SRS.
2. $\{D_i\}$: Commit to each row using $g_1, \ldots, g_m$.

$$\underbrace{\begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ t_{21} & t_{22} & \cdots & t_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ t_{k1} & t_{k2} & \cdots & t_{km} \end{bmatrix}}_{C} \begin{array}{l} \Longrightarrow D_1 \\ \Longrightarrow D_2 \\ \vdots \\ \Longrightarrow D_k \end{array}$$

Prove via pairing C and $\{D_i\}$ correspond to the same matrix.

KZH2: Overview



Efficiency of KZH2

For a multilinear polynomial $f(\vec{X})$ with on boolean hypercube of size ℓ:

- commitment time $=$ $O(\ell)$ group operations.
- Proof size $= \sqrt{\ell}\ \mathbb{G}_1 + \sqrt{\ell}\ \mathbb{F}$
- Opening time $= \ell\ \mathbb{F}$ [1]
- Verifier time $= \sqrt{\ell}$ pairing $+$ MSM$(2\sqrt{\ell}) + \sqrt{\ell}\ \mathbb{F}$.

Efficiency of KZH2

For a multilinear polynomial $f(\vec{X})$ with on boolean hypercube of size ℓ:

- commitment time $=$ $O(\ell)$ group operations.
- Proof size $= \sqrt{\ell}\ \mathbb{G}_1 + \sqrt{\ell}\ \mathbb{F}$
- Opening time $= \ell\ \mathbb{F}$ [1]
- Verifier time $= \sqrt{\ell}$ pairing $+$ MSM$(2\sqrt{\ell}) + \sqrt{\ell}\ \mathbb{F}$.

The opening cost is optimal: just polynomial evaluation and no group operations

KZH-k: An overview of efficiency

We extend KZH matrix construction to tensors of higher degree $\Longrightarrow$
Lower verifier cost at the cost of higher opening.

- Commitment cost: $O(k \cdot \ell)$
- Opening cost: $O(\ell^{1/2})$ via pre-processing
- Verifier cost: $O(k \cdot \ell^{1/k})$

KZH-log(ℓ)

We extend KZH matrix construction to tensors of higher degree $\Longrightarrow$
Lower verifier cost at the cost of higher opening.

- Commitment cost: $O(\log \ell \cdot \ell)$
- Opening cost: $O(\ell^{1/2})$ via pre-processing
- Verifier cost: $O(\log \ell)$

Comparison of KZH-k vs Prior Work

Scheme	Supports	Opening time	Proof size	Verifier time
KZH	Multilinear	$n^{\frac{1}{2}}\,\mathbb{G}_1$	$2n^{\frac{1}{2}}\mathbb{G}_1$	$n^{\frac{1}{2}}\ \mathrm{P} + 2 \cdot \mathrm{MSM}(n^{\frac{1}{2}})$
KZH-$\log(n)$	Multilinear	$n^{\frac{1}{2}}\,\mathbb{G}_1$	$2\log(n)\,\mathbb{G}_1$	$2\log(n)\ \mathrm{P}$
[PST13]	Multivariate	$\mathrm{MSM}(n)$	$\log(n)\,\mathbb{G}_1$	$\log(n)\ \mathrm{P}$
[KZG10]	Univariate	$\mathrm{MSM}(n)$	$1\mathbb{G}_1$	$2\ \mathrm{P}$
[Lee21]	Multilinear	$n^{\frac{1}{2}}\ P$	$3\log(n)\,\mathbb{G}_T$	$\log(n)\,\mathbb{G}_T,\ \mathrm{P}$

Other properties that make KZH unique.

KZH offers free opening for points in the boolean hypercube $\implies$ an efficient VC instantiation

KZH offers cheap zk $\implies$ $O(k \cdot N^{\frac{1}{k}})$

Free opening at Boolean points

HydraProofs: Optimally Computing All Proofs in a Vector Commitment
(with applications to efficient zkSNARKs over data from multiple users)

Christodoulos Pappas
Hong Kong University of Science and Technology
cpappas@connect.ust.hk

Dimitrios Papadopoulos
Hong Kong University of Science and Technology
dipapado@cse.ust.hk

Charalampos Papamanthou
Yale University
charalampos.papamanthou@yale.edu

IEEE S&P 2025

Free opening at Boolean points

HydraProofs: Optimally Computing All Proofs in a Vector Commitment
(with applications to efficient zkSNARKs over data from multiple users)

Christodoulos Pappas
Hong Kong University of Science and Technology
cpappas@connect.ust.hk

Dimitrios Papadopoulos
Hong Kong University of Science and Technology
dipapado@cse.ust.hk

Charalampos Papamanthou
Yale University
charalampos.papamanthou@yale.edu

IEEE S&P 2025

Square root proof size

Does not support zero-knowledge

Free opening at Boolean points

IronDict:
Transparent Dictionaries from Polynomial Commitments

Hossein Hafezi[†1], Alireza Shirzad[†2], Benedikt Bünz[3], and Joseph Bonneau[4]

[1,3,4]New York University
[2]University of Pennsylvania

USENIX Security 2026

Aegon: Scalable and Self-Auditable Key Transparency

Hossein Hafezi
University of Cambridge

Alireza Shirzad
University of Pennsylvania

Benedikt Bünz
NYU

Kevin Lewi
Meta

Dillon George
Meta

Joseph Bonneau
NYU

KZH-Fold: Sublinear Accumulation Scheme for KZH

Goal

- Just to review: building IVC/PCD with sublinear proof requires a sublinear accumulation scheme
- We have built a PCS with sublinear proof, next we build an polynomial accumulation (PA) scheme for it.
- We'll later see how to compiler PIOP+PA into IVC/PCD.

KZH-Fold: Accumulation Scheme for KZH Verifier

The verifier function for KZH= degree 2 scalar multiplications + degree 1 pairing check $\overset{\text{Protostar compiler}}{\Longrightarrow}$

- Accumulator size. $O(\ell^{\frac{1}{2}})$
- Decider complexity. $O(\ell^{\frac{1}{2}})$
- Prover complexity. $O(\ell)$
- Verifier complexity. $3-4$ $\mathbb{G}_1$ scalar multiplications + $O(1)$ $\mathbb{F}$

KZH-k fold:

- $O(k \cdot \ell^{1/k})$ decider and accumulator size
- $k+1$ scalar multiplication in recursive circuit.

Comparing KZH-Fold with Prior Work

Scheme	Recursive Overhead	Decider	—acc—
Nova	2 group ops	$\mathrm{MSM}(n)$	$O(n)$
KZH2-fold	3 group ops	$n^{\frac{1}{2}}$ P	$O(n^{\frac{1}{2}})$
KZH-k fold	$k+1$ group ops	$k \cdot n^{\frac{1}{k}}$ P	$O(k \cdot n^{\frac{1}{k}})$
Halo	$O(\log n)$ group ops	$\mathrm{MSM}(n)$	$O(\log n)$

How to achieve IVC/PCD from PA?

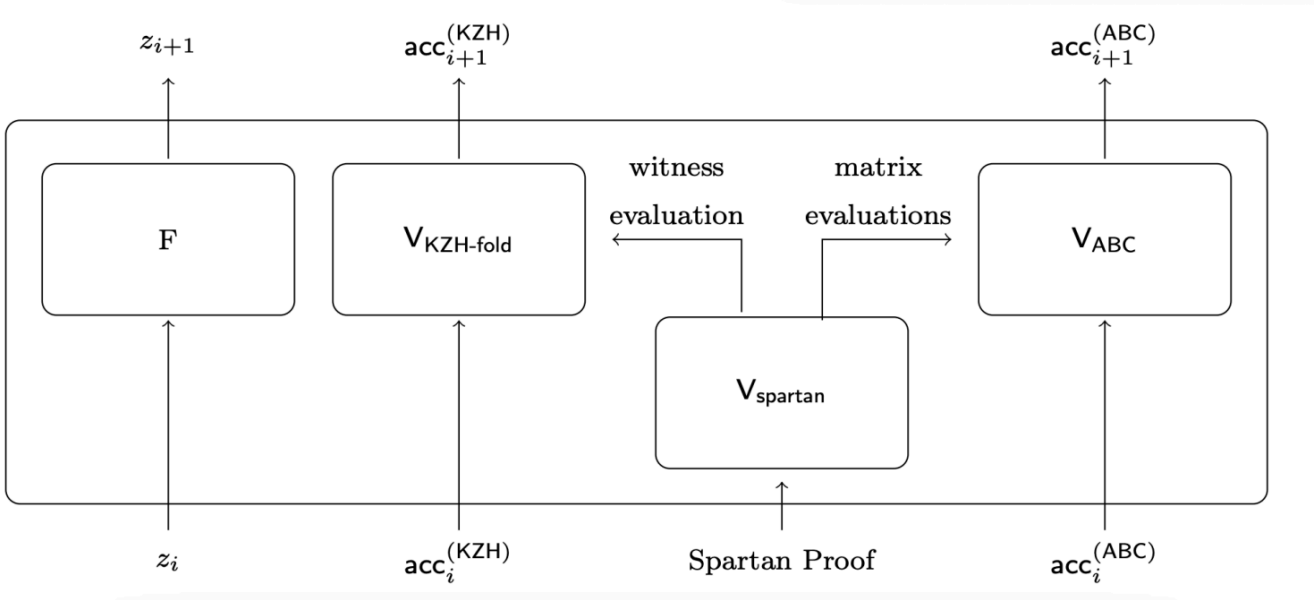
- BCLMS Compiler: NARK + NARK verifier accumulation ==> IVC/PCD
  - SNARK + SNARK verifier accumulation ==> Sublinear IVC/PCD
- NARK = PIOP + PCS
  - SNARK = $\mathscr{V}_{\mathsf{PIOP}}$ and $\mathscr{V}_{\mathsf{PCS}}$ are both succinct.
- Accumulation for $\mathscr{V}_{\mathsf{PIOP}}$ and $\mathscr{V}_{\mathsf{PCS}} \equiv$ Accumulation for $\mathscr{V}_{\mathsf{SNARK}}$

How to achieve IVC/PCD from PA?

PIOP (Spartan) + PCS (KZH) = SNARK (Spartan+KZH)

Spartan verifier accumulator + KZH-Fold = Spartan+KZH verifier accumulation

IVC circuit of Spartan+KZH-Fold



Comparison of Nova vs Spartan+KZH-Fold

Scheme	**Prover**	**Verifier**	$\|\text{Acc}\|$	**# Constraints**
Spartan+KZH-fold	16.5 s	135 ms	37 KB	$2^{21} \approx 2097k$
Nova	4.8 s	5.6 s	80.8 MB	1185k

Table: Spartan+KZH-Fold vs Nova for Circuit With 2000 Poseidon Hashes

New Paradigm in Design of Non-Uniform IVC/PCD

What is Non-Uniform IVC/PCD?

Standard IVC/PCD: repetitive computation of a function F

What is we have multiple function?!

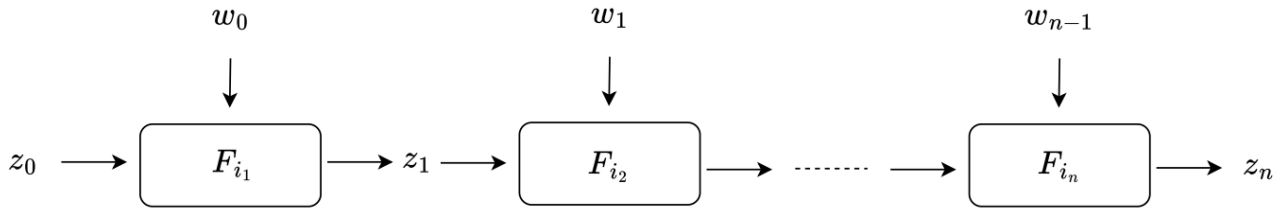
What is Non-Uniform IVC/PCD?

Standard IVC/PCD: repetitive computation of a function F

What is we have multiple function?!

Step function F can be one of multiple predefined instructions F_1, ..., F_k

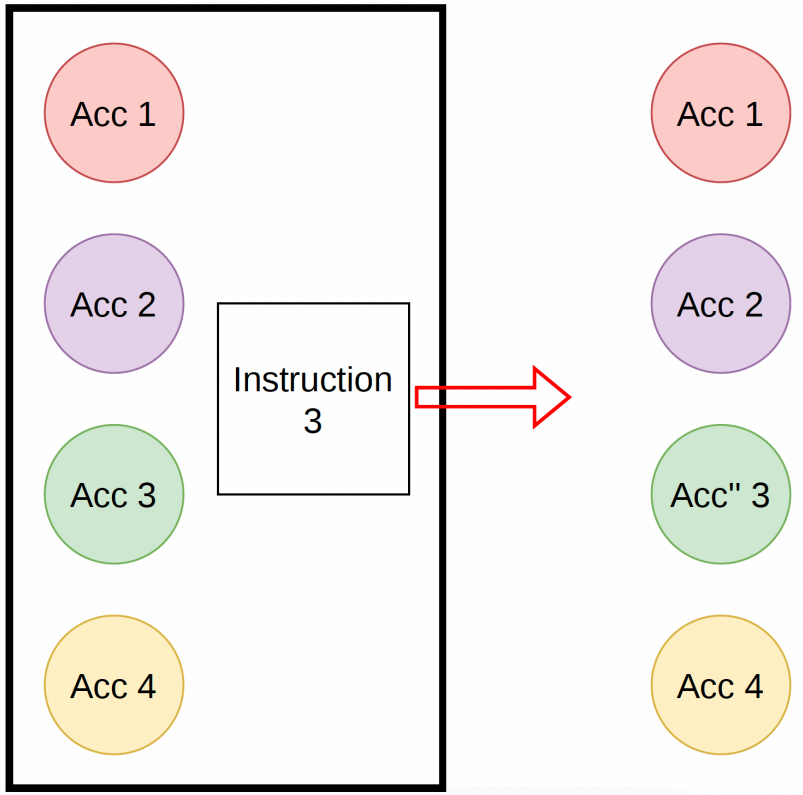
- Motivation: Verifiable CPU



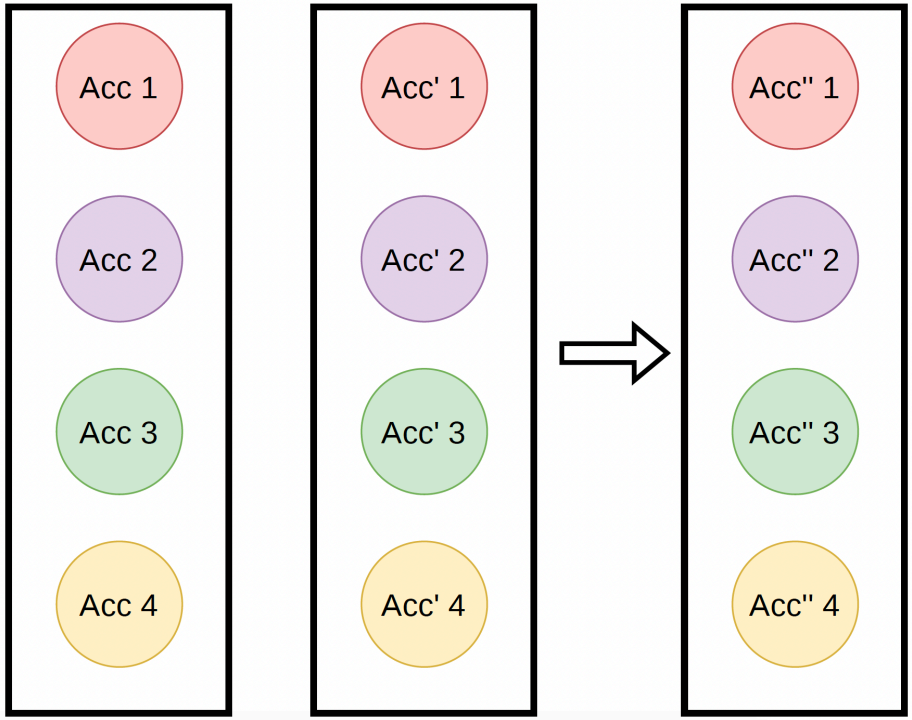
SuperNova: the State-of-the-Art for N-IVC

- Keep m different accumulators corresponding to m instructions.
- When a new instruction is received, accumulate with the corresponding and the remaining are untouched.

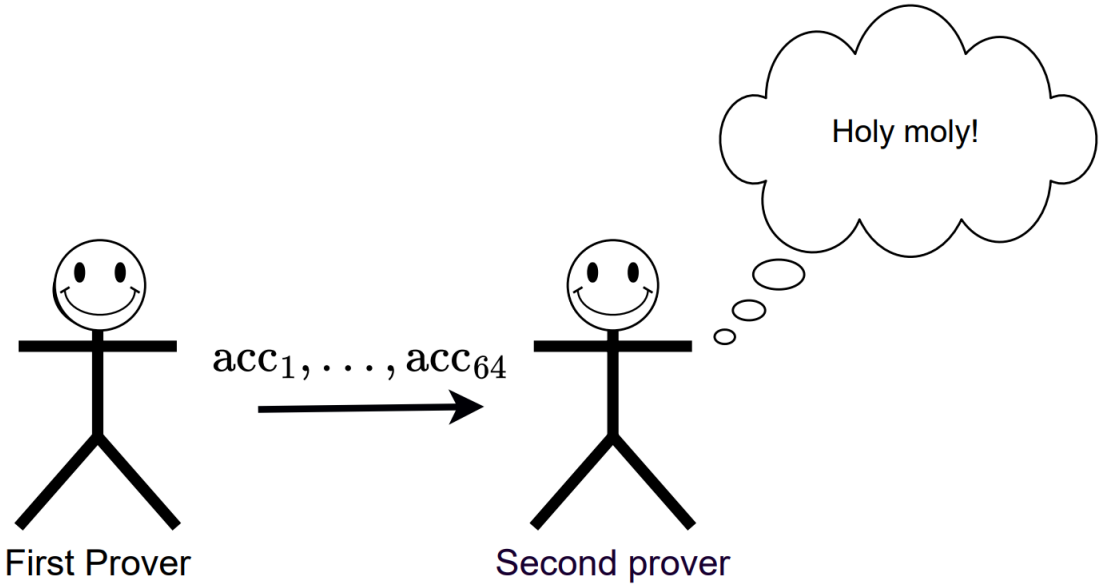
SuperNova: Previous State-of-the-Art



Why it doesn’t work for PCD



Distributed Proving of a zkVM with (Non-Uniform) IVC / PCD



- N-IVC/N-PCD with SuperNova $\Longrightarrow$ One accumulator per instruction

N-IVC/N-PCD from Polynomial Accumulation

High-level idea:

- PCS accumulation $\Longrightarrow$ more flexible than circuit accumulation
- Polynomials of different degrees can be accumulated.

Comparison to Supernova:

- directly accumulates circuit $\Longrightarrow$ one running accumulator for each instruction.

N-IVC/N-PCD from Polynomial Accumulation

F_i is translated into number of polynomial constraints via PIOP.

These polynomial are accumulated via polynomial accumulation

N-IVC/N-PCD from Polynomial Accumulation

F_i is translated into number of polynomial constraints via PIOP.

These polynomial are accumulated via polynomial accumulation

PIOP returns polynomial and polynomials are homogenous constraints

Comparison of N-IVC/N-PCD from PA+PIOP With Prior Work

Scheme	Prover Time	Verifier Time	Witness Size
SuperNova	$O(\sum \lvert F_i \rvert)\mathbb{G}$	$O(\sum \lvert F_i \rvert)\mathbb{G}$	$O(\sum_i \lvert F_i \rvert)$
Protostar	$O(\sum \lvert F_i \rvert)\mathbb{G}$	$O(\sum \lvert F_i \rvert)\mathbb{G}$	$O(\sum_i \lvert F_i \rvert)$
KiloNova	$O(\sum \lvert F_i \rvert)\mathbb{G}$	$O(\sum \lvert F_i \rvert)\mathbb{G}$	$O(\sum_i \lvert F_i \rvert)$
Spartan+PA	$\mathcal{P}_{\text{acc}}(\max_i \lvert F_i \rvert) + \sum_i \log \lvert F_i \rvert$	$\mathcal{D}_{\text{acc}}(\max_i \lvert F_i \rvert) + O(\sum_i \lvert F_i \rvert)\mathbb{F}$	$O(\lvert acc \rvert + \sum \log_i \lvert F_i \rvert)$

PA=KZH-Fold:

- Prover time: $O(\max_i \lvert F_i \rvert) + \sum_i \log \lvert F_i \rvert$
- Verifier time: $O(\sqrt{\max_i \lvert F_i \rvert}) + O(\sum_i \lvert F_i \rvert)\mathbb{F}$
- Witness size: $\sqrt{\max_i \lvert F_i \rvert} + \sum \log_i \lvert F_i \rvert$

ZKPROOF

Paving the path to adoption

Thank you!

ZKPROOF

Paving the path to adoption

Questions?