Scaling SNARK Provers

Carla Ràfols

Central European Conference on Cryptology 2025
June 19th, Budapest

Universitat
Pompeu Fabra
Barcelona

Scaling SNARK Provers: Motivation

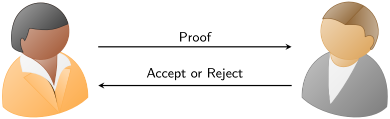
What are ZK Proofs?

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

x="Unsolved Sudoku"

4	2	6	5	7	1	3	9	8
8	5	7	2	9	3	1	4	6
1	3	9	4	6	8	2	7	5
9	7	1	3	8	5	6	2	4
5	4	3	7	2	6	8	1	9
6	8	2	1	4	9	7	5	3
7	9	4	6	3	2	5	8	1
2	6	5	8	1	4	9	3	7
3	1	8	9	5	7	4	6	2

w="Solved Sudoku"



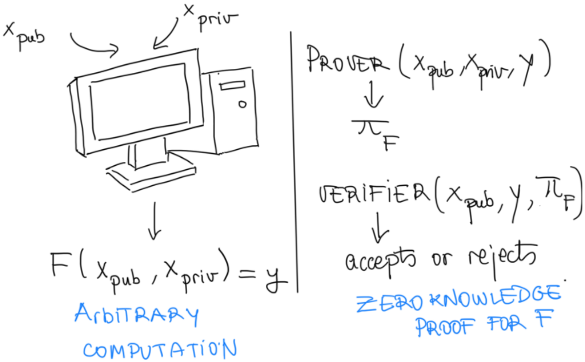
Peggy: (x, w)

Victor: x

A process in which a prover probabilistically convinces a verifier of the correctness of a mathematical proposition, and the verifier learns nothing else.

zkSNARK, (zk)**S**uccinct **N**on-Interactive **A**rgument of **K**nowledge: anything where the proof is less than $|w|$.

Zero-Knowledge Proofs & SNARKs



- ZKPs are proofs of computational integrity;
- ZKPs reveal nothing about private inputs of the computation;
- SNARKs (**Succinct Non-Interactive Arguments of Knowledge**) are **short** proofs, usually independent of computation size

$$|\pi_F| < |F|$$

Applications of SNARKs

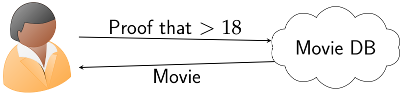
- Proving that any computation over encrypted, or compressed data is correct with very cheap verification!
  - **Privacy:** Hide but Verify.
  - **Scalability:** Compress but Verify.

Applications of SNARKs

Today

- Proving that any computation over encrypted, or compressed data is correct with very cheap verification!

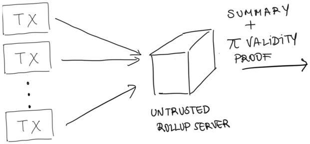
- **Privacy:** Hide but Verify.



Anonymous Credentials

Anonymous Financial Transactions

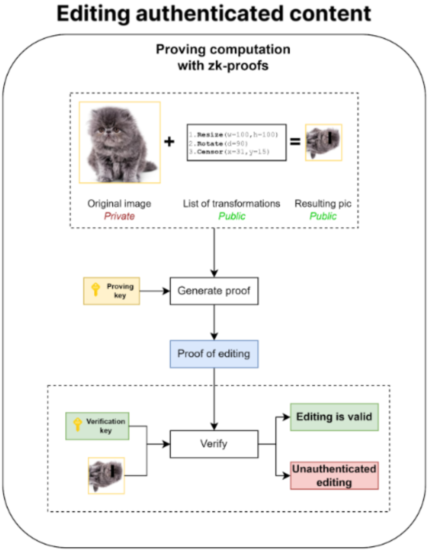
- **Scalability:** Compress but Verify.



Rollups

Applications of SNARKs

Today-ish



Credit: Roman Palkin

Using ZK Proofs to Fight Disinformation By Trisha Datta and Dan Boneh, Medium.

Applications of SNARKs

Tomorrow

zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy

Tianyi Liu
Texas A&M University and Shanghai

Xiang Xie
Shanghai Key Laboratory of

Yupeng Zhang
Texas A&M University

Zero-Knowledge Proofs of Training for Deep Neural Networks

Kasra Abbaszadeh
University of Maryland
College Park, MD, USA
kasraz@umd.edu

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

Jonathan Katz
Google
Washington, DC, USA

Dimitrios Papadopoulos
Hong Kong University of Science and Technology
Hong Kong, China

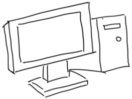
***ExpProof* : Operationalizing Explanations for Confidential Models with ZKPs**

Chhavi Yadav [*1] **Evan Monroe Laufer** [*2] **Dan Boneh** [2] **Kamalika Chaudhuri** [1]

How are many SNARKs built?

- **FRONTEND**

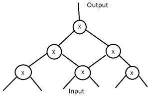
Computation



program

$\longrightarrow$

Computation Representation
e.g. Arith. Circuit, Arith. Circuit with Lookups



model with restricted operations

$\longrightarrow$ **Algebraic Relations**
R1CS, Plonkish, CCS
$e.g.\mathbf{A},\mathbf{B},\mathbf{C}$ s.t.
$\vec{z}$ satisfies circuit iff
$\mathbf{A}\vec{z} \circ \mathbf{B}\vec{z} = \mathbf{C}\vec{z}$

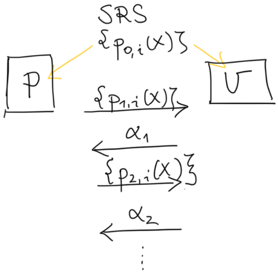
$\rightarrow$ **Polynomial Relations**

e.g.
$t(X)|A(X)B(X) - C(X)$

How are many SNARKs built?

- **BACKEND**

(Preprocessing) Polynomial IOP		**SNARK**



$\longrightarrow$

Polynomial commitment

$+$

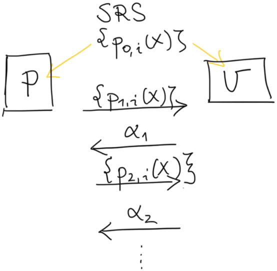
Fiat Shamir

SRS, π

How are many SNARKs built?

- **BACKEND**

(Preprocessing) Polynomial IOP



$\longrightarrow$
Polynomial commitment
$+$
Fiat Shamir

SNARK

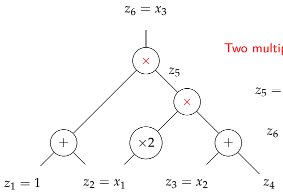
SRS, π

Compressing Step
Cryptography, Comp. Security

Key Idea:
Checking Polynomial Identities at Random Points.
Can be done succinctly with Polynomial Commitments.

Example: From Circuits to Algebraic Relations

Statement: $C(1, x_1, x_2, w) = x_3$ for some w, $\vec{x}$ public inputs.



Two multiplication gates g_1, g_2

$z_5 = (2z_2)(z_3 + z_4)$

$z_6 = (1 + z_2)z_5$

$$\mathbf{A}\vec{z} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{pmatrix} = \begin{pmatrix} 1 \\ z_2 \\ z_3 \\ z_4 \\ 2z_2 \\ 1 + z_2 \end{pmatrix}, \mathbf{B}\vec{z} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ z_3 + z_4 \\ z_5 \end{pmatrix}, \mathbf{C}\vec{z} = \mathbf{I}\vec{z} =$$

$$\mathbf{C}\vec{z} = \mathbf{I}\vec{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{pmatrix}$$

Statement true $\iff$

$$\mathbf{A}\vec{z} \circ \mathbf{B}\vec{z} = \mathbf{C}\vec{z}, \text{ and } \{z_1 = 1, z_2 = x_1, z_3 = x_2, z_6 = x_3\}$$

From Circuit to Algebraic Relations, Takeaway

Statement: $C(1, x_1, x_2, w) = x_3$ for some w, $\vec{x}$ public inputs.

1. **Public Input Relations:**
   $\{z_1 = 1, z_2 = x_1, z_3 = x_2, z_6 = x_3\}$
2. **Hadamard Product Relation**:
   $\vec{a} \circ \vec{b} = \vec{c}$
3. **Linear Relations**:
   $\vec{a} = \mathbf{A}\vec{z}$, $\vec{b} = \mathbf{B}\vec{z}$, $\vec{c} = \mathbf{C}\vec{z}$.

- Matrices are public, part of the circuit description.
- They are sparse, but of dimension of the extended witness size (inputs + multiplicative gates).

From Algebraic Relations to Polynomials

Inner Product Relations and the Univariate Sumcheck

- $\mathcal{R} = \{r_0, \ldots, r_{n-1}\} \subset \mathbb{F}_p^*$, multiplicative subgroup

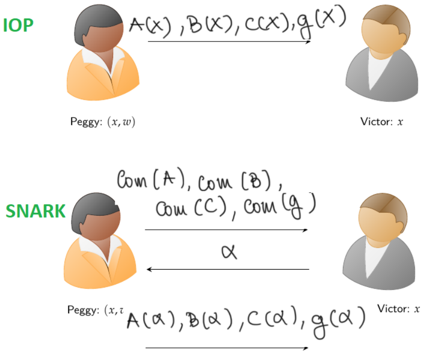
$$\lambda_i(X) = \prod_{j \neq i} \frac{(X - r_j)}{(r_i - r_j)}, \qquad t(X) = \prod_j (X - r_j).$$

Algebraic Formulation	Polynomial Formulation
Vector $\vec{y} = (y_0, \ldots, y_{n-1})$	Polynomial $Y(X) = \sum_{i=0}^{n-1} y_i \lambda_i(X)$
Public Input: $\vec{z}, \vec{x}$ agree on l positions	$Z(X) - Y(X)$ is divisible by $t_l(X)$
Hadamard Product $\vec{a} \circ \vec{b} = \vec{c}$	$A(X)B(X) - C(X)$ is divisible by $t(X)$
Inner product $z = \vec{f} \cdot \vec{g}$	[Ben-Sasson et al. 18] $\exists R(X)$, $deg\ R(X) \leq n - 2$. $t(X)$ divides $f(X)g(X) - n^{-1}z - XR(X)$

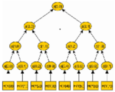
From IOPs to SNARKs

- We can immediately build a non-interactive IOP for any of these relations.

Statement : $\vec{a} \circ \vec{b} = \vec{c}$



Com(A)

1)  + Linear code

2) $A(\tau)\mathcal{P} = \sum_{i=0}^{n} a_i(\tau^i \mathcal{P})$

for public values $(\mathcal{P}, \tau\mathcal{P}, \dots, \tau^n\mathcal{P})$

$A(\alpha)B(\alpha) - C(\alpha) \stackrel{?}{=} g(\alpha)t(\alpha)$

From Algebraic Relations to Polynomials

How to prove Many Linear Relations?

Statement: $\vec{y} = \mathbf{M}\vec{z}$.

Plonk, Hyperplonk, Plonky
Permutation-based arguments
$\mathbf{M}$ is a permutation

$$\prod(X + y_i) = \prod(X + z_i).$$

Private Computation

Marlin, Fractal, Spartan
Lincheck-Based Arguments: Reduce many to one relation and use inner product

$$\vec{y} = \mathbf{M}\vec{z} \Longleftrightarrow r^\top \cdot \vec{y} = (\vec{r}^\top \mathbf{M})\vec{z},$$

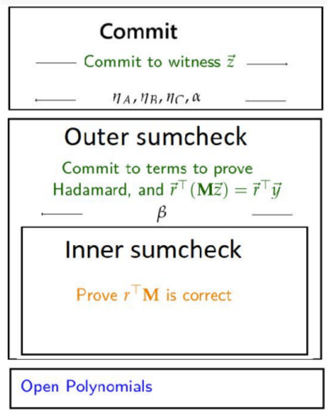
w.h.p. if $\vec{r}$ sufficiently random

Private and Public Computation

1) Private: $\vec{r}^\top \cdot \vec{y} = (\vec{r}^\top \mathbf{M})\vec{z}$
2) Public: $r^\top \mathbf{M}$ correct.

Example of Lincheck-based SNARKs

e.g. Marlin



$$\mathbf{M} = \begin{pmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \end{pmatrix}$$

$$\vec{r} = \begin{pmatrix} \eta_A \vec{\lambda}(\alpha) \\ \eta_B \vec{\lambda}(\alpha) \\ \eta_C \vec{\lambda}(\alpha) \end{pmatrix}.$$

$$\vec{r}^\top \mathbf{M} \leftrightarrow t(X) = \vec{r}^\top \mathbf{M} \vec{\lambda}(X)$$

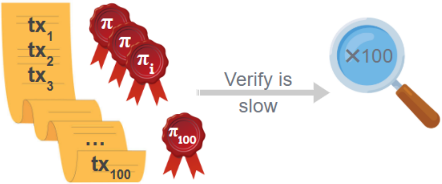
$$\Pi = (\pi_{succ}, \pi_{PC}, \pi_{Lin})$$

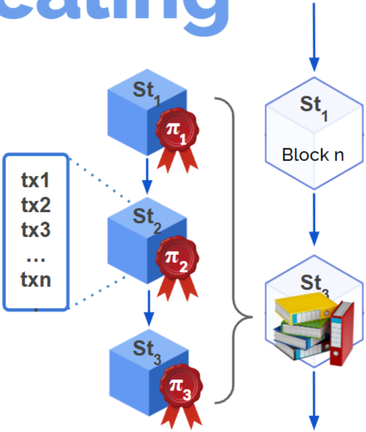
$$b_{succ} \wedge b_{PC} \wedge b_{Lin} \leftarrow \mathcal{V}(x, SRS_{\mathcal{V}}, \Pi)$$

Blockchain scaling

ZK Roll-ups

Periodically provide proofs for valid transaction batches





*Slides of Anca Nitulescu.

Disadvantadges "Monolithic" SNARKs

- No Incremental Proofs.

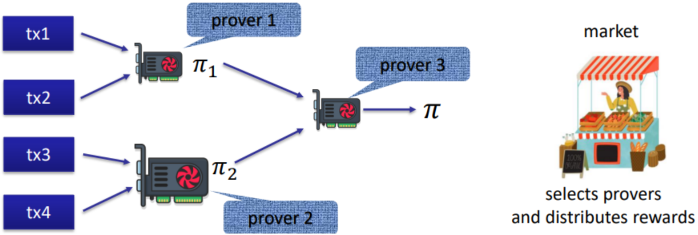
Disadvantadges "Monolithic" SNARKs

- No Incremental Proofs.
- ZK Markets*:

Disadvantadges "Monolithic" SNARKs

- No Incremental Proofs.
- ZK Markets*:

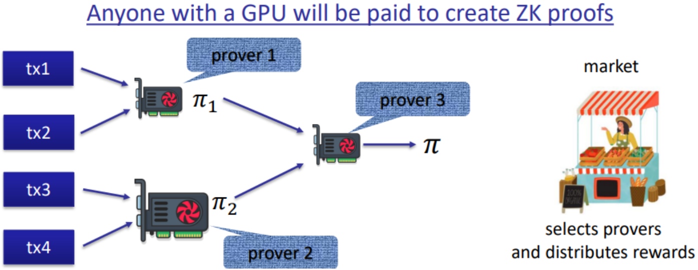
Anyone with a GPU will be paid to create ZK proofs



- Linear (or worse) memory in witness size.

Disadvantadges "Monolithic" SNARKs

- No Incremental Proofs.
- ZK Markets*:

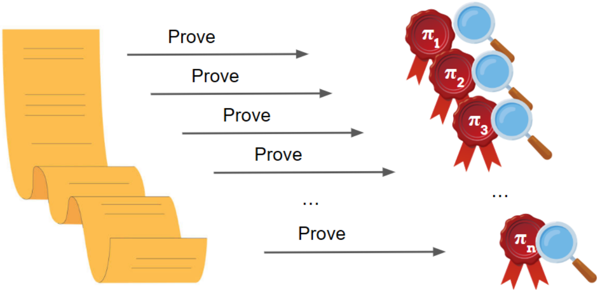


- Linear (or worse) memory in witness size.
- Prover complexity might not scale linearly, i.e. $O(n \log_2 n)$;
- Harder parallelization.

*Drawing of D.Boneh. ZKProof MOOC Course.

Proving Many Instances

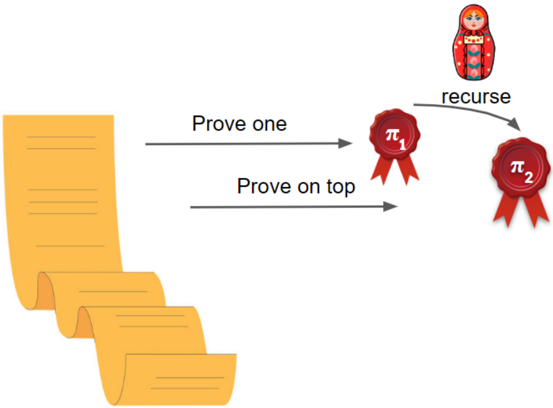
- What if instead of doing a single monolithic proof we cut computation in chunks?



Naive Strategy

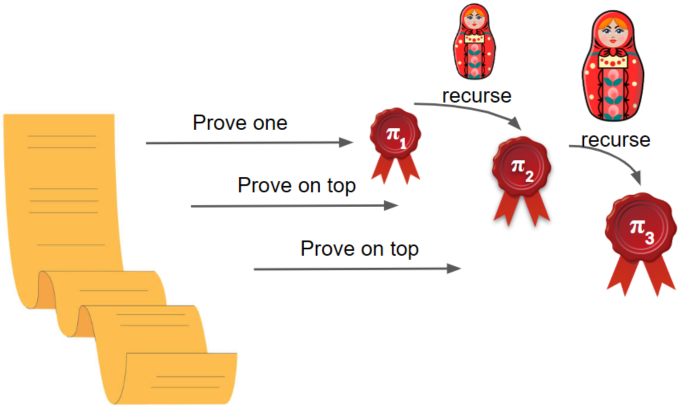
Recursive Proof Composition

Recursion



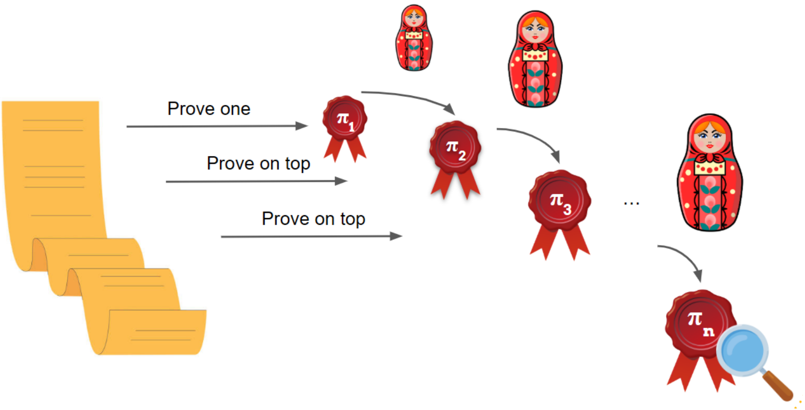
Recursion

Incrementally Verifiable Computation



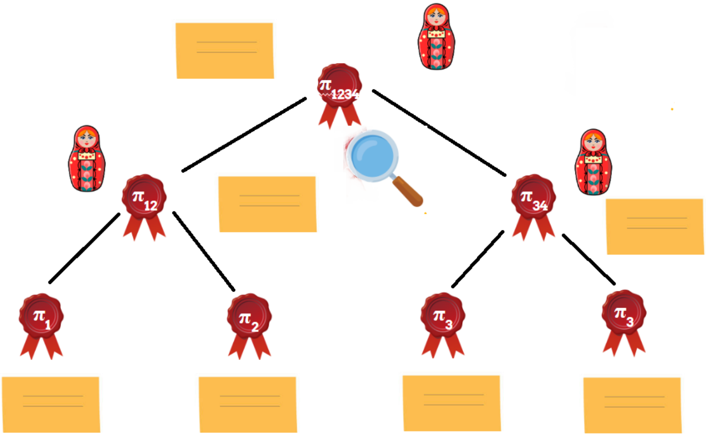
Recursion

Incrementally Verifiable Computation

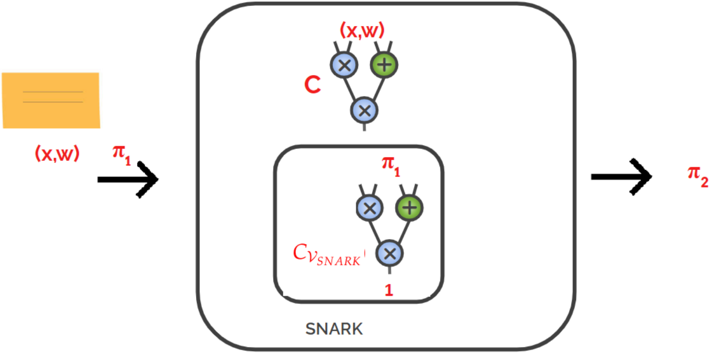


Recursion

Proof Carrying Data

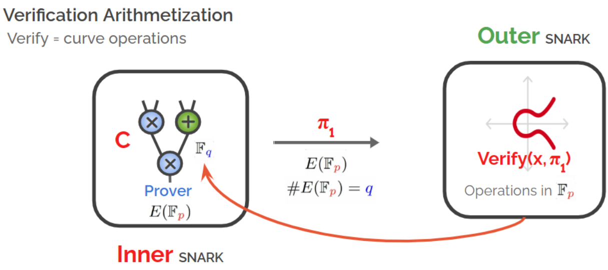


Recursion Overhead

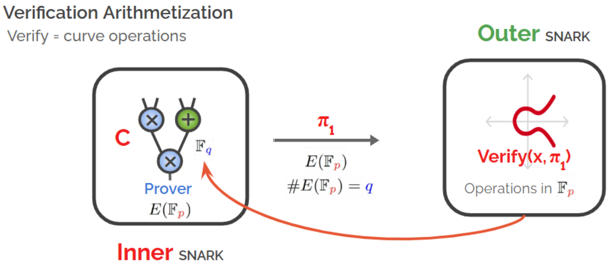


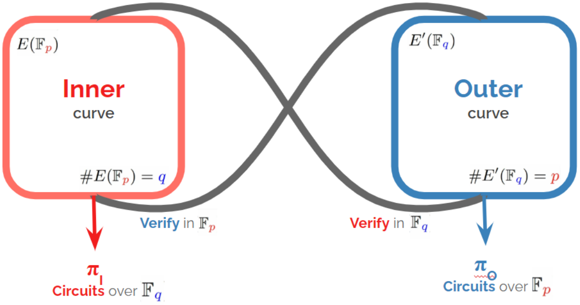
- At each step, proof of corresponding chunk + proof that the previous proof is accepted by the verifier of the snark.
- Total prover work increases with respect to naive approach.
- SNARK verifier must be a "small" circuit.

Proof Recursion in Elliptic Curves

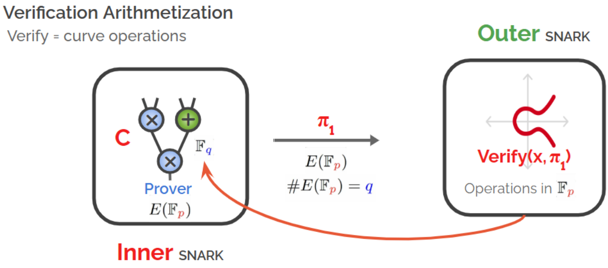


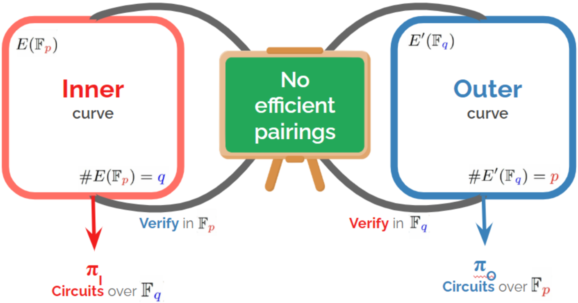
Proof Recursion in Elliptic Curves





Proof Recursion in Elliptic Curves





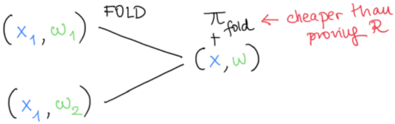
Folding/Accumulation

Amortizing Prover Work: Probabilistically Reduce Two Statements to one

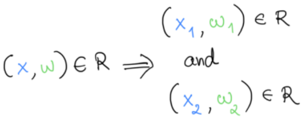
Folding Scheme:

NP language $\mathcal{L}$ with corresponding relation $\mathcal{R}$.

- Fold$(x_1, w_1, x_2, w_2) \rightarrow x, w, \pi_{\text{Fold}}$



- **(Knowledge soundness)**: If FoldVrfy$(x_1, x_2, x, \pi_{\text{Fold}}) \rightarrow 0/1$, then



Folding/Accumulation

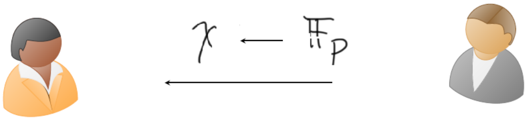
Example

x_i = "c_i opens to a polynomial $p_i(X)$ s.t $p_i(\gamma) = v_i$"

w_i = "coefficients of $p_i(X)$"

$\left(x_i = (c_i, \gamma, v_i),\ w_i = (p_i(X))\right)$ $i = 1, 2$

CLAIMS



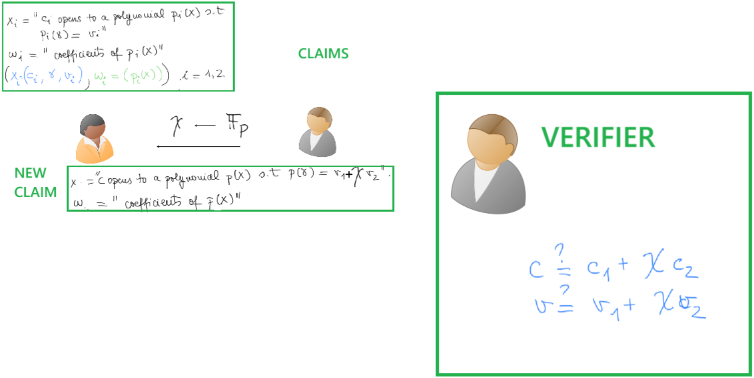
NEW CLAIM

x = "c opens to a polynomial $p(X)$ s.t $p(\gamma) = v_1 + \chi v_2$"

w = "coefficients of $p(X)$"

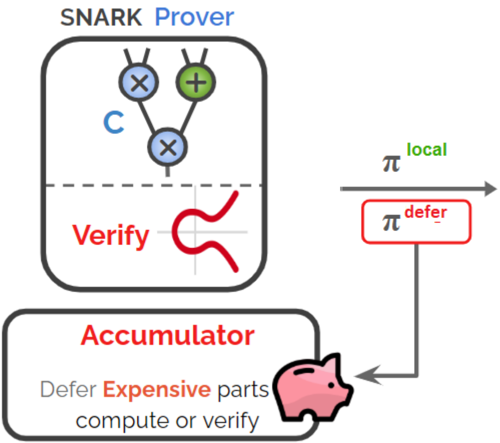
Folding/Accumulation

Example



Recursive Proofs via Folding/Accumulation

- Main idea: at each step execute only some cheap part the SNARK, and accumulate/fold expensive part. Expensive part is deferred to end of computation and only proven once.

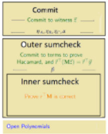


State-of-the-Art

$b_{succ} \wedge b_{PC} \wedge b_{Lin} \leftarrow \mathcal{V}(x, SRS_{\mathcal{V}}, \Pi)$

(1) **Full Recursion:**

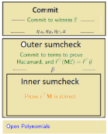
- π_i SNARK proofs
- V verifies π_i
- Fractal, Plonky2



(2) **Atomic Accumulation:**

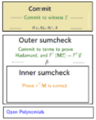
- π_i SNARK proofs
- V partially verifies π_i
- Halo

b_{PC} not fully checked.



(3) **Folding/Split Accumulation:**

- π_i commitment to witness + state s_i
- V verifies correct folding, i.e. RLC of commitments $-- >$ V small
- Nova, ...

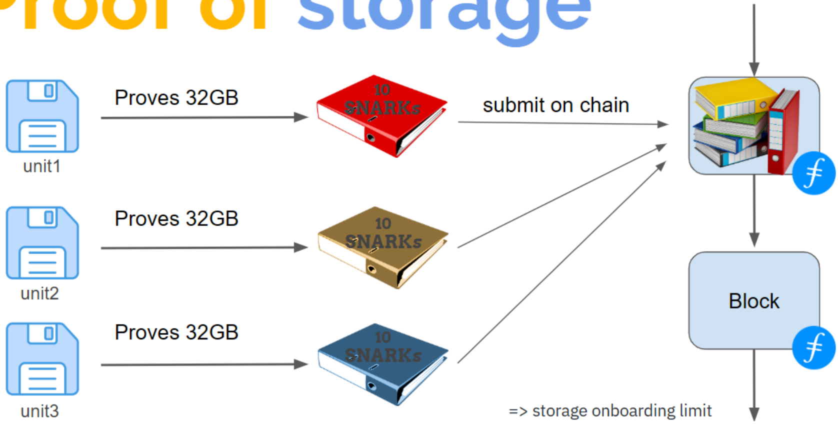


HOW MUCH OF SNARK PROVER IS EXECUTED

FLIP: Fold Inner Product

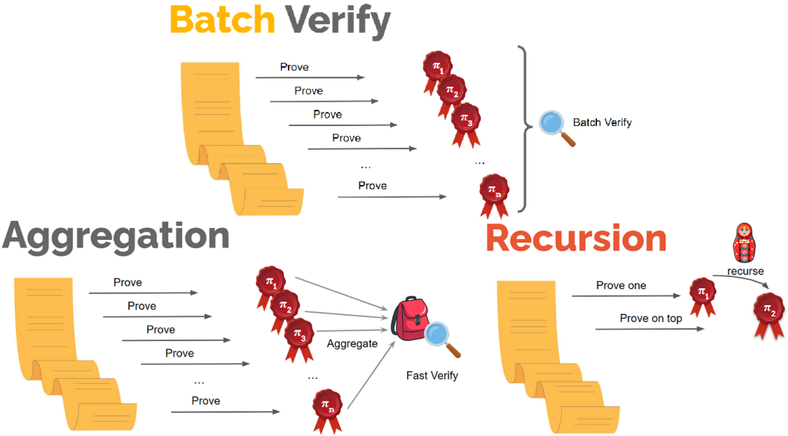
A. Nitulescu, N.Paslis and C. Ràfols. Flip and Prove R1CS. EPRINT IACR.

Proof of storage



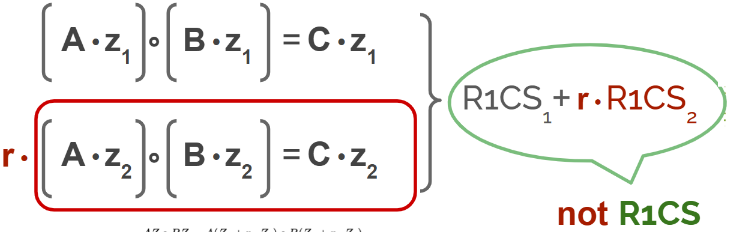
- Real-world example of computation naturally split in many chunks (R1CS instances), one single prover proves many such chunks.

Alternatives?



- Only in recursion with folding prover work is saved by amortization, but construction is complex (cycles).

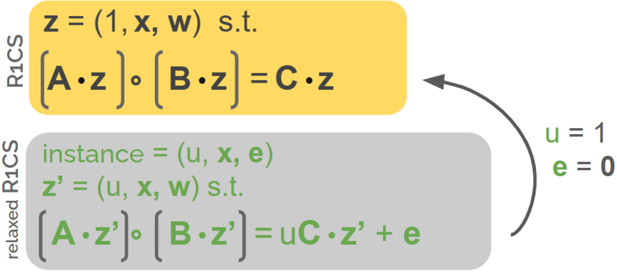
How to fold R1CS?



$$\begin{aligned} AZ \circ BZ &= A(Z_1 + r \cdot Z_2) \circ B(Z_1 + r \cdot Z_2) \\ &= AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2) \\ &\neq CZ. \end{aligned}$$

How to fold R1CS?

NOVA - Kothapalli, Setty, Tzialla'22



How to fold R1CS?

NOVA - Kothapalli, Setty, Tzialla'22

R1CS

$z = (1, \mathbf{x}, \mathbf{w})$ s.t.

$$[A \cdot z] \circ [B \cdot z] = C \cdot z$$

relaxed R1CS

instance = $(u, \mathbf{x}, \mathbf{e})$

$z' = (u, \mathbf{x}, \mathbf{w})$ s.t.

$$[A \cdot z'] \circ [B \cdot z'] = uC \cdot z' + e$$

$u = 1$
$e = 0$

$z_1 = (u_1, x_1, w_1)$

$$[A \cdot z_1] \circ [B \cdot z_1] = u_1 C \cdot z_1 + e_1$$

$r \cdot$

$z_2 = (u_2, x_2, w_2)$

$$[A \cdot z_2] \circ [B \cdot z_2] = u_2 C \cdot z_2 + e_2$$

relaxed R1CS

How to fold R1CS?

NOVA - Kothapalli, Setty, Tzialla'22

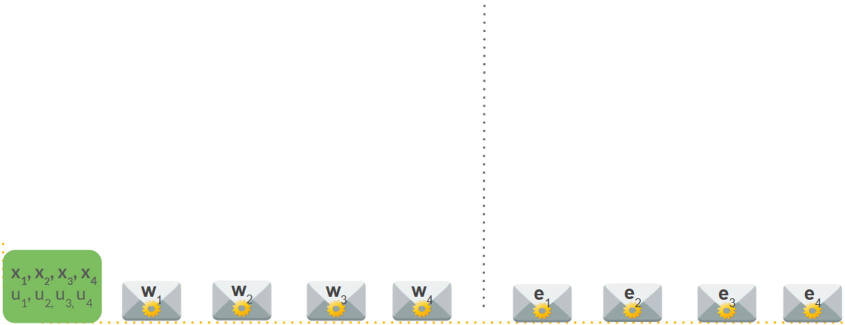
$$z_1 = (u_1, x_1, w_1) \qquad z_2 = (u_2, x_2, w_2)$$

$$\left[A \cdot \underbrace{(z_1 + r z_2)}_{z}\right] \circ \left[B \cdot (z_1 + r z_2)\right] = $$

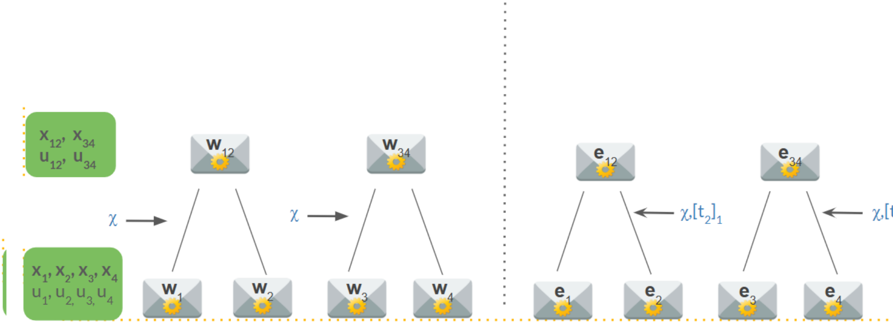
$$= (u_1 + r u_2) C \cdot (z_1 + r z_2) + e$$

$$\begin{aligned} AZ \circ BZ &= AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2) \\ &= (u_1 CZ_1 + E_1) + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (u_2 CZ_2 + E_2) \\ &= (u_1 + r \cdot u_2) \cdot C(Z_1 + rZ_2) + E \\ &= uCZ + E. \end{aligned}$$

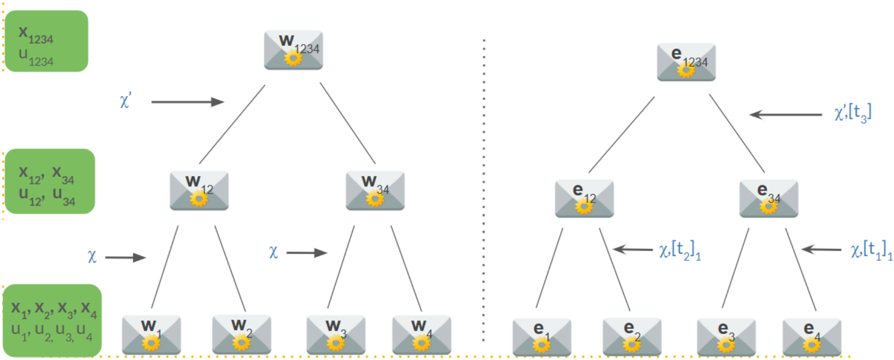
Nova Folding (without cycles)



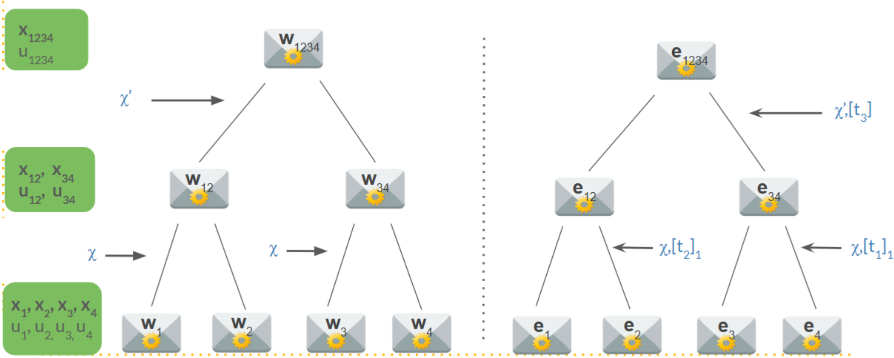
Nova Folding (without cycles)

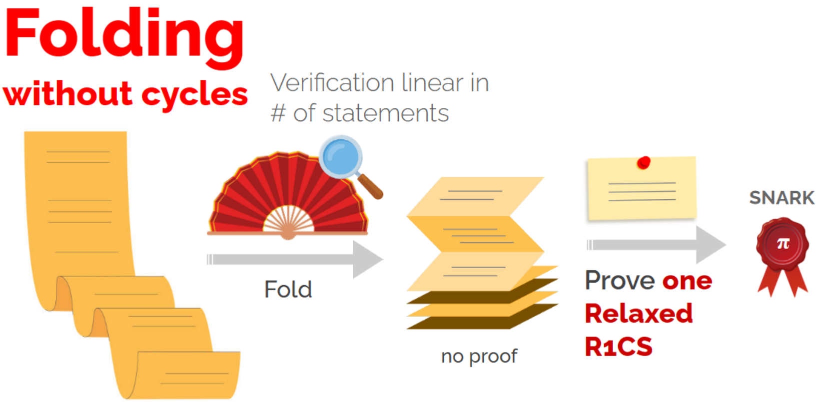


Nova Folding (without cycles)



Nova Folding (without cycles)





- How can we achieve other advantages of recursion: efficient verifier?

Vector Commitment

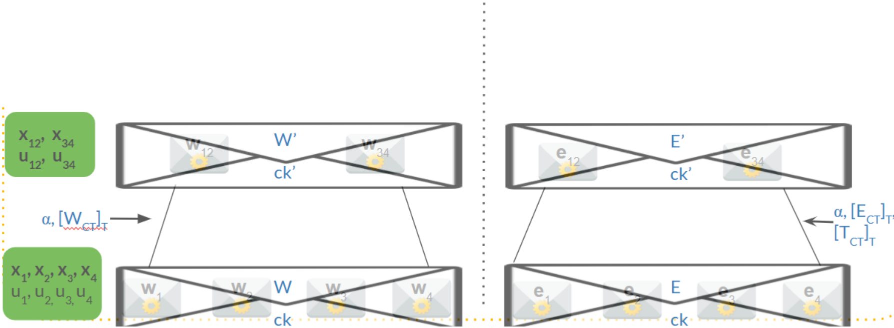
KeyGen(λ, n) $\rightarrow$ ck: $[1]_2, [\tau]_2, [\tau^2]_2, \dots [\tau^n]_2$

Target-Group Commitment

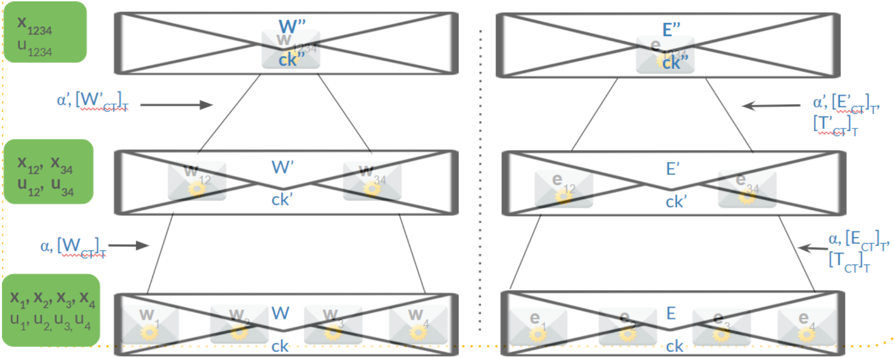
$\rightarrow$ $C = e([A_1]_1,[\tau]_2)e([A_2]_1,[\tau^2]_2)\dots e([A_n]_1,[\tau^n]_2)$

FLIP-style Folding

FLIP-style Folding



FLIP-style Folding



Conclusion

- No recursion, no cycles of elliptic curves.
- Cost of prover: one single relaxed R1CS + O(number of instances) pairings.
- Novel use of homomorphic properties of target group commitments to fold in parralel.

Holography Accumulation

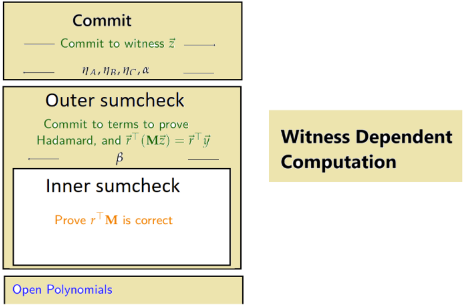
N.Paslis, C. Ràfols and A. Zacharakis. sooon in EPRINT IACR.

Research Question

- What are other meaningful settings in which we can accumulate/amortize prover work?

Research Question

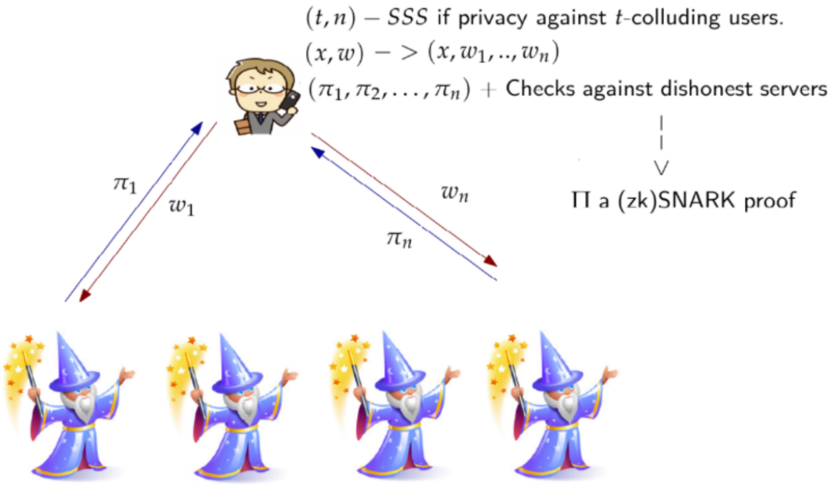
- What are other meaningful settings in which we can accumulate/amortize prover work?
- Idea: Leverage Public Computation in privacy preserving delegation of computation + Recursive Proofs?

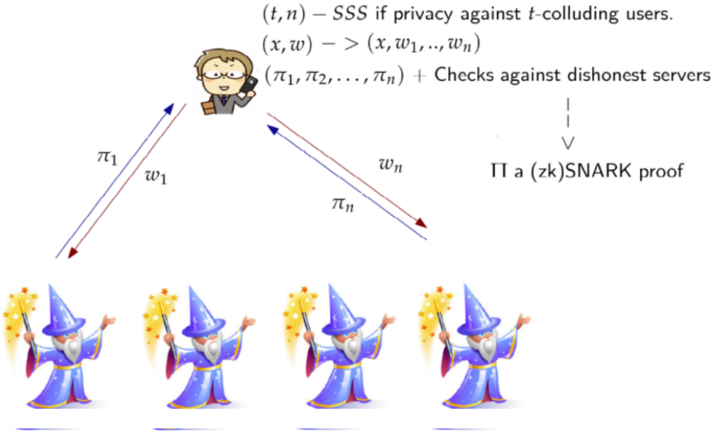


Mar-lin

Privacy Preserving SNARK Proof Delegation

Blueprint: (EOS,zkSaaS,..)

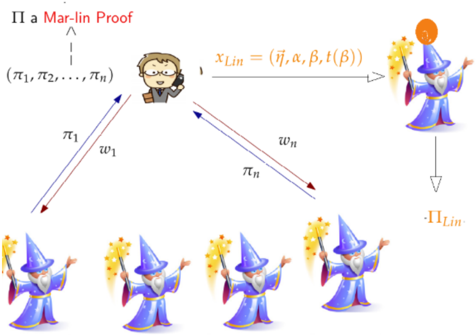




Scenario: Servers do computation as a service for many users, amortize some of the work?

Privacy Preserving SNARK Proof Delegation

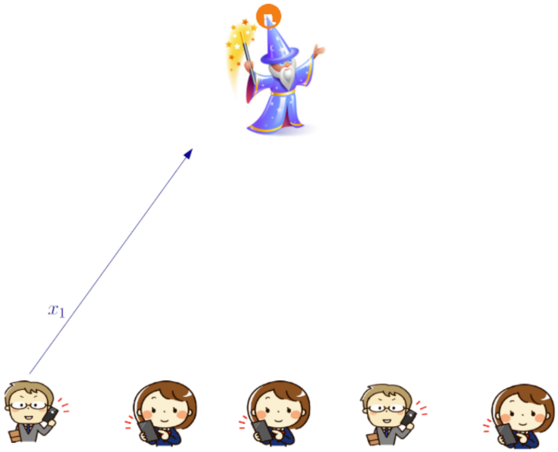
Revisited



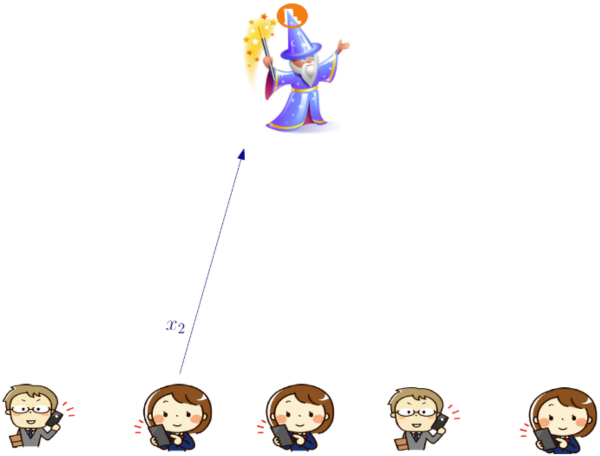
- Delegate public computation (INNER SUMCHECK) to a single powerful server.
- A Mar-lin proof can then be computed locally or delegated using privacy-preservir techniques.
- Verification checks $\Pi + \Pi_{Lin}$

IDEA: Accumulate INNER SUMCHECK to reduce computation per proof

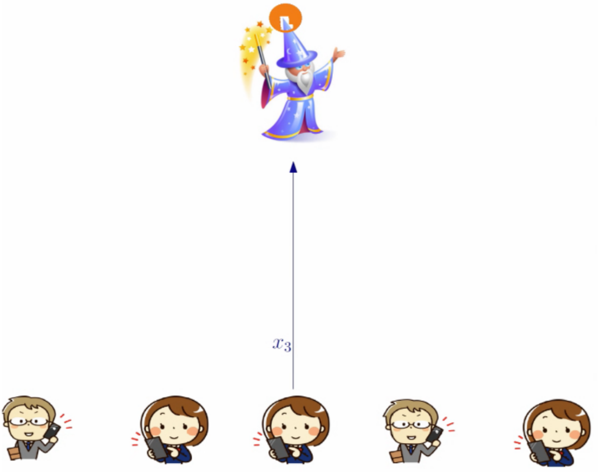
Public Computation as a Service



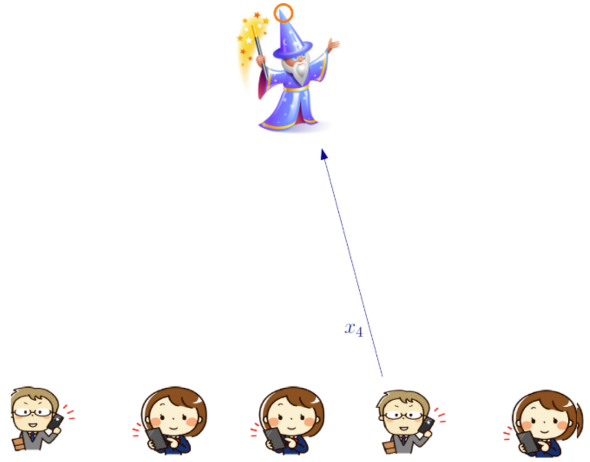
Public Computation as a Service



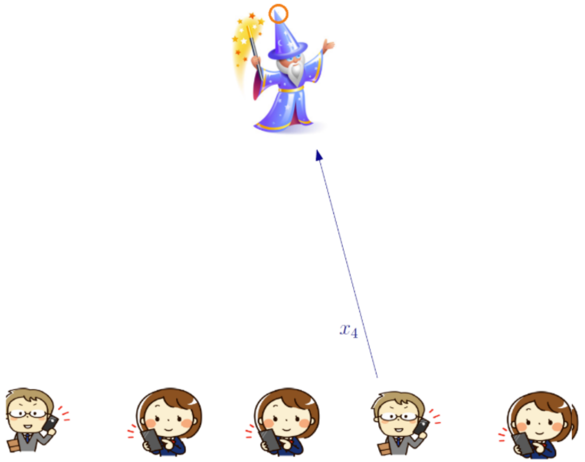
Public Computation as a Service



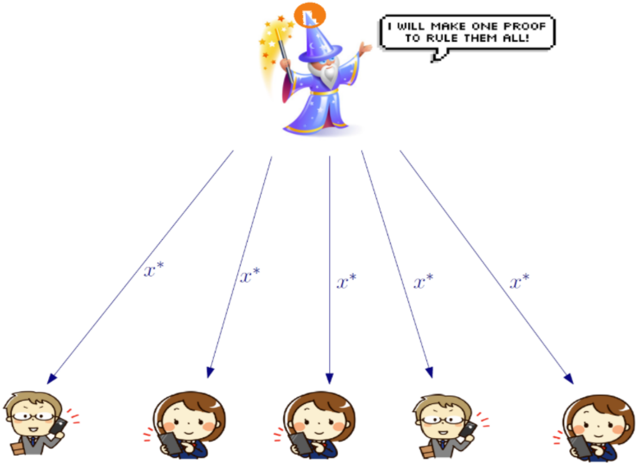
Public Computation as a Service



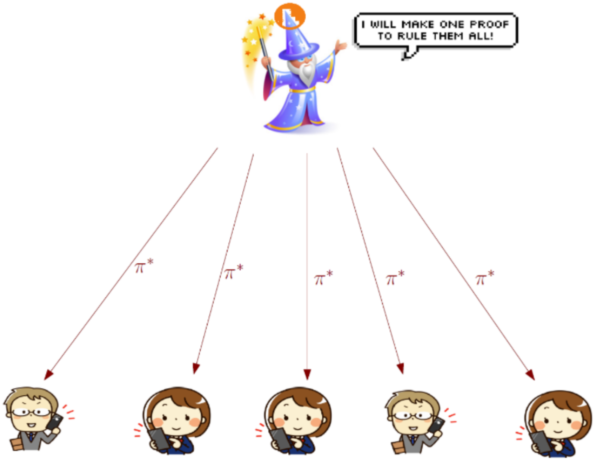
Public Computation as a Service



Public Computation as a Service

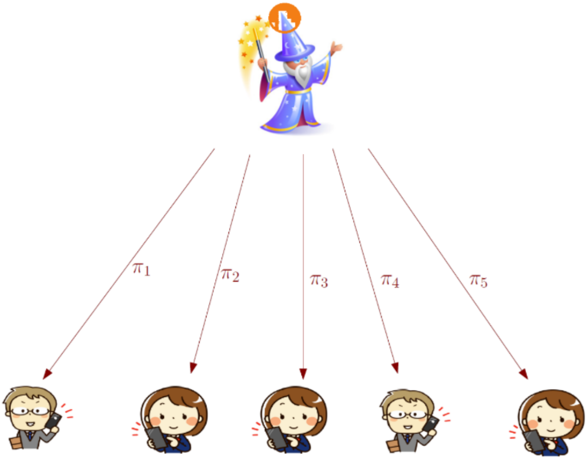


Public Computation as a Service



Users need to receive proof that final stateme
was derived from their individual statements.

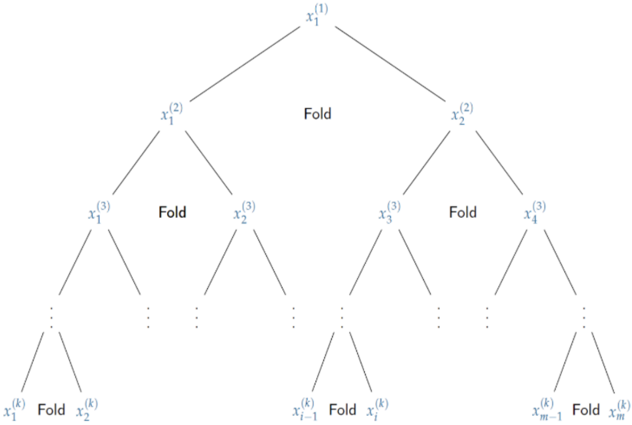
Public Computation as a Service



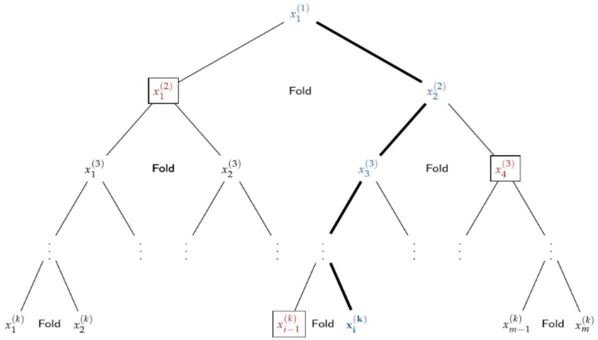
Naive solution:

Proof that asserts that x_i is included in final statement requires knowing all of user's statements

Folding Schemes with Local Verification



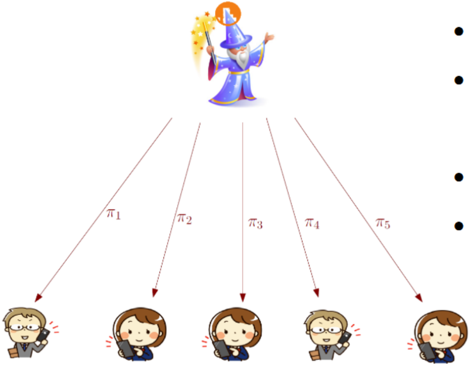
Folding Schemes with Local Verification



Give as *proof* the sibling statements & 2-folding proofs AND Prove only root statement.

Prover: $2m$ foldings + proof root./ Verifier: verify $\pi_i = O(\log m)$ + one proof.

Public Computation aas with Folding Schemes with Local Verification



- Tradeoff cost Server - Verifier Π_{Lin}
- Local/Private part: closer to Groth16 (dominated by 5 MSMs of size |m.gates|)
- Verifier checks $\Pi + \Pi^*_{Lin} + \pi_i$.
- Total Amortized Cost x Proof: essentially local cost.

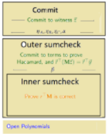
x_i

State-of-the-Art

$b_{succ} \wedge b_{PC} \wedge b_{Lin} \leftarrow \mathcal{V}(x, SRS_{\mathcal{V}}, \Pi)$

(1) **Full Recursion:**

- π_i SNARK proofs
- V verifies π_i
- Fractal, Plonky2

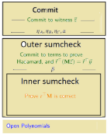


HOW MUCH OF SNARK PROVER IS EXECUTED

(2) **Atomic Accumulation:**

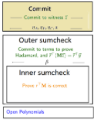
- π_i SNARK proofs
- V partially verifies π_i
- Halo

b_{PC} not fully checked.



(3) **Folding/Split Accumulation:**

- π_i commitment to witness + state s_i
- V verifies correct folding, i.e. RLC of commitments $-->$ V small
- Nova, ...



State-of-the-Art REVISITED

$b_{succ} \wedge b_{PC} \wedge b_{Lin} \leftarrow \mathcal{V}(x, SRS_{\mathcal{V}}, \Pi)$

(1)**Full Recursion:**

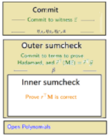
- π_i SNARK proofs
- V verifies π_i
- Fractal,Plonky2

(2)**Atomic Accumulation:**

- π_i SNARK proofs
- V partially verifies π_i
- Halo

b_{PC} not fully checked.

- Darlin: b_{Lin} not checked.



(3)**Folding/Split Accumulation:**

- π_i commitment to witness + state s_i
- V verifies correct folding, i.e. RLC of commitments $-- >$ V small
- Nova, ...

HOW MUCH OF SNARK PROVER IS EXECUTED

Take-away message

- SNARK computation is inherently expensive;
- "Amortization" of prover computation is a key element for scaling provers;
- We have identified three key scenarios where it plays a role:
  - Proving many instances of computation without recursion;
  - Privacy preserving computation of delegation;
  - Recursive proof composition with different tradeoffs.

Take-away message

- SNARK computation is inherently expensive;
- "Amortization" of prover computation is a key element for scaling provers;
- We have identified three key scenarios where it plays a role:
  - Proving many instances of computation without recursion;
  - Privacy preserving computation of delegation;
  - Recursive proof composition with different tradeoffs.

Credits: For the drawings on recursion, folding, and related, the slides are modified from original slides of Anca Nitulescu. She gives credits for clip arts by Iconfinder, Flaticon and juicyfish, and for illustrations to Disneyclips.