LatticeFold & its Applications to Succinct Proof Systems

Dan Boneh

Binyi Chen

Stanford University

(zk)SNARKs

E.g., SHA-3(w) = x

(zk)SNARK = A *succinct* ZK proof showing that $\exists w$ s.t. $C(x, w) = 0$

S$(C) \rightarrow (pp_C, vp_c)$

P(pp_C, x, w) $\xrightarrow{\boxed{\pi}}$ **V** $(vp_C, x, \pi) \rightarrow 0/1$

Properties:

- **Completeness**: honest P can compute valid π
- **Knowledge soundness**: malicious P* knows valid w if it can generate valid π
- **Zero knowledge**: π hide the witness w

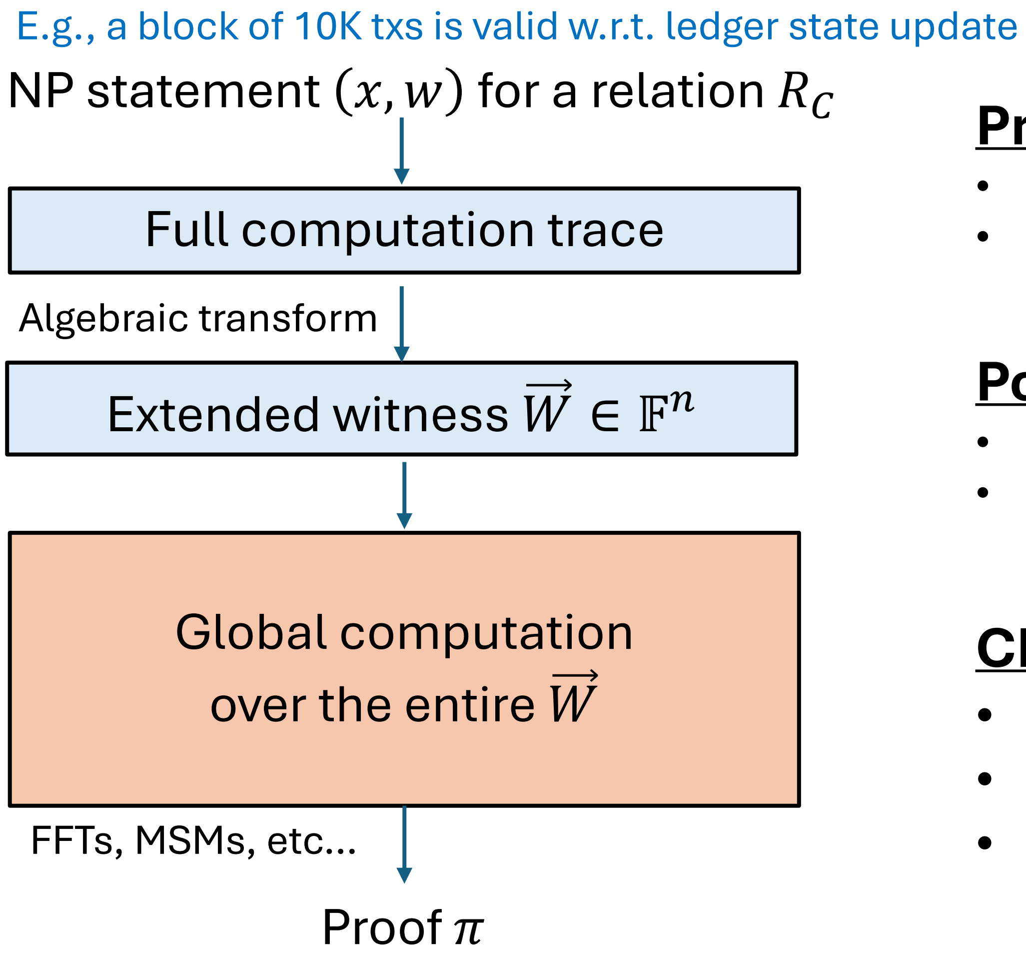
Key requirements for π: **Short** (i.e. $|\pi| \ll |w|$) + **Fast to verify** (e.g. $O(\log|F|)$ time)

Applications: Blockchain, Verifiable zkML/FHE, Fighting disinformation & more

[Xie+22, NT16, DB22, KHSS22, BBBF18, XCBFCK22……]

Challenges: Proving *expensive* statements (e.g., ML tasks) *efficiently*

Monolithic SNARKs [Bitansky-Canetti-Chiesa-Tromer12...]



Pre-quantum Schemes:

- Groth16, Plonk [GWC19], Marlin[CHMMVW20], Bulletproof[BBBPWM18]
- HyperPlonk[CBBZ22], Spartan[Setty19], etc...

Post-quantum Schemes:

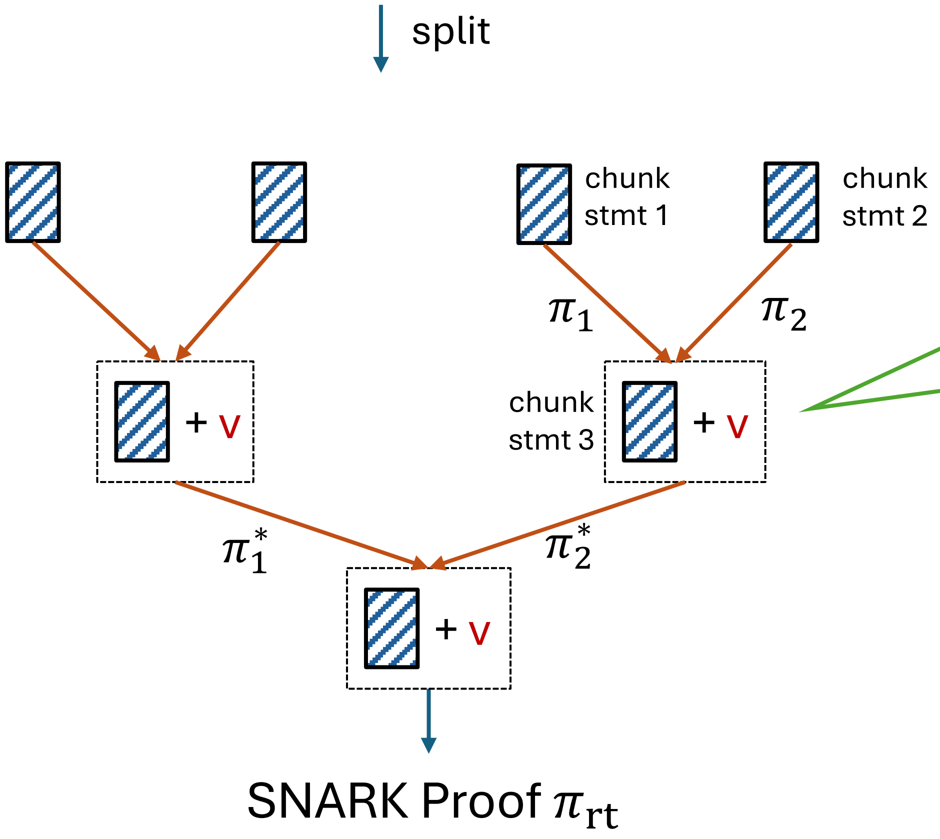
- STARK[BBHR18],Brakedown[GLSTW21],Ligero[AHIV17], Basefold[ZCF23] ...
- Lattice Bulletproofs[BLNS20,ACK21], LaBRADOR[BS22] ...

Challenges for proving expensive computation:

- Expensive global computation
- Large prover memory
- Harder parallelization + less streaming-friendly

Piecemeal SNARKs [Valiant08, BCTV14, BCCT12]

NP statement (x, w) for a relation R_C
e.g., a block of 10K txs is valid w.r.t. the ledger state



Ideas: e.g. Mangrove[NDCTB24], [Sou23]

- Split the statement into multiple small chunks
- Prove chunk statements using SNARK Recursion [Bitansky-Canetti-Chiesa-Tromer12]

SNARK Circuit:

(1) chunk stmt 3 is correct

(2) π_1, π_2 verify correctly

Pros:

- Minimal memory overhead
- Streaming/parallelization friendly

Problem: noticeable recursion overhead

- SNARK generation at each recursion step
- Concretely expensive SNARK verifier circuit

Folding Schemes [KST21,BCLMS20,KS23,BC23]

Committed NP Relation:

com: A commitment scheme

$$(x, w) \in R \quad \Rightarrow \quad (x' = (c, x), w) \in R'$$

if and only if

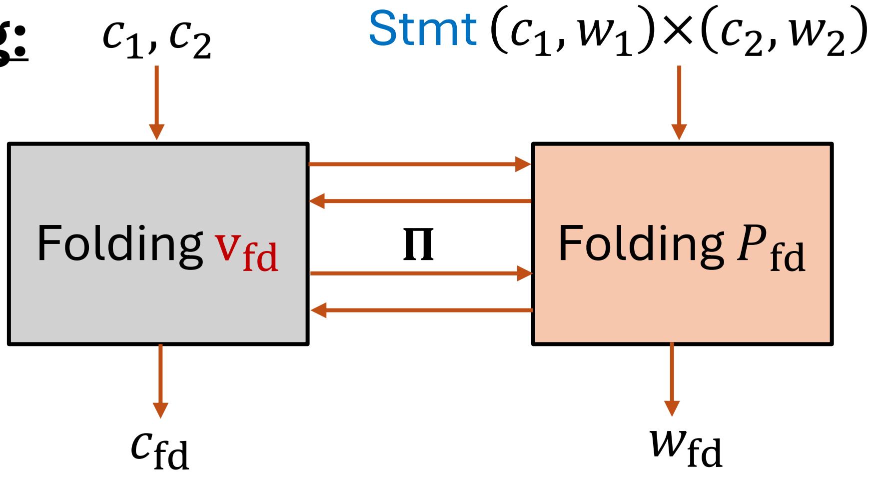
$$(x, w) \in R \wedge (c = \mathrm{com}(w))$$

Next: We omit public input x for notational convenience

Folding Schemes [KST21,BCLMS20,KS23,BC23]

E.g., $c_1 = \mathrm{com}(w_1) \wedge \phi(w_1) = 1$

Folding:



Goal: prove $(c_1, w_1)\times(c_2, w_2) \in R\times R$

Reduced goal: prove $(c_{\mathrm{fd}}, w_{\mathrm{fd}}) \in R$
P_{fd} and v_{fd} can be made non-interactive

Completeness: If $(c_1, w_1)\times(c_2, w_2) \in R\times R$, then $(c_{\mathrm{fd}}, w_{\mathrm{fd}}) \in R$ for honest execution
Knowledge soundness: If $(c_{\mathrm{fd}}\ w_{\mathrm{fd}}) \in R$ for P^*'s output w_{fd}, then P^* also knows w_1, w_2

Generalization: Reduction of knowledge [Kothapalli and Parno23]

Input relation: $R_1 \coloneqq R\times R$

$$(c_{\mathrm{in}}, w_{\mathrm{in}}) \coloneqq (c_1, w_1)\times(c_2, w_2)$$

Output relation: $R_2 \coloneqq R$

$$(c_{\mathrm{out}}, w_{\mathrm{out}}) \coloneqq (c_{\mathrm{fd}}, w_{\mathrm{fd}})$$

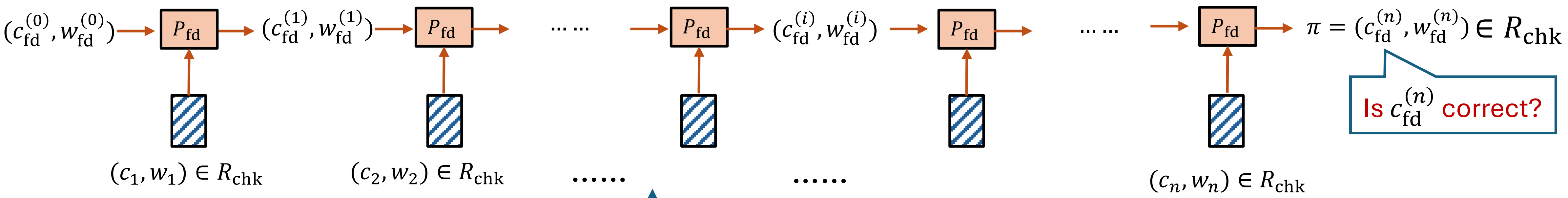
SNARKs from Folding [KST21,BCLMS20,KS23,BC23]

Similar strategies used in SNARGs for P and BARGs[Choudhuru-Jain-Jin21, Waters-Wu22]

Piecemeal SNARK: Prove a chain of computations (can extend to a tree of computations)

SNARK P computes:

SNARK V checks



split

NP statement (x, w) for a relation R_C

Fix:

- Set $\mathrm{x} = \mathrm{H}(c_n, \mathrm{H}(c_{n-1}, \dots, \mathrm{H}(c_1))$ as public input
- SNARK P also sends $(c_1, \dots, c_n)$
- V checks $\mathrm{x} = \mathrm{H}(c_n, \mathrm{H}(c_{n-1}, \dots, \mathrm{H}(c_1))$ and computes $c_{\mathrm{fd}}^{(n)}$ by *iteratively* calling folding v_{fd} given $c_1, \dots, c_n$

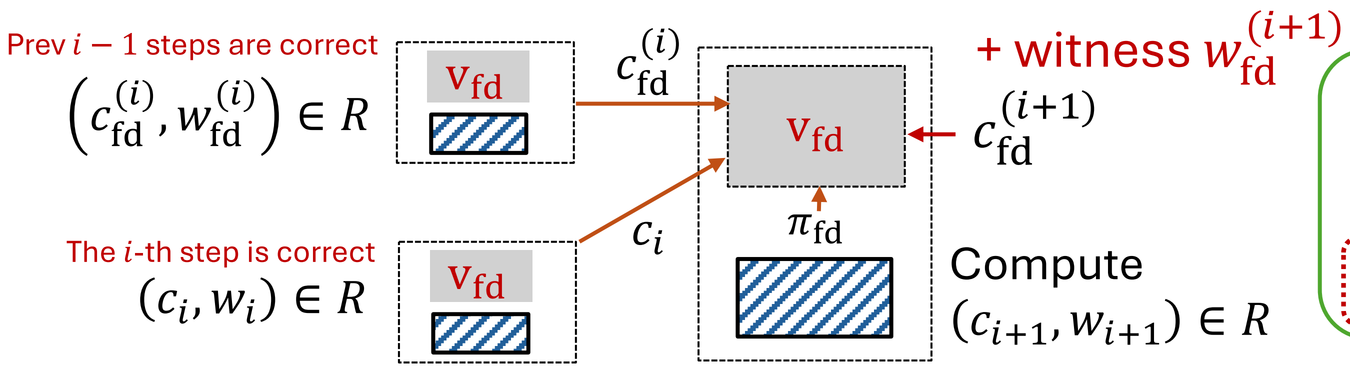
Caveat: proof/verifier complexity linear to n

Idea: Delegate the verifier work into the folded relation

SNARKs from Folding [KST21,BCLMS20,KS23,BC23]

Piecemeal SNARK: Prove a chain of computations (can extend to a tree of computations)

R: an expanded relation to R_{chk}



Omit public input hash checks for simplicity

Relation R :

(1) $c_{i+1} = \mathrm{com}(w_{i+1})$

(2) local computation ▨ is correct

(3) Folding verifier $\mathrm{v}_{\mathrm{fd}}\left(c_i, c_{\mathrm{fd}}^{(i)}, c_{\mathrm{fd}}^{(i+1)}; \pi_{\mathrm{fd}}\right) = 1$

Why faster than SNARK recursion? A folding scheme could be more efficient than a SNARK

Folding verifier v_{fd}:
- $\approx$ check $c_{\mathrm{fd}}^{(i+1)} = c_i + r \cdot c_{\mathrm{fd}}^{(i)}$ for some scalar r
- much simpler than a SNARK verifier!

Simpler relation R

Folding prover P_{fd}:
- $w_{\mathrm{fd}}^{(i+1)} = w_i + r \cdot w_{\mathrm{fd}}^{(i)}$: linear combination of field elems
- much faster than a SNARK prover!

Faster folding for relation R than SNARK proving

Folding Schemes: State-of-the-Art

Committed NP statement $(c, w) \in R$

- Instance c: a short $\mathrm{com}(w)$ to witness w
- com is *linearly-homomorphic* for easy folding

  e.g., $\mathrm{com}(a) + \mathrm{com}(b) = \mathrm{com}(a + b)$

<u>State-of-the-art:</u>

- *Pedersen commitments*
  - *Linearly-homomorphic*
  - *Pairing-free*
  - *No trusted setup*

<u>Alternative Option:</u>

Recursive SNARKs from hash-based STARKs

Less efficient: need full SNARK recursion

<u>Security:</u>

- Based on DLOG assumptions & not *post-quantum secure*

<u>Efficiency:</u>

- Require *cycle curves*
- Prover: many group-exponentiations over a large field
  - Wasteful as real data units usually small (e.g. 32-bit)
- The folding verifier circuit $\mathrm{v_{fd}}$:
  - Elliptic curve scalar multiplications : (
  - *Non-native* field-op simulations : (

    implement arithmetic in $\mathbb{F}_q$ as a circuit over $\mathbb{F}_p$

Can we construct a folding scheme with

- Post-quantum security
- Ultra-fast prover
- Efficient verifier circuit (e.g., no need for non-native field emulation)

Contributions

LatticeFold: The *first* lattice-based folding scheme

- Based on the Module Short-Integer-Solution (MSIS) assumption
- Competitive efficiency vs existing folding schemes
  - Linear-time prover + succinct verifier circuit
  - Relatively small fields (e.g., 32-bit or 64-bit)
- Native simulation of ring operations in circuits
  - More friendly for applications like verifiable FHEs/MLs

Technical contribution:

New folding techniques for lattice-based commitments

Warmup:

Folding for

Relation R **:**

(1) $c_{i+1} = \mathrm{com}(w_{i+1})$

(2) local computation is correct

(3) Folding verifier $\mathrm{v}_{\mathrm{fd}}(c_1, c_2, c_{\mathrm{fd}}; \pi_{\mathrm{fd}}) = 1$

Commitments Opening Relation

Folding for Ajtai Commitment Openings

Committed NP statement $(c, w) \in R$

- Instance c: a short $\mathrm{com}(w)$ to witness w
- com is *linearly-homomorphic* for easy folding

speed ≈ Poseidon hash over fast fields [GKRRS19]

How about Ajtai *binding* commitments?[Ajt96,99]

$$\underbrace{A \leftarrow_{\$} \mathbb{Z}_q^{\lambda \times n}}_{n} \; \vec{w} = c \in \mathbb{Z}_q^{\lambda}$$ **Compact**

$\vec{w} \in \mathbb{Z}_q^n$

$\vec{w}_i \in (-\beta, \beta)$ for $i \in [n]$

Binding for "small-norm" $\vec{w}$ (under ~~SIS~~ Module-SIS assumption)

Generalization

$\mathbb{Z} \Rightarrow R \coloneqq \mathbb{Z}[X]/(X^d + 1)$

$\mathbb{Z}_q \Rightarrow R_q \coloneqq R/qR$

[LM'07,PR'07]

Homomorphic property: (over small-norm messages)

$$c_1 + c_2 = A \times \left(\vec{w}_1 + \vec{w}_2\right) = A \; (\vec{w}_1 + \vec{w}_2)$$

How to commit to $\vec{w}$ w/ large norms?

Dealing with Arbitrary Witness

How to commit to an arbitrary witness $\vec{w}$ w/ large norms?

Comm open relation: Our full-fledged protocol fold a similar relation

$$\tilde{R}^{\beta}_{\text{ajtai}} := \{(c; (\vec{w}, \vec{v})): (c = A\vec{v}) \wedge (\|v\| < \beta) \wedge (\vec{w} = G\times\vec{v})\}$$

Gadget matrix

E.g. $w_1 = [1, 2, 2^2, \dots, 2^{k-1}] \times \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \cdot \\ \cdot \\ \cdot \\ \vec{v}_k \end{bmatrix}$

Next, assume that $\vec{w}$ is always **low-norm** in the first place!

Comm open relation:

$$R^{\beta}_{\text{ajtai}} := \{(c, \vec{w}) : c = A\vec{w} \wedge \|w\| < \beta\}$$

The infinite norm of $w \in \mathbb{Z}^n$
$\|w\| := \max(|w_i|)^n_{i=1}$

Folding for Ajtai Commitment Openings

Comm open relation: $R^{\beta}_{\text{ajtai}} := \{(c, \vec{w}) : c = A\vec{w} \wedge \|w\| < \beta\}$

The infinite norm of $w \in \mathbb{Z}^n$
$\|w\| := \max(|w_i|)_{i=1}^{n}$

Naïve approach:

$(c_1, w_1) \in R^{\beta}_{\text{ajtai}} \rightarrow$

$(c_2, w_2) \in R^{\beta}_{\text{ajtai}} \rightarrow$

$\boxed{\text{Folding } P_{\text{fd}}}$

$r \in \mathbb{Z}_q$ is a random scalar

$$\rightarrow \begin{pmatrix} c_{\text{fd}} := c_1 + r \cdot c_2 \\ w_{\text{fd}} := w_1 + r \cdot w_2 \end{pmatrix} \notin R^{\beta}_{\text{ajtai}}!$$

Problems:

- $\|w_{\text{fd}}\|$ can be larger than β (even if $\|r\|$ is small)
- c_{fd} no longer binding after $\|w_{\text{fd}}\|$ exceeds threshold

  Can't support many folding steps

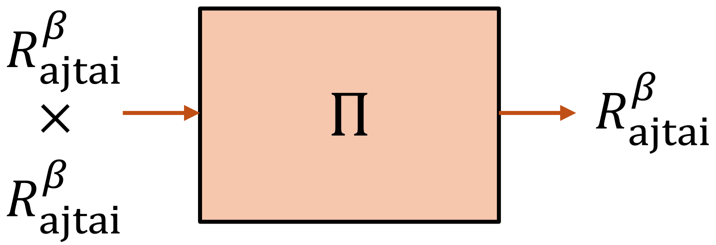
Thoughts:

Make $\|w_1\|, \|w_2\|$ smaller before random LinComb?

Our Strategy

Relation: $R^{\beta}_{\text{ajtai}} := \{(c, \vec{w}) : c = A\vec{w} \wedge \|w\| < \beta\}$

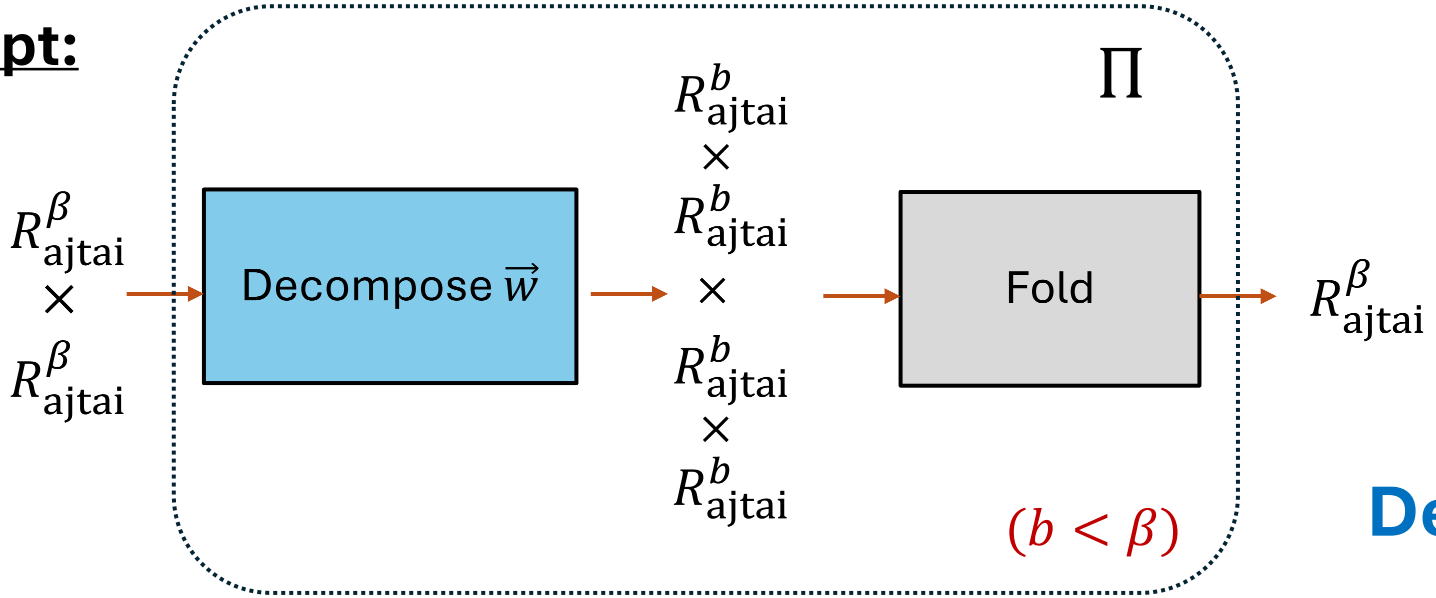
<u>Recall our goal:</u> reduction of knowledge Π Nice property of RoK! [Kothapalli and Parno23]



<u>Sequential composition:</u>

$$R_1 \xrightarrow[\Pi_a]{} R_2 \xrightarrow[\Pi_b]{} R_3 \Rightarrow R_1 \xrightarrow[\Pi_b \circ \Pi_a]{} R_3$$

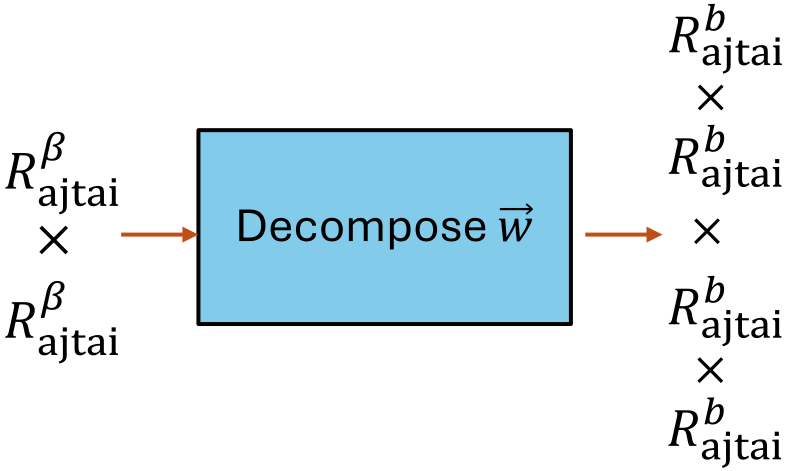
<u>Attempt:</u>



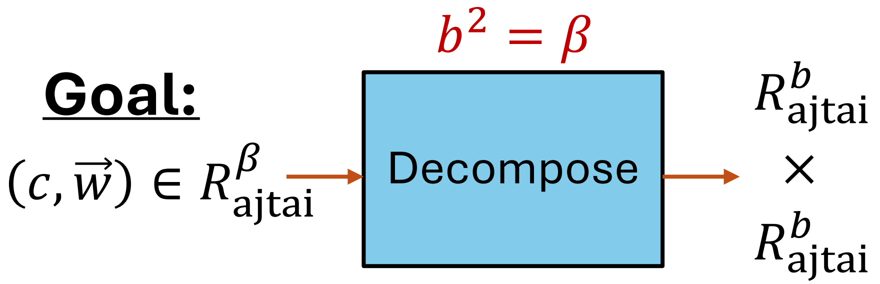
How to instantiate Decompose and Fold?

Roadmap

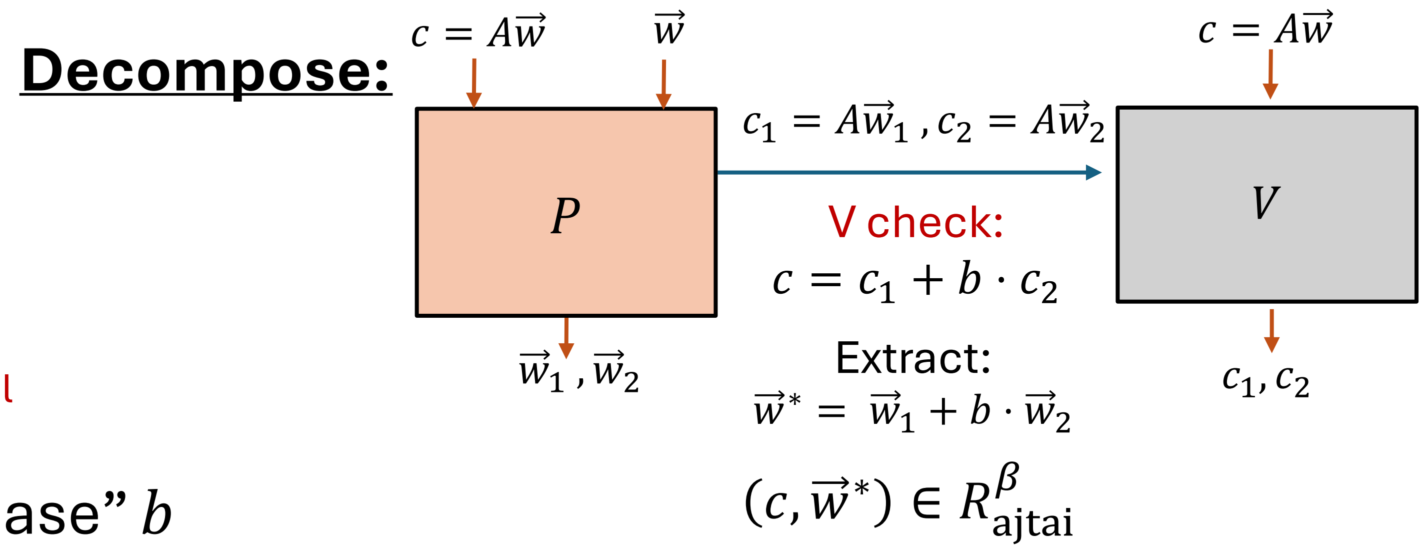
- Decomposition Protocol
- Fold Protocol



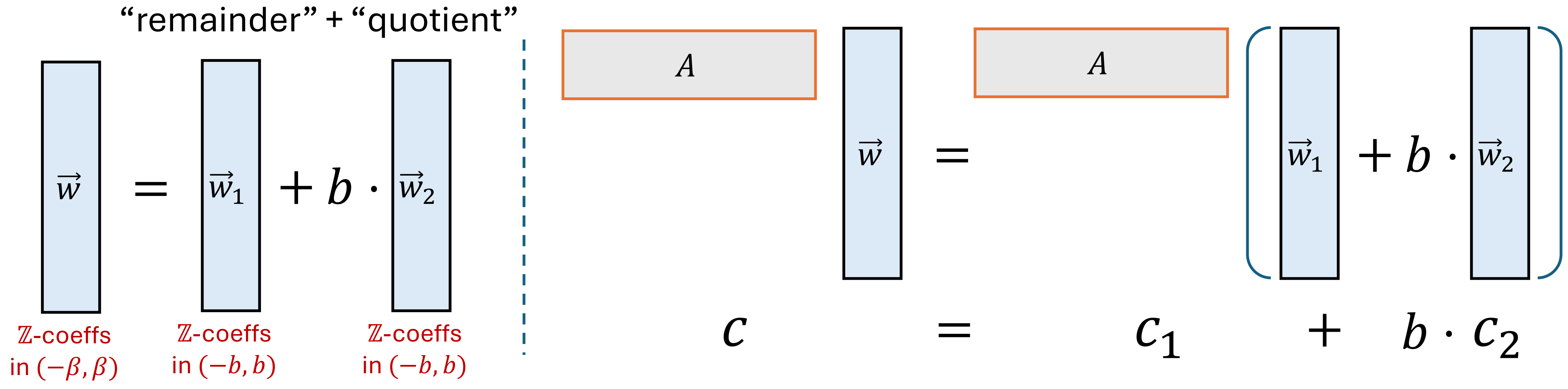
Norm Control with Decomposition



RoK from $R^{\beta}_{\text{ajtai}} \times R^{\beta}_{\text{ajtai}}$ to $\left(R^{b}_{\text{ajtai}}\right)^4$
is a parallel composition of the above protocol



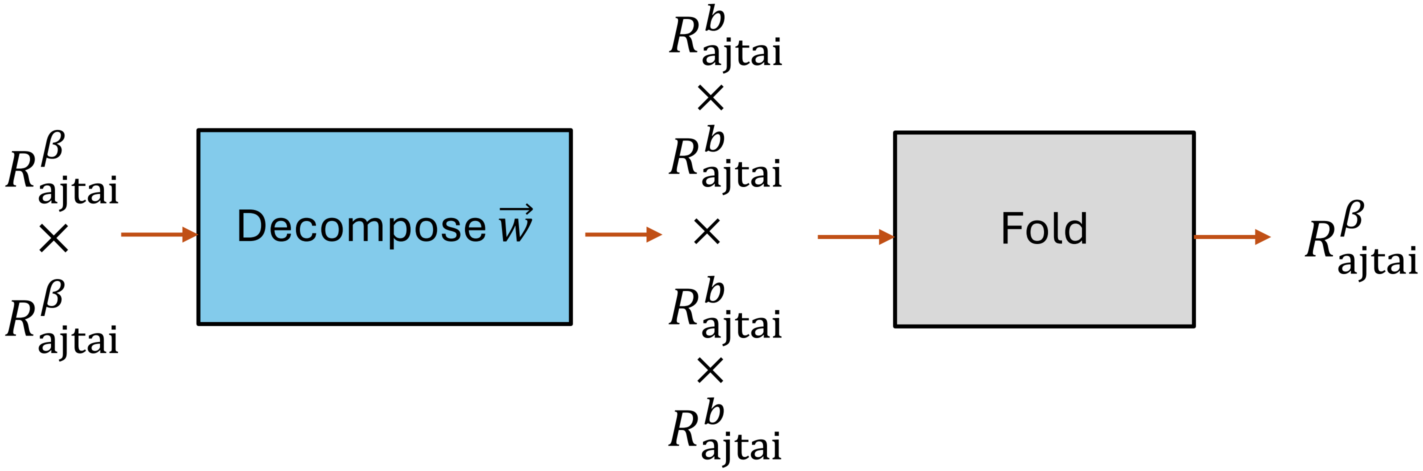
"Write" the big vector $\vec{w}$ using "base" b



Roadmap 

- Decomposition Protocol

- Fold Protocol



Folding: Naïve Approach

Goal:

$R^{b}_{\text{ajtai}} \times \cdots \cdots \times R^{b}_{\text{ajtai}} \rightarrow$ Fold $\rightarrow R^{\beta}_{\text{ajtai}}$

$b \ll \beta$

Naïve idea:

$(c_1, w_1) \in R^{b}_{\text{ajtai}} \rightarrow$ Folding P_{fd}

$(c_2, w_2) \in R^{b}_{\text{ajtai}} \rightarrow$ Folding P_{fd}

$r \in \mathbb{Z}_q$ is a **small** random scalar

$$\begin{pmatrix} c_{\text{fd}} \coloneqq c_1 + r \cdot c_2 \\ w_{\text{fd}} \coloneqq w_1 + r \cdot w_2 \end{pmatrix} \in R^{\beta}_{\text{ajtai}}!$$

Completeness ✓

Knowledge extraction: Rewind P^*_{fd} to obtain $w^x_{\text{fd}}, w^y_{\text{fd}}$ for $c^x_{\text{fd}} = c_1 + r_x \cdot c_2$ and $c^y_{\text{fd}} = c_1 + r_y \cdot c_2$

$$\begin{pmatrix} w^x_{\text{fd}} = w_1 + r_x \cdot w_2 \\ w^y_{\text{fd}} = w_1 + r_y \cdot w_2 \end{pmatrix} \Longrightarrow$$

Solve linear eqs for w_1, w_2

Extracted witness: $\boxed{(c_2, w_2) \notin R^{b}_{\text{ajtai}}!}$

$$w_2 = \underbrace{\left(w^y_{\text{fd}} - w^x_{\text{fd}}\right)} \cdot \underbrace{\left(\boldsymbol{r_y - r_x}\right)^{-1}}$$

The *norm* can be much larger than *b*!

Same for w_1

Roadmap 

- Decomposition Protocol

- Fold Protocol
  - Naïve extraction + argue smallness of the extracted witness

Using Range proof: witness $w \in [-b, b]^n$

(Batched) Range proof via Sumcheck

$c' = c_1$ & c_2 in folding $\qquad\qquad$ $\vec{w}' = w_1$ & w_2 in folding

Goal: Given input commitment c', prove knowledge of $\vec{w}' = (f_1, f_2, \ldots, f_n) \in \mathbb{Z}^n$

- $c' = A\vec{w}'$
- $\vec{w}' = (f_1, f_2, \ldots, f_n)$ **has norm smaller than** b
- Efficient (folding) verifier circuit

Our strategy: Combine naïve folding & extraction + Range proof protocol

- $c' = A\vec{w}'$ (achieved by naïve folding + extraction)
- $\vec{w}' = (f_1, f_2, \ldots, f_n)$ **has small norms**

Our solution: A range-proof protocol from Sumcheck

Review of the Sumcheck Protocol [LFKN92]

Goal: Given a "committed" m-variate poly $g(x_1, \ldots, x_m)$, convince V that $\sum_{x \in \{0,1\}^m} g(\vec{x}) = s$

Naïve verifier: query g at every $x \in \{0,1\}^m$ and check the sum

$\Omega(2^m)$ complexity : (

Sumcheck protocol [LFKN92]

- m-round interactive protocol between P and V
  - V sends a random challenge $r_i \in \mathbb{F}$ in each round
- At the end of the protocol, V queries g at a *single* random point

Sumcheck: $\Sigma_{\vec{x} \in \{0,1\}^m} g(\vec{x}) = s$

Sumcheck protocol [LFKN92] $O(m)$-time verifier

EvalCheck: $g(\vec{r}_1, \ldots, \vec{r}_m) = t'$ at a random $\vec{r} \in \mathbb{Z}_q^m$

A reduction from Sumcheck to Eval stmt

History: Key ingredient for proving $PH \subseteq IP$ and inspires the proof of $IP = PSPACE$

Goal: Given input commitment c', prove knowledge of $\vec{w}' = (f_1, f_2, \dots, f_n) \in \mathbb{Z}^n$

- $\vec{w}' = (f_1, f_2, \dots, f_n)$ **has norm smaller than** b

Our solution: A range-proof protocol from Sumcheck

Step 1: Rephrase the range-proof statement as a Sumcheck statement

Step 2: Construct a folding protocol for the Sumcheck statement

Step 1: Reducing Range proof to Sumcheck

Range proof: Prove knowledge of a witness $\vec{w}' = (f_1, f_2, \dots, f_n) \in \mathbb{Z}^n$ s.t.

Can extend to elements in ring $R = \mathbb{Z}[X]/(X^d + 1)$

$f_1 \in \mathbb{Z}$	f_2	…	…	…	f_{n-1}	f_n
$\in (-b, b) \subseteq \mathbb{Z}$	$\in (-b, b)$				$\in (-b, b)$	$\in (-b, b)$

$\boxed{h(x) = 0 \Leftrightarrow x \in (-b, b)}$

$h(x) := x(x+1) \cdot (x-1) \cdots \big(x + (b-1)\big)\big(x - (b-1)\big)$ over $\mathbb{Z}_q := \left[-\frac{q}{2}, \frac{q}{2}\right]$ and $q > 2b$ is a prime

$h(f_1) = 0$	$h(f_2) = 0$	…	…	…	$h(f_{n-1}) = 0$	$h(f_n) = 0$

Embed $\vec{w}'$ to the Boolean hypercube of a multilinear polynomial $f(x_1, \dots, x_{\log n})$

$\vec{x}$	$00 \dots 00$	$00 \dots 01$	…	…	…	$11 \dots 10$	$11 \dots 11$
$h(f(\vec{x}))$	$h(f_1) = 0$	$h(f_2) = 0$	…	…	…	$h(f_{n-1}) = 0$	$h(f_n) = 0$

Zero-check to sum-check [CBBZ23, Setty20]

Sumcheck: prove that $\Sigma_{\vec{x} \in \{0,1\}^{\log n}} g(\vec{x}) = 0$ where $g(\vec{x}) := h\big(f(\vec{x})\big) \cdot eq_\alpha(\vec{x})$ for a rand $\alpha \in \mathbb{Z}_q^{\log n}$

Step 2: Sumcheck Folding

Range proof: witness $\vec{w}' = (f_1, f_2, \dots, f_n) \in [-b, b]^n$

$g(\vec{x}) \coloneqq h\big(f(\vec{x})\big) \cdot eq_\alpha(\vec{x})$

Sumcheck: $\Sigma_{\vec{x} \in \{0,1\}^{\log n}} g(\vec{x}) \ = 0$

Step 1 ✓

Sumcheck protocol [LFKN92]

Prover time: $\approx O(bn)$

Verifier time: $O(b \log n)$

EvalCheck: $g(\vec{r}) = t'$ at a random $\vec{r} \in \mathbb{Z}_q^{\log n}$

EvalCheck: $f(\vec{r}) = t$ (and verifier can check $g(\vec{r}) = \ h(t) \cdot eq_\alpha(\vec{r}) = t'$ itself)

<u>**Problem:**</u> How to check $f(\vec{r}) = t$ given the comm of f?

- Send $(f_1, f_2, \dots, f_n)$ to the folding verifier to check it? $O(n)$ folding verifier : (

Observation: EvalStmt $f(\vec{r}) = t$ is easy to fold!

Folding Evaluation Statements

Observation: $f(\vec{r}) = t$ is easy to fold!

Multilinear extension: $f(\vec{r}) = \Sigma_{\vec{x}\in\{0,1\}^{\log n}} f(\vec{x}) \cdot \boxed{eq_{\vec{r}}(\vec{x})}$ efficiently computable

Translate to SumChk Stmt $\left\{ \begin{array}{l} f_1(\vec{r}_1) =_? t_1 \\ f_2(\vec{r}_2) =_? t_2 \end{array} \right.$ $\rightarrow$ SumCheck $\rightarrow$ $\left. \begin{array}{l} f_1(\vec{r}_o) =_? t_{o,1} \\ f_2(\vec{r}_o) =_? t_{o,2} \end{array} \right\}$ $f_{\text{fd}}(\vec{r}_o) =_? t_o$

$f_{\text{fd}} \coloneqq f_1 + \rho \cdot f_2$ for rand ρ

$\vec{r}_o$: sumcheck challenge

How does it help to check $f(\vec{r}) = t$ given the comm of f?

- Fold the evaluation statement without checking!

Folding for Ajtai Commitment Openings

Solution: Expand relation R_{ajtai} to include the evaluation statement

$(c = \text{com}(f)) \wedge (f(\vec{r}) = t)$

Online statement

R^b_{ajtai} $\boxed{c_1 =_? \text{com}_b(f_1)}$

R^b_{eval} $\boxed{\begin{array}{c} c_2 =_? \text{com}_b(f_2) \\ \wedge \\ f_2(\vec{r}_2) =_? t_2 \end{array}}$

Accumulated statement

MatMul + RangePf + EvalStmt $\rightarrow$

Naïve Fold
+
SumCheck for
RangeProof & EvalStmt

$\rightarrow$ $\boxed{\begin{array}{c} c_{\text{fd}} =_? \text{com}_\beta(f_{\text{fd}}) \\ \wedge \\ f_{\text{fd}}(\vec{r}_o) =_? t_o \end{array}}$ R^β_{eval}

New accumulated statement

Verifier: $O(b\log n)$

The knowledge soundness proof is more subtle than intuition

- A malicious prover can *adaptively* choose the output witness after seeing the challenges
- $\Rightarrow$ The extracted input witnesses could *depend on* the sumcheck challenges

Subtleties & Optimizations

Sumcheck over Rings: [CCKP19, BCS21]

- Ajtai commitments over ring $R_q := \mathbb{Z}_q[X]/(X^d + 1)$ for concrete efficiency
- Small-norm random folding scalar chosen from $S \subseteq R_q$ for negligible soundness error
- Implication: Run Sumcheck over rings

Supporting Small Modulus:

- We want a small modulus q for better efficiency
  - Efficient CPU/GPU ops; no big-number arithmetics
  - More efficient packing of real-world data

Folding for NP-complete relation

Arithmetic over a ring → Great fit for Verifiable ML/FHE

Relation R :

(1) $c_{i+1} = \mathrm{com}(w_{i+1})$

(2) local computation is correct

(3) Folding verifier $\mathrm{v_{fd}}(c_1, c_2, c_{\mathrm{fd}}; \pi_{\mathrm{fd}}) = 1$

needs to express computation

Efficiency Estimates

$R_q := \mathbb{Z}_q[X]/(X^{64}+1) \cong \mathbb{F}_{q^4}^{16}$; q: a 64-bit prime
C_{chk}: chunk circuit size (e.g. 2^{20} gates over $\mathbb{F}_{q^4}$)
Norm bound: $\beta \approx 2^{16}$; Base: $b = 2$

	LatticeFold	**Existing schemes**
Folding prover:	speed ≈ fast hash Compute Ajtai commitments $O(\lvert C_{\text{chk}}\rvert)$ multiplications over R_q Can reuse fast FHE impl!	Pedersen commimtents $O(\lvert C_{\text{chk}}\rvert)$-sized Multi-Scalar-Muls
Folding verifier:	Sumcheck verifier $O(b \cdot \log\lvert C_{\text{chk}}\rvert)$ hashes and R_q-ops **native-ops** in the circuit over R_q Competitive circuit sizes	ECC scalar-mul + (Sumcheck V) **non-native field ops** in the circuit i.e., arithmetic in $\mathbb{F}_q$ as a circuit over $\mathbb{F}_p$

Piecemeal SNARK proof: ≈2 folding instance-witness pairs What if it's still large?
E.g., splitting a stmt of size 2^{40} to 2^{20} chunks → 2^{20}-sized chunk stmts

Solution: Use a PQ-secure STARK to prove the correctness of the folding statement
$< 100\text{KB}$ and 2ms verifier (STIR[ACFY24]) $< 5\text{KB}$ w/ Hyperplonk+KZG[CBBZ23]

Summary & Open Problems

Takeaway:

- The *first* lattice-based folding scheme based on Ajtai commitments
- Gives memory-efficient, plausibly PQ-secure SNARKs, with fast provers
- Generic techniques for folding lattice-based commitments w/ norm constraints

Open problems:

- *Compact* + *homomorphic* lattice commitments with no norm constraints
- Folding table lookup relations (e.g., from Lasso [Setty-Thaler-Wahby23])
- Efficient implementation

Concurrent work: [Bünz-Mishra-Nguyen-Wang24]

- Purely from hashing; no lattice crypto
- General optimization techniques for piecemeal SNARKs (apply to LatticeFold)
- Larger verifier circuit; only supports bounded-depth folding (attack exists)

Thank you!

https://eprint.iacr.org/2024/257.pdf

Expecting updates soon!