November 18, 2024

Succinct Arguments over Towers of Binary Fields

↑ Diamond & Posen 2023

Outline:

0. Why SNARKs matter to industry - folk
1. Overview of Modern SNARK construction
2. Mathematical Preliminaries
   - A. Multilinear Polynomials
   - B. Towers of Binary Fields
3. The Binius IOP (Bird's Eye)
   - A. Zerocheck
   - B. Virtual Polynomials
   - C. Add Gadget Example
4. Small Field Multilinear Poly Commit Scheme (Tease only)

Goal: To introduce a technical audience to a SOTA SNARK proof system and shed light on what "cryptography" and "algorithms" industry cares for

Don't be shy to interrupt me with questions

⓪ Why SNARKs matter to industry-folk (Prelude)

Suppose you have

- Program P
- Input I

and you want to know Output $O := P(I)$

you generally have two options

✱ Run P(I) yourself

Pro: Confident about the answer, 100% accuracy

Con: Expensive 100% cost

✱✱ Ask someone else to run for you

Pro: Free 0% cost

Con: No guarantee on output integrity 0% accuracy

SNARKs give us an almost perfect best of both worlds

✱✱✱ Ask someone else to run for you AND give you a SNARK proof

Pro: Close to free < 0.1 % cost

Pro: Great guarantee on output integrity > 99.99...9 % accuracy

SNARKs enable verifiable computation, particularly great when you have asymmetric computational power

Main Use Case Today

- Blockchains — Scaling Throughput

Ethereum is a very slow computer; natively can validate 15 transactions per second

BUT... it can validate a SNARK proof of 10000 transactions very quickly

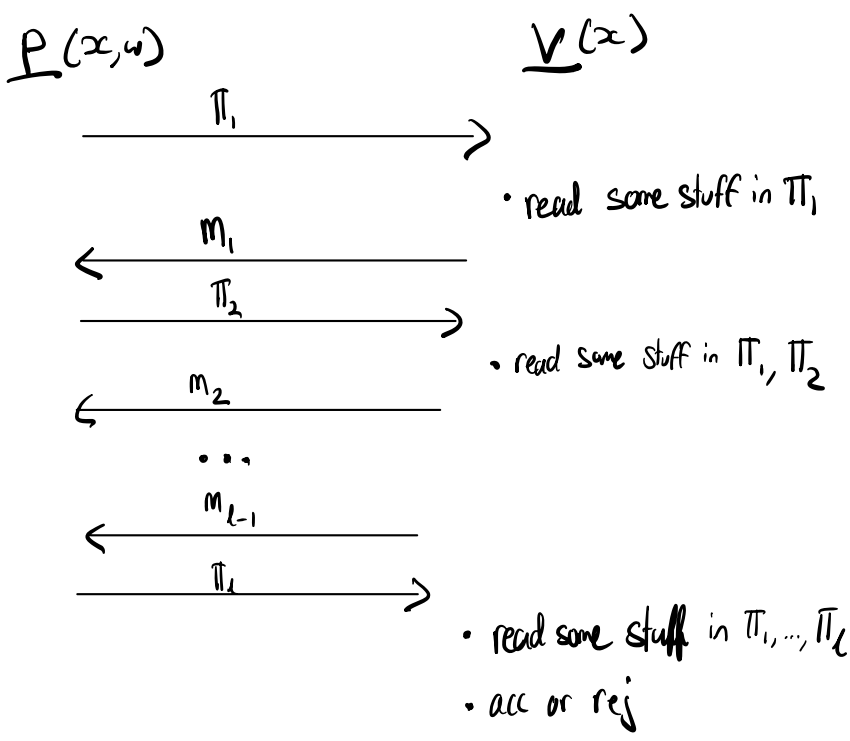
(Many other uses, but not our focus)

① Overview of Modern SNARK Construction

Key Tools:

- "public-coin" Interactive Oracle Proof (IOP)
- Commitment Scheme
- Hash function

What is an IOP?



pub-coin
means $m_1, \ldots, m_{\ell-1}$ are random and V has no private randomness

Notes: $\pi_1, \ldots, \pi_\ell$ are oracles

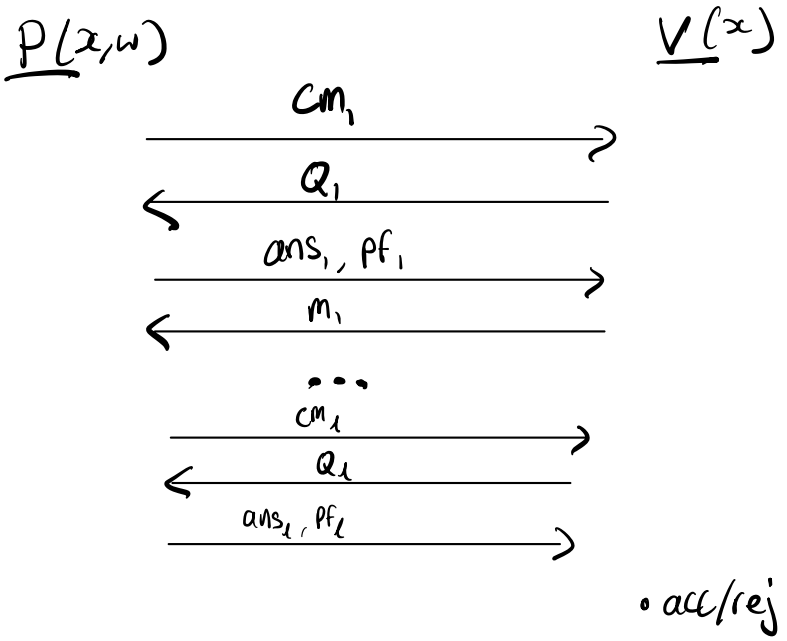
Much like PCPs, we can replace π_i with an appropriate commitment c_i and answer verifier queries with opening proofs

Doing so, we compile an IOP into a succinct interactive proof
↑
if V isn't querying everywhere

IOP is good if:

- complete
- (knowledge)-sound
- succinct
- few rounds
- good prover efficiency
- good verifier efficiency

IOP + Commitment Scheme $\Rightarrow$ IP



Note: different types of IOPs

- Simplest to think of $\Pi_1, \ldots, \Pi_\ell$ as vectors (so use VCS)
- In practice, we often want to use $\Pi_1, \ldots, \Pi_\ell$ as polynomials (so use PCS)

Back to Modern SNARK constructions...

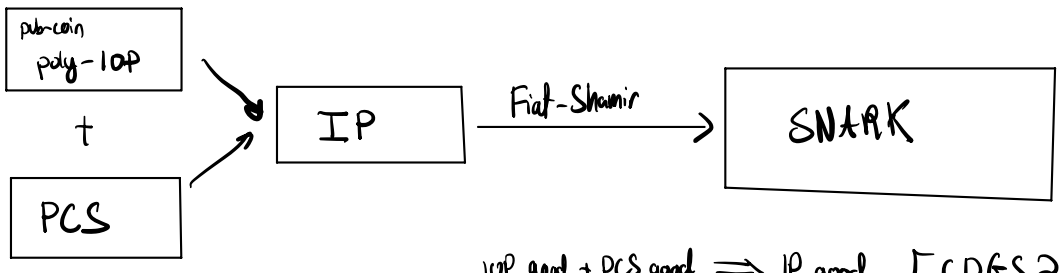
- Pick some NP-complete relation R (usually R1CS or "Plonkish" relation)
- Construct a Poly-IOP for R (oracles are polynomials)
- Compile to IP with appropriate PCS

Front-End:

- Reduce arbitary computation to an R-instance

Back-End:

[BCS16 transformation]



IOP good + PCS good $\Rightarrow$ IP good [CDGS23]

(2) Mathematical Preliminaries:

(A) Multilinear Polynomials

Defn: Multilinear polynomial is linear in each variable

ex ✓ $f(x_0, x_1, x_2) = x_0 + x_1x_2 + x_0x_1 + x_0x_1x_2$

ex ✗ $f(x_0, x_1, x_2) = x_0 + x_0^2x_1 + x_1x_2$ $(\deg_{x_0} f = 2 > 1)$

Fact: A multilinear polynomial on n vars can be uniquely specified by its evaluations on the n-dimensional boolean hypercube

Given $f: \{0,1\}^n \to \mathbb{F}$, $\exists!\ \tilde{f} \in \mathbb{F}[X_0, \dots, X_{n-1}]^{\leq 1}$ s.t.

$$\forall x \in \{0,1\}^n,\quad f(x) = \tilde{f}(x)$$

Defn: We call $\tilde{f}$ the multilinear extension of f

ex Sps $f(0,0) = a_{0,0}$, $f(1,0) = a_{1,0}$, $f(0,1) = a_{0,1}$, $f(1,1) = a_{1,1}$

Then $\tilde{f}(x_0, x_1) = a_{0,0} L_{0,0}(x_0, x_1) + a_{1,0} L_{1,0}(x_0, x_1) + a_{0,1} L_{0,1}(x_0, x_1) + a_{1,1} L_{1,1}(x_0, x_1)$

$$L_{b_0,b_1}(x_0, x_1) = \begin{cases} 1, & \text{if } (x_0, x_1) = (b_0, b_1) \\ 0, & \text{if } (x_0, x_1) \in \{0,1\}^2 \setminus \{(b_0, b_1)\} \end{cases}$$

$L_{0,0}(x_0, x_1) = (1-x_0)(1-x_1)$

$L_{1,0}(x_0, x_1) = x_0(1-x_1)$

$L_{0,1}(x_0, x_1) = (1-x_0)x_1$

$L_{1,1}(x_0, x_1) = x_0x_1$

$\exists$ 4 Lagrange Basis Polynomials.

$\forall r_0, r_1 \in \mathbb{F}^2$

evaluation: $$\tilde{f}(r_0, r_1) = \sum_{b_0, b_1 \in \{0,1\}} a_{b_0,b_1} L_{b_0,b_1}(r_0, r_1)$$

a_{b_0,b_1}: Lagrange coeff; L_{b_0,b_1}: Lagrange Basis Polynomial.

Why low degree extension useful? Short answer: amplifies small differences btwn $\vec{a}$ vectors

Long answer: Stay tuned

(B) Towers of Binary Fields

[Credit: Ben Diamond slides]

- Recall all finite fields of same size are isomorphic
  - BUT we still want to choose our specific fields carefully.
    ↳ affects efficiency!

We choose a sequence of fields $\mathbb{F}_2 \subset \mathbb{F}_4 \subset \mathbb{F}_{16} \subset \cdots \subset \mathbb{F}_{2^{128}}$

$$= T_0 \subset T_1 \subset T_2 \subset \cdots \subset T_7$$

where $T_i \cong \mathbb{F}_{2^{2^i}}$ and can be represented as a 2^i bit-string

Formally:

Wiedemann Tower [1987]

$$T_0 := \mathbb{F}_2$$

$$T_1 := T_0[X_0] / (X_0^2 + X_0 + 1) \cong \mathbb{F}_4$$

$$T_2 := T_1[X_1] / (X_1^2 + X_0 X_1 + 1) \cong \mathbb{F}_{16}$$

$$T_3 := T_2[X_2] / (X_2^2 + X_1 X_2 + 1) \cong \mathbb{F}_{256}$$

$\cdots$

$$T_{i+1} := T_i[X_i] / (X_i^2 + X_{i-1} X_i + 1)$$

$X_i^2 + X_i X_{i-1} + 1$ always irreducible

Fan-Paar 1997: efficient multiplication and inversion in Wiedemann's Tower

- $O(n^{1.59})$ where $n = 2^i$ the bit-width (Karatsuba-like)

Why not use classical approach: $\mathbb{F}_{2^{128}} \cong \mathbb{F}_2[X] / (X^{128} + \cdots)$

- Support multiple data types (data widths) in arithmetization
- RS codes alphabets are intermediate size $\approx \mathbb{F}_{2^{16}}$
- Intermediate fields need to be efficient
  - efficiently "packable": assemble 16 $\mathbb{F}_2$ values into 1 $\mathbb{F}_{2^{16}}$ value
  - efficient "small" by "large" mult: $\mathbb{F}_{2^{16}} \times \mathbb{F}_{2^{128}}$ mult FASTER than $\mathbb{F}_{2^{128}} \times \mathbb{F}_{2^{128}}$
  - efficiently "unpackable"

optimal in tower case
costs in classical case

$\mathbb{F}_{2^{16}}$ always a 16-dimensional $\mathbb{F}_2$-vector space but identifiable?

- In classical approach you see 128 bits for some $\mathbb{F}_{2^{128}}$ element

$$[1011001 \cdots 1] \cdot [X^0, \ldots, X^{127}] \rightarrow \text{some degree 127 poly}$$

is it in subfield $\mathbb{F}_{2^{64}}$? need to run some computation

- In tower approach, you see 128 bits for some $\mathbb{F}_{2^{128}}$ element

$$[1011001000000 \cdots 0] \cdot [1, X_0, X_1, X_0X_1, X_2, X_0X_2, X_1X_2, X_0X_1X_2, X_3, \ldots, X_0X_1X_2X_3X_4X_5X_6]$$

gives some multilinear in 7 variables

- The first 64 bits do not contribute to any monomial terms with X_6
- The last 64 bits all contribute to monomials terms with X_6

Element in $\mathbb{F}_{2^{64}}$ $\longleftrightarrow$ last 64 bits are all 0.

Similarly can pack 2 $\mathbb{F}_{2^{64}}$ values into 1 $\mathbb{F}_{2^{128}}$ value using first/last 64 bits

Generally T_{i+k} is a 2^k-dimensional T_i-vector space with basis elements

$$[1, X_i, X_{i+1}, X_iX_{i+1}, X_{i+2}, \ldots, X_iX_{i+1} \cdots X_{i+k-1}]$$

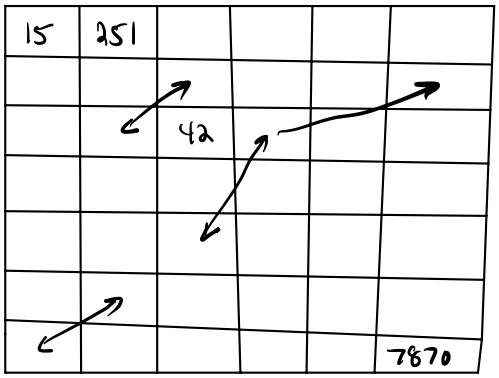
- So, small-by-large mult is 2^k small-by-smalls (good b/c recall same-size mult is super-linear, $O(n^{1.59})$)

cost of 128×128 mult $= 128^{1.59} = 32^{1.59} \cdot 4^{1.59} > 32^{1.59} \cdot 4 =$ cost of 4 32×32 mults $=$ cost of 128×32 mult

(3) Bird's Eye Perspective of the Binius IOP

- Plonkish Arithmetization is our chosen NP-complete relation

Instance:



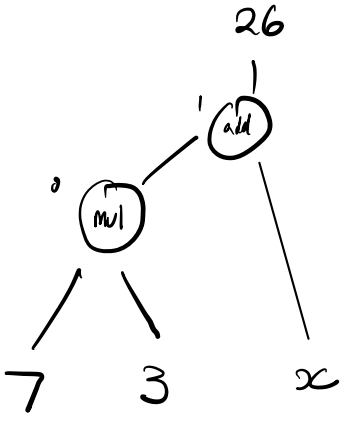
- a partially filled at 2D table
- some copy constraints ↗ indicates two cells must have same value
- a gate polynomial of arity # cols, g

Witness: a fully filled out table such that

- agrees with partial filling
- copy constraints hold
- $\forall \overrightarrow{row_i}$, $g(\overrightarrow{row_i}) = 0$ // gate constraints hold.

- Can reduce from Arithmetic Circuit-SAT (Circuit sat over some $\mathbb{F}$, not necessarily $\mathbb{F}_2$)
  "even more" NP-Hard (!!)

ex//



eg instance is "find x st $7 \cdot 3 + x = 26$"

↓ (Karp) Reduction

L	R	O	isMul
7	3		1
		26	0

,

$$g(L,R,O,b) = b \cdot (LR - O) + (1-b) \cdot (L + R - O)$$

In practice: most front-ends turn arbitrary computations into some $\mathbb{F}_p$-arithmetic circuit (for large prime p) and then (kind of) apply this generic reduction

This sucks... every cell is treated as an $\mathbb{F}_p$-element

Usually most columns are 1s and 0s

Binius arithmetization ⟶ allows you to choose T_i for each column

Better for many reasons:

i) Witness smaller in memory → bandwidth is bottleneck for Hardware acceleration

ii) Eventually we "commit to each column" → (non-trivial, this work) it is more efficient to commit to n small values than n big values

(A) Zerocheck (for Gate Constraints)

Setting: I want to prove to you that the 10^{th} fibonacci is 34

Instance:

A	A'	A''
0	1	
		34

Gate constraint:

$$g(A, A', A'') = A'' - A' - A$$

want for each row, $g(\text{row}) = 0$

"Copy" Constraints

$$\forall i \quad A[i+2] = A'[i+1] = A''[i]$$

(↗ diagonals are equal)

witness:

row	A	A'	A''
(0,0,0)	0	1	1
(1,0,0)	1	1	2
(0,1,0)	1	2	3
(1,1,0)	2	3	5
(0,0,1)	3	5	8
(1,0,1)	5	8	13
(0,1,1)	8	13	21
(1,1,1)	13	21	34

lsb ... msb

(the witness is also called the trace)

$D := g(A, A', A'')$

0	← $g(0,1,1)$
0	← $g(1,1,2)$
0	← $g(1,2,3)$
0	
0	
0	
0	
0	← $g(13,21,34)$

↑ gate evaluations

$\vec{r} = (r_0, r_1, r_2)$ | $\tilde{A}(\vec{r})$ | $\tilde{A}'(\vec{r})$ | $\tilde{A}''(\vec{r})$

$\tilde{D}(\vec{r})$ ← $g(\tilde{A}(\vec{r}), \tilde{A}'(\vec{r}), \tilde{A}''(\vec{r}))$

Goal: Show that vector D is all zeroes
But that would require Verifier to know full witness (computation trace)

Idea: Pretend $\tilde{D}$ is multilinear polynomial on 3-vars where the

$\vec{v} \in \{0,1\}^3$'th entry $D[\vec{v}]$ is evaluation of $\tilde{D}(\vec{v})$

Then zero-test $\tilde{D}(x_0, x_1, x_2)$ @ random point $\vec{r} = (r_0, r_1, r_2) \in (\mathbb{F}_{2^{128}})^3$

$\tilde{D}(\vec{r}) = 0 \Rightarrow$ whp $D = \vec{0}$

Why fast to evaluate $\tilde{D}(\vec{r})$?

- Poly commitment + opening proofs are succinct and efficiently verifiable
- Prover sends verifier commitments to all columns as multilinear polynomials.
  ↑ (not all, you will see)

Defn: Zerocheck is an IOP for a claim that a specific multivariate polynomial evaluates to 0 over the hypercube

Here, take $h(x_0, x_1, x_2) := g\left(\tilde{A}(x_0, x_1, x_2), \tilde{A}'(x_0, x_1, x_2), \tilde{A}''(x_0, x_1, x_2)\right)$

P shows V that $\forall \vec{v} \in \{0,1\}^3$, $h(\vec{v}) = 0$

Common mistake:

NOT proving $h(x_0, x_1, x_2) = 0$ (b/c h not necessarily multilinear

eg $g(A, A', A'') = A'' - A'A$

However

$$\tilde{D}(x_0, x_1, x_2) = \sum_{\vec{v} \in \{0,1\}^3} L_{\vec{v}}(x) \cdot \underbrace{h(v)}_{\text{constant}}$$

is the multilinear poly whose hypercube evals, by design, agree with h.

We check that $\tilde{D}(x_0, x_1, x_2) = 0$

I will stop unravelling Zerocheck here.

On YouTube I have 2 uploaded videos about

- zerocheck and zerocheck → sumcheck reduction
- the sumcheck protocol

that are presented in the context of Binius. I encourage you to check these out

OK... so we have a way to prove gate constraints.

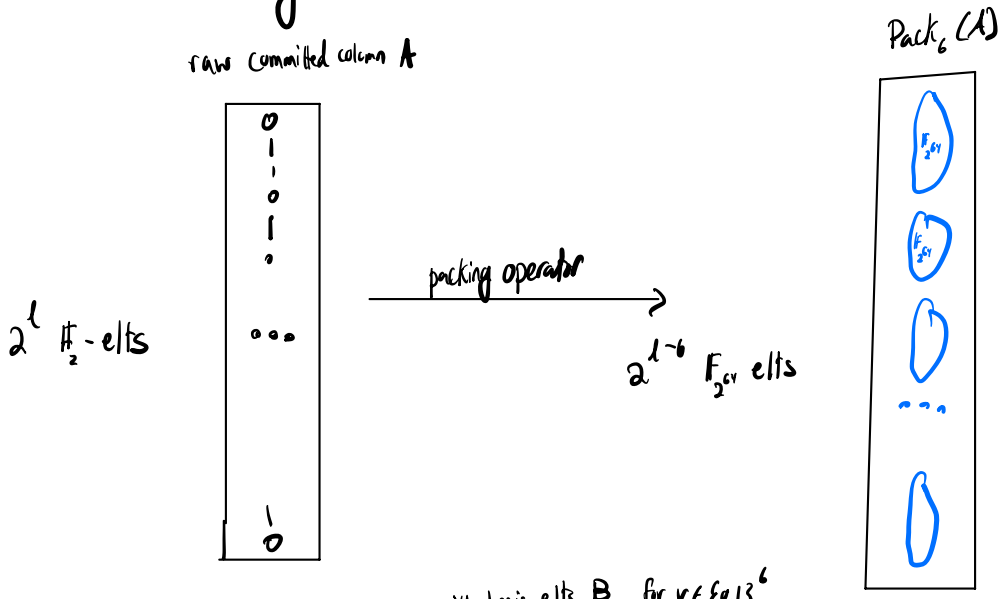
How about copy constraints?

Binius doesn't do copy constraints exactly...

(B) Virtual Multilinears:

Roughly a virtual multilinear is a way to manipulate / "reshape" committed multilinear in some useful way. We highlight two

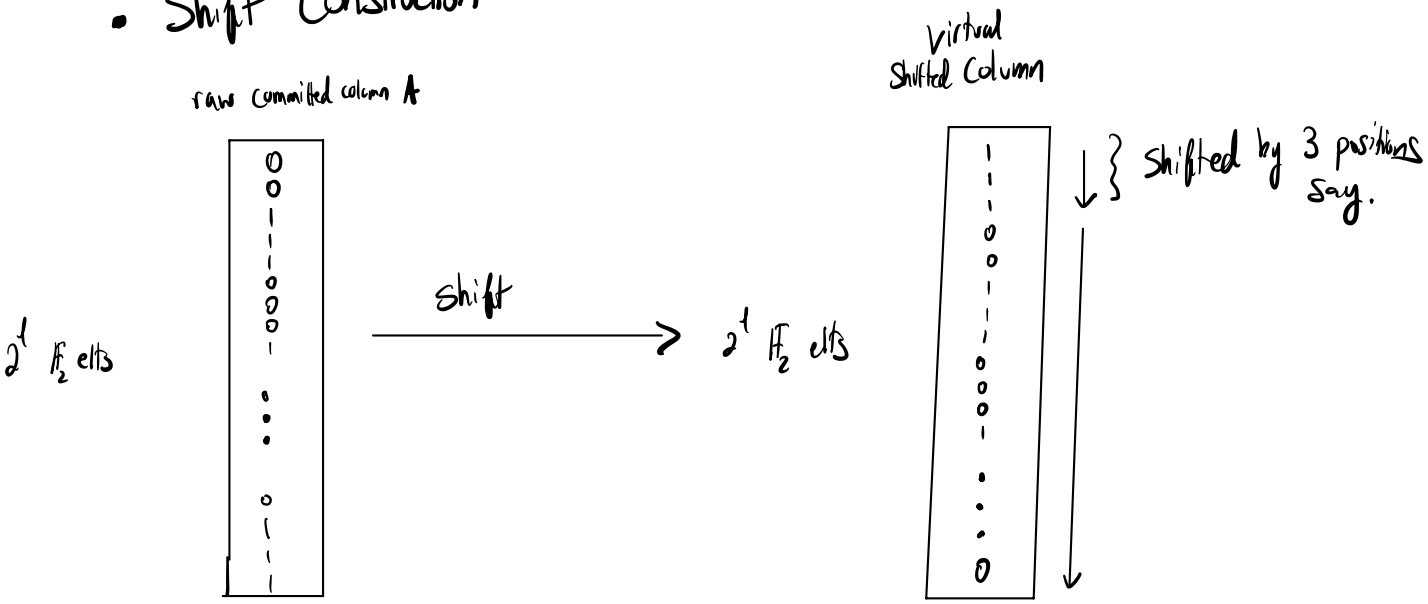
• Packing Construction



Recall $\mathbb{F}_{2^{64}}$ is a 64-dim $\mathbb{F}_2$-vectorspace, with basis elts β_v for $v \in \{0,1\}^6$

$$\widetilde{\text{pack}}_6(A)(X_0,\dots,X_{\ell-7}) = \sum_{\vec{v}\in\{0,1\}^6} \tilde{A}(\vec{v}, X_0, X_1, \dots, X_{\ell-7}) \cdot \beta_{\vec{v}}$$

• Shift Construction



(Can be circular or zero-fill, can be blockwise)

can ignore

$$\text{Shift}_{b,0}(A)(X_0,\dots,X_{\ell-1}) = \sum_{\vec{v}\in\{0,1\}^b} \tilde{A}(\vec{v}, X_b,\dots,X_{\ell-1}) \cdot \widetilde{\text{s-ind}}(\vec{v}, X_0,\dots,X_{b-1})$$

✡ Shift lets us Compare cols without committing to something new.

No need for copy-constraints in Fib example!

Commit to A and let A', A'' be virtual shifts of A

Problem 1: Shift doesn't work? Fine modify like so

A	$\text{Shift}_1(A)$	$\text{Shift}_2(A)$	B
0	1	1	1
1	1	2	1
1	2	3	1
2	3	5	1
3	5	8	1
5	8	13	1
8	13	21	1
13	21	34	1
21	34	0	0
34	0	0	0

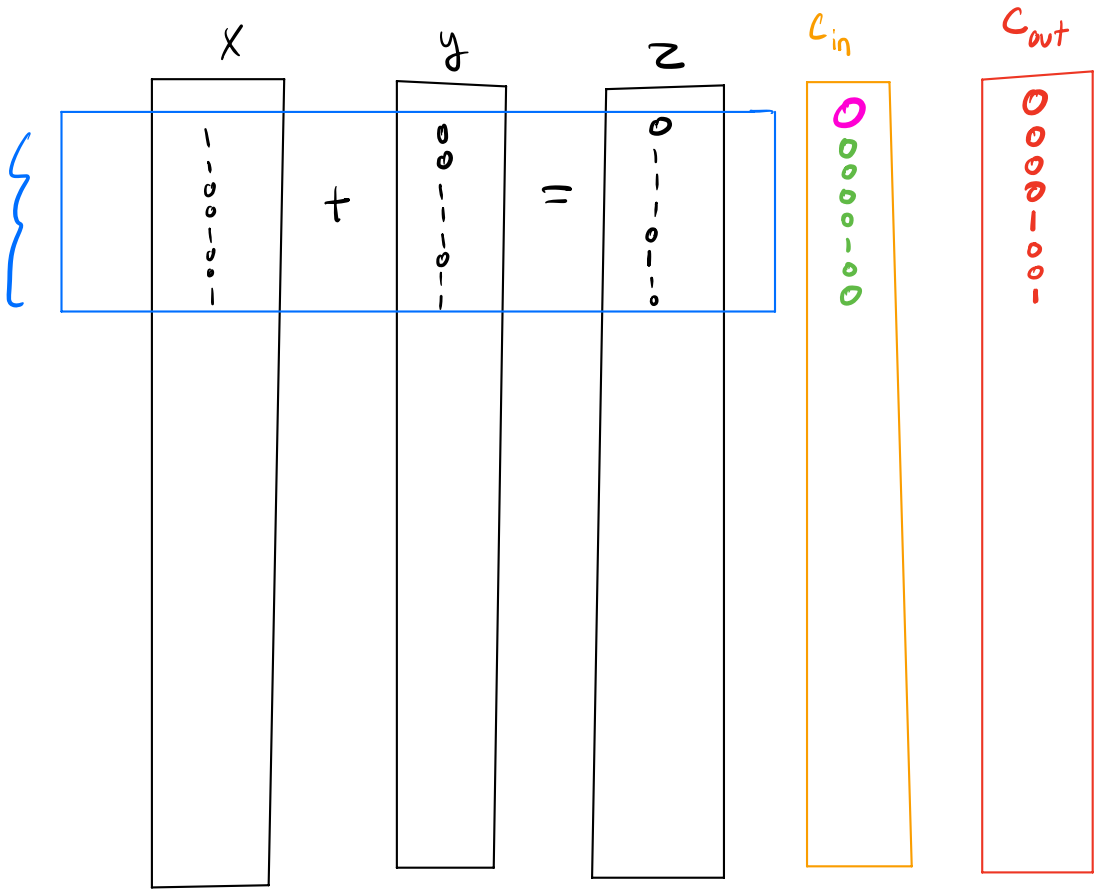
$$g(A, A', A'', B) = B \cdot (A'' - A' - A)$$

- highlighted values are instance
- commit to A only

Problem 2: Subtraction is happening in binary field (just XOR of bits)

no good!

ⓒ Add Gadget Example — Just a few $\mathbb{F}_2$ operations

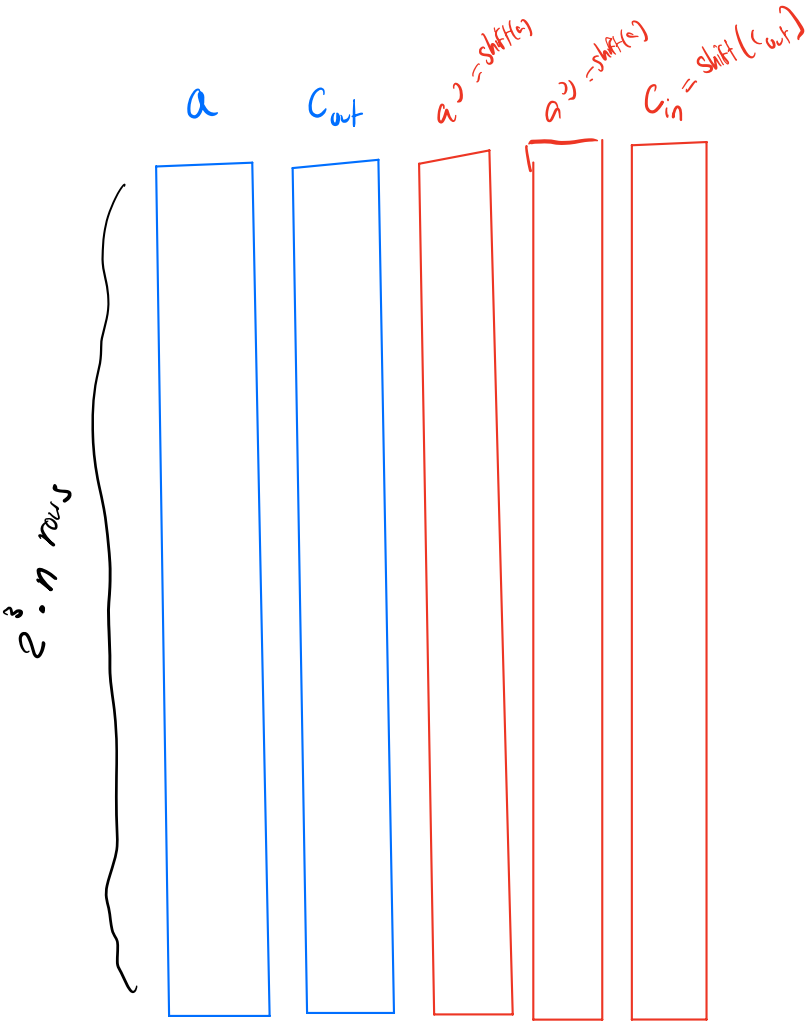


- commit c_{out}
- Declare c_{in} = shift-down of c_{out} by 1 position, with zero-fill, block-size $2^3 = 8$
- Constrain:
  - $z = x + y + c_{in}$ (over $\mathbb{F}_2$)
  - $c_{out} = xy + xc_{in} + yc_{in}$ (over $\mathbb{F}_2$) $c_{out} = 1$ iff ≥ 2 of $x, y, c_{in} = 1$

(multiple constraints ok, batch into single constraint with random linear combination)

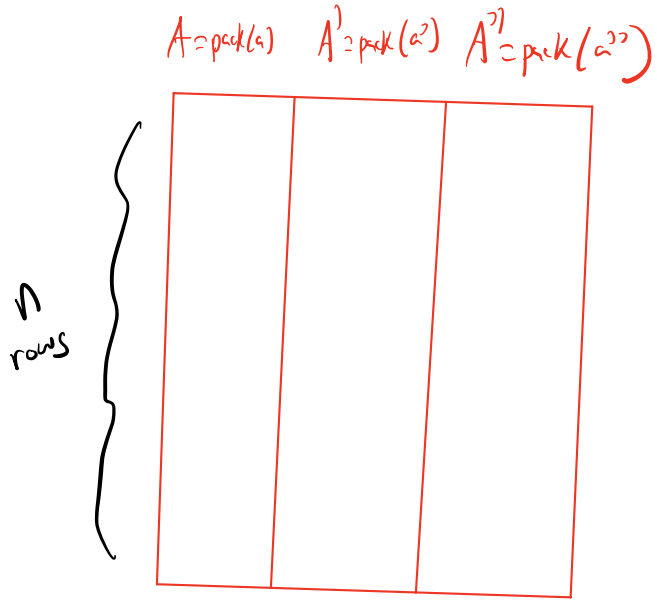
Just 2 constraints, shift, and extra column.

What does fibonacci look like now? (Ignore Problem 1)



↑ gate constraints on this trace

$8n$ $\mathbb{F}_2$ elements



↑ theoretically, one could put gate constraints on this trace too

n $\mathbb{F}_{2^{2^3}} = \mathbb{F}_{2^8} = \mathbb{F}_{256}$ elements

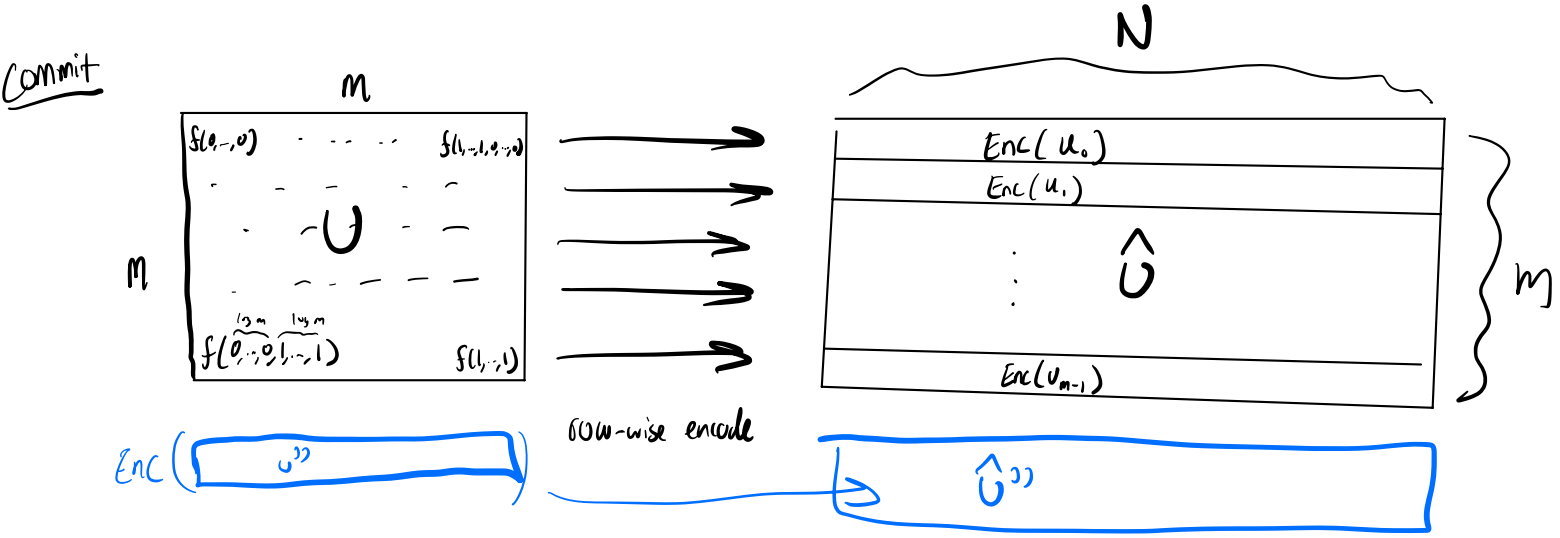
Commit to a and c_{out}

(4) Small-Field Multilinear Poly Commit Scheme

Let's recall the Brakedown PCS (Simplified)

- Let C be systematic linear code over $\mathbb{F}_q$ with constant rate and relative distance
- Let $m = \sqrt{n}$ (for simplicity say n power of 2)
- Let $f(x_0, \ldots, x_{\ell-1})$ be multilinear to commit where $\ell = \log n$

Commit



Eval

To eval $f(r)$ for some $r \in \mathbb{F}^{\log n}$:

- Let $q_1, q_2 \in \mathbb{F}_q^m$ st $q_1 = \bigotimes_{i=0}^{\log m - 1} [1 - r_i, r_i]$

$$q_2 = \bigotimes_{i=\log m}^{\log n - 1} [1 - r_i, r_i]$$

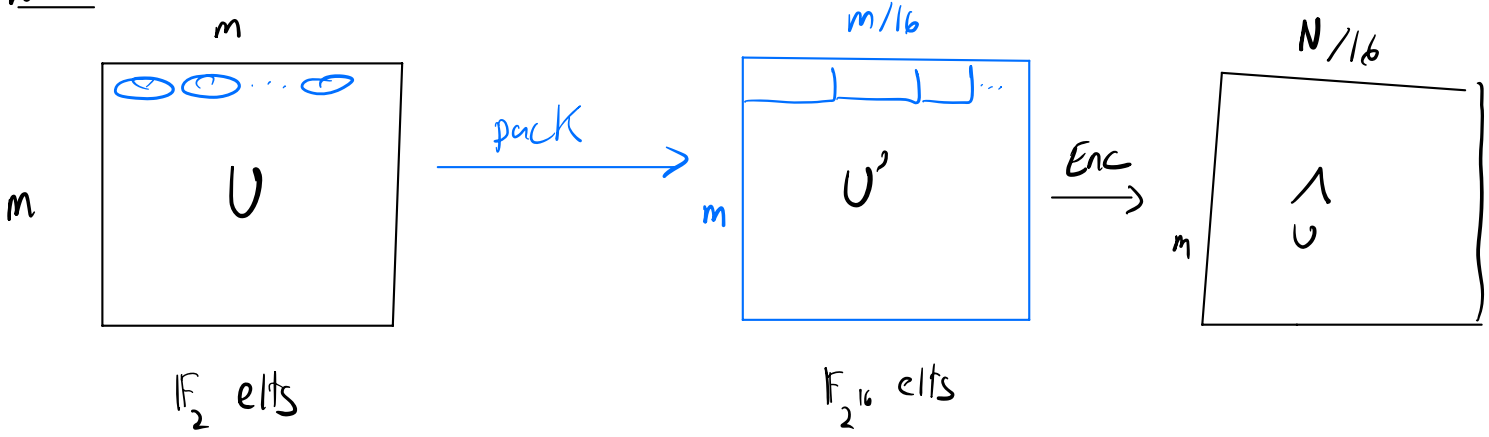
- P sends u'' to V claimed to be q_1-linear combination of U rows
- V checks consistency btwn $\mathrm{Enc}(u'')$ and q_1-linear combination of $\hat{U}$ (at $O(\lambda)$ columns only because code has good distance)
- V computes $f(r) = \langle u'', q_2 \rangle$

If $f \in \mathbb{F}_{2^{16}}[X_0, \ldots, X_{\ell-1}]^{\leq 1}$, then use normal Brakedown

what if $f \in \mathbb{F}_2[X_0, \ldots, X_{\ell-1}]$?

- Do not want 16x blowup in size
- Maybe also wanted to <u>constrain</u> all values are in $\mathbb{F}_2$

<u>Idea:</u>



Commitment is $\hat{U}$ (merkle root)

To eval @ $r = (r_0, \ldots, r_{\ell-1}) \in (\mathbb{F}_{2^{128}})^{\ell}$

- P sends $u'' = q_1$ - linear combination of U (unpacked)
- V checks consistency btwn Enc(pack (u'')) and q_1-l.c. of $\hat{U}$'s
  at $O(\lambda)$ column blocks
- V computes $f(r) = \langle u'', q_2 \rangle$

Okay, I can pack $\mathbb{F}_2$ elements into $\mathbb{F}_{2^{16}}$ elements, easy

How on earth do I pack $\mathbb{F}_{2^{128}}$ elements into $\mathbb{F}_{2^{16}}$ elements ?!?

We don't actually use RS codes over the $\mathbb{F}_{2^{16}}$ alphabet,

our symbols are actually "tower algebra" elements :)

Formally

$$A_{\iota,\kappa,\tau} := T_\tau[Y_0, \ldots, Y_{\kappa-1}] \Big/ \left(Y_0^2 + X_{\iota-1}Y_0 + 1,\ Y_0^2 + Y_0Y_1 + 1,\ \ldots,\ Y_{\kappa-1}^2 + Y_{\kappa-2}Y_{\kappa-1} + 1\right)$$

In Picture:

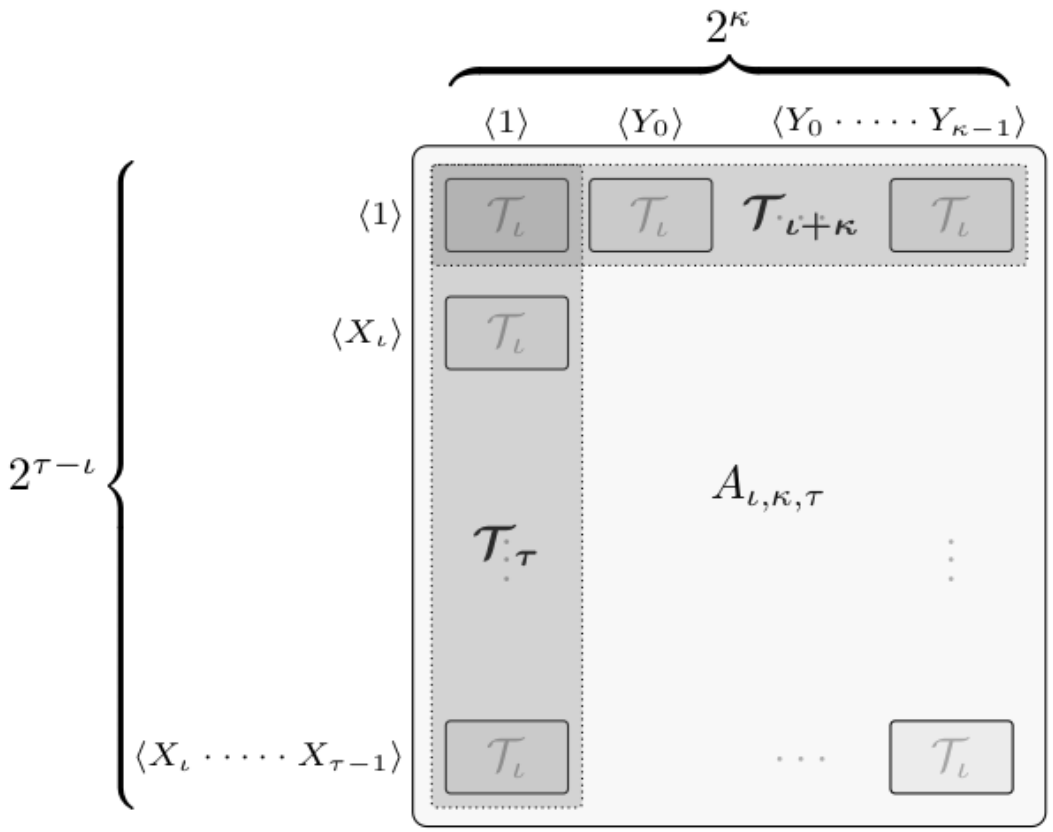


Figure 1: A depiction of our "tower algebra" data structure.

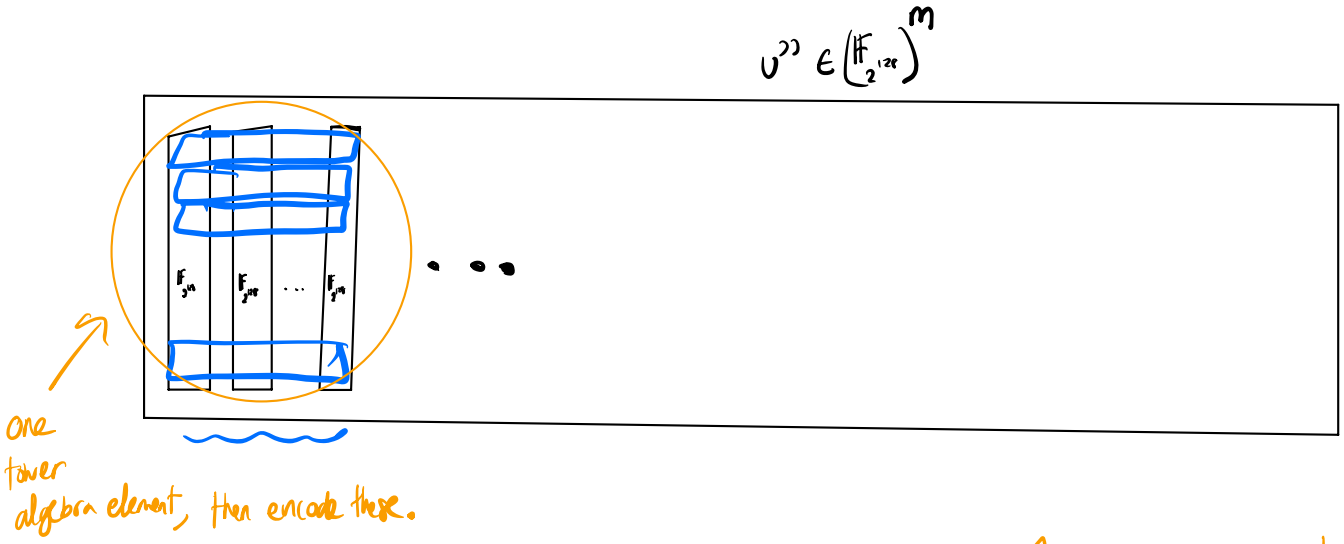
- So bits get packed into what is essentially 16-bit elements
- but big field elts get packed into a fully general $A_{\iota,\kappa,\tau}$ element

Can do

- $T_\tau \times A_{\iota,\kappa,\tau}$

and - $A_{\iota,\kappa,\tau} \times T_{\iota+\kappa}$

So how do we check consistency?



Compare $Enc(embed(U''))$ with q_1-l.c. of $\hat{U}$ rows at random cols.

q_1 elts are $\mathbb{F}_{2^{128}} = T_\tau$, $\hat{U}$ symbols are $A_{L,K,\tau}$ elts

so can left-multiply and everything type checks

Of note: Need a new proximity gap result for optimal efficiency

TLDR the binding holds b/c of a statement that says

if one row of $\hat{U}$ is δ-far from code

then random l.c. of $\hat{U}$ rows is δ-far from code whp

(Proximity Gap survey would need a whole new talk)

Want δ high so we can do fewer column-consistency checks to achieve λ bits of security

$\Rightarrow$ smaller proof sizes :)