Interactive Oracle Proofs of Proximity for Algebraic Codes

Sarah Bordage

Ecole Polytechnique, Institut Polytechnique de Paris / LIX & Inria Saclay

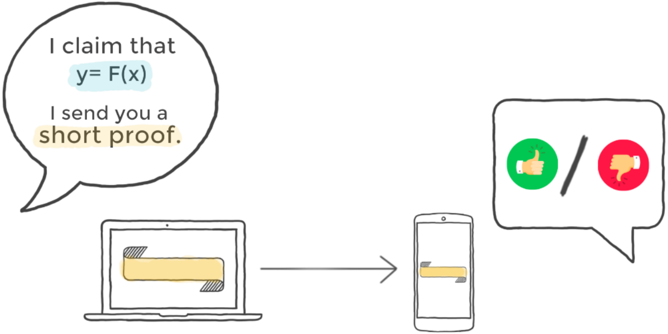
Based on joint work with Daniel Augot and Jade Nardi

May 18, 2022

- Motivations and context
- Local testers and proofs of proximity
- IOP of Proximity for Reed-Solomon codes: the FRI protocol
- IOP of Proximity for multivariate codes

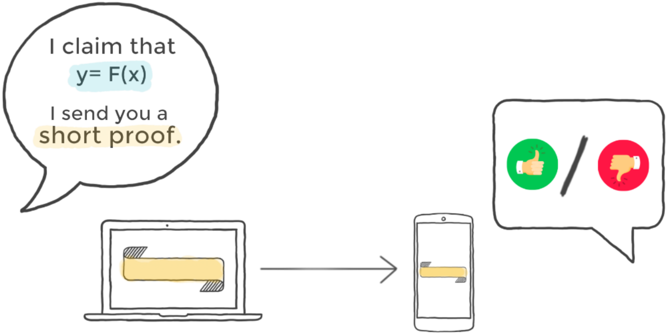
Motivations and context

Verifiable computing



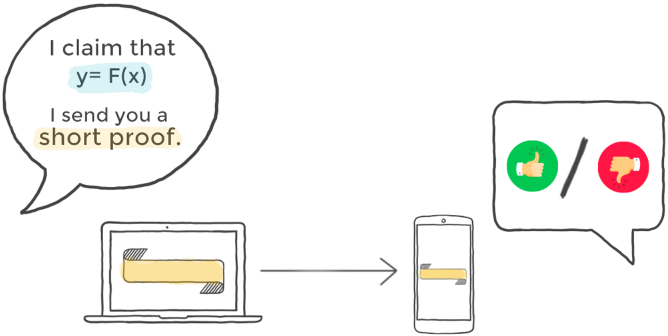
- **Completeness**: Verifier $\mathcal{V}$ always accepts valid proof of correct statement
- **Soundness**: Cheating prover $\widetilde{\mathcal{P}}$ is unable to produce convincing proof of false statement

Verifiable computing



- **Completeness**: Verifier $\mathcal{V}$ always accepts valid proof of correct statement
- **Soundness**: Cheating prover $\widetilde{\mathcal{P}}$ is unable to produce convincing proof of false statement
- **Additional requirements**: zero-knowledge, proof of knowledge ("ZK-SNARK")

  "short" proofs, "fast" proof generation, "fast" verification



- **Completeness**: Verifier $\mathcal{V}$ always accepts valid proof of correct statement
- **Soundness**: Cheating prover $\widetilde{\mathcal{P}}$ is unable to produce convincing proof of false statement
- **Additional requirements**: zero-knowledge, proof of knowledge ("ZK-SNARK")

"short" proofs, "fast" proof generation, "fast" verification

NP class

NP is the set of languages $\mathcal{L}$ for which the instances $x \in \mathcal{L}$ have membership proofs w verifiable in poly-time by a **deterministic** Turing machine.

NP class

NP is the set of languages $\mathcal{L}$ for which the instances $x \in \mathcal{L}$ have membership proofs w verifiable in poly-time by a **deterministic** Turing machine.

If we tolerate a margin of error, can we inspect only a small portion of a proof ?

Genesis: The PCP Theorem

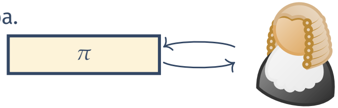
NP class

NP is the set of languages $\mathcal{L}$ for which the instances $x \in \mathcal{L}$ have membership proofs w verifiable in poly-time by a **deterministic** Turing machine.

If we tolerate a margin of error, can we inspect only a small portion of a proof ?

Let $\mathcal{R}$ be a **NP** relation, $\mathcal{L}(\mathcal{R}) := \{x \mid \exists w, (x, w) \in \mathcal{R}\}$.

- **Probabilistic verifier** $\mathcal{V}$ has input x and *oracle access* to a **probabilistically checkable proof** (PCP) π.
- **Completeness**: If $(x, w) \in \mathcal{R}$, then $\mathcal{V}^{\pi}(x)$ accepts with probability 1.
- **Soundness**: If $x \notin \mathcal{L}(\mathcal{R})$, then for all $\tilde{\pi}$, $\mathcal{V}^{\tilde{\pi}}(x)$ accepts with small proba.



→ Encoding of witnesses so that any PCP of a false statement has **errors almost everywhere**.

Probabilistically checkable proofs are **locally testable proofs**.

Checking computations in polylogarithmic time

PCP Theorem [..., AS92, ALMSS98, ...]

Every problem in **NP** has **polynomial-size** probabilistically checkable proofs verifiable by reading a **constant number of bits**.

[Kilian92, Micali95]

Based on the PCP theorem: there are **polylogarithmic-size non-interactive arguments** for **NP** (in the ROM).

Notable application of probabilistic proof systems (PCPs, IPs, and variants):
super fast verification of long computations.

Encoding computations and the role of proximity testing

- **Arithmetization**: Reduce **computational problem** (captured by relation $\mathcal{R}$) to an **algebraic problem** involving **low-degree polynomials** over $\mathbb{F}$ so that:

  $(x, w) \in \mathcal{R} \iff$ some polynomials (related to w) satisfy some polynomial equations $(\star)$ (related to x).

Encoding computations and the role of proximity testing

- **Arithmetization**: Reduce **computational problem** (captured by relation $\mathcal{R}$) to an **algebraic problem** involving **low-degree polynomials** over $\mathbb{F}$ so that:

$$(\mathsf{x}, \mathsf{w}) \in \mathcal{R} \iff \text{some polynomials (related to } \mathsf{w}\text{) satisfy some polynomial equations } (\star) \text{ (related to } \mathsf{x}\text{).}$$

- On input (x, w), Prover $\mathcal{P}$ computes a PCP π for the statement "$(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$".

  Roughly speaking, π is an encoding of w using low-degree polynomial functions.

Encoding computations and the role of proximity testing

- **Arithmetization**: Reduce **computational problem** (captured by relation $\mathcal{R}$) to an **algebraic problem** involving **low-degree polynomials** over $\mathbb{F}$ so that:

  $(\mathrm{x}, \mathrm{w}) \in \mathcal{R} \iff$ some polynomials (related to w) satisfy some polynomial equations $(\star)$ (related to x).

- On input (x, w), Prover $\mathcal{P}$ computes a PCP π for the statement "$(\mathrm{x}, \mathrm{w}) \in \mathcal{R}$".

  Roughly speaking, π is an encoding of w using low-degree polynomial functions.

- Prover $\mathcal{P}$ commits to π.

Encoding computations and the role of proximity testing

- **Arithmetization**: Reduce **computational problem** (captured by relation $\mathcal{R}$) to an **algebraic problem** involving **low-degree polynomials** over $\mathbb{F}$ so that:

  $(\mathsf{x}, \mathsf{w}) \in \mathcal{R} \iff$ some polynomials (related to w) satisfy some polynomial equations $(\star)$ (related to x).

- On input (x, w), Prover $\mathcal{P}$ computes a PCP π for the statement "$(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$".

  Roughly speaking, π is an encoding of w using low-degree polynomial functions.
- Prover $\mathcal{P}$ commits to π.
- Verifier $\mathcal{V}$ asks for certain symbols of π and (probabilistically) checks:

  Consistency test: the message associated to π is consistent with $(\star)$,
  Proximity test: π is close to a certain polynomial code C.

  (Note that $\mathcal{V}$ *does not know* w.)

Encoding computations and the role of proximity testing

- **Arithmetization**: Reduce **computational problem** (captured by relation $\mathcal{R}$) to an **algebraic problem** involving **low-degree polynomials** over $\mathbb{F}$ so that:

$$(\mathrm{x}, \mathrm{w}) \in \mathcal{R} \iff \text{some polynomials (related to w) satisfy some polynomial equations } (\star) \text{ (related to x).}$$

- On input (x, w), Prover $\mathcal{P}$ computes a PCP π for the statement "$(\mathrm{x}, \mathrm{w}) \in \mathcal{R}$".

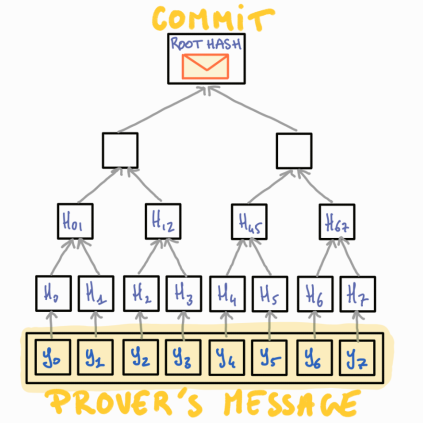
  Roughly speaking, π is an encoding of w using low-degree polynomial functions.
- Prover $\mathcal{P}$ commits to π.
- Verifier $\mathcal{V}$ asks for certain symbols of π and (probabilistically) checks:

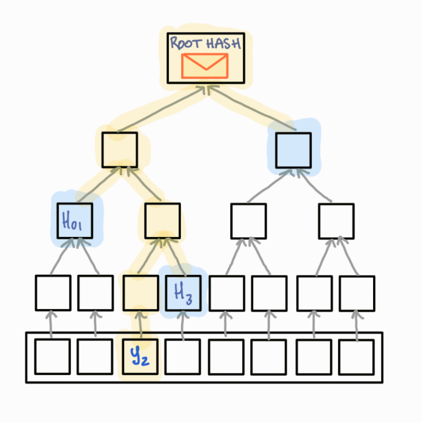
  Consistency test: the message associated to π is consistent with $(\star)$,

  Proximity test: π is close to a certain polynomial code C.

  (Note that $\mathcal{V}$ *does not know* w.)

In practice, oracles are replaced by **cryptographic commitments** (Merkle trees)





commit = 1 hash

opening at a single location = log(|oracle|) hashes

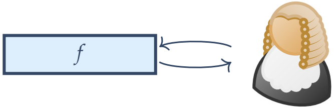
Local testers and proofs of proximity

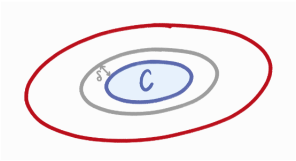
Given some domain D, a (linear) code $C \subseteq \mathbb{F}^D$ is a $\mathbb{F}$-vector space of functions from D to $\mathbb{F}$.

Local tester for a code

A **local tester** for a code $C \subseteq \mathbb{F}_q^D$ is a probabilistic algorithm $\mathcal{V}$ that is given **oracle access** to $f \in \mathbb{F}^D$, samples $Q \subset D$, queries f on Q and satisfies:

Completeness:	if $f \in C$,	$\mathcal{V}^f(C)$ always accepts.
Soundness:	if f is δ-far from C,	$\Pr[\mathcal{V}^f(C) \text{ accepts}] \leq \varepsilon$.





Codes with sublinear local testers are **locally testable codes**.

Multivariate codes

Def Multivariate polynomial codes

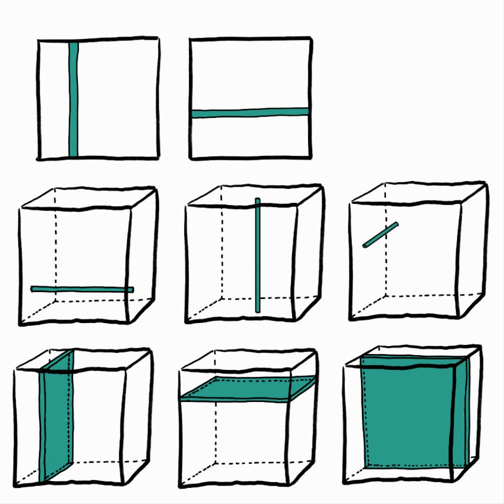
Let $L \subseteq \mathbb{F}$ and $d < |L|$.

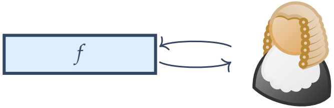
- **Tensor product of RS codes:**

  $\mathsf{RS}[L, d]^{\otimes m} = \{f : L^m \to \mathbb{F} \mid f$ evaluation of a poly in $\mathbb{F}[X_1, \ldots, X_m]$ with individual degrees $< d\}$
- **Reed-Muller codes:**

  $\mathsf{RM}[L, d, m] = \{f : L^m \to \mathbb{F} \mid f$ evaluation of a poly in $\mathbb{F}[X_1, \ldots, X_m]$ of total degree $< d\}$

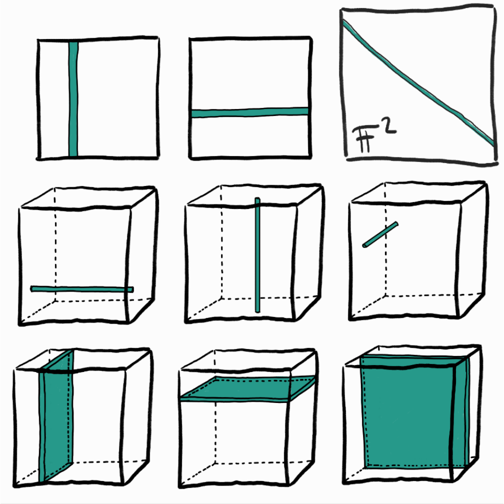
Remark: We consider m-wise tensor products to simplify the presentation.

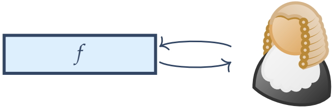




[BFL91, BFLS91, FHS94, PS94, ...]

Individual degree: query $d+1$ points along a random axis-parallel line.

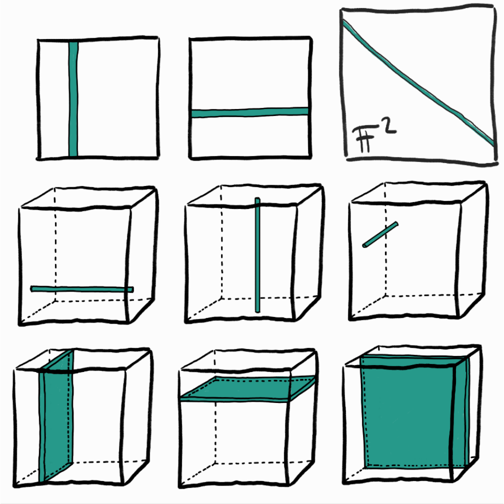


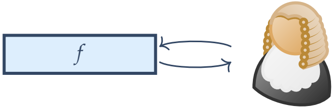


[BFL91, BFLS91, FHS94, PS94, ...]
Individual degree: query $d+1$ points along a random axis-parallel line.

[RS96, AS92, ALMSS98, ...]
Total degree: query $d+1$ points along a random line in $\mathbb{F}^m$.
(Require evaluation domain = $\mathbb{F}^m$)

Multivariate polynomial codes are locally testable



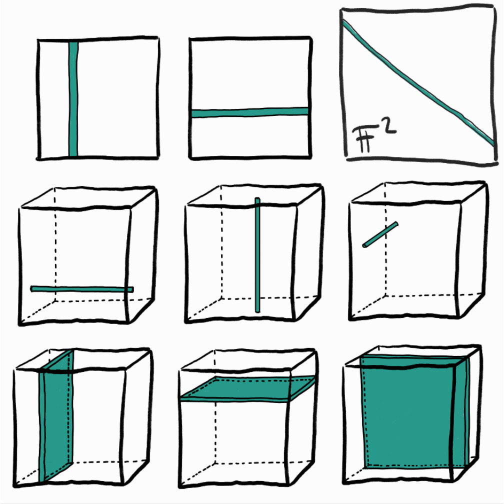


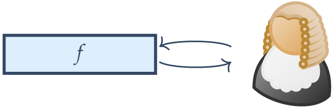
[BFL91, BFLS91, FHS94, PS94, ...]
Individual degree: query $d+1$ points along a random axis-parallel line.

[RS96, AS92, ALMSS98, ...]
Total degree: query $d+1$ points along a random line in $\mathbb{F}^m$.
(Require evaluation domain = $\mathbb{F}^m$)

With only oracle access to f
→ **require at least d queries**

Multivariate polynomial codes are locally testable





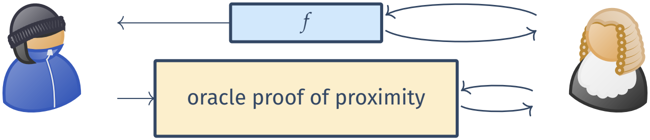
[BFL91, BFLS91, FHS94, PS94, ...]
Individual degree: query $d + 1$ points along a random axis-parallel line.

[RS96, AS92, ALMSS98, ...]
Total degree: query $d + 1$ points along a random line in $\mathbb{F}^m$.
(Require evaluation domain = $\mathbb{F}^m$)

With only oracle access to f
→ **require at least d queries**

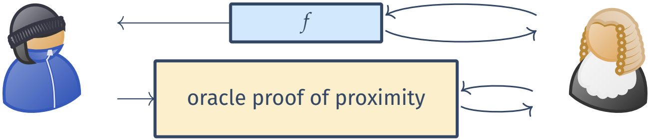
Most works on probabilistic proof systems use multivariate polynomials.

Probabilistically Checkable Proof of Proximity (PCPP):



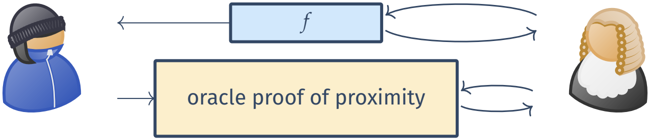
- **Relevant measures**: prover time, verifier time, proof length, query complexity

Probabilistically Checkable Proof of Proximity (PCPP):



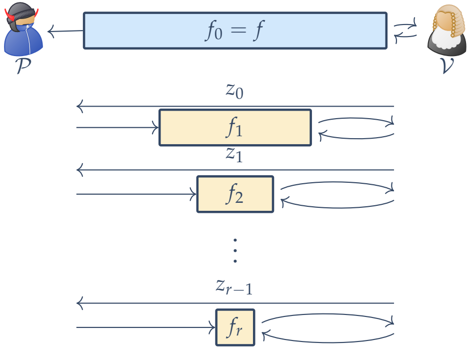
- **Relevant measures**: prover time, verifier time, proof length, query complexity
- For multivariate codes: PCPs of Proximity enable **constant query complexity**, but **prover time is too high** for interesting applications.

Probabilistically Checkable Proof of Proximity (PCPP):



- **Relevant measures**: prover time, verifier time, proof length, query complexity
- For multivariate codes: PCPs of Proximity enable **constant query complexity**, but **prover time is too high** for interesting applications.
- Also enable proximity testing with sublinear query complexity for **non-locally testable codes**

  e.g. Reed-Solomon codes [BS08]



Def IOPP for a Code

An **Interactive Oracle Proof of Proximity (IOPP)** $(\mathcal{P}, \mathcal{V})$ for C with soundness error $s : (0,1] \to [0,1]$ satisfies:

Completeness

If $f \in C$, then $\exists \mathcal{P}\ \Pr[\langle \mathcal{P}(f), \mathcal{V}^f \rangle = 1] = 1$.

Soundness

If f is δ-far from C,
Then, for all unbounded $\widetilde{\mathcal{P}}$, $\Pr[\langle \widetilde{\mathcal{P}}, \mathcal{V}^f \rangle = 1] \leq s(\delta)$.

Relevant measures: prover time, proof length, verifier time, query complexity, round complexity

Def Reed-Solomon code

Given domain $L \subseteq \mathbb{F}$, degree bound $d < |L|$, $\quad RS L, d := \left\{ f_{|L} : L \to \mathbb{F} \mid f \in \mathbb{F}[X], \deg f < d \right\}$.

Without help from a prover: $d + 1$ **queries are necessary** and sufficient.

Def Reed-Solomon code

Given domain $L \subseteq \mathbb{F}$, degree bound $d < |L|$, $\quad RS L, d := \left\{ f_{|L} : L \to \mathbb{F} \mid f \in \mathbb{F}[X], \deg f < d \right\}$.

Without help from a prover: $d+1$ **queries are necessary** and sufficient.
But for applications to probabilistic proofs, $|L| = \Theta(d)$ and $d \approx$ size of the computation to be verified.

Def Reed-Solomon code

Given domain $L \subseteq \mathbb{F}$, degree bound $d < |L|$, $RSL, d := \left\{ f_{|L} : L \to \mathbb{F} \mid f \in \mathbb{F}[X], \deg f < d \right\}$.

Without help from a prover: $d + 1$ **queries are necessary** and sufficient.
But for applications to probabilistic proofs, $|L| = \Theta(d)$ and $d \approx$ size of the computation to be verified.

FRI protocol [BBHR18]

IOP of Proximity for RS[L, d] where L is a subgroup of $(\mathbb{F}, +)$ or $(\mathbb{F}^\times, \times)$ of large smooth order with

- **logarithmic query complexity,** (with respect to $|L|$)
- **logarithmic verifier,**
- **linear prover.**

The FRI protocol is a crucial building-block of some proof systems deployed in the real-world with **post-quantum security** and **no trusted setup** ("Stark" proofs [BBHR19]).

	Code	Prover	Verifier	Query	Length	Rounds
[BBHR18]	RS	$< 8N$	$< 8 \log N$	$< 2 \log N$	$< N$	$< \log N$
[ABN21]	$\text{RS}^{\otimes m}$	$< 8N$	$< 8 \log N$	$< 2 \log N$	$< N$	$< \log N$
[ABN21]	RM	$< (2m+7)N$	$< 2^m \left(\frac{5}{4} + \frac{7}{m}\right) \log N$	$< \frac{2^m}{m} \log N$	$< \frac{N}{2^m - 1}$	$< \frac{\log N}{m}$

Inspired from the **FRI** protocol, we can construct **interactive oracle proofs of proximity** (IOPP) for multivariate polynomial codes that are **fast to generate** and **exponentially faster to verify**.

Block length is N, number of variables is m.

Complexities counted in $\mathbb{F}$-ops and field elements.

Remark: regarding SNARKs applications, constant rate codes ➞ shorter proofs ($m =$ constant)

IOP of Proximity for Reed-Solomon codes: the FRI protocol

Halve the size of the problem

Idea: recursively halve the size of the problem via “random folding”.

Assume $\mathbb{F}$ has a multiplicative subgroup L of order 2^n, $\mathrm{char}(\mathbb{F}) \neq 2$.
The square map $q : x \mapsto x^2$ is 2-to-1 from L to $q(L)$.

Halve the size of the problem

Idea: recursively halve the size of the problem via "random folding".

Assume $\mathbb{F}$ has a multiplicative subgroup L of order 2^n, $\text{char}(\mathbb{F}) \neq 2$.
The square map $q : x \mapsto x^2$ is 2-to-1 from L to $q(L)$.

Reduce proximity to RS$[L, d]$ ⟶ proximity to RS$[q(L), d/2]$.

Halve the size of the problem

Idea: recursively halve the size of the problem via "random folding".

Assume $\mathbb{F}$ has a multiplicative subgroup L of order 2^n, char$(\mathbb{F}) \neq 2$.
The square map $q : x \mapsto x^2$ is 2-to-1 from L to $q(L)$.

Reduce proximity to RS$[L, d]$ ⟶ proximity to RS$[q(L), d/2]$.

Given **arbitrary** function $f : L \to \mathbb{F}$,

Halve the size of the problem

Idea: recursively halve the size of the problem via "random folding".

Assume $\mathbb{F}$ has a multiplicative subgroup L of order 2^n, $\mathrm{char}(\mathbb{F}) \neq 2$.
The square map $q : x \mapsto x^2$ is 2-to-1 from L to $q(L)$.

Reduce proximity to RS[L, d] ⟶ proximity to RS[$q(L), d/2$].

Given **arbitrary** function $f : L \to \mathbb{F}$,

- Decompose f into two parts:

$$f(x) = g_0(x^2) + x g_1(x^2) \quad \text{where } \deg g_i \leq \frac{\deg f}{2}.$$

If $\deg f < d$, then
$\deg g_0, \deg g_1 < d/2$.

Halve the size of the problem

Idea: recursively halve the size of the problem via “random folding”.

Assume $\mathbb{F}$ has a multiplicative subgroup L of order 2^n, $\text{char}(\mathbb{F}) \neq 2$.
The square map $q : x \mapsto x^2$ is 2-to-1 from L to $q(L)$.

Reduce proximity to RS$[L, d]$ ⟶ proximity to RS$[q(L), d/2]$.

Given **arbitrary** function $f : L \to \mathbb{F}$,

- Decompose f into two parts:

$$f(x) = g_0(x^2) + x g_1(x^2) \quad \text{where } \deg g_i \leq \frac{\deg f}{2}.$$

- For $z \in \mathbb{F}$, define **Fold** $[f, z] : q(L) \to \mathbb{F}$ by

$$\textbf{Fold}\,[f, z]\,(y) = g_0(y) + z g_1(y).$$

If $\deg f < d$, then
$\deg g_0, \deg g_1 < d/2$.

Halve the size of the problem

Idea: recursively halve the size of the problem via "random folding".

Assume $\mathbb{F}$ has a multiplicative subgroup L of order 2^n, $\mathrm{char}(\mathbb{F}) \neq 2$.
The square map $q : x \mapsto x^2$ is 2-to-1 from L to $q(L)$.

Reduce proximity to RS$[L, d]$ ⟶ proximity to RS$[q(L), d/2]$.

Given **arbitrary** function $f : L \to \mathbb{F}$,

- Decompose f into two parts:

$$f(x) = g_0(x^2) + x g_1(x^2) \quad \text{where } \deg g_i \leq \frac{\deg f}{2}.$$

- For $z \in \mathbb{F}$, define **Fold** $[f, z] : q(L) \to \mathbb{F}$ by

$$\textbf{Fold}\,[f, z]\,(y) = g_0(y) + z g_1(y).$$

If $\deg f < d$, then
$\deg g_0, \deg g_1 < d/2$.

How to compute Fold $[f, z]$?

Any $y \in q(L)$ has 2 distinct square roots $x, -x \in L$.
Linear system $\implies g_0(y) = \frac{f(x)+f(-x)}{2}$ and $g_1(y) = \frac{f(x)-f(-x)}{2x}$.

The 3 key properties

Key properties of folding operators

1. Completeness:

$$f \in \mathsf{RS}[L, d] \implies \textbf{Fold}[f, z] \in \mathsf{RS}[q(L), d/2] \qquad \text{for all } z \in \mathbb{F}.$$

2. Local computability:

Each entry of **Fold** $[f, z]$ depends on only 2 entries of f, and is computable in $O(1)$ field operations.

Key properties of folding operators

1. Completeness:

$f \in \mathsf{RS}[L, d] \implies \mathbf{Fold}[f, z] \in \mathsf{RS}[q(L), d/2]$ for all $z \in \mathbb{F}$.

2. Local computability:

Each entry of **Fold** $[f, z]$ depends on only 2 entries of f, and is computable in $O(1)$ field operations.

3. Distance preservation:

f is far from $\mathsf{RS}[L, d] \implies \mathbf{Fold}[f, z]$ is far from $\mathsf{RS}[q(L), d/2]$ w.h.p. over z.

Distance and random combinations [RVW13, AHIV17, BBHR18, BKS18, BGKS20, BCIKS20]

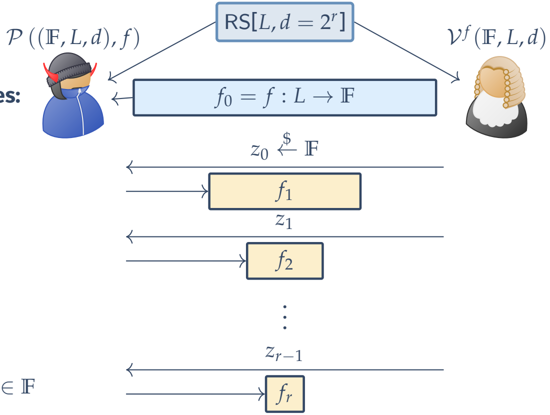
Let $V \subseteq \mathbb{F}^L$ be a linear code, $g_0, g_1 \in \mathbb{F}^L$, and $\delta \in (0, \delta_0)$. (δ_0 const. depends on distance of V)

Assume either g_0 or g_1 is δ-**far** from V.

Then $g_0 + z g_1$ is $\approx \delta$-**far** from V w.h.p. over z.

IOPP: COMMIT phase

Honest prover computes:

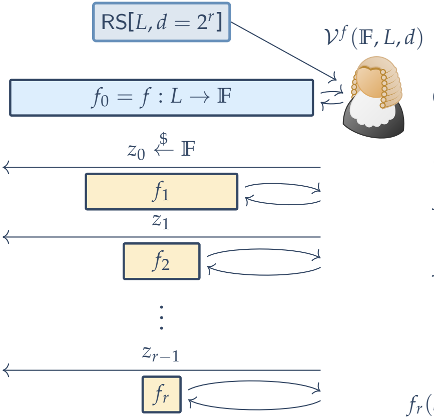


$f_1 = \textbf{Fold}[f_0, z_0]$

$f_2 = \textbf{Fold}[f_1, z_1]$

$\vdots$

$f_r = \textbf{Fold}[f_{r-1}, z_{r-1}] \equiv c \in \mathbb{F}$



Global consistency test:

Sample $s \in L$ and check

$$f_1(s^2) \stackrel{?}{=} \textbf{Fold}\,[f_0, z_0]\,(s^2)$$

$$f_2(s^4) \stackrel{?}{=} \textbf{Fold}\,[f_1, z_1]\,(s^4)$$

$$\vdots$$

$$f_r(s^{2^r}) \stackrel{?}{=} \textbf{Fold}\,[f_{r-1}, z_{r-1}]\,(s^{2^r})$$

Final test: $f_r \stackrel{?}{\equiv} c \in \mathbb{F}$

IOPP: FRI protocol

$\mathsf{RS}[L, d = 2^r]$

$\mathcal{P}\left((\mathbb{F}, L, d), f\right)$ $\mathcal{V}^f\left(\mathbb{F}, L, d\right)$

Honest prover computes:

$f_0 = f : L \to \mathbb{F}$

Global consistency test:

$z_0 \xleftarrow{\$} \mathbb{F}$

Sample $s \in L$ and check

$f_1 = \textbf{Fold}\left[f_0, z_0\right]$ f_1 $f_1(s^2) \stackrel{?}{=} \textbf{Fold}\left[f_0, z_0\right](s^2)$

z_1

$f_2 = \textbf{Fold}\left[f_1, z_1\right]$ f_2 $f_2(s^4) \stackrel{?}{=} \textbf{Fold}\left[f_1, z_1\right](s^4)$

$\vdots$ $\vdots$ $\vdots$

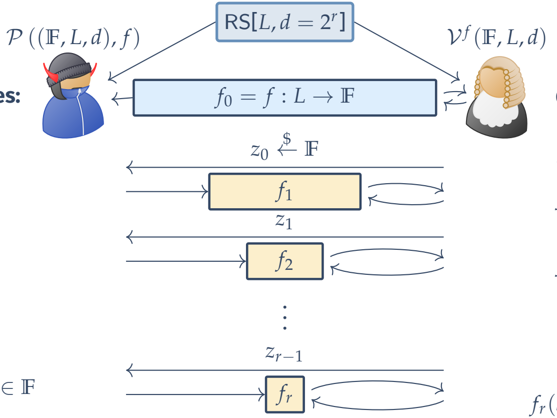
z_{r-1}

$f_r = \textbf{Fold}\left[f_{r-1}, z_{r-1}\right] \equiv c \in \mathbb{F}$ f_r $f_r(s^{2^r}) \stackrel{?}{=} \textbf{Fold}\left[f_{r-1}, z_{r-1}\right](s^{2^r})$

Final test: $f_r \stackrel{?}{\equiv} c \in \mathbb{F}$

Completeness

If $f \in \mathsf{RS}[L, d]$, $\mathcal{V}$ accepts.



Honest prover computes:

$f_1 = \textbf{Fold}[f_0, z_0]$

$f_2 = \textbf{Fold}[f_1, z_1]$

$\vdots$

$f_r = \textbf{Fold}[f_{r-1}, z_{r-1}] \equiv c \in \mathbb{F}$

Global consistency test:

Sample $s \in L$ and check

$f_1(s^2) \stackrel{?}{=} \textbf{Fold}[f_0, z_0](s^2)$

$f_2(s^4) \stackrel{?}{=} \textbf{Fold}[f_1, z_1](s^4)$

$\vdots$

$f_r(s^{2^r}) \stackrel{?}{=} \textbf{Fold}[f_{r-1}, z_{r-1}](s^{2^r})$

Final test: $f_r \stackrel{?}{\equiv} c \in \mathbb{F}$

Efficiency (local computability is key)

Prover running time is linear in $|L|$,
Query complexity and verifier time are logarithmic.

Folding preserves distance to the code

$\Downarrow$

Soundness of FRI [BBHR18, BKS18, BGKS20, BCIKS20]

Let $\varepsilon, \delta > 0$ such that $\varepsilon < \sqrt{\rho}/20$ and $\delta < 1 - \sqrt{\rho} - \varepsilon$. $\left(\rho = \frac{d}{|L|}\right)$

Suppose f is δ-far from RS$[L, d]$. Then, after t repetitions of the QUERY phase,

$$\Pr[\mathcal{V}\text{ accepts}] \leq \underbrace{\frac{d^2}{(2\varepsilon)^7 |\mathbb{F}|}}_{\text{err}_{\text{commit}}} + \underbrace{(1-\delta)^t}_{\text{err}_{\text{query}}}.$$

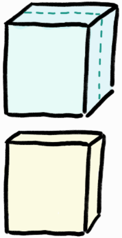
IOPs of Proximity for multivariate codes

Folding tensor product of RS codes

The tensor structure enables to fold along **one dimension at a time**.

The tensor structure enables to fold along **one dimension at a time**.

- Start by folding along the first dimension:

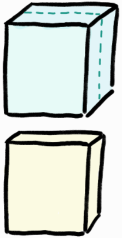


Folding tensor product of RS codes

The tensor structure enables to fold along **one dimension at a time**.

- Start by folding along the first dimension:
  - Write $f : \prod_{i=1}^{m} L_i \to \mathbb{F}$ as

$$f(x_1, x_2, \ldots, x_m) = g_0(x_1^2, x_2, \ldots, x_m) + x_1 g_1(x_1^2, x_2, \ldots, x_m)$$



Folding tensor product of RS codes

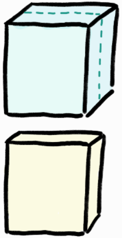
The tensor structure enables to fold along **one dimension at a time**.

- Start by folding along the first dimension: $(q : x \mapsto x^2)$
  - Write $f : \prod_{i=1}^{m} L_i \to \mathbb{F}$ as

$$f(x_1, x_2, \ldots, x_m) = g_0(x_1^2, x_2, \ldots, x_m) + x_1 g_1(x_1^2, x_2, \ldots, x_m)$$

  - For $z \in \mathbb{F}$, define **Fold** $[f, z] : q(L_1) \times \prod_{i=2}^{m} L_i \to \mathbb{F}$ by

$$\textbf{Fold}\,[f, z]\,(y, \boldsymbol{x}) = g_0(y, \boldsymbol{x}) + z g_1(y, \boldsymbol{x})$$



The tensor structure enables to fold along **one dimension at a time**.

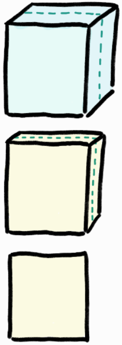
- Start by folding along the first dimension: $(q : x \mapsto x^2)$
  - Write $f : \prod_{i=1}^{m} L_i \to \mathbb{F}$ as

$$f(x_1, x_2, \ldots, x_m) = g_0(x_1^2, x_2, \ldots, x_m) + x_1 g_1(x_1^2, x_2, \ldots, x_m)$$

  - For $z \in \mathbb{F}$, define **Fold** $[f, z] : q(L_1) \times \prod_{i=2}^{m} L_i \to \mathbb{F}$ by

$$\textbf{Fold}\,[f, z]\,(y, \boldsymbol{x}) = g_0(y, \boldsymbol{x}) + z g_1(y, \boldsymbol{x})$$

  - After $\log d$ rounds, expected x_1-degree $= 0$



Folding tensor product of RS codes

The tensor structure enables to fold along **one dimension at a time**.

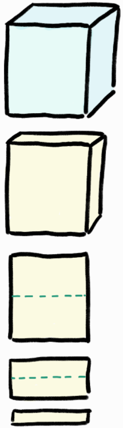
- Start by folding along the first dimension: $(q : x \mapsto x^2)$
  - Write $f : \prod_{i=1}^{m} L_i \to \mathbb{F}$ as

$$f(x_1, x_2, \ldots, x_m) = g_0(x_1^2, x_2, \ldots, x_m) + x_1 g_1(x_1^2, x_2, \ldots, x_m)$$

  - For $z \in \mathbb{F}$, define **Fold** $[f, z] : q(L_1) \times \prod_{i=2}^{m} L_i \to \mathbb{F}$ by

$$\textbf{Fold}\,[f, z]\,(y, \boldsymbol{x}) = g_0(y, \boldsymbol{x}) + z g_1(y, \boldsymbol{x})$$

  - After $\log d$ rounds, expected x_1-degree $= 0$
- Repeat for each of the other $m - 1$ variables.



Folding tensor product of RS codes

The tensor structure enables to fold along **one dimension at a time**.

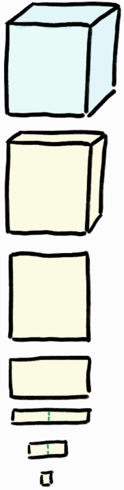
- Start by folding along the first dimension: $(q : x \mapsto x^2)$
  - Write $f : \prod_{i=1}^{m} L_i \to \mathbb{F}$ as

$$f(x_1, x_2, \ldots, x_m) = g_0(x_1^2, x_2, \ldots, x_m) + x_1 g_1(x_1^2, x_2, \ldots, x_m)$$

  - For $z \in \mathbb{F}$, define **Fold** $[f, z] : q(L_1) \times \prod_{i=2}^{m} L_i \to \mathbb{F}$ by

$$\textbf{Fold}\,[f, z]\,(y, \boldsymbol{x}) = g_0(y, \boldsymbol{x}) + z g_1(y, \boldsymbol{x})$$

  - After $\log d$ rounds, expected x_1-degree $= 0$
- Repeat for each of the other $m - 1$ variables.
- After a total of $m \log d$ rounds, final code has dimension 1.



Folding tensor product of RS codes

The tensor structure enables to fold along **one dimension at a time**.

- Start by folding along the first dimension: $(q : x \mapsto x^2)$
  - Write $f : \prod_{i=1}^{m} L_i \to \mathbb{F}$ as

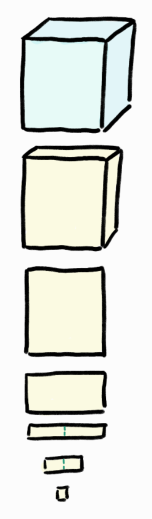
$$f(x_1, x_2, \ldots, x_m) = g_0(x_1^2, x_2, \ldots, x_m) + x_1 g_1(x_1^2, x_2, \ldots, x_m)$$

  - For $z \in \mathbb{F}$, define **Fold** $[f, z] : q(L_1) \times \prod_{i=2}^{m} L_i \to \mathbb{F}$ by

$$\textbf{Fold}\,[f, z]\,(y, \boldsymbol{x}) = g_0(y, \boldsymbol{x}) + z g_1(y, \boldsymbol{x})$$

  - After $\log d$ rounds, expected x_1-degree $= 0$
- Repeat for each of the other $m - 1$ variables.
- After a total of $m \log d$ rounds, final code has dimension 1.

☑ **Completeness** ☑ **Local computability** ☑ **Distance preservation**



Folding Reed-Muller code

In the total degree case, we fold along **every dimension at the same time**.

Folding Reed-Muller code

In the total degree case, we fold along **every dimension at the same time**.

Divide the size of the problem by 2^m: $\mathrm{RM}[L, d, m] \rightarrow \mathrm{RM}[q(L), d/2, m]$. $(q : x \mapsto x^2)$

Folding Reed-Muller code

In the total degree case, we fold along **every dimension at the same time**.

Divide the size of the problem by 2^m: $\mathrm{RM}[L, d, m] \rightarrow \mathrm{RM}[q(L), d/2, m]$. $(q : x \mapsto x^2)$

Lemma: multivariate decomposition

Let $f(\boldsymbol{X}) \in \mathbb{F}[X_1, \dots, X_m]$.
There is a unique sequence of polynomials $(g_{\boldsymbol{u}})_{\boldsymbol{u} \in \{0,1\}^m}$ such that

$$f(\boldsymbol{X}) = \sum_{\boldsymbol{u} \in \{0,1\}^m} \boldsymbol{X}^{\boldsymbol{u}} g_{\boldsymbol{u}}(X_1^2, \dots, X_m^2), \qquad \deg g_{\boldsymbol{u}} \leq \left\lfloor \frac{\deg f - w_{\mathrm{H}}(\boldsymbol{u})}{2} \right\rfloor$$

In the total degree case, we fold along **every dimension at the same time.**

Divide the size of the problem by 2^m: $\mathrm{RM}[L,d,m] \to \mathrm{RM}[q(L),d/2,m]$. $(q : x \mapsto x^2)$

Lemma: multivariate decomposition

Let $f(\boldsymbol{X}) \in \mathbb{F}[X_1,\ldots,X_m]$.
There is a unique sequence of polynomials $(g_{\boldsymbol{u}})_{\boldsymbol{u}\in\{0,1\}^m}$ such that

$$f(\boldsymbol{X}) = \sum_{\boldsymbol{u}\in\{0,1\}^m} \boldsymbol{X}^{\boldsymbol{u}} g_{\boldsymbol{u}}(X_1^2,\ldots,X_m^2), \qquad \deg g_{\boldsymbol{u}} \leq \left\lfloor \frac{\deg f - w_{\mathrm{H}}(\boldsymbol{u})}{2} \right\rfloor$$

The folding of $f : L^m \to \mathbb{F}$ w.r.t $\boldsymbol{z} \in \mathbb{F}^m$ is a function

$$\textbf{Fold}\,[f,\boldsymbol{z}] : q(L)^m \to \mathbb{F}$$

defined as a random linear combination of the $g_{\boldsymbol{u}}$'s.

Technical subtlety: need to be careful about the distinct degree bounds on the $g_{\boldsymbol{u}}$'s.

In the total degree case, we fold along **every dimension at the same time.**

Divide the size of the problem by 2^m: $\mathrm{RM}[L, d, m] \to \mathrm{RM}[q(L), d/2, m]$. $(q : x \mapsto x^2)$

Lemma: multivariate decomposition

Let $f(\boldsymbol{X}) \in \mathbb{F}[X_1, \ldots, X_m]$.
There is a unique sequence of polynomials $(g_{\boldsymbol{u}})_{\boldsymbol{u}\in\{0,1\}^m}$ such that

$$f(\boldsymbol{X}) = \sum_{\boldsymbol{u}\in\{0,1\}^m} \boldsymbol{X}^{\boldsymbol{u}} g_{\boldsymbol{u}}(X_1^2, \ldots, X_m^2), \qquad \deg g_{\boldsymbol{u}} \leq \left\lfloor \frac{\deg f - w_{\mathrm{H}}(\boldsymbol{u})}{2} \right\rfloor$$

The folding of $f : L^m \to \mathbb{F}$ w.r.t $\boldsymbol{z} \in \mathbb{F}^m$ is a function

$$\textbf{Fold}\,[f, \boldsymbol{z}] : q(L)^m \to \mathbb{F}$$

defined as a random linear combination of the $g_{\boldsymbol{u}}$'s.

Technical subtlety: need to be careful about the distinct degree bounds on the $g_{\boldsymbol{u}}$'s.

☑ **Completeness** ☑ **Local computability** (with $l = 2^m$) ☑ **Distance preservation**

[ABN21]

Distance-preserving folding operators for each code of a sequence of codes $(C_i)_{0 \leq i \leq r}$
$\Longrightarrow$ IOP of Proximity for the code C_0.

[ABN21]

Distance-preserving folding operators for each code of a sequence of codes $(C_i)_{0\leq i\leq r}$
$\Longrightarrow$ IOP of Proximity for the code C_0.

Theorem [ABN21]

$\mathrm{RS}[L,d]^{\otimes m}$ has an IOPP $(\mathcal{P},\mathcal{V})$ satisfying

$$\begin{bmatrix} \text{\# rounds} & = \log d^m \\ \text{\# queries} & = 2\log d^m + 1 \\ \text{prover time} & \leq 8|L^m| \\ \text{verifier time} & \leq 8\log d^m \\ \text{proof length} & < |L^m| \end{bmatrix}$$

Theorem [ABN21]

$\mathrm{RM}[L,d,m]$ has an IOPP $(\mathcal{P},\mathcal{V})$ satisfying

$$\begin{bmatrix} \text{\# rounds} & = \log d \\ \text{\# queries} & = 2^m \log d + 1 \\ \text{prover time} & < (2m+7)|L^m| \\ \text{verifier time} & < 2^m(\frac{5}{4}m+7)(\log d) \\ \text{proof length} & < |L^m|/(2^m-1) \end{bmatrix}$$

Remark: we also need $L \subset \mathbb{F}$ to be a multiplicative or additive subgroup of $\mathbb{F}$.

Conclusion

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).
- Inspired by FRI protocol for RS codes, we gave IOPPs for multivariate codes with **similar efficiency**.

Conclusion

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).
- Inspired by FRI protocol for RS codes, we gave IOPPs for multivariate codes with **similar efficiency**.
- The ability to define **distance-preserving folding operators** is sufficient to construct efficient IOPPs.

Conclusion

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).
- Inspired by FRI protocol for RS codes, we gave IOPPs for multivariate codes with **similar efficiency**.
- The ability to define **distance-preserving folding operators** is sufficient to construct efficient IOPPs.
- Folding-based approach also applies to AG codes [BLNR22] (smaller alphabets, fewer restrictions on field structure)

Conclusion

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).
- Inspired by FRI protocol for RS codes, we gave IOPPs for multivariate codes with **similar efficiency**.
- The ability to define **distance-preserving folding operators** is sufficient to construct efficient IOPPs.
- Folding-based approach also applies to AG codes [BLNR22] (smaller alphabets, fewer restrictions on field structure)

Conclusion

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).
- Inspired by FRI protocol for RS codes, we gave IOPPs for multivariate codes with **similar efficiency**.
- The ability to define **distance-preserving folding operators** is sufficient to construct efficient IOPPs.
- Folding-based approach also applies to AG codes [BLNR22] (smaller alphabets, fewer restrictions on field structure)

Open questions:

- Would $\{\mathrm{RS}^{\otimes m}$, RM, AG$\}$-based succinct arguments improve concrete efficiency?

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).
- Inspired by FRI protocol for RS codes, we gave IOPPs for multivariate codes with **similar efficiency**.
- The ability to define **distance-preserving folding operators** is sufficient to construct efficient IOPPs.
- Folding-based approach also applies to AG codes [BLNR22] (smaller alphabets, fewer restrictions on field structure)

Open questions:

- Would $\{\mathrm{RS}^{\otimes m}, \mathrm{RM}, \mathrm{AG}\}$-based succinct arguments improve concrete efficiency?
- Could we improve soundness/queries trade-offs?

Conclusion

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).
- Inspired by FRI protocol for RS codes, we gave IOPPs for multivariate codes with **similar efficiency**.
- The ability to define **distance-preserving folding operators** is sufficient to construct efficient IOPPs.
- Folding-based approach also applies to AG codes [BLNR22] (smaller alphabets, fewer restrictions on field structure)

Open questions:

- Would $\{\mathrm{RS}^{\otimes m}, \mathrm{RM}, \mathrm{AG}\}$-based succinct arguments improve concrete efficiency?
- Could we improve soundness/queries trade-offs?
- Practical IOPP with sublogarithmic query complexity? (in theory, $O(1)$ queries)

Conclusion

Let's recap:

- Proximity tests for linear codes play a role in verifiable computing and ZK proofs (ex: FRI [BBHR18]).
- Inspired by FRI protocol for RS codes, we gave IOPPs for multivariate codes with **similar efficiency**.
- The ability to define **distance-preserving folding operators** is sufficient to construct efficient IOPPs.
- Folding-based approach also applies to AG codes [BLNR22] (smaller alphabets, fewer restrictions on field structure)

Open questions:

- Would $\{\text{RS}^{\otimes m}, \text{RM}, \text{AG}\}$-based succinct arguments improve concrete efficiency?
- Could we improve soundness/queries trade-offs?
- Practical IOPP with sublogarithmic query complexity? (in theory, $O(1)$ queries)

Thank you!