Review of IOPP to Algebraic Geometry Codes

Parsa Tasbihgou, Rayan Sandid

January 2024

1 Motivation

In this paper the authors introduce an interactive oracle proof of proximity to algebraic geometry codes by generalizing the FRI protocol.
Using Reed-Solomon codes alongside the FRI protocol allows us to create a linear-size IOP for R1CS. However this IOP has some inherent limitations. By replacing Reed-Solomon codes with algebraic geometry codes and the FRI protocol with the IOPP in this paper we try to overcome some of these limitations:

- The alphabet size must be larger than the block length of the code.
  $|F| \geq$ block length.
- The field has to have a specific algebraic structure. Namely it has to have a large cyclic subgroup of smooth order.
- Due to the large size of the field, the field operations are costly.

In this paper the authors try to solve or relax some of these limitations. The field size can be constant and therefore the field operations have constant time complexity, also the specific algebraic structure of the field is relaxed to having a root of unity of relatively small order.

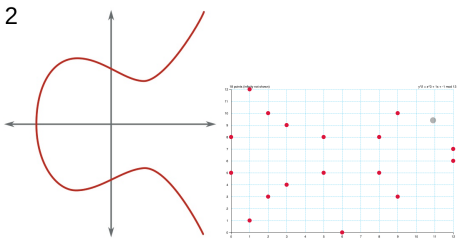
2 Mathematical background

We start our review of the mathematical backgrounds of this paper by a brief summery of algebraic curves.

2.1 Algebraic curve $\mathcal{X}$

The formal definition of an algebraic curve is a projective algebraic variety of dimension 1. However, for our purposes in this project an informal definition of an algebraic curve suffices:

Definition 2.1 (Algebraic curve) *Informally, an algebraic curve is a set of points in space that are exactly the common roots of a system of polynomials.*



(a) $\mathbb{R}$ (b) $\mathbb{F}_{13}$

Figure 1: (a) and (b) are respectively the graphs of the curve defined by $y^2 = x^3 + x - 1$ in fields $\mathbb{R}$ and $\mathbb{F}_{13}$

2.2 Divisors

$Div(\mathcal{X})$ the divisor group of the curve $\mathcal{X}$ is the free module of the points on curve $\mathcal{X}$ over the ring of integers.

Definition 2.2 (Divisor) *Informally, a divisor D of $\mathcal{X}$ is a formal sum over the points on $\mathcal{X}$ with integer coefficients.*

$$D = \sum_{P \in \mathcal{X}} n_P \cdot P.$$

A divisor is called **effective** if all the points have non-negative coefficients. Divisors D and D' are compared $D \leq D'$ iff $D' - D$ is effective ($D' - D \geq 0$).

Definition 2.3 (Canonical divisor) *Let f be a (rational) function defined on $\mathcal{X}$, the canonical divisor of f is defined, $div_{\mathcal{X}}(f) = \sum_{P \in \mathcal{X}} v_f(P) \cdot P$.* [1]

2.3 Riemann-Roch Space

Given a curve and a divisor on that curve the **Riemann-Roch space** $L_{\mathcal{X}}(D)$ is a $\mathbb{F}$-vector subspace of $\mathbb{F}(\mathcal{X})$.

Definition 2.4 (Riemann-Roch space)

$$L_{\mathcal{X}}(D) = \{f \in \mathbb{F}(\mathcal{X}) | div_{\mathcal{X}}(f) + D \geq 0\} \cup \{0\}.$$

2.4 Group action on curves

In this subsection we give a quick reminder of group actions and how they extend to curves.
A group G is said to act on a set Ω via an operator $\cdot : G \times \Omega \to \Omega$ iff for all $\omega \in \Omega, a, b \in G$, $1_G \cdot \omega = \omega$ and $a \cdot (b \cdot \omega) = (a + b) \cdot \omega$.

Definition 2.5 (Orbit) *The orbit of an element $\omega \in \Omega$ in a group action are all the elements accessible from ω using the action of G. $Orb_G(\omega) = \{x \in \Omega | \exists a \in G(a \cdot \omega = x)\}$.*

Definition 2.6 (Quotient curve) *Let $\mathcal{X}$ be a curve and G a finite abelian subgroup of $Aut(\mathcal{X})$, we define the quotient of $\mathcal{X}$ over G as $\mathcal{X}/G = \{Orb_G(P) | P \in \mathcal{X}\}$. We can verify that $\mathcal{X}/G$ is a curve and we have the projection map $\pi : \mathcal{X} \to \mathcal{X}/G, \pi(P) = Orb_G(P)$.*

This concludes our review of the mathematical background for this paper.

3 Algebraic geometry codes

An algebraic geometry code is a type of linear code more specifically, it is an evaluation code meaning that code-words are obtain by evaluating a set of functions over a set of points on some geometric object X. Moreover, we can show that any evaluation code is an algebraic geometry code.
To define an algebraic geometry code we need a curve $\mathcal{X}$, an evaluation domain $\mathcal{P} = (P_1, P_2, ..., P_n)$ of size n (a set of points on the curve) and a divisor D.
Then we can define the AG code $C(\mathcal{X}, \mathcal{P}, D)$ to be the image of the evaluation map $ev : L_{\mathcal{X}}(D) \to \mathbb{F}^n, ev(f) = (f(P_1), f(P_2), ..., f(P_n))$.
In particular Reed-Solomon code $RS[\mathbb{F}, \mathcal{P}, d]$ is a special case of AG codes $C(\mathbb{P}^1, \mathcal{P}, dP_\infty)$.

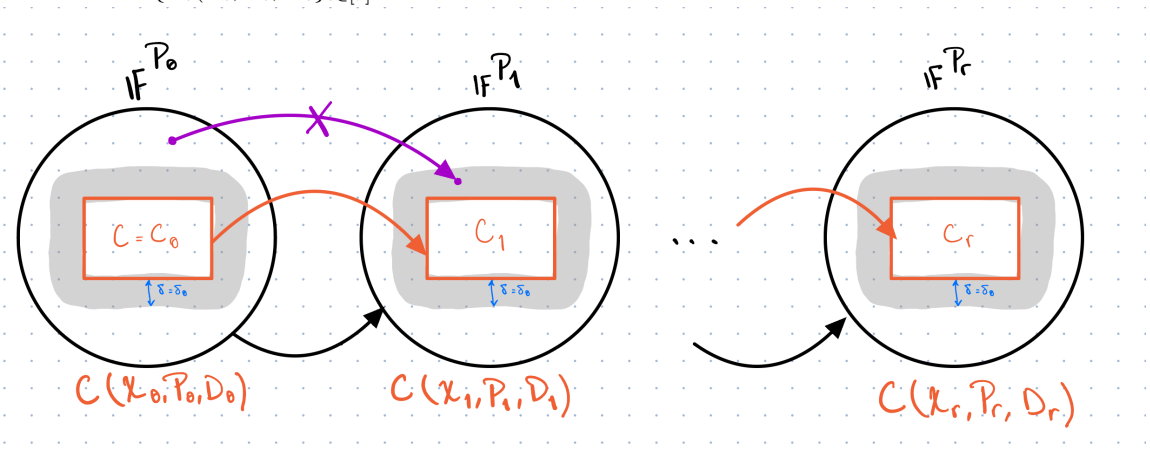
[1] $v_f(P)$ is the valuation of f at point P. The valuation is 0 if f is non-zero, positive if f is 0 and negative if f has poles.

4 Generalizing the FRI protocol

In this section we will discuss how to generalize the FRI protocol for AG codes.
We remember that FRI has 3 key ideas:

- Splitting polynomials:
  Decomposing the space of degree $\leq d$ polynomials ($\mathbb{F}^d[x]$) into two copies of the space of degree $\leq d/2$ polynomials ($\mathbb{F}^{d/2}[x]$).
  To extend this idea of decomposing the function space, we will decompose the Riemann-Roch space $L_{\mathcal{X}}(D)$.
- Randomized folding:
  The (randomized) folding operator reduces the evaluation domain (block lenght). In addition to the decomposition of the function space this operator reduces the function space, therefore, the code is reduced over all parameters. To extend this idea, we will introduce a randomized folding operator that combines the components from the decomposition.
- Distance preservation:
  The folding operator is such that if f is δ-far from the code $C = RS[\mathbb{F}, \mathcal{P}, d]$ then Fold[f, z] is with high probability "almost" δ-far from the code $C' = RS[\mathbb{F}, \mathcal{P}, d/2]$.

We want to iteratively reduce the code $C = C(\mathcal{X}, \mathcal{P}, D)$ so that at the final step we have a code that we can check membership efficiently. The approach is similar to the FRI protocol and we need a sequence of codes $\{C_i(\mathcal{X}_i, \mathcal{P}_i, D_i\}_{i\in[r]}$.



Let $G \leq Aut(\mathcal{X})$ be a finite solvable subgroup of the automorphism group of $\mathcal{X}$. Since G is solvable there is a normal sequence $G = G_0 \triangleright G_1 \triangleright \ldots \triangleright G_r = \{1\}$. Let $\Gamma_i = G_i/G_{i+1}$, by definition Γ_i is abelian and hence it acts on the curve $\mathcal{X}_i$ to create the quotient curve $\mathcal{X}_{i+1} = \mathcal{X}_i/\Gamma_i$. Remember that each group action Γ_i comes with a projection map π_i. Now we define a $(\mathcal{X}, G)$-sequence of curves and projections as follows:

$$\mathcal{X} = \mathcal{X}_0 \xrightarrow{\pi_0} \mathcal{X}_1 \xrightarrow{\pi_1} \mathcal{X}_2 \xrightarrow{\pi_2} \ldots \xrightarrow{\pi_{r-1}} \mathcal{X}_r = \mathcal{X}/G$$

By applying the projection maps iteratively on the evaluation domain $\mathcal{P}$ we get a sequence of evaluation domains.

$$\mathcal{P} = \mathcal{P}_0 \xrightarrow{\pi_0} \mathcal{P}_1 \xrightarrow{\pi_1} \mathcal{P}_2 \xrightarrow{\pi_2} \ldots \xrightarrow{\pi_{r-1}} \mathcal{P}_r = \mathcal{P}/G$$

Notice that the number of points on the curve $\mathcal{X}_r = \mathcal{X}/G$ is $\frac{|\mathcal{X}|}{|G|}$. Similarly $|\mathcal{P}_r| = \frac{|\mathcal{P}|}{|G|}$.

4.1 Decomposing the Riemann-Roch space

Theorem 4.1 (Kani's theorem) *Given a function $f : \mathcal{P}_i \to \mathbb{F}$, define p_i functions $\{f_j : \mathcal{P}_{i+1} \to \mathbb{F}\}_{j\in[p_i]}$ and divisors $\{E_j\}_{j\in[p_i]}$ such that f is the evaluation of a function in $L_{X_i}(D_i)$ iff f_j is the evaluation of some function in $L_{X_{i+1}}(E_j)$.*

$$f = \sum_{j=0}^{p-1} \mu^j f_j \circ \pi_i$$

Divisors E_j and functions μ_i are explicitly expressed in terms of D_i and $\mathcal{X}_i$.

Kani's theorem allows us to decompose $L_{\mathcal{X}_i}$ into a sequence smaller Riemann-Roch spaces $L_{\mathcal{X}_{i+1}}$ therefore, a proximity test to a code defined by a divisor D_i is broken into several proximity tests to codes defined by E_j. Notice that in each step we want to test proximity to a single code therefore, we need to choose a divisor D_i such that all Riemann-Roch spaces $L_{\mathcal{X}_{i+1}}(E_j)$ are subspace of $L_{\mathcal{X}_{i+1}}(D_i)$

The orange edges in Figure 2 represent decomposition of valid functions. The blue edges represent decomposition of an invalid (far) function when at least one of the components is out of the code C_{i+1} therefore we can hope to catch it later.
The purple edges represent the decomposition of an invalid function but all components are in the code C_{i+1} so the protocol can never catch this in the future. We need to prevent this kind of decomposition in order to preserve distance.

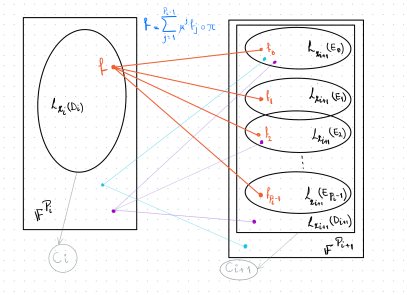


Figure 2: Decomposition via Kani's theorem

4.1.1 Balancing functions

To prevent such decompositions we use balancing functions ν_j. By definition $f_j \in L_{\mathcal{X}_{i+1}}$ iff
$f_j \in L_{\mathcal{X}_{i+1}}(D_{i+1})$ and $\nu_j f_j \in L_{\mathcal{X}_{i+1}}(D_{i+1})$. So it is enough to test proximity of f_j and $\nu_j f_j$ to C_{i+1}.
The existence of balancing functions depends on the Weierstrass semigroup of $support(D_{i+1})$. If balancing functions exist for $_{i+1}$ we say it is D_i-compatible.

4.2 Fold operator

Now we can define the randomized folding operator similar to the FRI protocol. The grey term is added for balancing functions.

$$\mathsf{Fold}[f, \vec{z}] = \sum_{j=0}^{p_i-1} z_1^j f_j + \sum_{j=0}^{p_i-1} z_2^{j+1} \nu_{i+1,j} f_j$$

4.3 Completeness

Since $L_{\mathcal{X}_{i+1}}(E_j) \subseteq L_{\mathcal{X}_{i+1}}(D_{i+1})$, any linear combination of $L_{\mathcal{X}_{i+1}}(E_j)$ is a linear subspace of $L_{\mathcal{X}_{i+1}}(D_{i+1})$ so if $f \in C_i$ then for any $\vec{z} \in \mathbb{F}^2$, $\mathsf{Fold}[f, \vec{z}] \in C_{i+1}$.

4.4 IOPP for AG codes

- **Setup**
  The prover and the verifyer agree on:
  - Starting curve $\mathcal{X}$
  - Group $G \subset Aut(\mathcal{X})$
  - Sequence $(\mathcal{X}, G)$
  - Functions μ_i
  - Divisors D_i.
- **Commit** The verifyer sends random field elements z_i to the prover and the prover sends the foldings $f_{i+1} = Fold[f_i, z_i]$as oracles to the verifyer.
- **Query** First, the verifyer does a round consistency check to see if all the oracles are really the folding of the previous one. The verifyer does this by comparing the two functions at a random point.
  Second, the veryfier reads the last code-word entirely and checks if it is in the last code C_r.