Elliptic curve primer: https://jeremykun.com/2014/02/08/introducing-elliptic-curves/
Blog post on implementing ECFFT part 1: https://solvable.group/posts/ecfft/

ECFFT Part 1:

https://arxiv.org/pdf/2107.08473.pdf

1. Introduction

- In classical FFT, we convert evaluation table representation of polynomial into coefficient form in $O(n)$ with classical inverse FFT.
- With elliptic curve group, don't know how to do this as fast. They instead *extend* the evaluation of polynomial on subset S to another subset S' of $\mathbb{F}_q$.
    - Like using FFT and inverse-FFT to use evaluations on multiplicative subgroup S to deduce evaluations at a coset of S.
    - Similarly, they do ECFFT and inverse. But intermediate representation after inverse-ECFFT is not coefficient form of polynomial.
- They use fast algorithm for extending polynomial evaluations to do in $O(n \log n)$:
- For polynomials of low degree, represented as evaluations over special sets:
    - Polynomial addition
    - Polynomial multiplication
    - Degree computation
- Note, addition is trivially $O(n)$. But for mult, only trivial if result has degree < n. They can get higher degree results by extending evaluations to a larger set.
- Degree computation non-trivial since polynomials not represented directly by coefficients.
- Only representation of polynomials allowing these three operations to be computed in $O(n \log n)$ operations for general q and $n \leq q^{O(1)}$

- They also give fast algorithms for other operations like division.

- Converting between new representation and standard coefficient representation is $O(n \log^2 n)$.

ECFFT - Informal Explanation

- Similarities:
    - ECFFT uses degree-2 maps that are 2-to-1 on the special sets of points.
    - ECFFT uses ability to express polynomial of degree < n in terms of two polynomials of degree $< n/2$

- Differences:
    - The domains are not groups of $\mathbb{F}_p$.
    - The 2-to-1 maps may vary as we recurse, whereas FFT uses only squaring.

- The maps are degree-2 rational maps (ratio of degree-2 polynomials) instead of degree 2 polynomials. Gives more freedom to search for 2-to-1 maps on special sets of points.
-

Elliptic Curves as a Source for FFTrees over Arbitrary Finite Fields

- Main properties of elliptic curves for $\mathbb{F}_q$ used:
  - The number of points on the curve E can be almost any number in range $[q + 1 \pm 2\sqrt{q}]$
  - These points form an elliptic curve group (which will be abelian). Using previous point, we can find subgroups G of elliptic curve groups of size $n = 2^k$ for $n = O(\sqrt{q})$.
  - If $H < G$ are subgroups of an elliptic curve E over $\mathbb{F}_q$, there is an $|H|$-to-1 map ϕ called an isogeny with kernel H that maps points of curve E to points of a different curve E' over same field $\mathbb{F}_q$. So the size of image of G is $|G|/|H|$.
- Initial set of points $G^{(0)}$ inside curve E^0. Each step compresses group of points $G^{(i))}$ to half the size $G^{(i+1)}$ using the 2-to-1 isogeny $\phi^{(i)}$
- Remaining gap: points in each group $G^{(i)}$ are pairs $(x, y) \in \mathbb{F}_q^2$, but we are interested in univariate polynomials and evaluation sets over just $\mathbb{F}_q$.
  - They pick curves in a particular format (extended Weierstrass form) so that shifting and projecting $G^{(i)}$ to the x coordinate gives a set $L^{(i)} \subset \mathbb{F}_q$ with the same size as $G^{(i)}$, and also the isogeny map $\phi^{(i)}$ becomes a degree-2 rational map that is 2-to-1 from $L^{(i)}$ onto $L^{(i+1)}$.
- Summary: Abundance of elliptic curve groups of many sizes over any large finite field means we can always find a subgroup of smooth size (concretely, 2^k size). Isogenies and their projections give 2-to-1 degree-2 rational maps from sets of size 2^k to sets of size 2^{k-1} for all needed k.

3. Polynomial decompositions and FFTrees

- Classical FFT decomposes degree d polynomial $P(X)$ into two degree $d/2$ polynomials (containing even / odd degree terms):
  - $P(X) = P_0(X^2) + X \cdot P_1(X^2)$
- Can generalize, replacing X^2 with a rational function. (Simplify description of next Lemmas by setting parameter $\delta = 2$)
  - **Lemma 3.1:**
    - Let $\psi(X) = \frac{u(X)}{v(X)}$ be a degree 2 rational map over $\mathbb{F}_q$, and let d be a multiple of 2.
    - Then for any degree $\leq d$ polynomial $P(X)$, there is a unique pair $(P_0(X), P_1(X))$ of degree $\leq d/2$ polynomials such that:

- $P(X) = \left(P_0(\psi(X)) + X \cdot P_1(\psi(X))\right) \cdot v(X)^{\frac{d}{2}-1}$
- **Lemma 3.2:**
  - Let $s_0, s_1, t \in \mathbb{F}_q$ be such that $\psi(s_0) = \psi(s_1) = t$ with $s_0 \neq s_1$.
  - Then the mapping $M : (P(s_0), P(s_1)) \mapsto (P_0(t), P_1(t))$ is linear and invertible.
  - In particular, M is a 2×2 matrix whose entries depend only on $s_0, s_1, v(s_0), v(s_1)$.
    - When implementing, we know these in advance so we can pre-compute the matrix M.
  - So when we have degree 2 ψ that is 2-to-1 from S to $T = \psi(S)$, we can use the evaluations of $P_0(X)$ and $P_1(X)$ at the points of T to get the evaluations of $P(X)$ on S.

FFTrees

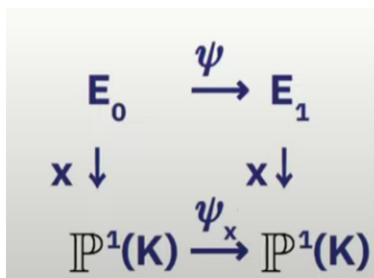
- **Definition 3.3:**
  - Let q be a prime power and k an integer. An FFTree over $\mathbb{F}_q$ of depth k is a sequence of subsets $L^{(0)}, L^{(1)}, \ldots, L^{(k)} \subseteq \mathbb{F}_q$, along with a sequence of degree 2 rational functions $\psi^{(i)}(X)$ over $\mathbb{F}_q$ such that:
    - $|L^{(0)}| = 2^k, |L^{(1)}| = 2^{k-1}, \ldots, |L^{(k)}| = 1$
    - $\psi^{(i)}(L^{(i)}) = L^{(i+1)}$ (so $\psi^{(i)}$ is 2-to-1 from $L^{(i)}$ to $L^{(i+1)}$)
  - This forms a binary tree.
    - The root is the single element of $L^{(k)}$
    - The leaves are the elements of $L^{(0)}$
    - The parent of an element $s \in L^{(i)}$ is the element $\psi^{(i)}(s) \in L^{(i+1)}$.
- An FFTree of depth k is useful for polynomials of degree up to $2^k - 1$.
- Elliptic curves used to find FFTrees over any $\mathbb{F}_q$ of depth $\Omega(\log q)$.

4. FFTrees from Elliptic Curves

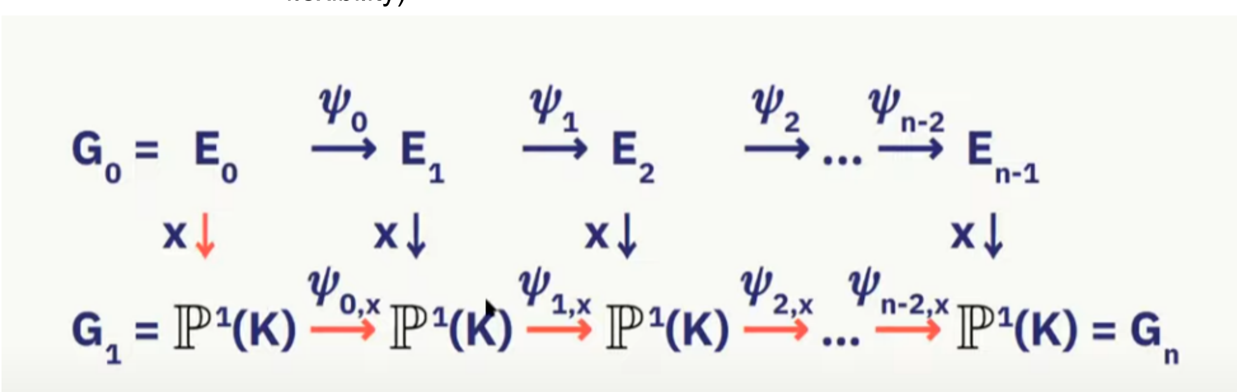
(Notes from lecture below: zkStudyClub: Elliptic Curve Fast Fourier Transform)
https://www.youtube.com/watch?v=kQZvBXLZ8dM&ab_channel=ZeroKnowledge

Isogeny (replacing the squaring map)

- **An isogeny is a morphism between elliptic curves that is also a group homomorphism.**
  - (Morphism definition from https://www.hyperelliptic.org/tanja/conf/summerschool08/slides/Maps.pdf)
  - A morphism $\phi : E \to E'$ between elliptic curves (in projective coordinates) is a polynomial mapping $\phi : (x : y : z) \mapsto (\phi_0(x, y, z) : \phi_1(x, y, z) : \phi_2(x, y, z))$

- The ϕ_i are homogeneous polynomials satisfying the equation of target curve E'.
- In affine coordinates (normalizing z =1), ϕ is a rational map:
  - $\phi : (x, y) \mapsto \left(\frac{\phi_0(x,y,1)}{\phi_2(x,y,1)}, \frac{\phi_1(x,y,1)}{\phi_2(x,y,1)}\right)$
- Equivalent definition: An isogeny $\phi : E \to E'$ of elliptic curves is a non-constant rational map that sends the identity $0_E \in E$ to the identity $0_{E'} \in E'$.
- **Theorem: Any finite subgroup of an elliptic curve is the kernel of some isogeny**
  - So to get 2-to-1 isogeny $\phi : E \to E'$, use a subgroup of size 2 from E.
  - Example:
    - $E_0 : y^2 = x^3 + ax^2 + b^2x$
    - $E_1 : y^2 = x^3 + (a + 6b)x^2 + (4ab + 8b^2)x$
    - 2-to-1 isogeny: $\phi : (x, y) \mapsto \left(x - 2b + \frac{b^2}{x}, y(1 - \frac{b}{x^2})\right)$
    - $\ker(\phi) = \{(0, 0), \infty\}$
      - (remember, for homogeneous Weirstrass form, $\infty = [0 : 1 : 0]$)
- **Theorem: x coordinate of any isogeny depends only on x coordinate of the input**
  - $\phi : E_0 \to E_1$
  - $\phi : (x, y) \mapsto (\phi_x(x), \phi_y(x, y)))$
  - Commutative diagram:
    - 

  - Apply on chain of elliptic curves $E_0 \to E_1 \to E_2 \to \ldots$
    - (The curves don't necessarily need to be different but this way gives more flexibility)



    - The ECFFT computation will go through the bottom path (So we can work with univariate polynomials instead of bivariate)
    - The bottom maps will replace the "squaring" of classical FFT.

- What are the evaluation points at each step?
    - Traveling along the top, at E_{n-1} we have 2 points. Each time we go backwards, the number of points doubles.
    - Since diagram is commutative, the pre-image of each bottom step is the x-coordinate of the pre-image of the top map.

5. Representing polynomials via FFTrees

- The basic idea is to represent a polynomial by the leaves of a subtree in a fixed FFTree
- To represent degree $< n$ polynomial, we write its evaluations on the leaves of a sub-FFTree with at least n leaves.

6. Fast polynomial algorithms from FFTrees

- Fix one FFTree here.
- Main algorithm is EXTEND(S, S'), which does low degree extension of polynomial evaluations from sub-FFTree S to another sub-FFTree S'. All the other algorithms are based on EXTEND.
    - With $|S| = |S'| = n$, EXTEND(S, S') takes $O(n \log n)$ field ops.
    - More concretely, say S, S' the odd and even indices of the FFTree leaves L. Given evaluations of degree $< n/2$ polynomial Q on S, EXTEND computes evaluations of Q on S' as follows.
        - Base case:
            - $|S| = |S'| = 1$ and Q is constant, so evaluations of Q on S and S' are the same.
        - Recursion step:
            - Use 2-to-1 isogeny ϕ to decompose evaluations of Q on S into evaluations of Q_0 and Q_1 on $\psi(S)$.
            - Apply EXTEND twice to get evaluations of Q_0 and Q_1 on $\psi(S')$.
            - Then invert the decomposition of Q to recover evaluations of Q on S'.
- Other algorithms include polynomial multiplication, degree computation, polynomial division and remainder, and ENTER/EXIT to convert between evaluation and coefficient forms.
- The ENTER operation (coefficients -> evaluations on FFTree leaves) works similarly as classic FFT, but it calls EXTEND in the recursion step. So the resulting runtime is a bit worse than classical FFT: $O(n \log^2 n)$.

ECFFT Part 2:

https://www.math.toronto.edu/swastik/ECFFT2.pdf

1. Introduction

- Main question: Which finite fields can be used to create transparent, scalable, and concretely efficient proof systems?
  - Classic applications of arithmetization from 90s (MIP = NEXP, PCP, etc) work with any large enough finite field. However, proofs are impractically large and prover/verifier also infeasible to implement in real life.
  - **Scalable** proof systems:
    - Proving time (# of field operations) scales quasi-linearly in T
    - Verification time scales polylog in T
  - Around ~2010, scalable PCPs for NEXP developed. But the finite field $\mathbb{F}$ needs to be FFT-friendly, meaning $\mathbb{F}$ must contain subgroup of size 2^k (either mult or additive subgroup is ok)
    - These still not used in practice because prover/verifier time and soundness error still too large
  - Lastly, IOPs for NEXP developed with proving time $O(TlogT)$, verification time $O(logT)$, though still need FFT-friendly finite field.
  - Summary: Early constructions work for any large enough finite field, but scalable PCP/IOP require FFT-friendly finite field. So is FFT-friendliness needed for scalability? Main result says no!

1.1 Main Results

- **Arithmetic intermediate representation (AIR)**
  - Idea is to encode the trace of a computation algebraically.
  - AIR with complexity m, length T, over $\mathbb{F}$, contains
    - A total of m gates to specify a set of low degree multivariate constraints.
    - Cyclic group D of size T.
    - Idea is that this encodes state transition validity checks
  - AIR witness is functions $f_1, \ldots, f_w : D \to \mathbb{F}$.
    - Idea is that this encodes trace of a computation correctly evolving according to the state transition function.
  - Satisfiable AIR instances are NEXP-complete. Same if restricted to FFT-friendly fields.
- AIR used for STARKs:
  - Specific computations: ethSTARK
  - Domain specific languages: Winterfell
  - VMs: Cairo
- Previously, had scalable and transparent IOP for AIR over FFT-friendly fields. Below main theorem removes the FFT-friendly requirement.
- **Main Theorem:** For any finite field $\mathbb{F}$ and $T \leq \sqrt{|\mathbb{F}|}$, satisfiability of AIR instance over $\mathbb{F}$ with size m, length T can be verified by a strictly scalable and transparent IOP of knowledge with advice.
  - Prover: $T \cdot (O(logT) + poly(m))$
  - Verifier: $\lambda \cdot (O(logT) + poly(m))$ with knowledge soundness error $2^{-\lambda}$
- Result applies to other systems like succinct R1CS.

- Can be augmented to achieve perfect zero knowledge
- Applying e.g. Kilian-Micali to remove interaction results in post quantum security

Fast IOPs of Proximity for Reed–Solomon and Elliptic Curve codes

- Given oracle access to function $f : D \to F$, distinguish between f being low degree polynomial (a Reed-Solomon codeword), or f far away from RS codeword.
- Strictly scalable IOPs used FRI protocol. But FRI needs FFT-friendly field.
    - For function of blocklength $n = |D|$, FRI gets:
        - Prover $O(n)$
        - Verifier $O(\lambda \log n)$ for soundness error $2^{-\lambda}$.
- To prove main theorem, they extend FRI protocol to work over **all** fields with $|\mathbb{F}| >= \Omega(\sqrt{n})$
- **FRI over all fields:** For any finite field $\mathbb{F}$ of size q, and integer n a power of 2 $\leq \sqrt(q)$, and integers t and rate $\rho = 2^{-R}$ for some integer R:
    - Exists a subset $D' \subseteq \mathbb{F}$, $|D| = n$, s.t. Family of RS codes of rate ρ evaluated over D' has an IOP of proximity with
        - Prover $O(n)$
        - Verifier $O(t \log n)$
        - Query complexity $t \log n$
        - Soundness: If f is δ-far from a codeword, probability of accepting f is at most $\left(\max\{(1-\delta), \sqrt{\rho}\} - o(1)\right)^t$

Applications to concrete scalability:

- Non-FFT-friendly finite fields used in practice. E.g. secp256k1 for bitcoin's ECDSA uses a prime field.
- Say prover wants to prove they correctly processed a batch of ECDSA signatures over this prime field $\mathbb{F}_p$.
- Prover would need to arithmetize over some FFT-friendly field $\mathbb{F}_q$, then simulate the field operations of $\mathbb{F}_p$ within $\mathbb{F}_q$. Overhead can be ~100x.
- Main theorem allows us to arithmetize over the native, non-FFT friendly field. Can stay in the original prime field F_p. However, new construction uses more complicated elliptic curves instead of plain polynomials so there is a tradeoff.
    - Not yet implemented, but they speculate that the new constructions will be better

1.2 Why do PCPs and IOPs require FFT-friendliness?

- Need a cyclic group D of size 2^k for below reasons.
    - **1. Super-efficient Reed-Solomon encoding**
        - Need to compute low degree extensions of witness polynomials
        - Can use EXTEND algorithm from ECFFT part 1
    - **2. Codewords invariant to cyclic shifts**
        - Just needs cyclic group as domain, but doesn't seem to require size 2^k.

- **3. Polylogarithmic verification requires sparse domain polynomials**
    - Need to evaluate vanishing polynomials of several subsets of D.
    - Just need any multiplicative subgroup.
- **4. Low-degree testing**
    - Needed to use FRI, which required FFT-friendly domain.

1.3 Elliptic curves save the day, again

- To get analogues of 2, 3, 4 above, need to get deeper into elliptic curve group structure and Riemann-Roch spaces.
- They also give randomized near-linear time algorithm to do the precomputations for ECFFT.
- Similar as in ECFFT part 1, in non-FFT-friendly finite field, we can get a subgroup of size 2^k. In ECFFT part 2, they use something more specific: can get a **cyclic** subgroup of size 2^k.

Arithmetization and automorphisms

- In classical FFT-friendly field IOP, for efficient arithmetization, they use invariance of polynomials under certain linear transformation.
    - For example, for a group $G \subset \mathbb{F}_q$ generated by g and a degree d polynomial $f : G \to \mathbb{F}_q$, we have that $f(g \cdot x)$ is also a degree d polynomial. That is, the space of degree $\leq d$ functions is invariant under the permutation $x \mapsto g \cdot x$.
- Say we want to arithmetize using cyclic group H that is generated by a point h on an elliptic curve. Natural permutation here is $x \mapsto x + h$.
    - Need space of functions invariant under this permutation. Ends up that a certain Riemann-Roch space will work.

Appendix A.5: Divisors and Riemann-Roch spaces

- RR space for elliptic curve: rational functions ($f(x) = \frac{P(x)}{Q(x)}$ where P, Q are polynomials) over the elliptic curve with zeroes and poles constrained by a “divisor” D.
    - Given a rational function, its divisor $\operatorname{div}(f)$ is a formal linear combination of points on the curve:
        - $\sum n_i P_i$ where n_i are integers, P_i are on the curve.
        - Negative n_i means a pole, positive n_i is a zero (both with multiplicity)
    - Partially ordered: $D \geq D'$ iff $D - D'$ is non-negative.
- Riemann-Roch space for divisor D is below set of rational functions:
    - $\{f : \operatorname{div}(f) \geq -D\} \cup \{0\}$

3. Scalable IOPs for AIRs over any large field

The EC backbone

- Backbone of constructions is chain of 2-isogenies shown to exist in ECFFT part 1. They use a strengthened and more explicitly-described version of this isogeny chain for their IOPs.

The IOP Protocol

- At a high level, steps are similar to https://eprint.iacr.org/2021/582, but with rational functions and points on the curve replacing univariate polynomials and points in $\mathbb{F}_q$.

- 1. Prover does EC version of a "low-degree extension" of a satisfying AIR-witness and sends it to Verifier.
    - Codewords are functions from a Riemann-Roch space of an elliptic curve. These codewords agree with the AIR-witness on a certain subset of points T.
- 2. Verifier samples one random field element per AIR constraint and sends it to the prover.
    - This randomly picks out a random linear combination of the constraints. Prover needs to convince verifier that their witness satisfies this random linear combination.
    - Ends up that we can reduce this to showing that a certain random linear combination $\hat{f}^r$ belongs to a certain Riemann-Roch space.
- 3. Prover represents the rational function $\hat{f}^r$ as two univariate polynomials. Then sends evaluations of these polynomials at points of T.
- 4. Verifier samples a certain challenge point q on the elliptic curve. (known as a DEEP query)
- 5. Prover sends back evaluations of its witness functions as well as the random linear combination $\hat{f}^r$ based on the challenge point q.
    - Verifier checks that these evaluations are consistent with the AIR constraints.
- 6. Prover and Verifier run a batched FRI protocol to check that the witness and random linear combination functions are all low degree.

4. Sequence of Elliptic Curve isogenies

- This section explicitly constructs the isogeny chain shown to exist in ECFFT pt 1.
    - Strengthen from initial subgroup of size 2^k to a *cyclic* group isomorphic to $\mathbb{Z}/2^k\mathbb{Z}$.
    - Each isogeny kernel is the set of two points $\{0, \infty\}$.
- Also gives a quasi-linear time algorithm that, for integers q, p, yields an elliptic curve E_0 over $\mathbb{F}_q$ with an order 2^k cyclic subgroup.