Electronic Colloquium on Computational Complexity, Report No. 103 (2021)

Elliptic Curve Fast Fourier Transform (ECFFT) Part I: Fast Polynomial Algorithms over all Finite Fields

Eli Ben-Sasson* Dan Carmon* Swastik Kopparty † David Levit*

July 18, 2021

Abstract

Over finite fields $\mathbb{F}_q$ containing a root of unity of smooth order n (smoothness means n is the product of small primes), the Fast Fourier Transform (FFT) leads to the fastest known algebraic algorithms for many basic polynomial operations, such as multiplication, division, interpolation and multi-point evaluation. These operations can be computed by constant fan-in arithmetic circuits over $\mathbb{F}_q$ of quasi-linear size; specifically, $O(n \log n)$ for multiplication and division, and $O(n \log^2 n)$ for interpolation and evaluation.

However, the same operations over fields with no smooth order root of unity suffer from an asymptotic slowdown, typically due to the need to introduce "synthetic" roots of unity to enable the FFT. The classical algorithm of Schönhage and Strassen [SS71] incurred a multiplicative slowdown factor of $\log \log n$ on top of the smooth case. Recent remarkable results of Harvey, van der Hoeven and Lecerf [HvdHL17, HvdH19a] dramatically reduced this multiplicative overhead to $\exp(\log^*(n))$.

We introduce a new approach to fast algorithms for polynomial operations over all large finite fields. The key idea is to replace the group of roots of unity with a set of points $L \subset \mathbb{F}_q$ suitably related to a well-chosen elliptic curve group over $\mathbb{F}_q$ (the set L itself is *not* a group). The key advantage of this approach is that elliptic curve groups can be of *any* size in the Hasse–Weil interval $[q + 1 \pm 2\sqrt{q}]$ and thus can have subgroups of large, smooth order, which an FFT-like divide and conquer algorithm can exploit. Compare this with multiplicative subgroups over $\mathbb{F}_q$ whose order must divide $q - 1$. By analogy, our method extends the standard, multiplicative FFT in a similar way to how Lenstra's elliptic curve method [Len87] extended Pollard's $p - 1$ algorithm [Pol74] for factoring integers.

For polynomials represented by their evaluation over subsets of L, we show that multiplication, division, degree-computation, interpolation, evaluation and Reed–Solomon encoding (also known as low-degree extension) *with fixed evaluation points* can all be computed with arithmetic circuits of size similar to what is achievable with the classical FFTs when the field size q is special. For several problems, this yields the asymptotically smallest known arithmetic circuits even in the standard monomial representation of polynomials.

The efficiency of the classical FFT follows from using the 2-to-1 squaring map to reduce the evaluation set of roots of unity of order 2^k to similar groups of size $2^{k-i}, i > 0$. Our algorithms operate similarly, using isogenies of elliptic curves with kernel size 2 as 2-to-1 maps to reduce L of size 2^k to sets of size 2^{k-i} that are, like L, suitably related to elliptic curves, albeit different ones.

1 Introduction

The rocket fuel that powers modern fast algorithms for polynomial algebra is the Fast Fourier Transform (FFT). The original FFT, due to Cooley–Tukey [CT65][1], is a divide-and-conquer algorithm that evaluates a polynomial $P(X) = \sum_{i<n} a_i X^i \in \mathbb{C}[X]$, given by its sequence of coefficient $(a_0, \ldots, a_{n-1})$, on the nth

*StarkWare Industries Ltd. {eli,dancar,david}@starkware.co

†Department of Mathematics and Department of Computer Science, University of Toronto. Research supported in part by NSF grants CCF-1540634 and CCF-1814409, at Rutgers University. swastik.kopparty@gmail.com

[1]The history of this algorithm is much longer, and dates back to Gauss, see [HJB85].

ISSN 1433-8092

roots of unity in $\mathbb{C}$. It does so using $O(n \log n)$ arithmetic operations over $\mathbb{C}$ whenever n is an integer power of 2, or more generally, when n is a *smooth number* – a product of $O(1)$-sized primes. This immediately enables $O(n \log n)$ time multiplication of polynomials of degree $< n/2$ – by evaluation at the nth roots of unity, pointwise multiplication of these evaluations, and then interpolation from the nth roots of unity via the *inverse* FFT (iFFT) algorithm. Polynomial multiplication turns out to be the crucial operation for a wide variety of other algorithmic problems of polynomial algebra. See the books [vzGG13, BCS97] for a taste of the impact of the FFT on computer algebra.

Over finite fields $\mathbb{F}_q$, these ideas generalize to some extent [Pol71]. Define $M_q(n)$ to be the number of $\mathbb{F}_q$ operations needed for the fastest algorithm over $\mathbb{F}_q$ which takes as input the coefficients of two polynomials in $\mathbb{F}_q[X]$ of degree $< n$, and returns the coefficients of their product. Using the same FFT algorithm, if $\mathbb{F}_q$ contains an nth root of unity for smooth n, we have $M_q(n) = O(n \log n)$. More generally, we get the same upper bound on $M_q(n)$ even if a bounded degree extension field $\mathbb{F}_{q^{O(1)}}$ contains such a root of unity which generates a multiplicative subgroup of smooth order. However, most finite fields are not "special" in this way, which raises the following well-known open problem:

Open Question 1: Does the bound $M_q(n) = O(n \log n)$ hold for all prime powers q and all n?

Until recently, the best general upper bound on $M_q(n)$ was the classical result of Schönhage and Strassen [SS71] (see also Schönhage [Sch77] and Cantor–Kaltofen [CK91]), who showed that:

$$M_q(n) = O(n \log n \log\log n).$$

This algorithm involves introducing a synthetic root of unity and recursively running FFTs over more general rings. The algorithm is inspired by, and closely mirrors, the classical (Boolean) algorithm of Schönhage and Strassen for integer multiplication, which shows that $M_{\mathbb{Z}}(n)$, the Boolean circuit complexity of multiplying two n-bit integers presented in base 2, satisfies:

$$M_{\mathbb{Z}}(n) \leq O(n \log n \log\log n).$$

Remark 1.1 (Computational Model). Unless explicitly specified otherwise, we use the word "algorithm" to mean an algebraic algorithm that uses only field operations and field constants. In particular, we do not consider any precision issues or the cost of computing the constants used by the computation. This computational model is more commonly known as an arithmetic circuit or a straight-line program. When we refer to the running time of such an algorithm, we mean the size of the straight-line program or arithmetic circuit, which means we assign unit computational cost to each arithmetic operation over the ambient field.

As in the case of $\mathbb{C}$, the best known algorithms for a wide variety of algorithmic problems of polynomial algebra over $\mathbb{F}_q$ depend on polynomial multiplication over $\mathbb{F}_q$, and thus their running time depends on $M_q(n)$. Of particular interest are the following classical results.

1. Horowitz [Hor72b, Hor72a] gave an algorithm for polynomial interpolation at n points *with preprocessing* in time $O(M_q(n) \log^2 n)$. Here we are given a subset B of $\mathbb{F}_q$ of size n and a function $f : B \to \mathbb{F}_q$, and after doing arbitrary preprocessing of B, we want to compute the coefficients of the unique polynomial of degree $< n$ that interpolates f.

2. In the above mentioned paper, Horowitz [Hor72b] presented a fast algorithm for evaluating all elementary symmetric polynomials over n variables on a specific input $(\alpha_1, \ldots, \alpha_n)$ in time $O(M_q(n) \log n)$.

3. Subsequently, Borodin and Moenck [BM74] improved Horowitz's algorithm and gave an algorithm for polynomial interpolation at n points *without preprocessing* in time $O(M_q(n) \log n)$.

4. Along the way, Borodin and Moenck [BM74] also showed how to do multi-point evaluation of degree $< n$ polynomials at n arbitrary points in time $O(M_q(n) \log n)$.

In recent years, there have been some remarkable advances in our understanding of the complexity of multiplying polynomials over finite fields. These advances closely track breakthroughs on the fundamental problem of understanding the complexity of multiplying integers in the Boolean circuit or (multi-tape) Turing Machine model. The starting point for all these recent advances was the result of Fürer [Für07] (see also [DKSS08]) who showed that $M_{\mathbb{Z}}(n) = O(n \log n \cdot 2^{O(\log^* n)})$. Soon after, Harvey, van der Hoeven and Lecerf [HvdHL17] simplified and improved the constant in the exponent in Fürer's bound on $M_{\mathbb{Z}}(n)$, while also developing an $\mathbb{F}_q$-analogue of this algorithm to show that $M_q(n) = O(n \log n \cdot 2^{O(\log^* n)})$. Harvey and van der Hoeven [HvdH19a] further improved the constant in the exponent in the bound on $M_q(n)$.

Finally, Harvey and van der Hoeven [HvdH21] proved the breakthrough $M_{\mathbb{Z}}(n) = O(n \log n)$, settling a long-standing conjecture. There they discussed the reasons why their results do not extend to a similar bound on $M_q(n)$. Nevertheless, their results do imply (via Kronecker substitutions, see Section 1.2 of [HvdH19a]) that multiplication of degree n polynomials over $\mathbb{F}_q$ for $n = q^{O(1)}$, can be done in time $O(n \log q(\log n + \log\log q))$ in the Turing machine model, which seems to be as good a bound one can hope to deduce in the Turing Machine model from the conjectured $M_q(n) = O(n \log n)$.

Returning to $M_q(n)$, Harvey and van der Hoeven showed in [HvdH19b, Theorem 9.2], which is a companion paper to [HvdH21], that under a number theoretic conjecture on the least prime in arithmetic progressions, $M_q(n)$ is indeed $O(n \log n)$.

Summarizing, the recent wave of results come extremely close to answering Open Question 1 unconditionally, but we are not quite there yet.

1.1 Our Results

The main contribution of our paper is a new approach to fast polynomial algorithms via a new polynomial representation that works over all large finite fields. The approach is very closely related to the classical FFT algorithm, but instead of working with subgroups of $\mathbb{F}_q$ of smooth order (be they multiplicative or additive), it works with *elliptic curve groups* with large, smooth order subgroups, *which exist for all* $\mathbb{F}_q$.

Our approach is unrelated to all the recent results mentioned above, and unconditionally yields some new results that would follow if $M_q(n) = O(n \log n)$ were true.

The new representation for polynomials suggested here is essentially the evaluation tables of the polynomials at carefully chosen subsets of $\mathbb{F}_q$. These sets are related to some subgroup of some elliptic curve over $\mathbb{F}_q$. This is the analogue of taking multiplicative/additive subgroups of large, smooth order, which is only possible when q is special—either a power of a constant prime or such that $q-1$ is divisible by a large smooth factor.

In the classical multiplicative subgroup based FFT, we can convert the evaluation table representation into the standard coefficient representation in time $O(n \log n)$ via the classical inverse FFT. Unfortunately, in our elliptic curve group case, we do not know how to do this conversion as fast. What we can do instead is to quickly *extend* the evaluation of the polynomial on our chosen subset S to another subset S' of $\mathbb{F}_q$. This is the analogue of using a combination of FFT and inverse-FFT (with some scaling) to use the evaluations of some low degree polynomial at a multiplicative subgroup S to deduce the evaluations of that low degree polynomial at some coset of S. In fact, the way we compute the low degree extension to the subset S' is also a combination of some FFT-like transform (which we call the ECFFT) and the inverse transform. It just so happens that the intermediate representation, i.e., the result of our iFFT-analogue, is not the standard monomial expansion of the polynomial, but some other representation. In this respect, our approach resembles the additive FFT-like transforms of [LCH14] which also lead to non-monomial representations supporting fast operations (see also [GM10, Can89]); however, their algorithms have S, S' being additive subgroups of $\mathbb{F}_q$, and require $\mathbb{F}_q$ to have constant characteristic to have $O(n \log n)$ running time.

We systematically exploit the above-mentioned fast algorithm for extending polynomial evaluations on special sets to develop fast algorithms[2] for a variety of polynomial computation problems, giving the following results, defined formally in Section 6:

[2]We remind the reader that the model of computation is algebraic circuits (and for one problem, algebraic decision trees), where the circuit may depend arbitrarily on n and q. The preprocessing cost of setting up this circuit for a given n or q, which

1. When polynomials of degree less than n over $\mathbb{F}_q$, $n \leq q^{O(1)}$, are represented as evaluations over special sets, the following three operations can all be done in time $O(n \log n)$:

   (a) addition,

   (b) multiplication, and

   (c) degree computation[3]

   Note that addition trivially takes $O(n)$ time for polynomials evaluated on any set of points, as does multiplication of polynomials whose degrees sum to less than n; the crux here is that polynomials can still be multiplied in quasi-linear time even if their product has degree above n, by extending the evaluations to a larger set, supporting higher degrees. Degree compuation is also non-trivial, as the polynomials are not represented directly by their coefficients.

   As far as we know, this is the only known representation of polynomials that allows all the above three operations to be computed in $O(n \log n)$ algebraic operations for general q and $n \leq q^{O(1)}$.

   A folklore question, which was recently resolved by the breakthrough on integer multiplication [HvdH21], asked to find a representation of integers that supports addition, multiplication and comparison in $O(n \log n)$ time. Our result can be viewed as a positive answer to the analogous question for polynomials over arbitrary finite fields.

2. We develop fast algorithms for other basic operations on these representations, such as division with remainder and Chinese remaindering, modulu fixed polynomials.

3. Converting between our new representation and the standard representation of polynomials by their monomial coefficients (in both directions) can be done in time $O(n \log^2 n)$.

Armed with these tools for working with polynomials in the new representation, we get the following new results for classical problems that have nothing to do with the new representation. All these results improve on the state of the art by a multiplicative $\exp(\log^* n)$ factor, and are consequences of the conjectured bound $M_q(n) = O(n \log n)$. See Section 7 for the formal statements.

1. We give an $O(n \log^2 n)$ time algorithm to evaluate all n elementary symmetric polynomials on n inputs, provided $n \leq q^{O(1)}$. It was not known how to do this in general for all $n \leq q^{O(1)}$ even for the computation of just the $n/2$-th elementary symmetric polynomial.

2. Given an arbitrary set B of n points, we give an $O(n \log^2 n)$ time algorithm for interpolating a polynomial (and representing it in the standard monomial basis) from its evaluation on B (we allow preprocessing based on B).

3. We give an $O(n \log^2 n)$ time algorithm for multi-point evaluation of a degree $< n$ polynomial at an arbitrary set B of n points (here, too, we allow preprocessing based on B).

4. Combining the above two results, we get a an $O(n \log^2 n)$ time algorithm for computing low-degree extensions of function evaluated at n arbitrary points to n other arbitrary points. The two sets of points are assumed to be known in advance, and preprocessed to derive constants used by the algorithm.

We believe this representation will have further uses in the development of fast algorithms for polynomial algebra. The most compelling question here is whether these methods can improve the bound on $M_q(n)$ itself. It is also interesting to see if we can do away with the need for preprocessing in the above algorithms.

in our case involves searching for a suitable elliptic curve, is not included in the complexity bounds. Under standard number theoretic heuristics, this preprocessing can be done by a randomized Turing machine in $O(n \cdot \mathsf{poly}(\log n, \log q))$ time. Details will appear in [BCKL21].

[3]The formal model for this is Algebraic Decision Tree (since the output is an integer), and by "running time" for this model we mean the depth of this tree.

Further applications in Part II [BCKL21]: The applications of FFT-like divide and conquer for polynomials is not limited to the design of fast algorithms. In a sequel to this paper (which is oriented towards applied cryptography), we explore applications of the Elliptic Curve based Fast Fourier Transforms to interactive oracle proofs (IOPs), IOPs of proximity (IOPPs) for algebraic geometry codes and scalable transparent arguments of knowledge (STARK) systems, generalizing the use of the standard FFT in PCPPs for Reed–Solomon codes [BS08], the FRI protocol for proving proximity to Reed–Solomon codes [BBHR18], and the STARK protocol and analogous transparent IOP based proof systems for verifying general computation [BBHR19, BCR+19, COS20, Sta21]. Because of applications of the latter two to cryptography in the real world, where the natural field of definition of the problems is specified by external sources, there is a natural need to prove computational integrity statements about computations of length n executed over specific finite fields $q \gg n$. For example, the q used in the ECDSA algorithm that is part of the Bitcoin standard is such that $q-1$ has no large smooth factor, and this is also the case for any q which is a "safe prime" which means that $(q-1)/2$ is a large prime. Indeed, such examples were the original motivation for looking for generalizations of FFTs to all fields, and resolving it requires a deeper scrutiny of the ECFFT, used here only as a "black-box", and several other ideas.

1.2 ECFFT – Informal Explanation

The standard FFT algorithm exploits the structure of the group of 2^{k}-th roots of unity and its subgroups, using the squaring map $x \mapsto x^2$ to simultaneously (i) project the group of size n to a subgroup of half the size and (ii) split a polynomial of degree n into two polynomials of half the degree, expressed using the squaring map.

Let $n = 2^{\mathsf{k}}$, and suppose we are working in a field $\mathbb{F}$ which contains all n of the nth roots of 1. Let $L^{(0)} \subseteq \mathbb{F}$ denote all the nth roots of 1, assuming we wish to represent polynomials of degree $< n$ by evaluating them on $L^{(0)}$. Let $\psi(X) = X^2$ be the squaring map. For each i, let $L^{(i+1)} = \psi(L^{(i)})$. Thus $L^{(i)}$ is the set of $\frac{n}{2^i}$th roots of unity in $\mathbb{F}$ and ψ is a 2-to-1 map of degree 2 from $L^{(i)}$ onto $L^{(i+1)}$. Thus far we have described how ψ is used to "compress" an evaluation set $L^{(i)}$ to a smaller evaluation set $L^{(i+1)}$ of half the size. Simultaneously, ψ can be used to "split" a polynomial presented in the standard monomial basis thus:

$$P(X) = \sum_{i<n} a_i \cdot X^i = \left(\sum_{i<n/2} a_{2i} \cdot \psi(X)^i \right) + X \cdot \left(\sum_{i<n/2} a_{2i+1} \cdot \psi(X)^i \right) = P_0(\psi(X)) + X \cdot P_1(\psi(X)).$$

The FFT evaluates P on $L^{(0)}$ by recursively evaluating both $P_0(Y)$ and $P_1(Y)$ on $y \in \psi(L^{(0)}) = L^{(1)}$ and then combining the results using $O(n)$ operations via the formula above. The running time $F(n)$ satisfies the recursive formula $F(n) = 2 \cdot F(n/2) + O(n)$ leading to $O(n \log n)$ running time.

The essential elements we preserve in our ECFFT are the usage of degree-2 maps $\psi^{(i)}$ that are 2-to-1 maps on special sets of points $L^{(i)}$ of size $\frac{n}{2^i}$, along with the ability to express a polynomial $P(X)$ of degree $< n$ in terms of two other polynomials $P_0(\psi^{(i)}(X)), P_1(\psi^{(i)}(X))$ of degree $< n/2$, such that the value of $P(x), x \in L^{(i)}$ can be obtained "locally" from the values of $P_0(\psi^{(i)}(x)), P_1(\psi^{(i)}(x))$. Thus, we use such maps and sets of points to describe new FFTrees. An FFTree (see Definition 3.3) is an "FFT-inspired" object that is a layered binary tree whose nodes residing at the ith layer are labeled by the members of $L^{(i)}$, and such that the 2-to-1 map $\psi^{(i)}$ defines directed edges from two elements $s_0, s_1 \in L^{(i)}$ to $t = \psi^{(i)}(s_0) = \psi^{(i)}(s_1) \in L^{(i+1)}$.

So far we have listed similarities between the FFT and our new ECFFT, so let us now describe the differences. First, our set $L^{(i)}$ is not a multiplicative group, and in fact it is not a group at all (soon, in Section 1.3, we'll explain what $L^{(i)}$ actually is). But examining the classical FFT, we could do its first step using *any* degree-2 polynomial $\psi(X)$ which is 2-to-1 on some set of points $L^{(0)}$ (mapping it to an arbitrary set of points $L^{(1)}$ of size $n/2$). The group structure is useful for knowing, recursively, that we can find further 2-to-1 maps from $L^{(1)}$ to $L^{(2)}$ and so on. A second point of difference is that our 2-to-1 maps may vary with i, whereas the classical FFT uses only squaring[4] to move from $L^{(i)}$ to $L^{(i+1)}$. Finally, the maps $\psi^{(i)}$ we

[4]When n is factored into different prime factors (say, $n = 2^a \cdot 3^b$) one would also use different maps in the FFT (say, squaring and cubing) to move between $L^{(i)}$ and $L^{(i+1)}$, and varying maps are also used in additive FFTs [GM10, Can89, LCH14].

use are not degree-2 polynomials but rather degree-2 rational maps, ratios of two degree-2 polynomials. We show that any such map is just as good for the purpose of splitting a polynomial into two subpolynomials of half the degree (see Lemma 3.1), and using rational maps rather than polynomials gives us more degrees of freedom when searching for 2-to-1 maps on special sets of points. These points, and the way they are obtained, are our next, and main, point in this intuitive description of the ECFFT.

1.3 Elliptic Curves as a Source for FFTrees over Arbitrary Finite Fields

Elliptic curves are a vast topic of study, with wide-ranging impact across mathematics (e.g., [Wil95]), and we shall not attempt to describe their importance here. An elliptic curve E over the finite field $\mathbb{F}_q$ is defined by a suitable polynomial $C(X,Y) \in \mathbb{F}_q[X,Y]$, and the solutions $(x,y) \in \mathbb{F}_q^2$ of $C(X,Y) = 0$ are the points of interest (the description here is intentionally simplified, see Section 4.1 for a formal and accurate definition). Elliptic curves have some remarkable properties that have led to a number of significant and surprising applications in theoretical computer science. A small selection of notable examples include: Lenstra's elliptic curve method for factoring integers [Len87]; Schoof's deterministic algorithm for finding square roots modulo a prime [Sch85]; cryptosystems, starting with Miller's EC Diffie–Hellman (ECDH) key exchange [Mil86] and Koblitz's EC integrated encryption scheme (ECIES) [Kob87, ABR99] and including Vanstone's EC digital signature algorithm (ECDSA) [Van92] and applications based on pairings, such as Joux's one-round 3-way key agreement [Jou04] and the Boneh–Franklin identity based encryption protocol [BF03].

We remark that Lenstra's method for factoring integers using elliptic curves [Len87] in particular was a major inspiration for this paper. Lenstra's method is a generalization of Pollard's $p-1$ algorithm for factoring [Pol74]: The $p-1$ method only works when, for some prime factor p, the multiplicative group $\mathbb{F}_p^\times$ has a special property, which is only true for few primes p. Lenstra's method extends the $p-1$ method to all possible p's by replacing the group $\mathbb{F}_p^\times$ with elliptic curves. Very similarly, the standard FFT works inside $\mathbb{F}_q$ only when the field has special roots of unity, which is true only for sporadic q, and this paper extends core applications of FFT to all prime powers q by replacing the group $\mathbb{F}_q^\times$ with elliptic curves.

The main properties of elliptic curves that we use are:

- The number of points on the curve E can be nearly any number in the range $[q \pm 2\sqrt{q} + 1]$ (see Section 4.1.4 for a precise discussion of the number of points).
- These points form an abelian group, called, appropriately, an *elliptic curve group*. Varying over curves, and acknowledging the previous point, elliptic curve groups could be of nearly any size in $[q \pm 2\sqrt{q} + 1]$. In particular, we can find subgroups G of elliptic curve groups of size $n = 2^k$ for $n = O(\sqrt{q})$ (see Theorem 4.4 and Claim 4.6).
- 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 from the points of the curve E to points on a different curve E' over $\mathbb{F}_q$. Thus, the image of G under the isogeny is of size $|G|/|H|$.

The observations above give us nearly all that we need. We can find a set of points $G^{(0)}$ inside a curve $E^{(0)}$ that is a group of size $2^{\mathsf{k}} \leq O(\sqrt{q})$ irrespective of the exact nature of q, and we have at our disposal isogenies that "compress" groups of points $G^{(i)}$ to groups $G^{(i+1)}$ half the size via 2-to-1 maps $\phi^{(i)}$, where the new group $G^{(i+1)}$ belongs to a different curve $E^{(i+1)}$. The only remaining gap is that elements in the groups $G^{(i)}$ are *pairs* $(x,y) \in \mathbb{F}_q^2$ whereas we are interested in univariate polynomials and evaluation sets over $\mathbb{F}_q$. The final ingredient is to pick curves represented in a certain format (extended Weierstrass form) such that suitably shifting and then projecting $G^{(i)}$ to the x coordinate gives a set $L^{(i)} \subset \mathbb{F}_q$ that is the same size as $G^{(i)}$ and, crucially, the isogeny map $\phi^{(i)}$ gives rise to a degree-2 rational map that is 2-to-1 from $L^{(i)}$ onto $L^{(i+1)}$ (see Proposition 4.1 and Theorem 4.9).

Remark 1.2. The degree-2 (or higher degree) maps so obtained are generalizations of *Lattés maps* [Lat18] (see [Sil07]). Lattés maps are the rational maps arising from the x-coordinate mapping of isogenies from an elliptic curve to *itself*. The rational maps that underlie the FFTree construction arise from the x-coordinate

mapping of isogenies from an elliptic curve E to some other elliptic curve E', which may or may not equal E.

Summarizing, the abundance of elliptic curve groups of various sizes over any large finite field assures us that we'll find a subgroup of smooth size; isogenies and their projections give 2-to-1 degree-2 rational maps from sets of size 2^{k} (in $\mathbb{F}_q$) to sets of size $2^{\mathsf{k}-1}$ for all needed k, and thereby we have the needed FFTree structure which leads to efficient FFT-like running times for all finite fields.

Organization of paper The following Section 2 gives notation. Section 3 defines and discusses (i) the FFTree data structure and (ii) the polynomial decomposition lemma (using rational maps); these two ingredients are needed to abstract and generalize the classical FFT algorithm to arbitrary sets of points and maps. Section 4 instantiates FFTrees and decomposition maps using elliptic curve and projections of isogenies, showing that the necessary data structures exist over all large finite fields. Section 5 defines the way we represent polynomials for efficient operations – by evaluating them over the special sets of points that arise from the previously defined FFTrees. Section 6 presents fast algorithms for fundamental operations applied to polynomials that are represented in this special way. Finally, Section 7 uses these efficient algorithms to efficiently solve "classical" problems about polynomials, like interpolation, evaluation over general sets of points, and computation of elementary symmetric polynomials.

2 Notation

2.1 Functions and Polynomials

For $g : D \to R$ a function and $S \subset R$ denote by $g^{-1}(S)$ the set of g-preimages of S, namely $g^{-1}(S) = \{x : g(x) \in S\}$, and for $u \in R$ let $g^{-1}(u) = g^{-1}(\{u\})$. Likewise for $D' \subset D$ we let $g(D') = \{g(x) : x \in D'\}$.

For a set $A \subseteq \mathbb{F}_q$, we define *the vanishing polynomial* of A to be the polynomial $Z(X) \in \mathbb{F}_q[X]$ given by:

$$Z(X) = \prod_{\alpha \in A} (X - \alpha).$$

We define:

$$B(X) \mathsf{\ rem\ } A(X)$$

to be the unique polynomial with degree $< \deg(A)$ which is congruent to $B(X) \bmod A(X)$.

When $B(X), A(X)$ are coprime polynomials, we define

$$(B(X))^{-1}_{A(X)}$$

to be the unique polynomial $C(X)$ with degree $< \deg(A)$ such that $B(X) \cdot C(X) \equiv 1 \pmod{A(X)}$.

2.2 Projective Space

We denote by $\mathbb{P}^n(\mathbb{F}_q)$ (or simply $\mathbb{P}^n$) the *n-dimensional projective space* over $\mathbb{F}_q$; only $\mathbb{P}^1$ and $\mathbb{P}^2$ will appear in the paper. Points in $\mathbb{P}^n$ are given by homogenized coordinates $[x_1 : x_2 : \cdots : x_{n+1}]$ where at least one x_i is non-zero, and with the equivalence relation

$$[x_1 : x_2 : \cdots : x_{n+1}] \sim [cx_1 : cx_2 : \cdots : cx_{n+1}], \quad \forall c \neq 0.$$

Points in the *affine* space $\mathbb{F}_q^n$ are given by affine coordinates $(x_1, \ldots, x_n)$, and in this paper we equate such points with their standard embedding into projective space, i.e.

$$(x_1, \ldots, x_n) = [x_1 : \cdots : x_n : 1].$$

Thus, $\mathbb{P}^n$ is the disjoint union of $\mathbb{F}_q^n$ and a copy of $\mathbb{P}^{n-1}$ "at infinity", i.e. with an additional $x_{n+1} = 0$ coordinate. In particular, $\mathbb{P}^1(\mathbb{F}_q) = \mathbb{F}_q \cup \{\infty\}$, where ∞ denotes the unique point at infinity, $[1 : 0]$.

We will refer to the two coordinates of the affine plane $\mathbb{F}_q^2$ as x and y. For a point $P \in \mathbb{F}_q^2$, we will denote its x, y coordinates by P_x, P_y, respectively. For a point $P \in \mathbb{P}^2$, the coordinates P_x, P_y will only be defined if it is an affine point, according to the above notation.

2.3 Rational functions

Rational functions over $\mathbb{F}_q$ are quotients $R(X) = P(X)/Q(X)$ where $P(X), Q(X) \in \mathbb{F}_q[X]$ are coprime polynomials and Q is non-zero. Rational functions form a field, denoted by $\mathbb{F}_q(X)$.

Rational functions can be considered as maps from $\mathbb{P}^1$ to itself, where zeros of Q are mapped to ∞ and are called *poles* of the rational function, with multiplicity equal to their multiplicity as zeros of Q. Depending on whether $\deg(P) - \deg(Q)$ is positive, negative, or zero, the point ∞ is either a pole of multiplicity $\deg(P) - \deg(Q)$, a zero of multiplicity $\deg(Q) - \deg(P)$, or mapped to the ratio between the leading coefficients of P and Q, correspondingly.

The *degree* of R is defined as $\deg(R) \coloneqq \max(\deg(P), \deg(Q))$, and is equal to both the total number of zeros and the total number of poles of R, including at ∞, counted with multiplicity.

3 Polynomial decompositions and FFTrees

In this section we show that any rational map can be used to decompose a polynomial into lower degree polynomials, in a way similar to how the squaring map is used in FFTs (see Lemma 3.1). We then define a generalized notion of FFT-like sets of evaluation points (Section 3.2). In the next section we shall instantiate both of these—rational maps and FFTrees—using elliptic curve groups.

3.1 Polynomial decompositions based on rational functions

Let V_d be the $\mathbb{F}_q$-linear subspace of $\mathbb{F}_q[X]$ consisting of polynomials of degree strictly less than d. A crucial component in the standard FFT is the decomposition of a polynomial $P(X) = \sum_{i<d} a_i X^i \in V_d$ into two polynomials in $V_{d/2}$, one containing the terms of even degree and the other containing the terms of odd degree:

$$P(X) = \left(\sum_{i<d/2} a_{2i} \left(X^2\right)^i \right) + X \cdot \left(\sum_{i<d/2} a_{2i+1} \cdot \left(X^2\right)^i \right) = P_0(X^2) + X \cdot P_1(X^2). \tag{1}$$

The results of this section generalize this partition by replacing X^2 with any rational function. Later, we shall instantiate the results of this section with rational functions coming from projections of isogenies of elliptic curves. We state the decomposition lemma next; its proof appears in Appendix A.

Lemma 3.1 (Decomposition)**.** *Let $\psi(X) \in \mathbb{F}_q(X)$ be a rational map given by:*

$$\psi(X) = \frac{u(X)}{v(X)},$$

where $u(X), v(X) \in \mathbb{F}_q[X]$ are relatively prime polynomials. Let $\delta = \deg(\psi) = \max\{\deg(u), \deg(v)\}$. Let d be a multiple of δ. Then for every $P(X) \in V_d$, there is a unique tuple:

$$(P_0(X), P_1(X), \ldots, P_{\delta-1}(X)) \in (V_{d/\delta})^\delta$$

such that:

$$P(X) = \left(\sum_{i=0}^{\delta-1} X^i \cdot P_i(\psi(X)) \right) \cdot v(X)^{\frac{d}{\delta}-1}. \tag{2}$$

The next statement says that, as in the case of the standard FFT, moving between the two representations of Eqs. (1) and (2) is done via a set of δ-local invertible linear transformations.

Lemma 3.2 (Locality and invertibility)**.** *Let $t \in \mathbb{F}_q$. Keeping the notation of the previous lemma, suppose $\psi^{-1}(t) = \{s_0, \ldots, s_{\delta-1}\}$ is a set of elements of $\mathbb{F}_q$ of size exactly δ. Then the transformation*

$$M_t : \mathbb{F}_q^\delta \to \mathbb{F}_q^\delta, \quad M_t(P(s_0), \ldots, P(s_{\delta-1})) \mapsto (P_0(t), \ldots, P_{\delta-1}(t)) \tag{3}$$

is linear and invertible.

Proof. The assumption $t \in \mathbb{F}_q$ and, in particular, $t \neq \infty$, implies $v(s_j) \neq 0$ for each s_j. The relationship between the $P(s_j)$ and the $P_i(t)$ is captured by the following system of linear equations:

$$P(s_j) = \left(\sum_{i=0}^{\delta-1} s_j^i \cdot P_i(t) \right) \cdot v(s_j)^{\frac{d}{\delta}-1}.$$

Inspection shows that the underlying matrix is a nonsingular Vandermonde matrix with rows scaled by nonzero scalars. □

For the rest of this paper, we will focus on the $\delta = 2$ case, although everything generalizes to larger δ. We briefly instantiate the above lemmas in this case, to expose the similarity to the classical FFT.

Let $\psi(X)$ be a degree 2 rational function. Suppose d is even. Fix any $P(X) \in V_d$, and consider the two polynomials $P_0(X), P_1(X)$ given by Lemma 3.1. Then we have the following decomposition that resembles the classical FFT case of Eq. (1):

$$P(X) = (P_0(\psi(X)) + XP_1(\psi(X))) \cdot (v(X))^{\frac{d}{2}-1},$$

and so, for any $s \in \mathbb{F}_q$:

$$P(s) = (P_0(\psi(s)) + sP_1(\psi(s))) \cdot (v(s))^{\frac{d}{2}-1}. \tag{4}$$

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 Lemma 3.2 implies that the values $P(s_0), P(s_1)$ determine $P_0(t), P_1(t)$ and vice versa (this uses the fact that $s_0 \neq s_1$), and the transformation between the two pairs of values is computed by multpilication by an invertible 2×2 matrix, whose coefficients depend only on the values of $s_0, s_1, v(s_0)$, and $v(s_1)$.

Thus, when we have a degree 2 rational function ψ that is 2-to-1 from S to $\psi(S) = T$, finding evaluations of a polynomial $P(X)$ at the points of S is equivalent to finding evaluations of $P_0(X)$ and $P_1(X)$ at the points of T.

3.2 FFTrees

We now define FFTrees, a structure abstracting out relevant properties of evaluation sets and maps between them, which suffice to simulate an FFT-like algorithm.

Definition 3.3 (FFTrees)**.** *Let q be a prime power, and let k be an integer. An* FFTree *over $\mathbb{F}_q$ of* depth k *is a collection of subsets $L^{(0)}, L^{(1)} \ldots, L^{(\mathsf{k})} \subseteq \mathbb{F}_q$ along with degree 2 rational functions $\psi^{(i)}(X) \in \mathbb{F}_q(X)$ such that:*

1. *$|L^{(i)}| = 2^{\mathsf{k}-i}$.*
2. *$\psi^{(i)}(L^{(i)}) = L^{(i+1)}$ (and so $\psi^{(i)}$ is a 2-to-1 map from $L^{(i)}$ to $L^{(i+1)}$).*

Let $\mathcal{F}$ denote the rooted, layered, binary tree, whose layers are indexed by $i \in \{0, 1, \ldots, \mathsf{k}\}$. The set of vertices in layer i is $L^{(i)}$. The root of $\mathcal{F}$ is the unique element of $L^{(\mathsf{k})}$. The leaves of $\mathcal{F}$ are all the vertices in $L^{(0)}$. For each $i < \mathsf{k}$, the parent of the vertex $s \in L^{(i)}$ of the i-th layer is the vertex $\psi^{(i)}(s) \in L^{(i+1)}$ of the $(i+1)$st layer.

Because of the decomposition lemma, evaluations of a polynomial on $L^{(i)}$ can be deduced from evaluations of 2 related lower degree polynomials on $L^{(i+1)}$, and this serves as the basis for fast "divide and conquer" algorithms.

Our eventual use of FFTrees will be as follows. We will first fix an FFTree over $\mathbb{F}_q$. We will use L to denote $L^{(0)}$. Let $\mathsf{K} = |L| = 2^{\mathsf{k}}$. Then for any $n \leq \mathsf{K}$, polynomials of degree $< n$ will be represented by evaluations at specific subsets of L of size $O(n)$. The FFTree structure will then enable fast algorithms for working with these representations.

Thus any given FFTree will be useful for working with polynomials of degree up to $2^{\mathsf{k}} - 1$. Therefore it is interesting to find FFTrees with as large depth k as possible.

In the next section, we use elliptic curves to show the existence of FFTrees over $\mathbb{F}_q$ with depth $\Omega(\log q)$.

4 FFTrees from Elliptic Curves

In this section we prove the existence of FFTrees of depth $\Omega(\log q)$ in any finite field $\mathbb{F}_q$. Specifically, we show that there exist FFTrees over $\mathbb{F}_q$ whose base set $L^{(0)}$ has size $\Omega(\sqrt{q})$. We start by recounting the necessary definitions and results regarding elliptic curves. In Section 4.2 we then prove our main results about existence of FFTrees using rational maps that are projections of isogenies.

4.1 Background on elliptic curves and isogenies

In this subsection we provide a brief overview of the necessary definitions and theorems regarding elliptic curves. Further details and proofs can be found in most basic texts on the subject. Except where specifically noted, all results can be found in [Sil09] or [Was08].

4.1.1 Elliptic curve in Weierstrass form

An *elliptic curve* E is a smooth, projective, algebraic curve of genus 1, with a special marked point O, defined over a field. In this paper all curves will be defined over the finite field $\mathbb{F}_q$. Every elliptic curve can be presented in *extended Weierstrass form* as the set of planar points $(x, y) \in \mathbb{F}_q^2$ satisfying a cubic equation

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \tag{5}$$

or equivalently

$$F(X, Y) := Y^2 + a_1XY + a_3Y - X^3 - a_2X^2 - a_4X - a_6 = 0 \tag{6}$$

parameterized by a_1, a_2, a_3, a_4, a_6, together with the marked point $O = [0 : 1 : 0] \in \mathbb{P}^2(\mathbb{F}_q)$, called its *point at infinity*.

4.1.2 The group law

The points of an elliptic curve E form an abelian group, in which O is the neutral element, and any three distinct points $P, Q, R \in E$ satisfy $P + Q + R = O$ iff they are colinear. If $P = Q \neq R$, the condition is that the tangent to E at P passes through R, and if $P = Q = R$ the condition is that the tangent at P to E is doubly tangent at the point.

Lines passing through O are either the line at infinity (which is doubly tangent to E at O), or lines of the form $X = c$. Thus $P + Q = O$, i.e. $P = -Q$, iff their coordinates satisfy $P_x = Q_x$ and $P_y \neq Q_y$; or $P = Q \neq O$ and the line $X = P_x$ is tangent to E at P; or $P = Q = O$. Note that in both affine cases, we also have $Q_y = -a_1P_x - a_3 - P_y$, since P_y, Q_y are the two (not necessarily distinct) roots of a monic quadratic in y with linear coefficient $a_1P_x + a_3$.

4.1.3 Isogenies and x-projection

For a curve E in extended Weierstrass form, let $\pi : E \to \mathbb{P}^1$ denote the projection to the x-coordinate, defined by $\pi(O) = \infty \in \mathbb{P}^1$ and $\pi(P) = P_x \in \mathbb{F}_q$ for $P \in E \setminus \{O\}$. Additionally, as noted in Section 4.1.2, for any $P, Q \in E$, $\pi(P) = \pi(Q)$ if and only if $P = \pm Q$, thus the preimages $\pi^{-1}(\pi(P)) = \{\pm P\}$ are either sets of size two, or a singleton $\{P\}$ when $2P = O$. In particular, it follows that for any subset $C \subset E$ such that C is disjoint from $-C = \{-P : P \in C\}$, the map $\pi|_C$ is 1-to-1 from C to $\mathbb{F}_q$.

Let E, E' be elliptic curves over the same field. An *isogeny* between the curves is a rational map $\phi : E \to E'$ satisfying $\phi(O) = O'$, where O' is the neutral element of E'. We follow [Was08, Chapters 2.9, 12.2] to give an algebraic, rather than geometric, description of isogenies. When E, E' are in extended Weierstrass form, ϕ can be expressed in a *standard form*:

Proposition 4.1. *Let $\phi : E \to E'$ be an isogeny between two curves in extended Weierstrass form. Then, in coordinates, we may write*

$$\phi(x, y) = (\psi(x), \xi(x, y)),$$

where $\psi : \mathbb{P}^1 \to \mathbb{P}^1$ is a rational function. Equivalently, if $\pi : E \to \mathbb{P}^1, \pi' : E' \to \mathbb{P}^1$ are the x-projection maps in each curve, then there exists a unique rational function ψ such that the diagram

$$\begin{array}{ccc} E & \xrightarrow{\phi} & E' \\ {\scriptstyle\pi}\downarrow & & \downarrow{\scriptstyle\pi'} \\ \mathbb{P}^1 & \xrightarrow{\psi} & \mathbb{P}^1 \end{array}$$

is commutative.

This fact appears to be folklore, and is most commonly discussed only in the special case of curves in *short* Weierstrass form $E : y^2 = x^3 + Ax + B$, where $\xi(x, y)$ can also be expressed as y times a rational function—see [Was08, Chapter 2.9] for a discussion of this case. When focusing only on the x-coordinate, the same proof is valid also for the extended Weierstrass form. For completeness, a full proof of this fact is included in Appendix B.1.

Definition 4.2. *Let $\phi : E \to E'$ be an isogeny between two curves in extended Weierstrass form, and let ψ be as in Proposition 4.1. We define $\deg \phi \coloneqq \deg \psi$, i.e. the degree of the isogeny ϕ is defined to be equal to the degree of ψ as a rational function. The isogeny ϕ is called* separable *if the derivative (in x) of ψ is not identically zero.*

The term *d-isogeny* is shorthand for degree d isogeny.

An important property of isogenies is that they are also group homomorphisms, with finite kernels. If ϕ is separable, then $|\ker \phi| = \deg \phi$. The converse is also true, and is a crucial part of our construction:

Proposition 4.3 ([Sil09, III.4.12])**.** *Let E be an elliptic curve and let $H < E$ be a finite subgroup of E. There is a unique elliptic curve E' and a separable $|H|$-isogeny $\phi : E \to E'$ with $\ker \phi = H$.*

See also [Vél71] for an explicit construction of such isogenies. We will apply the proposition for groups H with $|H| = 2$, but all our results generalize to larger H. In this case ϕ is 2-isogeny, meaning ψ is a degree 2 rational function.

4.1.4 Group size and structure

The group E is abelian, and it is always of rank at most 2, i.e. it is isomorphic to a product of at most 2 cyclic groups

$$E \simeq \mathbb{Z}/m_1\mathbb{Z} \times \mathbb{Z}/m_2\mathbb{Z}$$

with $m_1 \mid m_2$ and $m_1 \cdot m_2 = |E|$.

Hasse's theorem states that for every elliptic curve E, the order of the group $|E|$ belongs to a range of length $4\sqrt{q}$ centered at $q+1$, that is,

$$q - 2\sqrt{q} + 1 \leq |E| \leq q + 2\sqrt{q} + 1.$$

By a theorem of Deuring [Deu41], any number in this range is indeed attainable as the size of an elliptic curve, in the case where q is prime. Waterhouse [Wat69, Theorem 4.1] provides the complete characterization of achievable sizes for the prime power case. We will require a much weaker form, about possible factors of $|E|$. The following is the simplest case of Waterhouse's theorem:

Theorem 4.4. *Let $N = q+1-t$ be an integer such that $|t| \leq 2\sqrt{q}$ and t is coprime to q. Then there exists an elliptic curve $E/\mathbb{F}_q$ with $|E| = N$.*

4.2 An FFT-friendly sequence of rational maps coming from elliptic curves

As noted in Section 3, the depth (or size) of an FFTree limits the degrees of the polynomials which it can be used to evaluate. Thus, we would like to find the largest FFTree possible: if smaller degrees are sufficient, we can always use a subtree instead. We will denote by $\widehat{\mathsf{K}}_q$ the largest possible size of an FFTree which can be obtained by our method. More rigorously, we define

Definition 4.5. *Let q be a prime power. Define $\widehat{\mathsf{K}}_q$ to be the largest power of 2 such that there exists an elliptic curve E defined over $\mathbb{F}_q$ whose size satisfies $\widehat{\mathsf{K}}_q \mid |E|$ and $|E| > 2\widehat{\mathsf{K}}_q$.*

We claim that $\widehat{\mathsf{K}}_q$ is in fact fairly large with respect to q:

Claim 4.6. *Let $q \geq 7$ be a prime power. Then $\widehat{\mathsf{K}}_q > \sqrt{q}$. Equivalently, for any $\mathsf{K} = 2^{\mathsf{k}} \leq 2\sqrt{q}$, there exists an elliptic curve E defined over $\mathbb{F}_q$ with $\mathsf{K} \mid |E|$ and $|E| > 2\mathsf{K}$. If q is even, then $\widehat{\mathsf{K}}_q \geq \frac{q}{4}$.*

Before we prove Claim 4.6, having defined and bounded $\widehat{\mathsf{K}}_q$, we are now able to precisely state the main theorem of this section:

Theorem 4.7 (Existence of large FFTrees)**.** *Let q be a prime power, and let the integer k be such that $\mathsf{K} = 2^{\mathsf{k}} \leq \widehat{\mathsf{K}}_q$; in particular, one may take K to be any power of two up to $2\sqrt{q}$ for $q \geq 7$.*

Then there exists an FFTree over $\mathbb{F}_q$ with depth k.

We now proceed with building up the infrastructure towards proving Theorem 4.7.

Proof of Claim 4.6. We will ignore at first the condition $|E| > 2\mathsf{K}$. By Theorem 4.4, it is enough to show that there exists an integer t such that $\mathsf{K} \mid q+1-t$, $|t| \leq 2\sqrt{q}$, and t is coprime to q.

Since the closed interval $[q - 2\sqrt{q} + 1, q + 2\sqrt{q} + 1]$ has length at least $2\mathsf{K}$, it must contain at least two integers $q+1-a, q+1-(a+\mathsf{K})$ which are both divisible by K. Note that at least one of $a, a+\mathsf{K}$ must be coprime to the characteristic p of $\mathbb{F}_q$: indeed, if $p \neq 2$, this follows since their difference $\mathsf{K} = 2^{\mathsf{k}}$ is not divisible by p, whereas if $p = 2$, then $\mathsf{K} \mid q+1-a$ implies both $a, a+\mathsf{K}$ are odd and thus coprime to q—and in fact $a = 1$ simply works, yielding a curve of size q (also known as an "anomalous" curve) and showing $\widehat{\mathsf{K}}_q \geq \frac{q}{4}$. Thus we can always choose at least one of $a, a+\mathsf{K}$ as our candidate for t, for which a corresponding curve exists.

Finally, to assert $|E| > 2\mathsf{K}$, note that $|E| > q - 2\sqrt{q}$ and $2\mathsf{K} \leq 4\sqrt{q}$, thus for all $q \geq 36$ we get

$$|E| > q - 2\sqrt{q} \geq 6\sqrt{q} - 2\sqrt{q} \geq 2\mathsf{K}$$

as claimed. The finitely many cases of $7 \leq q < 36$ can be manually checked to verify that indeed for each such q there is an elliptic curve E with size exactly $3\widehat{\mathsf{K}}_q$. □

Remark 4.8. Claim 4.6 is false for $q = 2, 4, 5$: since $2\sqrt{q}$ is not much smaller than q for these prime powers, for the largest K below $2\sqrt{q}$, we have $q + 2\sqrt{q} + 1 < 3\mathsf{K}$, and therefore no curve has order divisible by K and greater than $2\mathsf{K}$.

See also [SS17] for an overview of practical algorithms for finding such curves. We note that restricting the size of K further, e.g. $\mathsf{K} \leq \sqrt{q}$ or even $\mathsf{K} = o(\sqrt{q})$, greatly increases the number of possible curves, and similarly decreases the difficulty of finding one.

Starting from a curve as guaranteed by Claim 4.6, we now construct a chain of curves and isogenies with useful properties.

Theorem 4.9. *For any prime power q and any $1 < \mathsf{K} = 2^{\mathsf{k}} \leq \widehat{\mathsf{K}}_q$, there exist elliptic curves $E_0, E_1, \ldots, E_{\mathsf{k}}$ over $\mathbb{F}_q$ in extended Weierstrass form, a subgroup $G_0 \subseteq E_0$ of size K, 2-isogenies $\phi_i : E_i \to E_{i+1}$ and rational functions $\psi^{(i)} : \mathbb{P}^1 \to \mathbb{P}^1$ of degree 2, such that the following diagram is commutative:*

$$\begin{array}{ccccccc} E_0 & \xrightarrow{\phi_0} & E_1 & \xrightarrow{\phi_1} & \cdots & \xrightarrow{\phi_{\mathsf{k}-1}} & E_{\mathsf{k}} \\ \downarrow{\scriptstyle \pi_0} & & \downarrow{\scriptstyle \pi_1} & & & & \downarrow{\scriptstyle \pi_{\mathsf{k}}} \\ \mathbb{P}^1 & \xrightarrow{\psi^{(0)}} & \mathbb{P}^1 & \xrightarrow{\psi^{(1)}} & \cdots & \xrightarrow{\psi^{(\mathsf{k}-1)}} & \mathbb{P}^1 \end{array} \tag{7}$$

where:

- *π_i are the projection maps to the x-coordinate of each curve;*
- *$\ker(\phi_i) \subseteq G_i \coloneqq \phi_{i-1} \circ \cdots \circ \phi_0(G_0)$ for all i; and*
- *G_0 has a coset C such that $C \neq -C$ (as elements of the quotient group E_0/G_0).*

Remark 4.10. The existence of the coset C with $C \neq -C$ will be crucial in the derivation of Theorem 4.11, i.e. in the construction of the FFTree structure.

Proof. By the definition of $\widehat{\mathsf{K}}_q$, there exists an elliptic curve E_0 over $\mathbb{F}_q$ with exactly N points, where $\mathsf{K} \mid N$ and $N > 2\mathsf{K}$. Since E_0 is abelian, it has a subgroup of any order dividing N, in particular of order K. However, since we want to ensure the existence of coset C with $C \neq -C$, we may need to choose G_0 more carefully.[5] The proof that an appropriate G_0 exists is technical and not of particular importance, and the interested reader may find it in Appendix B.2. We note that the condition that $N > 2\mathsf{K}$ is required exactly to ensure the existence of such G_0 and C.

Having constructed G_0, we choose inside it a subgroup of size 2, and use Proposition 4.3 to find a new Weierstrass curve E_1 and 2-isogeny $\phi_0 : E_0 \to E_1$ whose kernel is the subgroup. Thus $G_1 = \phi_0(G_0)$ is a subgroup of E_1 of order $2^{\mathsf{k}-1}$, and we continue iteratively, at step i constructing E_{i+1} and ϕ_i such that the kernel of ϕ_i is a size 2 subgroup of G_i, the image of G_0 in E_i, which is of size $2^{\mathsf{k}-i}$. The iteration stops at E_{k}, where the image G_{k} of G_0 becomes a singleton.

By Proposition 4.1 and Definition 4.2, having written all curves E_i in extended Weierstrass forms, we find that there exist rational functions $\psi^{(i)}$, of degrees equal to $\deg \phi_i = 2$, which complete the commutative diagram as claimed. □

Focusing on the bottom row of (7), we obtain Theorem 4.7 as a direct corollary of Theorem 4.9. The following theorem is an equivalent reformulation of Theorem 4.7, directly recalling the definition of the FFTree.

Theorem 4.11. *Let q be a prime power, and let k be such that $\mathsf{K} = 2^{\mathsf{k}} \leq \widehat{\mathsf{K}}_q$. There exist subsets $L^{(0)}, L^{(1)}, \ldots, L^{(\mathsf{k})} \subseteq \mathbb{F}_q$ and degree 2 rational functions $\psi^{(i)}(X) = \frac{u^{(i)}(X)}{v^{(i)}(X)} \in \mathbb{F}_q(X)$ such that:*

1. *$|L^{(i)}| = 2^{\mathsf{k}-i}$.*
2. *$\psi^{(i)}$ is a 2-to-1 map from $L^{(i)}$ onto $L^{(i+1)}$.*

[5] As the proof shows, this is in fact only an issue when $\frac{N}{\mathsf{K}} = 4$; in other cases any choice of G_0 works.

Proof. The case $\mathsf{K} = 1$ is trivial. If $\mathsf{K} > 1$, apply Theorem 4.9 to find $E_i, \phi_i, \psi^{(i)}$ and G_0 as above, and let C be a coset of G_0 such that $C \neq -C$. For each i define C_i to be the image of C in E_i, i.e. $C_i = \phi_{i-1} \circ \cdots \circ \phi_1 \circ \phi_0(C)$. Since $\ker(\phi_{i-1} \circ \cdots \circ \phi_0) < G_0$, by the third isomorphism theorem, the map $\phi_{i-1} \circ \cdots \circ \phi_0$ induces an embedding $E_0/G_0 \hookrightarrow E_i/G_i$ which maps distinct cosets of G_0 to distinct cosets of G_i, and C to C_i. In particular $C \neq -C$ as cosets of G_0 implies to $C_i \neq -C_i$ as cosets of G_i. Define $L^{(i)} = \pi_i(C_i)$. Note that since $C_i, -C_i$ are cosets, $C_i \neq -C_i$ means they are disjoint, and thus π_i is a 1-to-1 map from C_i onto $L^{(i)}$. In particular $|L^{(i)}| = |C_i| = 2^{\mathsf{k}-i}$.

Finally, since the diagram is commutative and ϕ_i is a 2-to-1 map from C_i onto C_{i+1}, $\psi^{(i)}$ is a 2-to-1 map from $L^{(i)}$ onto $L^{(i+1)}$. □

Remark 4.12. Not to miss the forest for the trees, we clarify some features of this elliptic curve based construction. A careful examination of the proof in Appendix B.2 shows that $C = -C$ holds for at most 4 different cosets. The rest of the cosets appear in pairs $\{C^{(j)}, -C^{(j)}\}$, each pair projecting through π_0 to a different (and disjoint) $L_j^{(0)} = \pi_0(C^{(j)}) = \pi_0(-C^{(j)})$. Thus, our construction actually yields at least $\frac{N}{2\mathsf{K}} - 2$ different FFTrees, with pairwise disjoint vertices from all trees at every fixed level, but with the same rational functions $\psi^{(i)}$ across all trees.

Thus, there exists not only a single FFTree, but an entire FFForest of disjoint FFTrees all sharing the same maps. The algorithms in Section 6 will all be described for the case of a single FFTree and subsets of its vertices, but we note that many of them can also be applied without additional complexity on sets taken from two (or $O(1)$) different FFTrees belonging to the same FFForest. Note that the total number of leaves in this FFForest is $\Omega(q)$, or, more accurately, $\frac{q}{2} - O(\sqrt{q} + \mathsf{K})$.

5 Representing polynomials via FFTrees

In this section, we show how to use FFTrees to get a nice representation for polynomials that supports fast operations.

We begin by fixing an FFTree for the rest of this section. Thus we have sets $L^{(0)}, L^{(1)}, \ldots, L^{(\mathsf{k})} \subseteq \mathbb{F}_q$, and degree-2 rational functions $\psi^{(i)} : L^{(i)} \to L^{(i+1)}$. We let $L = L^{(0)}$ and let $\mathsf{K} = |L| = 2^{\mathsf{k}}$. Also recall the associated binary tree $\mathcal{F}$ whose set of leaves is L.

All the data structures and algorithms for polynomials that we describe will be in the context of this FFTree. While the exact details of how this FFTree is obtained are not important for anything in this section, it will be helpful to recall the parameters of FFTrees that are achievable via Theorem 4.7.

5.1 Evaluation tables

We shall represent polynomials by their evaluations on various special sets of points, so we introduce a special notation that will emphasize the sets of evaluation points used. Concretely, an *evaluation table* is specified by the following data:

- a set $S \subseteq \mathbb{F}_q$,
- a function $f : S \to \mathbb{F}_q$.

We denote the associated evaluation table by $\langle f \wr S \rangle$, pronounced "f on S".

For a polynomial or rational function $P(X) \in \mathbb{F}_q(X)$ with $P(X)$ defined on S, we define the associated evaluation table $\langle P \wr S \rangle$ to be the evaluation table $\langle P|_S \wr S \rangle$, where $P|_S$ is the function from S to $\mathbb{F}_q$ given by evaluation of P. Looking ahead, we shall use evaluation tables for operations like

- Adding, multiplying and dividing, as in this example: given $\langle f \wr S \rangle$, $\langle g \wr S \rangle$, $\langle h \wr S \rangle$, $\langle P \wr S \rangle$ for some $P(X) \in \mathbb{F}_q[X]$, we can compute $\left\langle \frac{f + P(X)g}{h} \wr S \right\rangle$.
- Restricting an evaluation table $\langle f \wr S \rangle$ to a subset $S_0 \subseteq S$, denoting the restricted table by $\langle f \wr S_0 \rangle$

- Partitioning a set S into $S = S_0 \cup S_1$, and "splitting" $\langle f \wr S \rangle$ into $\langle f_0 \wr S_0 \rangle$ and $\langle f_1 \wr S_1 \rangle$, as well as doing the inverse operation of forming the combined evaluation table $\langle f \wr S \rangle = \langle f_0 \wr S_0 \rangle \cup \langle f_1 \wr S_1 \rangle$, where $f : S \to \mathbb{F}_q$ is given by:
$$f|_{S_0} = f_0,$$
$$f|_{S_1} = f_1.$$

5.2 Basic sets

We now identify some important subsets of L.

Definition 5.1 (Basic sets)**.** *We define a* **basic** *set to be a subset S of L which is the set of all descendants in L of some vertex of $\mathcal{F}$.*

Equivalently, it is a set of size 2^a for some integer a, such that if we let g denote the composed function $\psi^{(a-1)} \circ \psi^{(a-2)} \circ \cdots \circ \psi^{(1)} \circ \psi^{(0)}$, then $S = g^{-1}(u)$ for some $u \in L^{(a)}$.

We have the following important property of basic sets: they can be partitioned into two basic sets of equal size.

Lemma 5.2. *Any basic set S of size $2^a \geq 2$ can be partitioned to two basic sets $S_0 \cup S_1$, where each S_i has size 2^{a-1}.*

The proof is immediate from Definitions 5.1 and 3.3: if S is the set of all descendants in L of the vertex $u \in \mathcal{F}$, then letting $\{u_0, u_1\}$ be the children of u, we can take S_i to be the set of all descendants in L of u_i. We shall call S_0 and S_1 the *moieties* of S. Note that the two moieties are equivalent, and can be labeled S_0, S_1 or S_1, S_0 interchangeably.

The following property of sets and polynomials with respect to moieties of basic sets will also prove to be important in the paper, especially for algorithms related to modular arithmetic:

Definition 5.3. *Let S be a basic set, and let $A \subset \mathbb{F}_q$ be an arbitrary set. We say A is* half-disjoint *from S if it is disjoint from at least one moiety of S. Similarly, we say a polynomial $P(X)$ is half-disjoint from S if its set of zeros is disjoint from at least one moiety of S.*

We now consider representations of polynomials by evaluation tables. Since nonzero polynomials of degree $< n$ cannot vanish in n points, we immediately get the following fundamental fact. For distinct polynomials $P(X), Q(X) \in \mathbb{F}_q[X]$ with $\deg(P), \deg(Q) < n$, and a set S with $|S| = n$, we have that

$$\langle P \wr S \rangle \neq \langle Q \wr S \rangle .$$

Thus, for a fixed set S with $|S| = n$, $\langle P \wr S \rangle$ is a way of representing a polynomial P with degree $< n$. The key to our fast algorithms for working with such a representation is to choose S to be a basic set.

We now define a standard representation for polynomials (in the context of the fixed FFTree). This standard representation will support fast operations, and will be used when we describe applications to classical problems.

For each $a \leq \mathsf{k}$, we arbitrarily pick a basic set U_a with size 2^a such that:

$$U_0 \subseteq U_1 \subseteq \cdots \subseteq U_{\mathsf{k}} = L.$$

We will call this U_a the *standard* basic set of size 2^a.

For a polynomial $P(X)$ and an integer a with $2^a > \deg(P)$, we define the *standard representation of P at scale a*, denoted $\langle P \rangle_a$, to be $\langle P \wr U_a \rangle$.

For a polynomial $P(X)$, we define **the** *standard representation of P*, to be the $\langle P \rangle_{a_0}$, where a_0 is the smallest integer with $2^{a_0} > \deg(P)$.

This standard representation will be our data structure for representing polynomials. In the next section, we show how the FFTree enables fast operations for this representation of polynomials.

6 Fast polynomial algorithms from FFTrees

As in the previous section, we assume that we have fixed an FFTree. Again, the exact details of how this FFTree is obtained is not important for anything in this section, but it will be helpful to recall the parameters of FFTrees that are achievable via Theorem 4.7.

In this section we give a number of fast algorithms for working with polynomials $P(X)$ represented using evaluation tables $\langle P \wr S \rangle$, where S is a basic set. Inspection will reveal that nearly all of these algorithms can be converted to arithmetic circuits over $\mathbb{F}_q$ with constant fan-in and size that matches the proclaimed running time (the only exception is the computation of polynomial degree, which outputs an integer, not a field element). Thus, henceforth when we say an algorithm "runs in time $t(n)$" we shall allow it to receive advice that will be explicitly stated, and also mean that it can be computed by an arithmetic circuit over $\mathbb{F}_q$ with $t(n)$ gates (and constant fan-in). In particular, we assume each basic arithmetic operation $(+, -, \times, /)$ over $\mathbb{F}_q$ has constant computational cost. While the algorithms of this section use division of elements in $\mathbb{F}_q$ for clarity, by inspecting the details it can be seen that they can be reformulated to avoid division by taking advice in a different form (for example, taking $\left\langle \frac{1}{f} \wr S \right\rangle$ as advice instead of $\langle f \wr S \rangle$ as advice).

Algorithmic notations We use the notation $\mathsf{ALG}_{P_1,P_2,\ldots}(I_1, I_2, \ldots)$ for our algorithms/circuits. ALG is the name of the algorithm, the subscript elements $P_1, P_2, \ldots$ denote fixed parameters that affect constants of the algorithm/circuit and the inputs $(I_1, I_2, \ldots)$ are given inside the parenthesis, and are variables. In particular, any data which depends only on $P_1, P_2, \ldots$ can be assumed to be included as part of the circuit, or given by a precomputation advice, and our running times exclude the time required to obtain these parameters and constants. Furthermore, q and the FFTree that we fixed are always assumed to be part of the fixed parameters of the algorithm.

Directory of algorithms Below we give a list of the algorithms in this section.

1. $\mathsf{EXTEND}_{S,S'}$ which does low degree extension of polynomial evaluations from a basic set S to another basic set S'. EXTEND is the basis for all the remaining algorithms in this section.
2. MULT, which multiplies polynomials in the new representation (allowing for the possibility of the degree growing). Addition is trivially done in linear time so we do not explicitly describe it.
3. MEXTEND, a version of EXTEND for monic polynomials of known, fixed degree.
4. DEGREE, which computes the degree of a polynomial given in the new representation.
5. REDC, which performs Montgomery reduction—a technical operation that helps with the remaining operations.
6. MOD, which performs modular reduction, reducing a given polynomial in the new representation modulo a fixed polynomial.
7. DIV, which finds the quotient after division by a fixed polynomial.
8. ENTER and EXIT, which convert between the new representation and the standard monomial representation.
9. CRT which computes one direction of the Chinese Remainder Theorem, constructing a polynomial from its residues modulo two fixed and relatively prime polynomials. (The other direction of the CRT can be done by MOD.)

6.1 Low degree extension

Our first primitive extends the evaluation of P from one basic set to another basic set of the same size in time $O(n \log n)$ (i.e., via an arithmetic circuit over $\mathbb{F}_q$ with constant fan-in and $O(n \log n)$ gates). In other words, the algorithm performs Reed–Solomon encoding in quasi-linear time, as long as the message is provided by the evaluation of P on a basic set, and is encoded by evaluating P on a constant collection of basic sets. Such low-degree extensions are often used to produce interactive proofs and interactive oracle proofs.

Theorem 6.1 (Low-degree extension)**.** *For any two basic sets $S, S' \subset \mathbb{F}_q$ with $|S| = |S'| = n$, there is an algorithm that runs in time $O(n \log n)$, denoted $\mathsf{EXTEND}_{S,S'}$, which when given as input:*

- *$\langle P \wr S \rangle$, where $P(X) \in \mathbb{F}_q[X]$ with $\deg(P) < n$,*

outputs $\langle P \wr S' \rangle$.

For the proof of this theorem (and only for this proof) we need a generalization of basic sets:

Definition 6.2 (i-basic sets)**.** *We define an i-**basic** set to be a subset S of $L^{(i)}$ which is the set of all descendants in $L^{(i)}$ of some vertex of $\mathcal{F}$.*

Equivalently, an i-basic set is a subset S of $L^{(i)}$ of size 2^a for some integer a, such that if we let g denote the function

$$\psi^{(a+i-1)} \circ \psi^{(a+i-2)} \circ \cdots \circ \psi^{(i+1)} \circ \psi^{(i)},$$

then $S = g^{-1}(u)$ for some $u \in L^{(a+i)}$.

Notice that 0-basic sets are simply basic sets per Definition 5.1. In our proof, stated next, we shall use the property that for every i-basic set S, $\psi^{(i)}(S)$ is an $(i+1)$-basic set T of size $|S|/2$, and we say T *lies above* S and is *induced by* $\psi^{(i)}$.

Proof of Theorem 6.1. We give a more general algorithm $\mathsf{EXTEND}_{S,S',i}$ to solve the analogous extension problem where S and S' are i-basic sets with $|S| = |S'| = n$. The algorithm $\mathsf{EXTEND}_{S,S'}$ claimed in Theorem 6.1 is obtained by fixing $i = 0$, i.e., $\mathsf{EXTEND}_{S,S'}(\langle \pi \wr S \rangle) = \mathsf{EXTEND}_{S,S',0}(\langle \pi \wr S \rangle)$.

The $\mathsf{EXTEND}_{S,S',i}$ algorithm uses the map $\psi^{(i)}$ to reduce the extension problem for i-basic sets of size $n = 2^a$ to two analogous extension problems for $(i+1)$-basic sets of size $n/2$, and then proceeds recursively, by induction on a.

Let $T = \psi^{(i)}(S)$, $T' = \psi^{(i)}(S')$ be the $(i+1)$-basic sets above S, S', respectively, which are induced by $\psi^{(i)}$. By Lemma 3.1, there are unique polynomials $P_0(X), P_1(X)$ of degree $< n/2$ with:

$$P(X) = \left(P_0(\psi^{(i)}(X)) + X P_1(\psi^{(i)}(X))\right) (v^{(i)}(X))^{\frac{n}{2}-1}. \tag{8}$$

$\mathsf{EXTEND}_{S,S',i}$ first computes $\langle P_0 \wr T \rangle$ and $\langle P_1 \wr T \rangle$. (Since $|T| = n/2$, these uniquely determine $P_0(X)$ and $P_1(X)$). Then it runs $\mathsf{EXTEND}_{T,T',i+1}$ on this to get $\langle P_0 \wr T' \rangle$ and $\langle P_1 \wr T' \rangle$, and combines the results to get $\langle P \wr S' \rangle$.

The algorithm takes as advice $\left\langle (v^{(i)}(X))^{\frac{n}{2}-1} \wr S \right\rangle$, which can be precomputed since it only depends on S and $\psi^{(i)}$, along with whatever advice is needed in the recursive calls.

Algorithm $\mathsf{EXTEND}_{S,S',i}$:
Input: an evaluation table $\langle \pi \wr S \rangle$

1. If $n = 1$ (recall that $n = |S| = |S'|$), then
   (a) Let $S = \{s\}$ and $S' = \{s'\}$.

(b) Define

$$\pi' : S' \to \mathbb{F}_q$$

by $\pi'(s') = \pi(s)$.

(c) Return $\langle \pi' \wr S' \rangle$.

2. Let $T = \psi^{(i)}(S), T' = \psi^{(i)}(S')$ be the sets that lie above S and S' respectively.

3. For each $t \in T$:

   (a) Define s_0, s_1 to be the $\psi^{(i)}$-preimages of t (noticing they are distinct because S is a basic set)

   (b) Compute $(\pi_0(t), \pi_1(t)) = M_t(\pi(s_0), \pi(s_1)))$ where M_t is defined in Eq. (3).

4. Form the evaluation tables $\langle \pi_0 \wr T \rangle$ and $\langle \pi_1 \wr T \rangle$.

5. Let $\langle \pi'_0 \wr T' \rangle$ and $\langle \pi'_1 \wr T' \rangle$ be the evaluation tables returned by:

$$\mathsf{EXTEND}_{T,T',i+1}(\langle \pi_0 \wr T \rangle)$$

$$\mathsf{EXTEND}_{T,T',i+1}(\langle \pi_1 \wr T \rangle)$$

6. For each $s' \in S'$, define $\pi'(s')$ by

$$\pi'(s') = \left(\pi'_0(\psi^{(i)}(s')) + s' \cdot \pi'_1(\psi^{(i)}(s'))\right) v^{(i)}(s')^{\frac{n}{2}-1}. \tag{9}$$

7. Return $\langle \pi' \wr S' \rangle$.

Correctness: Suppose $P(X) \in \mathbb{F}_q[X]$ is a polynomial of degree $< n$. We want to show that $\mathsf{EXTEND}_{S,S',i}(\langle P \wr S \rangle)$ returns $\langle P \wr S' \rangle$.

The main claim is that when the input $\langle \pi \wr S \rangle$ is $\langle P \wr S \rangle$, the functions $\pi_0, \pi_1 : T \to \mathbb{F}_q$ computed by the algorithm satisfy:

$$\langle \pi_0 \wr T \rangle = \langle P_0 \wr T \rangle,$$
$$\langle \pi_1 \wr T \rangle = \langle P_1 \wr T \rangle,$$

where P_0, P_1 are as in Equation (8). This is trivially correct for $a = 0$ (i.e., when $n = 1$) so we focus henceforth on larger values of $n = 2^a$.

Take any t in T, and take $s_0, s_1 \in S$ with $\psi^{(i)}(s_0) = \psi^{(i)}(s_1) = t$. Using the fact that $\pi(s_0) = P(s_0)$ and $\pi(s_1) = P(s_1)$, and the definition of M_t from Eq. (3), Lemma 3.2 implies that $\pi_0(t) = P_0(t)$ and $\pi_1(t) = P_1(t)$. Thus

$$\langle \pi_0 \wr T \rangle = \langle P_0 \wr T \rangle,$$
$$\langle \pi_1 \wr T \rangle = \langle P_1 \wr T \rangle.$$

By induction on a, we conclude that $\mathsf{EXTEND}_{T,T',i+1}$ on $\langle P_0 \wr T \rangle$ and $\langle P_1 \wr T \rangle$ returns $\langle P_0 \wr T' \rangle$ and $\langle P_1 \wr T' \rangle$.

Thus

$$\langle \pi'_0 \wr T' \rangle = \langle P_0 \wr T' \rangle,$$
$$\langle \pi'_1 \wr T' \rangle = \langle P_1 \wr T' \rangle,$$

Using this along with Equations (9) and (4), we get that

$$\langle \pi' \wr S' \rangle = \langle P \wr S' \rangle,$$

as desired. This completes the proof of correctness.

Running time: By inspection, we see that our algorithm uses $O(n)$ arithmetic operations over $\mathbb{F}_q$ to reduce an instance of EXTEND of size n to two instances of size $n/2$. (Recall that the algorithm fixes various constants, like $v(s)^{n/2-1}$ and the values of the matrix M_t.) Thus the total running time $F(n)$ of this algorithm satisfies the recursion:

$$F(n) \leq 2F(n/2) + O(n).$$

We conclude the running time (or circuit size) is $O(n \log n)$ and this completes our proof. □

Remark 6.3. The EXTEND algorithm as described is defined for S, S' which are basic sets of the same size in the same FFTree. However, we note that it works just as well when S, S' are basic sets of the same size from two different $\mathsf{FFTrees}$ in the same $\mathsf{FFForest}$ (see also Remark 4.12).

6.2 Multiplication

We give a quick application of the previous algorithm to multiplication of polynomials in the new representation.

Theorem 6.4 (Multiplication)**.** *Let S be a basic set with $|S| = n$. Let $S_0 \subseteq S$ be a moiety of S.*
There is an algorithm MULT_{S,S_0}, which when given as input:

- $\langle P \wr S_0 \rangle$, *where* $P(X) \in \mathbb{F}_q[X]$ *with* $\deg(P) < n/2$, *and*
- $\langle Q \wr S_0 \rangle$, *where* $Q(X) \in \mathbb{F}_q[X]$ *with* $\deg(Q) < n/2$,

runs in time

$$O(n \log n)$$

and computes $\langle P \cdot Q \wr S \rangle$.

Proof. The algorithm is basically immediate given EXTEND. Let S_1 be the other moiety of S. We first run $\mathsf{EXTEND}_{S_0,S_1}$ on $\langle P \wr S_0 \rangle$ and $\langle Q \wr S_0 \rangle$ to get $\langle P \wr S_1 \rangle$ and $\langle Q \wr S_1 \rangle$. Combining these, we get $\langle P \wr S \rangle$ and $\langle Q \wr S \rangle$, and by pointwise multiplication we get $\langle P \cdot Q \wr S \rangle$. The running time comes from two invocations of EXTEND and $O(n)$ other operations, and is thus $O(n \log n)$. □

6.3 Monic polynomial extension

As noted before, for a set S of size n, the linear space of all possible evaluation tables $\langle P \wr S \rangle$ is in one-to-one correspondence with the space of all polynomials $P(X)$ of degree $< n$. It is also interesting to note that these spaces are in one-to-one correspondence with the set of all *monic* polynomials of degree *exactly* n. In fact, if $Z(X)$ is the vanishing polynomial of S, and $P(X), Q(X)$ are polynomials with $\deg(Q) < n = \deg(P)$ and P is monic, then $\langle P \wr S \rangle = \langle Q \wr S \rangle$ if and only if $P(X) = Q(X) + Z(X)$.

This property allows us to easily adapt the EXTEND algorithm into an extension algorithm for monic polynomials, which we call $\mathsf{MEXTEND}$.

Theorem 6.5 (Monic polynomial extension)**.** *For any two basic sets $S, S' \subset \mathbb{F}_q$ with $|S| = |S'| = n$, there is an algorithm that runs in time $O(n \log n)$, denoted $\mathsf{MEXTEND}_{S,S'}$, which when given as input:*

- $\langle P \wr S \rangle$, *where* $P(X) \in \mathbb{F}_q[X]$ *is monic with* $\deg(P) = n$,

outputs $\langle P \wr S' \rangle$.

Proof. Let $Z(X)$ be the vanishing polynomial of S. As noted above, for such polynomials $P(X)$, we have $\langle P \wr S \rangle = \langle P - Z \wr S \rangle$, and $\deg(P(X) - Z(X)) < n$. By the properties of EXTEND it thus follows that

$$\mathsf{EXTEND}_{S,S'}(\langle P \wr S \rangle) = \mathsf{EXTEND}_{S,S'}(\langle P - Z \wr S \rangle) = \langle P - Z \wr S' \rangle$$

and adding $\langle Z \wr S' \rangle$ pointwise yields

$$\mathsf{MEXTEND}_{S,S'}(\langle P \wr S \rangle) \coloneqq \mathsf{EXTEND}_{S,S'}(\langle P \wr S \rangle) + \langle Z \wr S' \rangle = \langle P \wr S' \rangle$$

as needed. The algorithm takes $\langle Z \wr S' \rangle$ as advice, calls EXTEND once and does an additional $O(n)$ operations, thus runs in time $O(n \log n)$. □

This algorithm can replace EXTEND in applications where the polynomials are known to be monic and of known degrees, with more efficient run times. For example, it can be used to multiply two monic polynomials of degree $n/2$, represented as evaluation tables on a set of size $n/2$, with the product similarly being a monic polynomial of degree n, represented as an evaluation table on a set of size n. If we were instead to multiply such polynomials using the standard EXTEND algorithm, we would have to represent each polynomial by its values on a set of size n, and their product on a set of size $2n$, and use extensions from n to $2n$ instead of extensions from $n/2$ to n, which would more than double the required run-time.

6.4 Degree Computation

The next operation we describe is that of computing the degree of a polynomial P represented by its evaluation on a basic set.

Theorem 6.6 (Degree Computation)**.** *Let S be a basic set of size $|S| = n$. There is an algorithm DEGREE_S, which when given as input:*

- $\langle P \wr S \rangle$*, where* $P(X) \in \mathbb{F}_q[X]$ *with* $\deg(P) < n$*,*

runs in time

$$O(n \log n)$$

and computes $\deg(P)$*.*

Proof. Let S_0, S_1 be the moieties of S, and let $Z_0(X)$ be the vanishing polynomial of S_0. The algorithm we give will assume that $\langle Z_0 \wr S_1 \rangle$ is given as advice: this is a fixed precomputation that depends only on S.

Algorithm DEGREE_S**:**
Input: An evaluation table $\langle \pi \wr S \rangle$

1. If $|S| = 1$ with $S = \{s\}$, then
   - if $\pi(s) \neq 0$ return 0; else, return $-\infty$.
2. Let $\langle g \wr S_1 \rangle = \mathsf{EXTEND}_{S_0,S_1}(\langle \pi \wr S_0 \rangle)$.
3. If $\langle g \wr S_1 \rangle = \langle \pi \wr S_1 \rangle$, then return $\mathsf{DEGREE}_{S_1}(\langle \pi \wr S_1 \rangle)$.
4. Otherwise, using $\langle \pi \wr S_1 \rangle$, $\langle g \wr S_1 \rangle$ and $\langle Z_0 \wr S_1 \rangle$, compute:

$$\left\langle \frac{\pi - g}{Z_0} \wr S_1 \right\rangle,$$

and return

$$\frac{n}{2} + \mathsf{DEGREE}_{S_1}\left(\left\langle \frac{\pi - g}{Z_0} \wr S_1 \right\rangle\right).$$

Correctness: The case $n = 1$ is trivial, and when P is the zero polynomial notice by inspection the result will be $-\infty$, as required.

Suppose $n > 1$. Let $P(X)$ be a polynomial with $0 \leq \deg(P) < n$. Let us consider the execution of the above algorithm on input $\langle P \wr S \rangle$.

- **Case 1:** $\deg(P) < n/2$. Then by the defining property of $\mathsf{EXTEND}_{S_0,S_1}$, we have that

$$\mathsf{EXTEND}_{S_0,S_1}\left(\langle P \wr S_0 \rangle\right) = \langle P \wr S_1 \rangle .$$

  Thus in the execution of the algorithm, we will have $\langle g \wr S_1 \rangle = \langle P \wr S_1 \rangle$, and thus in Step 3 the algorithm will return

$$\mathsf{DEGREE}_{S_1}\left(\langle P \wr S_1 \rangle\right),$$

  which equals $\deg(P)$ by induction, as desired.

- **Case 2:** $\deg(P) \geq n/2$. Let $P(X) = R(X) + Z_0(X) \cdot Q(X)$, where $\deg(R) < n/2$. Thus $\deg(P) = n/2 + \deg(Q)$.

  By the above relation between P and R, we have

$$\langle R \wr S_0 \rangle = \langle P \wr S_0 \rangle = \langle \pi \wr S_0 \rangle .$$

  By the defining property of $\mathsf{EXTEND}_{S_0,S_1}$, we get that $\langle g \wr S_1 \rangle = \langle R \wr S_1 \rangle$. Thus

$$\left\langle \frac{\pi - g}{Z_0} \wr S_1 \right\rangle = \left\langle \frac{P - R}{Z_0} \wr S_1 \right\rangle = \langle Q \wr S_1 \rangle ,$$

  which implies, by induction, that Step 4 returns

$$n/2 + \mathsf{DEGREE}_{S_1}\left(\langle Q \wr S_1 \rangle\right) = n/2 + \deg(Q) = \deg(P),$$

  as desired.

Running time: The algorithm calls one instance of EXTEND on an instance of size $O(n)$, does $O(n)$ operations, and makes one recursive call to itself on an instance of size $n/2$. Thus the running time $F(n)$ satisfies:

$$F(n) \leq O(n \log n) + F(n/2),$$

and thus $F(n) \leq O(n \log n)$, as claimed. □

6.5 Modular and Montgomery Reduction

6.5.1 Modular Reduction—theorem statement

The goal of this chapter is to present an algorithm that computes the remainder of the division of an input polynomial P (in the new representation) by a fixed polynomial A:

Theorem 6.7 (Modular Reduction)**.** *Let S be a basic set of size n, and let $A(X) \in \mathbb{F}_q[X]$ be a polynomial of degree at most $n/2$ which is half-disjoint from S, i.e. $A(X)$ has no zeroes in at least one moiety of S.*

There is an algorithm running in time $O(n \log n)$, denoted $\mathsf{MOD}_{S,A}$, which when given as input:

- *$\langle P \wr S \rangle$, where $P(X) \in \mathbb{F}_q[X]$ with $\deg(P) < n$,*

computes $\langle Q \wr S \rangle$, where $Q(X) \in \mathbb{F}_q[X]$ is given by:

$$Q(X) = P(X) \mathsf{\ rem\ } A(X).$$

Before presenting the proof and the algorithm, we introduce an auxiliary algorithm, which we call *Montgomery reduction*, inspired by Montgomery's [Mon85] algorithm for "modulo-free" modular multiplication, which we also describe briefly.

6.5.2 Montgomery Reduction

Montgomery's algorithm for multiplication is motivated by the observation that while the operation $a \bmod N$ for a generic (odd) integer N might be computationally expensive, the operation $a \bmod R$ where $R = 2^r \gtrsim N$ is very efficient, in computing systems based on binary representations.

In Montgomery's method, each residue $x \pmod N$ is represented instead by $xR \bmod N$. To get the representation of the product xy, i.e. $xyR \bmod N$, we first multiply the two representations to get an integer equivalent to $xyR^2 \pmod N$, and then apply the *reduction* algorithm REDC, which efficiently maps an integer t to $tR^{-1} \bmod N$, without explicitly computing the division by N. The reduction algorithm relies on having the constant number $(-N^{-1}) \bmod R$ as advice.

The representation $xR \bmod N$ can be transformed back to $x \bmod N$ by simply applying reduction. In the other direction, $x \bmod N$ can be transformed into $xR \bmod N$ by performing the full Montgomery multiplication (i.e. integer multiplication + reduction) between $x \bmod N$ and the constant $R^2 \bmod N$, which is again given as advice.

For our purposes, we want to perform modular arithmetic of polynomials. We observe that the vanishing polynomial $Z(X)$ of a basic set S is a natural analogue to the radix $R = 2^r$, as arithmetic operations on the tables $\langle P \wr S \rangle$ are equivalent to arithmetic operations on polynomials modulo $Z(X)$. Thus, we can attempt to create a version of REDC which transforms $\langle P \wr S \rangle$ into $\left\langle P \cdot Z^{-1} \wr S \right\rangle$, and then use this algorithm to perform general modular operations, such as MOD. In fact, we apply REDC directly only inside MOD.

Theorem 6.8 (Montgomery Reduction)**.** *Let S be a basic set with $|S| = n$. Let $S_0 \subseteq S$ be a moiety of S. Let $A(X) \in \mathbb{F}_q[X]$ be a polynomial of degree at most $n/2$ having no zeroes in S_0. Let $Z_0(X)$ be the vanishing polynomial of S_0.*

There is an algorithm running in time $O(n \log n)$, denoted $\mathsf{REDC}_{S,S_0,A}$, which when given as input:

- *$\langle P \wr S \rangle$, where $P(X) \in \mathbb{F}_q[X]$ satisfies $\deg(P) < n$,*

computes $\langle Q \wr S \rangle$, where $Q(X) \in \mathbb{F}_q[X]$ is a polynomial such that

- *$Q(X) \equiv P(X) \cdot Z_0(X)^{-1} \pmod{A(X)}$, and*
- *$\deg(Q) \leq \max(\deg(P) - n/2,\ \deg(A) - 1) < n/2$.*

Remark 6.9. If $\deg(P) < n/2 + \deg(A)$, then it follows that $\deg(Q) < \deg(A)$, and therefore

$$Q(X) = P(X) \cdot (Z_0(X))^{-1}_{A(X)} \ \mathsf{rem}\ A(X).$$

However, the last identity is not true in general when $n/2 + \deg(A) \leq \deg(P) < n$, since Q might not be of degree less than $\deg(A)$.

Proof. Let S_1 be the other moiety of S. The algorithm uses the values of $\langle Z_0 \wr S_1 \rangle$, $\langle A \wr S_0 \rangle$, $\langle A \wr S_1 \rangle$, which depend only on S, S_0 and A.

Algorithm $\mathsf{REDC}_{S,S_0,A}$**:**
Input: an evaluation table $\langle \pi \wr S \rangle$

1. From $\langle \pi \wr S_0 \rangle$ and $\langle A \wr S_0 \rangle$, compute $\left\langle \frac{\pi}{A} \wr S_0 \right\rangle$.
2. Let $\langle g \wr S_1 \rangle = \mathsf{EXTEND}_{S_0,S_1}(\left\langle \frac{\pi}{A} \wr S_0 \right\rangle)$.
3. From $\langle \pi \wr S_1 \rangle$, $\langle g \wr S_1 \rangle$, $\langle A \wr S_1 \rangle$, $\langle Z_0 \wr S_1 \rangle$, compute:
$$\langle h_1 \wr S_1 \rangle = \left\langle \frac{\pi - gA}{Z_0} \wr S_1 \right\rangle.$$
4. Compute:
$$\langle h_0 \wr S_0 \rangle = \mathsf{EXTEND}_{S_1,S_0}(\langle h_1 \wr S_1 \rangle).$$
5. Return $\langle h_0 \wr S_0 \rangle \cup \langle h_1 \wr S_1 \rangle$.

Proof of correctness: Let $P(X)$ be a polynomial of degree $< n$. We will analyze the above algorithm when its input $\langle \pi \wr S \rangle$ is taken to be $\langle P \wr S \rangle$. Let $G(X) \in \mathbb{F}_q[X]$ be the unique polynomial of degree $< n/2$ interpolating $\frac{\pi}{A}$ on S_0; namely:

$$\langle G \wr S_0 \rangle = \left\langle \frac{\pi}{A} \wr S_0 \right\rangle .$$

Then by the defining property of EXTEND, we get that:

$$\mathsf{EXTEND}_{S_0,S_1}(\left\langle \frac{\pi}{A} \wr S_0 \right\rangle) = \mathsf{EXTEND}_{S_0,S_1}(\langle G \wr S_0 \rangle) = \langle G \wr S_1 \rangle .$$

Thus in Step 2 of the algorithm, we will have

$$\langle g \wr S_1 \rangle = \langle G \wr S_1 \rangle .$$

By definition of $G(X)$, we have that $\frac{P(X)}{A(X)} - G(X)$ vanishes on S_0. Therefore $P(X) - G(X)A(X) \in \mathbb{F}_q[X]$ vanishes on S_0, and so $Z_0(X)$ divides $P(X) - G(X)A(X)$. Let $H(X) \in \mathbb{F}_q[X]$ be given by:

$$H(X) = \frac{P(X) - G(X)A(X)}{Z_0(X)}.$$

Note that

$$\deg(H) \leq \max\{\deg(P), \deg(A) + \deg(G)\} - \deg(Z_0) \leq \max(\deg(P) - n/2, \deg(A) - 1) < n/2. \tag{10}$$

The second inequality follows from the fact that $\deg(G) < \deg(Z_0) = n/2$, and the final inequality from the assumptions $\deg(P) < n$ and $\deg(A) \leq n/2$.

We have

$$\langle h_1 \wr S_1 \rangle = \left\langle \frac{\pi - gA}{Z_0} \wr S_1 \right\rangle = \left\langle \frac{P - GA}{Z_0} \wr S_1 \right\rangle = \langle H \wr S_1 \rangle .$$

Thus in Step 4 $\mathsf{EXTEND}_{S_1,S_0}$ yields

$$\langle h_0 \wr S_0 \rangle = \langle H \wr S_0 \rangle ,$$

and so the algorithm returns:

$$\langle h_0 \wr S_0 \rangle \cup \langle h_1 \wr S_1 \rangle = \langle H \wr S_0 \rangle \cup \langle H \wr S_1 \rangle = \langle H \wr S \rangle$$

and we have already shown in Eq. (10) that $H(X) = Q(X)$ is of the claimed degree.

Finally, from the definition of $H(X)$ we get

$$H(X)Z_0(X) = P(X) - G(X)A(X) \equiv P(X) \pmod{A(X)},$$

which after dividing by $Z_0(X)$ is equivalent to

$$H(X) \equiv P(X) \cdot Z_0(X)^{-1} \pmod{A(X)}.$$

This completes the proof of correctness.

Running time: The algorithm does $O(n)$ operations and invokes EXTEND twice on instances of size $n/2$. Thus the total running time is $O(n \log n)$. □

6.5.3 Modular Reduction—algorithm and proof

Proof of Theorem 6.7. Let S_0, S_1 be the moieties of S, and suppose without loss of generality that $A(X)$ has no zeros in S_0 (otherwise, it has no zeros in S_1 by assumption, and we may swap the labeling of the moieties).

Let $C(X) = Z_0(X)^2 \text{ rem } A(X)$, which has degree $\deg(C) < \deg(A)$. The algorithm uses the values of $\langle C \wr S \rangle$, that depend only on $A(X)$, S and S_0, as well as values used internally by the $\mathsf{REDC}_{S,S_0,A}$ sub-circuit.

Algorithm $\mathsf{MOD}_{S,A}$:
Input: an evaluation table $\langle \pi \wr S \rangle$

1. From $\langle \pi \wr S \rangle$, compute
$$\langle h \wr S \rangle = \mathsf{REDC}_{S,S_0,A}(\langle \pi \wr S \rangle).$$
2. From $\langle h \wr S \rangle$ and $\langle C \wr S \rangle$, compute
$$\langle h \cdot C \wr S \rangle .$$
3. Compute:
$$\langle g \wr S \rangle = \mathsf{REDC}_{S,S_0,A}(\langle h \cdot C \wr S \rangle).$$
4. Return $\langle g \wr S \rangle$.

Proof of correctness: Suppose $P(X) \in \mathbb{F}_q[x]$ with $\deg(P) < n$. We will analyze the above computation when its input $\langle \pi \wr S \rangle$ is taken to be $\langle P \wr S \rangle$. By Theorem 6.8 about REDC, we get that Step 1 computes $\langle h \wr S \rangle = \langle H \wr S \rangle$, where $H(X)$ is a polynomial with $\deg H(X) < n/2$ and satisfying

$$H(X) \equiv P(X) \cdot Z_0(X)^{-1} \pmod{A(X)}.$$

Thus Step 2 computes
$$\langle H \cdot C \wr S \rangle ,$$
where $H(X) \cdot C(X)$ is a polynomial with

$$\deg(H \cdot C) = \deg(H) + \deg(C) < n/2 + \deg(A)$$

and
$$H(X) \cdot C(X) \cdot Z_0(X)^{-1} \equiv P(X) \cdot Z_0(X)^{-1} \cdot Z_0(X)^2 \cdot Z_0(X)^{-1} \equiv P(X) \pmod{A(X)}.$$

Thus in Step 3, as noted in Remark 6.9, the algorithm returns $\langle Q \wr S \rangle$, where

$$Q(X) = \left(H \cdot C \cdot (Z_0(X))^{-1}_{A(X)} \right) \text{ rem } A(X) = P(X) \text{ rem } A(X),$$

as desired.

Running time: The algorithm invokes REDC twice on instances of size n, and does $O(n)$ other operations. Thus the running time is $O(n \log n)$. □

Remark 6.10. As noted earlier in this section, we have no further direct applications of REDC in this paper, and all calls to it are mediated by calls to MOD. Nonetheless, we note that it may hold individual interest for real-world applications, as it is naturally more than twice as fast as MOD, due to MOD containing two calls to REDC. Thus, applying REDC directly might be more efficient in certain situations.

6.6 Division

We give a quick application of the previous algorithm to finding the quotient of an input polynomial P (in the new representation) by a fixed polynomial A.

Theorem 6.11 (Division)**.** *Let S be a basic set with $|S| = n$.*
Let $A(X)$ be a polynomial with degree at most $n/2$ having no zeroes in S.
There is an algorithm DIV_S, which when given as input:

- $\langle P \wr S \rangle$*, where $P(X) \in \mathbb{F}_q[X]$ with $\deg(P) < n$, and*

runs in time

$$O(n \log n)$$

and computes $\langle Q \wr S \rangle$, where $Q(X)$ is the quotient when $P(X)$ is divided by $A(X)$.

Proof. The algorithm is basically immediate given MOD. Letting $R = \mathsf{MOD}_{S,A}(\langle P \wr S \rangle)$, the algorithm returns $\left\langle \frac{P-R}{A} \wr S \right\rangle$. □

6.7 Exiting to Standard Polynomial Representation

The next computation transforms a polynomial represented by its evaluation on a basic set to the set of coefficients that form the standard representation as $\sum_i a_i X^i$.

Theorem 6.12 (Exit to Standard Polynomial Representation)**.** *Let S be a basic set with $|S| = n$.*
There is an algorithm EXIT_S, which when given as input:

- $\langle P \wr S \rangle$*, where $P(X) \in \mathbb{F}_q[X]$ with $\deg(P) < n$,*

runs in time

$$O(n \log^2 n)$$

and computes the coefficients a_i of $P(X)$ in the standard expansion

$$P(X) = \sum_{i=0}^{n-1} a_i X^i.$$

Proof. Let S_0, S_1 be the moieties of S, and note that we may assume without loss of generality that $0 \notin S_0$. Thus $X^{n/2}$ has no roots in S_0, and in particular an algorithm $\mathsf{MOD}_{S,X^{n/2}}$ exists.

On input $\langle P \wr S \rangle$, the algorithm will compute $\langle U \wr S_0 \rangle$ and $\langle V \wr S_0 \rangle$, where $P(X) = U(X) + X^{n/2} \cdot V(X)$ with $\deg(U), \deg(V) < n/2$, in time $O(n \log n)$. Then by recursively calling EXIT_{S_0} on these two smaller instances and combining the results in the obvious way, we get the coefficients of $P(X)$ in time $O(n \log^2 n)$.

The algorithm uses as advice the values $\left\langle X^{n/2} \wr S_0 \right\rangle$, which depend only on S, as well as auxiliary values used by the MOD algorithm (namely, $\langle Z_0 \wr S_1 \rangle$, $\left\langle Z_0^2 \mathsf{\ rem\ } X^{n/2} \wr S \right\rangle$).

Algorithm EXIT_S:
Input: an evaluation table $\langle \pi \wr S \rangle$

1. If $|S| = 1$ with $S = \{s\}$, return $(\pi(s))$.
2. Let $\langle u \wr S \rangle = \mathsf{MOD}_{S,X^{n/2}}(\langle \pi \wr S \rangle)$.
3. Let
$$(a_0, a_1, \ldots, a_{\frac{n}{2}-1}) = \mathsf{EXIT}_{S_0}(\langle u \wr S_0 \rangle).$$

4. From $\langle \pi \wr S_0 \rangle$, $\langle u \wr S_0 \rangle$ and $\left\langle X^{n/2} \wr S_0 \right\rangle$, compute:

$$\langle v \wr S_0 \rangle = \left\langle \frac{\pi - u}{X^{n/2}} \wr S_0 \right\rangle.$$

5. Let

$$(b_0, b_1, \ldots, b_{\frac{n}{2}-1}) = \mathsf{EXIT}_{S_0}(\langle v \wr S_0 \rangle).$$

6. Return

$$(a_0, a_1, \ldots, a_{\frac{n}{2}-1}, b_0, b_1, \ldots, b_{\frac{n}{2}-1}).$$

Correctness: Suppose $P(X) \in \mathbb{F}_q[X]$ with $\deg(P) < n$. We will analyze what the above algorithm does when its input $\langle \pi \wr S \rangle$ is taken to be $\langle P \wr S \rangle$.

If $n = 1$ the algorithm is clearly correct.

Now assume $n > 1$. Write $P(X) = U(X) + X^{n/2} \cdot V(X)$, where $\deg(U), \deg(V) < n/2$.

Then $U(X) = P(X) \mathsf{\ rem\ } X^{n/2}$. By properties of MOD, we get that Step 2 computes $\langle u \wr S \rangle = \langle U \wr S \rangle$. Thus $\langle u \wr S_0 \rangle = \langle U \wr S_0 \rangle$.

Also note that $V(X) = \frac{P(X) - U(X)}{X^{n/2}}$. Then

$$\langle v \wr S_0 \rangle = \left\langle \frac{\pi - u}{X^{n/2}} \wr S_0 \right\rangle = \left\langle \frac{P - U}{X^{n/2}} \wr S_0 \right\rangle = \langle V \wr S_0 \rangle.$$

By induction, we get that the algorithm correctly computes the coefficients of $U(X)$ and $V(X)$, and by concatenating them together, it computes the coefficients of $P(X)$, as desired.

Running time: The algorithm makes one call to MOD on an instance of size n and two recursive calls to EXIT on instances of size $n/2$. Thus the running time $F(n)$ satisfies the recurrence:

$$F(n) \leq 2F(n/2) + O(n \log n),$$

and thus

$$F(n) \leq O(n \log^2 n).$$

□

6.8 Entering from Standard Polynomial Representation

The next algorithm is the inverse of EXIT, it transforms a polynomial given in standard representation to its evaluation over a basic set.

Theorem 6.13 (Entering from Standard Polynomial Representation)**.** *Let S be a basic set with $|S| = n$.*

There is an algorithm ENTER_S, which when given as input:

- $a_0, a_1, \ldots, a_{n-1} \in \mathbb{F}_q$,

runs in time

$$O(n \log^2 n)$$

and computes $\langle P \wr S \rangle$, where

$$P(X) = \sum_{i=0}^{n-1} a_i X^i.$$

Proof. If $|S| = 1$, the task is trivial.

Otherwise, let S_0, S_1 be the moieties of S. The algorithm is based on writing the polynomial $P(X)$ as:

$$P(X) = U(X) + X^{n/2} \cdot V(X),$$

where $\deg(U), \deg(V) < n/2$, and finding the evaluation tables of U, V on both S_0, S_1. A priori, this seems like reducing an ENTER instance of size n to 4 ENTER instances of size $n/2$ (leading to a quadratic running time), but in fact this can be done by 2 recursive calls to ENTER and 2 invocations of EXTEND.

The algorithm below takes $\left\langle X^{n/2} \wr S \right\rangle$ as advice. This can be precomputed since it only depends on S.

Algorithm ENTER_S**:**
Input: $(a_0, a_1, \ldots, a_{n-1}) \in \mathbb{F}_q^n$.

1. If $|S| = 1$ with $S = \{s\}$
   - Define $g : S \to \mathbb{F}_q$ by $g(s) = a_0$
   - Return $\langle g \wr S \rangle$
2. Let $\langle u_0 \wr S_0 \rangle = \mathsf{ENTER}_{S_0}(a_0, \ldots, a_{\frac{n}{2}-1})$.
3. Let $\langle u_1 \wr S_1 \rangle = \mathsf{EXTEND}_{S_0,S_1}(\langle u_0 \wr S_0 \rangle)$.
4. Let $\langle v_0 \wr S_0 \rangle = \mathsf{ENTER}_{S_0}(a_{n/2}, \ldots, a_{n-1})$.
5. Let $\langle v_1 \wr S_1 \rangle = \mathsf{EXTEND}_{S_0,S_1}(\langle v_0 \wr S_0 \rangle)$.
6. Let
$$\langle \pi \wr S \rangle = \left\langle u_0 + X^{n/2} v_0 \wr S_0 \right\rangle \cup \left\langle u_1 + X^{n/2} v_1 \wr S_1 \right\rangle.$$
7. Return $\langle \pi \wr S \rangle$.

Correctness: The correctness follows immediately from the discussion preceding the algorithm.

Running time: This algorithm makes two recursive calls to ENTER on instances of half the size, makes two invocations of EXTEND on instances of half the size, along with $O(n)$ other operations. Thus the running time $F(n)$ satisfies:

$$F(n) \leq 2F(n/2) + O(n \log n) + O(n),$$

and thus $F(n) \leq O(n \log^2 n)$, as claimed. □

6.9 Chinese Remaindering

The following operation receives as input two polynomials P, Q and computes the polynomial R whose remainders modulo two relatively prime polynomials A, B are P and Q, respectively.

Theorem 6.14 (Chinese Remaindering)**.** *Let S be a basic set with $|S| = n$. Let $S_0 \subseteq S$ be a moiety of S.*

Let $A(X), B(X)$ be relatively prime polynomials with degrees at most $n/2$. Suppose that both $A(X)$ and $B(X)$ are half-disjoint from S; the moieties having no zeroes of A and B may be the same moiety for both or a different one for each.

There is an algorithm $\mathsf{CRT}_{S,S_0,A,B}$, which when given as input:

- *$\langle P \wr S_0 \rangle$, where $P(X) \in \mathbb{F}_q[X]$ with $\deg(P) < n/2$, and*
- *$\langle Q \wr S_0 \rangle$, where $Q(X) \in \mathbb{F}_q[X]$ with $\deg(Q) < n/2$,*

runs in time

$$O(n \log n)$$

and computes $\langle R \wr S \rangle$ *where* R *is the unique polynomial of degree* $< \deg(A) + \deg(B)$ *such that* $R \equiv P \pmod{A}$ *and* $R \equiv Q \pmod{B}$.

Proof. By the usual proof of the Chinese Remainder Theorem, the desired $R(X)$ is of the form:

$$((P(X) \cdot G(X)) \mathsf{\ rem\ } A(X)) \cdot B(X) + ((Q(X) \cdot H(X)) \mathsf{\ rem\ } B(X)) \cdot A(X),$$

where $G(X) = (B(X)^{-1})_{A(X)}$, $H(X) = (A(X)^{-1})_{B(X)}$ depend only on A and B, and have degrees

$$\deg(G), \deg(H) < n/2.$$

Thus the algorithm simply extends $\langle P \wr S_0 \rangle$ and $\langle Q \wr S_0 \rangle$ to find $\langle P \wr S \rangle$ and $\langle Q \wr S \rangle$. Then, using $\langle G \wr S \rangle$ and $\langle H \wr S \rangle$ as advice (which can be precomputed, since they only depend on A, B and S), as well as $\langle A \wr S \rangle$ and $\langle B \wr S \rangle$, we compute:

$$\mathsf{MOD}_{S,A}(\langle P \cdot G \wr S \rangle) \cdot \langle B \wr S \rangle + \mathsf{MOD}_{S,B}(\langle Q \cdot H \wr S \rangle) \cdot \langle A \wr S \rangle$$

which is the desired output. Note that $\deg(P \cdot G), \deg(Q \cdot H) < n$, as MOD requires.

The run-time comes from two invocations of EXTEND, two invocations of MOD, and $O(n)$ other operations, and is thus $O(n \log n)$ overall. □

7 Applications to classical problems

The previous Section 6 presented fast algorithms (and arithmetic circuits) for manipulating polynomials represented by their evaluations on basic sets. This section uses those results to efficiently solve "classical" problems of algebraic computation, in which the polynomials are represented in the "classical" way, as sums of monomials. In all cases below we shall transition to a representation of polynomials by their evaluations on basic sets, and this will result in running times, *over any polynomially large field*, that are as good as those of special, classical-FFT-friendly, finite fields.

7.1 Elementary Symmetric Polynomial Evaluation

Theorem 7.1 (Evaluating Elementary Symmetric Polynomials)**.** *Let* $t < n < q^{O(1)}$. *There is an arithmetic circuit over* $\mathbb{F}_q$ *of size*

$$O(n \log^2 n)$$

which takes as input variables $\alpha_1, \ldots, \alpha_n$ *and computes*

$$\mathsf{Sym}_{n,t}(\alpha_1, \ldots, \alpha_n) := \sum_{J \subseteq [n], |J| = t} \prod_{j \in J} \alpha_j.$$

Proof. We follow the classical approach of computing elementary symmetric polynomials as coefficients of a certain product, except that we work with polynomials in the new representation.

The idea is to compute the coefficients, in the standard monomial representation, of the polynomial

$$P(X) = \prod_{i=1}^{n} (X - \alpha_i) = \sum_{i=0}^{n} (-1)^{n-i} \cdot \mathsf{Sym}_{n,n-i}(\alpha_1, \ldots, \alpha_n) X^i$$

We do this by first computing $\langle P \wr S \rangle$ for a big enough basic set S, and then running $\mathsf{EXIT}_S(P)$ to compute the coefficients of P. Details follow.

By adding some dummy 0 inputs α_i and increasing n by at most a factor 2, we may assume that n is a power of 2. Next, we claim that $\mathbb{F}_q$ can be assumed to contain a basic set of size at least $2n$. Indeed, this

can be done by replacing $\mathbb{F}_q$ by an $O(1)$-degree extension of $\mathbb{F}_q$ which is sufficiently large, of size $O(n^2)$, as needed for a basic set of size at least $2n$ to exist in $\mathbb{F}_q$ (cf. Section 4.2). Moving to a larger q increases the number of arithmetic operations by a factor of at most $O(1)$ because we assume $n < q^{O(1)}$.

Let $m = \log_2(2n)$ and fix arbitrary basic sets $U_0 \subseteq U_1 \ldots \subseteq U_m \subseteq \mathbb{F}_q$ with $|U_j| = 2^j$. We shall compute $\langle P \wr U_m \rangle$ in a bottom-up manner by computing products of terms $P_i(X) := X - \alpha_i, i \in [n]$, of increasing size. We start by computing

$$\langle P_1 \wr U_1 \rangle, \ldots, \langle P_n \wr U_1 \rangle$$

which takes time $O(1)$ for each term P_i (and total time $O(n)$).

Let $Q(X) = \prod_{i=i_0}^{i_0+2^j-1} P_i(X)$ and assume, inductively, that we have already computed $\langle Q' \wr U_j \rangle$ and $\langle Q'' \wr U_j \rangle$ where

$$Q'(X) = \prod_{i=i_0}^{i_0+2^{j-1}-1} P_i(X), \quad Q''(X) = \prod_{i=i_0+2^{j-1}}^{i_0+2^j-1} P_i(X).$$

We shall now compute $\langle Q \wr U_{j+1} \rangle$ as follows:

- Compute $\langle Q' \wr U_{j+1} \rangle$ using the EXTEND[6] algorithm
- Compute $\langle Q'' \wr U_{j+1} \rangle$ using the EXTEND algorithm
- Pointwise multiply the two to obtain $\langle Q \wr U_{j+1} \rangle = \langle Q' \cdot Q'' \wr U_{j+1} \rangle$

Since EXTEND runs in time $O(n \log n)$ and pointwise multiplication runs in time $O(n)$, the running time $F(n)$ for this algorithm satisfies:

$$F(n) \leq 2F(n/2) + O(n \log n) \leq O(n \log^2 n).$$

Finally, once we have $\langle P \wr U_m \rangle$, we can find its standard monomial expansion using EXIT_{U_m}, which also runs in time $O(n \log^2 n)$.

The desired output $\mathsf{Sym}_{n,t}$ is one of the coefficients in this standard monomial expansion, and is thus computed in time $O(n \log^2 n)$, as claimed. □

7.2 Multipoint evaluation over general sets of points

Previously we evaluated polynomials over basic sets in quasi-linear time (see Theorem 6.13). The next result shows that evaluating polynomials over general sets of points can also be done in (slightly worse) quasi-linear time.

Theorem 7.2 (Multipoint polynomial evaluation)**.** *Assume $n < q$. Given any set B of m points in $\mathbb{F}_q$, there exists an arithmetic circuit over $\mathbb{F}_q$ (that depends on B) of size*

$$O(n \log^2 n + m \log^2 m)$$

which takes as input $(a_0, \ldots, a_{n-1}) \in \mathbb{F}_q^n$ and computes $\langle P \wr B \rangle$ for $P(X) = \sum_{i=0}^{n-1} a_i X^i$.

Remark 7.3. Note that if $n \geq q$ then we can first reduce $P(X)$ modulo $X^q - X$ trivially in n steps and get back to the case $n < q$. Moreover, if $m > n$, then we can partition B into $O(m/n)$ sets of size at most $O(n)$, getting a run-time of $O(m \log^2 n)$, whereas if $m < n$, then we can decompose P into $O(n/m)$ polynomials of degree at most $O(m)$, getting a run-time of $O(n \log^2 m)$.

[6] Note that all polynomials computed in this algorithm are monic and of degrees equal to powers of 2. Thus, as noted in Section 6.3, it is natural to extend and multiply these polynomials using $\mathsf{MEXTEND}$ instead of EXTEND, allowing us to take $m = \log_2(n)$, start from evaluations at U_0, and cut down the running time by a factor of 2.

Proof. Let $B = \{b_1, \ldots, b_m\}$. The idea of the algorithm is based on the fact that

$$P(b_i) = P(X) \operatorname{\mathsf{rem}} (X - b_i).$$

To find $P(X) \operatorname{\mathsf{rem}} (X - b_i)$, we start with $P(X) \operatorname{\mathsf{rem}} \prod_{i=1}^{m}(X - b_i)$, and successively compute $P(X) \operatorname{\mathsf{rem}} \prod_{i \in I}(X - b_i)$ for smaller and smaller sets $I \subseteq [m]$. Details follow.

The algorithm starts by running ENTER_{U_a} to find

$$\langle P \wr U_a \rangle,$$

where $a = \log_2 n + O(1)$. This step runs in $O(n \log^2 n)$ time.

Next, we tweak B and U_a until they are of similar sizes, specifically, $2^{a-2} < |B| \leq 2^{a-1}$.

In the case $|B| \leq 2^{a-2}$, let $a' = \lceil \log_2 m \rceil + 1 < a$. We wish to assume that B is half-disjoint from U_a: if it is not the case, we may simply split B into two parts that are each half-disjoint, e.g. $B \cap U_{a-1}$ and $B \setminus U_{a-1}$. Then, assuming half-disjointness, we may run $\mathsf{MOD}_{U_a, \prod_{b \in B}(X-b)}(\langle P \wr U_a \rangle)$ in $O(n \log n)$ time to obtain

$$\left\langle P \operatorname{\mathsf{rem}} \prod_{b \in B} (X - b) \wr U_a \right\rangle.$$

The resulting polynomial will have degree strictly less than $|B| \leq 2^{a'-1}$, and we may restrict its evaluation table $\left\langle P \operatorname{\mathsf{rem}} \prod_{b \in B}(X - b) \wr U_a \right\rangle$ to

$$\left\langle P \operatorname{\mathsf{rem}} \prod_{b \in B} (X - b) \wr U_{a'-1} \right\rangle$$

at no cost while maintaining the fact that it represents $P \operatorname{\mathsf{rem}} \prod_{b \in B}(X - b)$. We then continue to evaluate this polynomial on B, replacing a with a', and noting that $2^{a'-2} < |B| \leq 2^{a'-1}$, and $a' \leq \min(\log_2(n), \log_2(m)) + O(1)$.

In the case $|B| > 2^{a-1}$, split B arbitrarily into l disjoint parts, each of size at most $2^{a-1} = O(n)$, and proceed on each part separately. As in the previous case we further require that each part be half-disjoint from U_a, and observe that again this requires at most one additional part (e.g. by taking one of the parts equal to $B \cap U_{a-1}$), and can be achieved using only $l = O(\frac{m}{n} + 1)$ parts, and note that this bound also covers the previous case (with 1 or 2 parts). The complexity of the remaining work done on each part will be multiplied by l to obtain the total complexity. We continue now with $|B|$ denoting a single part, of size at most 2^{a-1}, and half-disjoint from U_a. Again we have $a \leq \min(\log_2(n), \log_2(m)) + O(1)$. As in the previous case, the next step is to run $\mathsf{MOD}_{U_a, \prod_{b \in B}(X-b)}(\langle P \wr U_a \rangle)$ and restrict to U_{a-1}, yielding

$$\left\langle P \operatorname{\mathsf{rem}} \prod_{b \in B} (X - b) \wr U_{a-1} \right\rangle$$

in $O(n \log n)$ time.

We can now get the desired result by applying the following recursive step, for $j = a - 1, a - 2, \ldots, 1$: Suppose $A(X)$ is a product of $\leq 2^j$ different linear factors. Then we may write $A(X) = A'(X) \cdot A''(X)$, where $\deg(A'), \deg(A'') \leq 2^{j-1}$, and both A', A'' are half-disjoint from U_j. Then given $\langle P \operatorname{\mathsf{rem}} A \wr U_j \rangle$, we can compute

$$\langle P \operatorname{\mathsf{rem}} A' \wr U_j \rangle = \mathsf{MOD}_{U_j, A'}(\langle P \operatorname{\mathsf{rem}} A \wr U_j \rangle)$$
$$\langle P \operatorname{\mathsf{rem}} A'' \wr U_j \rangle = \mathsf{MOD}_{U_j, A''}(\langle P \operatorname{\mathsf{rem}} A \wr U_j \rangle)$$

in time $O(|U_j| \log |U_j|)$, and then restrict the tables to $\langle P \operatorname{\mathsf{rem}} A' \wr U_{j-1} \rangle$, $\langle P \operatorname{\mathsf{rem}} A'' \wr U_{j-1} \rangle$.

At layer j of the recursion we perform 2^{a-j} MOD_{U_j} operations, taking a total run time of $O(|U_a| \log |U_j|)$, and summing over all layers j we get a run time of

$$O(|U_a| \log^2 |U_a|) = O(\min(n \log^2 n, m \log^2 m))$$

per part. Multiplying by the number of parts $l = O(\frac{m}{n} + 1)$ and adding the $O(n \log^2 n)$ from ENTER, we get that the total run time is

$$O(n \log^2 n + m \log^2 m),$$

as claimed. □

7.3 Interpolation from general evaluation sets

In Section 6.7 we showed how to interpolate in quasi-linear time from evaluations on basic sets. The following result, the converse of the previous Theorem 7.2, obtains quasi-linear running time (with somewhat worse parameters) for interpolating from general evaluation sets.

Theorem 7.4 (Polynomial interpolation from general evaluation sets)**.** *Let $B \subseteq \mathbb{F}_q$ be a set of m points. There is an arithmetic circuit over $\mathbb{F}_q$ (depending on B) of size*

$$O(m \log^2 m)$$

which takes as input an evaluation table $\langle \pi \wr B \rangle$ and computes the coefficients a_i of the unique polynomial of degree $< m$:

$$P(X) = \sum_{i=0}^{m-1} a_i X^i,$$

such that $\langle P \wr B \rangle = \langle \pi \wr B \rangle$.

Proof Sketch. Since this algorithm is roughly the opposite of the previous algorithm, instead of applying MOD in each recursive step as done above, we use CRT to do fast Chinese remaindering to compute $\langle P \wr U \rangle$ for a basic set U, followed by calling EXIT($\langle P \wr U \rangle$) to get the desired standard polynomial representation. The running time and analysis are similar to that of the previous Theorem 7.2. □

Acknowledgements

Some of this research was done while SK was visiting StarkWare in 2019. SK is grateful to StarkWare for the warm hospitality and the electrifying atmosphere.

References

[ABR99] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHAES: An encryption scheme based on the diffie-hellman problem. Cryptology ePrint Archive, Report 1999/007, 1999. https://eprint.iacr.org/1999/007.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO*, volume 11694 of *Lecture Notes in Computer Science*, pages 701–732. Springer, 2019.

[BCKL21] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic Curve Fast Fourier Transform Part II: FRI and STARK over all finite fields. In preparation, 2021.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019.

[BCS97] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*. Springer-Verlag, Berlin, 1997.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[BM74] Allan Borodin and R. Moenck. Fast modular transforms. *J. Comput. Syst. Sci*, 8(3):366–386, 1974.

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput*, 38(2):551–607, 2008.

[Can89] David G Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A*, 50(2):285–300, 1989.

[CK91] Cantor and Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *ACTAINF: Acta Informatica*, 28, 1991.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 769–793. Springer, 2020.

[CT65] J. M. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comp.*, 19:297, 1965.

[Deu41] Max Deuring. Die typen der multiplikatorenringe elliptischer funktionenkörper. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 14(1):197–272, Dec 1941.

[DKSS08] Anindya De, Piyush P. Kurur, Chandan Saha, and Ramprasad Saptharishi. Fast integer multiplication using modular arithmetic. In ACM, editor, *STOC '08: proceedings of the 39th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17–20, 2008*, pages 499–506, pub-ACM:adr, 2008. ACM Press.

[Für07] Martin Fürer. Faster integer multiplication. In *STOC'07*, pages 57–66, 2007.

[GM10] Shuhong Gao and Todd Mateer. Additive fast fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010.

[HJB85] Michael T Heideman, Don H Johnson, and C Sidney Burrus. Gauss and the history of the fast fourier transform. *Archive for history of exact sciences*, 34(3):265–277, 1985.

[Hor72a] E. Horowitz. Errata: A fast method for interpolation with preconditioning. *Information Processing Letters*, 1(5):216, October 1972.

[Hor72b] Ellis Horowitz. A fast method for interpolation using preconditioning. *Information Processing Letters*, 1(4):157–163, June 1972.

[HvdH19a] David Harvey and Joris van der Hoeven. Faster polynomial multiplication over finite fields using cyclotomic coefficient rings. *Journal of Complexity*, 54:101404, 2019.

[HvdH19b] David Harvey and Joris van der Hoeven. Polynomial multiplication over finite fields in time $O(n \log n)$. Technical report, HAL, 2019. `http://hal.archives-ouvertes.fr/hal-02070816`.

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time o (n log n). *Annals of Mathematics*, 193(2):563–617, 2021.

[HvdHL17] David Harvey, Joris van der Hoeven, and Grégoire Lecerf. Faster polynomial multiplication over finite fields. *Journal of the ACM (JACM)*, 63(6):1–23, 2017.

[Jou04] Antoine Joux. A one round protocol for tripartite diffie-hellman. *J. Cryptology*, 17:263–276, 2004.

[Kob87] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Lat18] S. Lattès. Sur l'itération des substitutions rationnelles et les fonctions de *Poincaré. C. R. Acad. Sci., Paris*, 166:26–28, 1918.

[LCH14] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han. Novel polynomial basis and its application to reed-solomon erasure codes. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 316–325, 2014.

[Len87] H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):649–673, 1987.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 417–426, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[Pol71] John M Pollard. The fast fourier transform in a finite field. *Mathematics of computation*, 25(114):365–374, 1971.

[Pol74] J. M. Pollard. Theorems on factorization and primality testing. *Mathematical Proceedings of the Cambridge Philosophical Society*, 76(3):521–528, 1974.

[Pos11] Alexey Pospelov. Faster polynomial multiplication via discrete fourier transforms. In Alexander S. Kulikov and Nikolay K. Vereshchagin, editors, *CSR*, volume 6651 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2011.

[Sch77] A. Schönhage. Fast multiplication of polynomials over fields of characteristic 2. *Acta Inf.*, 7(4):395–398, 1977.

[Sch85] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44(170):483–494, 1985.

[Sil07] Joseph H Silverman. *The arithmetic of dynamical systems*, volume 241. Springer Science & Business Media, 2007.

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Graduate texts in mathematics. Springer, Dordrecht, 2nd edition, 2009.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

[SS17] Igor E. Shparlinski and Andrew V. Sutherland. Finding elliptic curves with a subgroup of prescribed size. *International Journal of Number Theory*, 13(1):133–152, February 2017.

[Sta21] StarkWare. ethstark documentation. Cryptology ePrint Archive, Report 2021/582, 2021. https://eprint.iacr.org/2021/582.

[Van92] Scott Vanstone. Responses to nist's proposal. *Communications of the ACM*, pages 50–52, 7 1992.

[Vél71] Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes-Rendus de l'Académie des Sciences, Série I*, 273:238–241, juillet 1971.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013.

[Was08] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition*. Chapman & Hall/CRC, 2 edition, 2008.

[Wat69] William C. Waterhouse. Abelian varieties over finite fields. *Annales scientifiques de l'École Normale Supérieure*, Ser. 4, 2(4):521–560, 1969.

[Wil95] Andrew Wiles. Modular elliptic curves and fermat's last theorem. *Annals of Mathematics*, 141(3):443–551, 1995.

A Proof of the decomposition lemma 3.1

Lemma 3.1 (Decomposition). *Let $\psi(X) \in \mathbb{F}_q(X)$ be a rational map given by:*

$$\psi(X) = \frac{u(X)}{v(X)},$$

where $u(X), v(X) \in \mathbb{F}_q[X]$ are relatively prime polynomials. Let $\delta = \deg(\psi) = \max\{\deg(u), \deg(v)\}$. Let d be a multiple of δ. Then for every $P(X) \in V_d$, there is a unique tuple:

$$(P_0(X), P_1(X), \ldots, P_{\delta-1}(X)) \in (V_{d/\delta})^\delta$$

such that:

$$P(X) = \left(\sum_{i=0}^{\delta-1} X^i \cdot P_i(\psi(X)) \right) \cdot v(X)^{\frac{d}{\delta}-1}. \tag{2}$$

Proof. For general $P_i(Y) \in V_{d/\delta}$, where

$$P_i(Y) = \sum_{j=0}^{d/\delta-1} a_{ij} Y^j,$$

consider the polynomial

$$P(X) = \sum_{i=0}^{\delta-1} X^i P_i(\psi(X)) \cdot v(X)^{\frac{d}{\delta}-1}.$$

Observe that

$$\begin{aligned} P(X) &= \sum_{i=0}^{\delta-1} X^i P_i(u(X)/v(X)) \cdot v(X)^{\frac{d}{\delta}-1} = \sum_{i=0}^{\delta-1} X^i \sum_{j=0}^{d/\delta-1} a_{ij} (u(X)/v(X))^j \cdot v(X)^{\frac{d}{\delta}-1} \\ &= \sum_{i=0}^{\delta-1} \sum_{j=0}^{d/\delta-1} a_{ij} X^i u(X)^j \cdot v(X)^{\frac{d}{\delta}-1-j}, \end{aligned} \tag{11}$$

and thus $P(X) \in V_d$. We shall use the following claim, proved below:

Claim A.1. *For every choice of $P_0(X), P_1(X), \ldots, P_{\delta-1}(X) \in V_{d/\delta}$, not all P_i being zero, the polynomial:*

$$P(X) = \sum_{i=0}^{\delta-1} X^i P_i(\psi(X)) \cdot v(X)^{\frac{d}{\delta}-1}$$

is nonzero.

Together with the fact that the dimension of $\left(V_{d/\delta}\right)^\delta$ equals the dimension of V_d, the theorem follows. □

Proof of Claim A.1. Reordering the right hand side of Eq. (11) gives

$$P(X) = \sum_{j=0}^{d/\delta-1} Q_j(X) u(X)^j v(X)^{d/\delta-1-j},$$

where $Q_j(X) = \sum_{i=0}^{\delta-1} a_{ij} X^i$ is a polynomial of degree $< \delta$.

Since $\deg(\psi) = \delta$, we have that either $\deg(u(X)) = \delta$ or $\deg(v(X)) = \delta$. Suppose $\deg(u(X)) = \delta$, the other case being similar.

The assumption that not all $P_i(X)$ are zero implies that not all $Q_j(X)$ are zero, so let j_0 be the minimal integer such that $Q_{j_0}(X)$ is a nonzero polynomial. Then $P(X)$ is divisible by $u(X)^{j_0}$, and

$$\frac{P(X)}{u(X)^{j_0}} = \sum_{j=j_0}^{d/\delta-1} Q_j(X)u(X)^{j-j_0}v(X)^{d/\delta-1-j},$$

Finally, we observe this polynomial is *nonzero* modulo $u(X)$, since modulo $u(X)$ it equals:

$$Q_{j_0}(X) \cdot v(X)^{d/\delta-1-j_0},$$

$v(X)$ is invertible modulo $u(X)$ (since $v(X)$ is relatively prime to $u(X)$), and $Q_{j_0}(X)$ is nonzero modulo $u(X)$ because it is a nonzero polynomial of degree strictly less than δ. This implies that $P(X)$ is a nonzero polynomial, completing the proof of the claim. □

B Proofs from Section 4

B.1 Proof of Proposition 4.1

Proposition 4.1. *Let $\phi : E \to E'$ be an isogeny between two curves in extended Weierstrass form. Then, in coordinates, we may write*

$$\phi(x, y) = (\psi(x), \xi(x, y)),$$

where $\psi : \mathbb{P}^1 \to \mathbb{P}^1$ is a rational function. Equivalently, if $\pi : E \to \mathbb{P}^1, \pi' : E' \to \mathbb{P}^1$ are the x-projection maps in each curve, then there exists a unique rational function ψ such that the diagram

$$\begin{array}{ccc} E & \xrightarrow{\phi} & E' \\ {\scriptstyle\pi}\downarrow & & \downarrow{\scriptstyle\pi'} \\ \mathbb{P}^1 & \xrightarrow{\psi} & \mathbb{P}^1 \end{array}$$

is commutative.

Proof. As noted in Section 4.1.2, $\pi'(Q) = \pi'(-Q)$ for all points $Q \in E'$. In fact, this equality holds for all the points of $E'(\overline{\mathbb{F}_q})$ – the set of all solutions of the curve equation in the algebraic closure of $\mathbb{F}_q$ (i.e., considering all solutions over all the finite field extensions of $\mathbb{F}_q$). The composition $\pi' \circ \phi : E \to \mathbb{P}^1$ can be represented as an element of $\mathbb{F}_q(X)[Y]/F(X, Y)$ where $F(X, Y) = 0$ is the equation that defines E (see Eq. (6)). Notice that $\mathbb{F}_q(X)[Y]/F(X, Y)$ is a degree 2 extension field of $\mathbb{F}_q(X)$ because F is a degree 2 polynomial in Y with coefficients in $\mathbb{F}_q(X)$, so we can write $\pi' \circ \phi(X, Y) = \psi(X) + Y \cdot \chi(X)$ for some $\psi, \chi \in \mathbb{F}_q(X)$. We know that ϕ is a group homomorphism, so $\pi'(\phi(-Q)) = \pi'(-\phi(Q)) = \pi'(\phi(Q))$ for all points $Q \in E(\overline{\mathbb{F}_q})$. In particular, since Q and $-Q$ have the same x coordinate but different y coordinates (unless $Q = -Q$), then $\chi(x) = 0$ for every x coordinate of a point in $E(\overline{\mathbb{F}_q})$ except for at most 4 points (see [Sil09, Exercise 3.7] or [Was08, Example 2.5]). Since there are infinitely many such points over the algebraic closure $\overline{\mathbb{F}_q}$ we conclude that χ is the constant 0 function and $\pi' \circ \phi(x, y) = \psi(x)$ or equivalently $\pi' \circ \phi = \psi \circ \pi$. □

B.2 Existence of an appropriate G_0 in the proof of Theorem 4.9

Recall that we have constructed a curve E_0 of size N with $\mathsf{K} \mid N$ and $N > 2\mathsf{K}$, where K is a power of 2. Our goal in this section is to show that there exists a subgroup $G_0 < E_0$ which is of size K, and such that there exists a coset C of G_0 with $C \neq -C$.

As noted in Section 4.1.4, E_0 is of rank at most 2, and there is an isomorphism

$$\tau : E_0 \leftrightarrow \mathbb{Z}/(m_1 2^{l_1}\mathbb{Z}) \times \mathbb{Z}/(m_2 2^{l_2}\mathbb{Z})$$

where m_1, m_2 are odd with $m_1 \mid m_2$, $l_1 \leq l_2$, $m_1 m_2 2^{l_1+l_2} = N$ and in particular $l_1 + l_2 \geq \mathsf{k}$. A subgroup G_0 of size K will necessarily be of the form

$$G_0 = \tau^{-1}\big((m_1 2^{l_1-k_1}\mathbb{Z})/(m_1 2^{l_1}\mathbb{Z}) \times (m_2 2^{l_2-k_2}\mathbb{Z})/(m_2 2^{l_2}\mathbb{Z})\big) \simeq \mathbb{Z}/2^{k_1}\mathbb{Z} \times \mathbb{Z}/2^{k_2}\mathbb{Z}$$

with $k_1 \leq l_1$, $k_2 \leq l_2$ and $k_1 + k_2 = \mathsf{k}$, and the quotient E/G_0 is then isomorphic to

$$E_0/G_0 \simeq \mathbb{Z}/(m_1 2^{l_1-k_1}\mathbb{Z}) \times \mathbb{Z}/(m_2 2^{l_2-k_2}\mathbb{Z}).$$

We wish to ensure that this group contains an element C such that $C \neq -C$, or equivalently, $2C \neq 0$. This is clearly the case for any choice of k_1, k_2, except if $m_1 = m_2 = 1$ and $l_1 - k_1, l_2 - k_2 \leq 1$, which are the cases where E_0/G_0 is isomorphic to either the trivial group, $\mathbb{Z}/2\mathbb{Z}$, or $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$. But since $m_1 m_2 2^{l_1-k_1+l_2-k_2} = \frac{N}{\mathsf{K}} > 2$, this happens only when $N = 4\mathsf{K}$ and for the choice $k_1 = l_1 - 1$ and $k_2 = l_2 - 1$. But, by the assumption $\mathsf{K} > 1$ and by $l_2 \geq l_1$, we find $l_2 \geq 2$, thus we may choose instead $k_1 = l_1$ and $k_2 = l_2 - 2$, to obtain $E_0/G_0 \simeq \mathbb{Z}/4\mathbb{Z}$, which indeed contains an element C with $C \neq -C$. □

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il