

Fast Computation on Encrypted Polynomials and Applications

Payman Mohassel

University of Calgary
pmohasse@cpsc.ucalgary.ca

Abstract. In this paper, we explore fast algorithms for computing on encrypted polynomials. More specifically, we describe efficient algorithms for computing the Discrete Fourier Transform, multiplication, division, and multipoint evaluation on encrypted polynomials. The encryption scheme we use needs to be *additively* homomorphic, with a plaintext domain that contains appropriate primitive roots of unity. We show that some modifications to the key generation setups and working with variants of the original hardness assumptions one can adapt the existing homomorphic encryption schemes to work in our algorithms.

The above set of algorithms on encrypted polynomials are useful building blocks for the design of secure computation protocols. We demonstrate their usefulness by utilizing them to solve two problems from the literature, namely the oblivious polynomial evaluation (OPE) and the private set intersection but expect the techniques to be applicable to other problems as well.

1 Introduction

Polynomials are powerful objects with numerous applications in computer science and cryptography in particular. It is easy to find their trace in a range of cryptographic primitives and protocols. A common approach is to use polynomials to represent the input data and then to perform the necessary computation on the new representation. Examples of this approach include well-studied problems such as private information retrieval [2], secret sharing [26], as well as more recent applications such as private set intersection [11], privacy-preserving set operations [20], and private keyword search [12].

In almost all of these constructions, the data is first represented using one or more polynomial and then various operations such as addition, multiplication, and point evaluation on the polynomials are performed in the relevant setting. Furthermore, due to the security requirements, in many cases the polynomials are encrypted, secret-shared, or committed to and the computation is performed on the encrypted versions. Currently, the *efficiency gains* in these protocols are often the result of focusing on the specific problem at hand and taking advantage of the unique properties of that problem.

However, faster algorithms for computing on encrypted polynomials, would potentially lead to more efficient protocols for such applications. *In fact, a natu-*

ral question we study in this paper is whether the existing computer algebra techniques – that have been around for over half a century – can be used to improve the efficiency of the cryptographic protocols for applications that use polynomials to represent their data. We are not aware of any previous work on this question. The advantage of such an approach compared to the customized improvements for specific problems is that the resulting techniques are more general and can be used in a range of applications. In essence, they provide a cryptographic toolkit that can be used by protocol designers in almost a black-box manner. For instance, as we will see in Section 4, our algorithms can be used (without much additional effort) to design simple and efficient protocols for private set intersection (PSI) and batch oblivious polynomial evaluation (OPE). The resulting PSI protocol is a logarithmic factor less efficient than the best existing constructions, while the Batch OPE construction is more efficient than the only other alternative (to the best of our knowledge) of repeating an OPE protocol multiple times. Nevertheless, we mostly present these protocols as simple demonstrations of applicability of our techniques to different privacy-preserving problems.

With the recent developments in designing fully-homomorphic encryption schemes [15], it is possible to run any plaintext algorithm (including fast computer algebra techniques) on encrypted data without the need for decryption (or interaction) and with efficiency that is asymptotically similar to the complexity of the original algorithms. However, the existing fully homomorphic schemes are not yet practical, and hence it is desirable to rely on more efficient schemes but with *limited* homomorphic properties.

When working with encryption schemes with limited homomorphic properties, we need to make sure that the computer algebra algorithms can be computed on encrypted data without the use of the decryption key. Also, since these algorithms work over rings with special properties, we need to ensure that the existing homomorphic encryption schemes can be adapted to work in the required plaintext domains.

In this paper, we demonstrate how given the right setup and the appropriate computational assumptions for the encryption schemes, one can implement some of the main computer algebra techniques for fast polynomial computation on encrypted data. We then show how to use these techniques to develop efficient protocols for several privacy-preserving problems studied in the literature. Next, we describe our contributions in more detail.

1.1 Our Contribution

Fast Computation on Encrypted Polynomials. We design efficient algorithms for performing different computational tasks on encrypted polynomials. Particularly, given polynomials that are encrypted using an *additively* homomorphic encryption scheme, we design algorithms for computing the encrypted Discrete Fourier Transform, polynomial multiplication, division¹, and multipoint evaluation all

¹ We consider the variants of polynomial multiplication and division where one polynomial is encrypted and the other is in plaintext. This variant appears to be sufficient for all the applications we have in mind.

with computational complexities that are linear in the size of the polynomials (upto a logarithmic factor). The constant factors in the complexities are fairly small and specified in the body of the paper.

In designing our algorithms we rely heavily on computer algebra techniques designed for fast symbolic computation. We show that the additive homomorphic properties of the underlying encryption scheme are sufficient to implement these techniques efficiently and non-interactively.

The encrypted Fast Fourier Transform (FFT) algorithm which is the starting point of our work, operates over a commutative ring that contains primitive n th roots of unity for appropriate values of n that are powers of two. We take a closer look at two widely used additively homomorphic encryption schemes in the literature, i.e. the Paillier encryption [25], and the El Gamal encryption [13] with messages in the exponent. We show that modulo some modifications to the key generation setups and working with variants of the original hardness assumptions, namely the *fourier DDH* and *fourier DCRA* assumptions (see Section 2), one can adapt these schemes to work in our protocols.

A number of other papers in the literature investigate the connection between secure computation and polynomials. For example, the work of [10] and [7] looks at evaluating public multivariate polynomials on parties' private inputs. Similar to us (and independently), Cheon et al [4] also consider the use of fast computer algebra techniques with the aim of designing more efficient protocols (multiparty PSI in this case). Our main goal, and what separates our work from the related papers, is to introduce an efficient framework for fast computation on encrypted polynomials with the intention that they can be used in privacy preserving applications.

We demonstrate the usefulness of our encrypted polynomial toolkit by applying it to several cryptographic problems.

Batch Oblivious Polynomial Evaluation. We first look at the oblivious polynomial evaluation problem [24]. In the OPE problem a sender holds a polynomial f of degree n over some ring R . A receiver holding an input $u \in R$ wants to learn $f(u)$ without learning anything else about the polynomial f and without revealing to the sender any information about u . OPE can be seen as a generalization of the 1-out-of- n oblivious transfer, and the private keyword search problem.

One can envision a wide range of applications that can take advantage of oblivious evaluation of polynomials. It is common practice to approximate a function by using a finite number of terms of its Taylor series and evaluating the polynomial corresponding to it. Through such approximations, OPE can be used to obliviously evaluate a variety of functions. One concrete example is the Taylor series approximation of the $\ln(x)$ function used in privacy-preserving data mining protocols [21]. OPE has also been used to design privacy-preserving protocols for machine learning methods such as neural learning [3].

In most applications, one is interested in performing the polynomial evaluation many times and on different input values. The naive solution of rerunning the protocol for each instance, which to the best of our knowledge is the only ex-

isting solution, is costly since the computation cost grows multiplicatively with the number of points being evaluated.

In Section 4.1, we show how our techniques for computation on encrypted polynomials help with designing a protocol for batch evaluation of $k < n$ OPE instances with only $O(n)$ communication and $O(n \log n + k(\log k)^2)$ computation compared to the $O(kn)$ complexity of the naive solution. *Private Set Intersection*. We also use our algorithmic ideas to solve the private set intersection (PSI) problem. The PSI problem involves two or more parties each with their own private data sets who want to learn which data items they share without revealing anything more about their data. A number of organizations dealing with sensitive data such as healthcare providers, insurance companies, law enforcement agencies, and aviation security need to perform such an operation on their data. A large body of work has been studying the design of PSI protocols [11,20,16,18,6,17,9,19,8,1].

In Section 4, we provide two efficient protocols for the PSI problem. While the computational efficiency of our (more efficient) construction is a *logarithmic factor* worse than the best existing works, our protocols are obtained with little effort and are almost immediate applications of the algorithms we introduce. In fact, our main motivation is *not to design more efficient PSI protocols* but to demonstrate the usefulness of the techniques discussed in this paper, in the design of privacy-preserving protocols.

Our batch OPE protocol can be used to solve the private set intersection problem with almost linear computation. In order to find the intersection of two datasets A and B , it is sufficient to represent A via a polynomial f_A (where elements of A are roots of f_A), and then obviously evaluate all elements of B at f_A . Those values evaluated to zero are in the intersection while the rest are not. The resulting PSI protocol requires $O(n)$ communication and $O(n(\log n)^2)$ computation.

Kissner and Song [20] designed a simple and elegant protocol for the private set intersection problem. Given two sets A and B of equal size n , the idea is to represent the sets using polynomials f_A and f_B . Then, one can show that roots of the polynomial $o = rf_A + sf_B$ for random polynomials r and s are either random or in the intersection of the two sets. This idea can be turned into a PSI protocol, and can easily be extended to work for the multiparty case. The main drawback of the construction, as pointed out in the literature, is that it requires computational complexity that is quadratic in the size of the datasets. However, in light of the new algorithms we designed for encrypted polynomial multiplication, the PSI protocol we derive has $O(n)$ communication and $O(n \log n)$ computation. The constant factors in the complexity are also small. In particular the computation involved consists of n encryptions, n decryptions, $2n \log n$ homomorphic additions/subtractions and $n \log n$ homomorphic multiplications.

2 Homomorphic Encryption and Hardness Assumptions

We use a semantically secure public-key encryption scheme that is also additively homomorphic. In particular, we call an encryption scheme E additively homomorphic if given two ciphertexts $E(m_1)$ and $E(m_2)$, we can efficiently compute an encryption of $m_1 + m_2$. We denote this by $E(m_1 + m_2) = E(m_1) +_h E(m_2)$. This implies that given an encryption $E(m)$ and a value c , we can efficiently compute a random encryption $E(cm)$; we denote this by $E(cm) = c \times_h E(m)$. For a vector \mathbf{v} we denote by $E(\mathbf{v})$ an entry-wise encryption of the vector. We define the encryption of a polynomial by the encryption of the vector of its coefficients. We can add two encrypted vectors (polynomials) by adding each encrypted component individually (we use the same notation $+_h$ for this operation as well). When measuring efficiency of our algorithms, we often count the number of homomorphic additions/subtractions and multiplications separately, since homomorphic addition/subtraction tends to be significantly faster for all existing encryption schemes.

There are a number of homomorphic encryption schemes each with their own special properties. Generalized versions of our protocols would work with any encryption scheme that is additively homomorphic as long as the domain of the plaintexts is a *commutative ring*. However, the protocols become simpler and more efficient when we can guarantee that the underlying ring contains a *primitive n th root of unity* for an appropriate choice of n that is a power of two. Hence, it is important to review the existing homomorphic encryption schemes and to determine whether they can be instantiated with plaintext domains that satisfy our requirement. We take a closer look at two widely used additively homomorphic encryption schemes in the literature, i.e. the Paillier encryption [25] and the El Gamal encryption [13] with messages in the exponent and introduce the two hardness assumptions *fourier DDH* and *fourier DCRA* which we need for our protocols. While these new assumptions are closely related to the standard DDH and DCRA assumptions, a better understanding of their hardness requires more careful analysis.

2.1 Additive Variant of El Gamal

Consider the additive variant of the El Gamal encryption scheme where messages are encoded in the exponent. In other words, a message m is encrypted by computing $E_{pk}(m, r) = (g^r, h^r g^m)$ where g is a generator for a cyclic group G_q over which the DDH assumption is hard. It is easy to see that multiplying two ciphertexts adds the two corresponding messages in the exponent. Decryption can be performed efficiently as long as the messages are small. For most of the applications we consider, it is sufficient to determine whether a ciphertext encrypts the message 0 or not, but otherwise there is no need to decrypt a ciphertext. The additive variant of the El Gamal encryption is sufficient for such applications. For example, in our private set intersection protocol (based on polynomial multiplication), it suffices for one party to test whether ciphertexts are encryptions of 0 without fully decrypting them. In applications such

as private keyword search where one is required to decrypt larger plaintexts (i.e. the payload), it is still possible to get around this limitation at the cost of a less efficient construction. In particular, one can use a different polynomial for each bit (or a constant number of bits) of the payload, and invoke the OPE protocol once for each polynomial. Nevertheless, the additive variant of the El Gamal scheme is not a suitable choice for all applications, and should be used with this limitation in mind.

In most cases, the DDH assumption is considered over a finite field \mathbb{F}_q , where q is a prime. In order to make sure the underlying finite field contains the necessary primitive roots of unity, we take advantage of the following lemma:

Lemma 1. [14] *For a prime power q and $n \in \mathbb{N}$, a finite field \mathbb{F}_q contains a primitive n th root of unity if and only if n divides $q - 1$.*

Based on this Lemma, if we choose q carefully such that $q = r2^\ell + 1$ for positive integers r and ℓ , then the corresponding finite field \mathbb{F}_q has n th roots of unity for any $n = 2^s$ for which $s \leq \ell$. The primes of this form are sometimes referred to as *Fourier primes*.

Definition 1 (Fourier Primes). *A prime number p is called a fourier prime if $p = r2^\ell + 1$ for two positive integers r and ℓ .*

Given the above lemma, in order to use the additive variant of El Gamal in our protocols, we need to make sure that the prime we use is in fact a fourier prime. In other words, we need to modify the key generation step in order to sample from the space of primes of the form $r2^\ell + 1$. It is not hard to show that successively testing $2^\ell + 1, 2 \cdot 2^\ell + 1, 3 \cdot 2^\ell + 1, \dots$, for primality is bound to find a fourier prime in a small number of iterations. See Chapter 18 of [14] for more details on efficiency of this step. In addition to generating a fourier prime, the key generation algorithm also needs to find a root of unity. The following lemma guarantees an efficient way of generating such roots of unity.

Lemma 2. *Let $p = r2^\ell + 1$ be a prime number where r and ℓ are positive integers. Let a be a nonsquare modulo p . $a^r \pmod p$ is a primitive 2^ℓ th root of unity in \mathbb{F}_p^* .*

While we are not aware of any algorithms for solving the discrete log problem or any attacks against the DDH assumption that takes advantage of the fact that the prime number we work with is a fourier prime, the effect of this extra assumption requires further analysis. In this paper, we denote this new variant of the DDH assumption with the *fourier DDH assumption*.

Definition 2 (Fourier DDH Assumption). *We denote by fourier DDH, the DDH assumption over the group \mathbb{F}_p where p is a fourier prime.*

2.2 Paillier's Encryption Scheme

Paillier's encryption [25] is another widely used additively homomorphic encryption scheme. In Paillier's encryption, plaintexts are in Z_N where $N = pq$ is

product of two prime numbers. Once again, in order to take advantage of efficient FFT algorithms, we need to make sure that we can find n th roots of unity for all values of n we might be interested in for our applications. Consider the following lemma:

Lemma 3 ([14], **Exercise 8.19**). *Let p, q be two distinct odd primes and $N = pq$. Z_N^* contains a primitive k th root of unity if and only if $k | \text{lcm}(p-1, q-1)$.*

More specifically, in order to use Paillier’s encryption scheme in our protocols, we need to make sure that at least one of the two primes p and q is a fourier prime. We can use the method discussed above to generate such a prime efficiently. Paillier’s encryption is based on the hardness of a mathematical assumption called decisional composite residuosity assumption (DCRA). We denote the variant of the assumption where at least one of the primes is a fourier prime with *fourier DCRA*.

3 Non-Interactive Computation on Encrypted Polynomials

In this section, we describe efficient and non-interactive algorithms for computing on encrypted polynomials. This collection of algorithms on encrypted data provides us with a useful toolkit for designing efficient privacy-preserving protocols for a number of problems.

We start by showing that the FFT algorithm and the interpolation on roots of unity (reviewed in Section A) can be efficiently extended to work on inputs that are encrypted using an *additively homomorphic encryption* scheme. More importantly, performing these computations on encrypted data can be done locally and without the use of the decryption algorithm. Then, we show how to use these two techniques to perform polynomial multiplication, division and multipoint evaluation on partially encrypted data (also non-interactively). The complexity of all the algorithms are linear, upto a logarithmic factor, in the degree of the polynomials we work with.

For simplicity, we assume that the plaintext domain of the encryption scheme is a commutative ring containing a primitive n th root of unity, for an appropriate value of n . As explained in Section 2, it is possible to modify both Paillier’s and the El Gamal encryption schemes to meet this requirement.

Computing the DFT of Encrypted Polynomials. Let f be a polynomial of degree d with coefficients in a finite Ring R , and let $w \in R$ be an n th root of unity where $d < n$ and $n = 2^k$. Given the encryption of the polynomial f via an additively homomorphic encryption scheme (and without the knowledge of the decryption key), we want to compute an encryption of f ’s DFT, namely encryption of the vector $\langle f(w), f(w^2), \dots, f(w^{n-1}) \rangle$. In essence, DFT can be seen as a special case of multipoint evaluation, at the powers of n th root of unity w .

Using *Horner’s rule* and the homomorphic properties of an additive encryption scheme, one can compute an encryption of each $f(w^i)$ via d homomorphic

multiplications and d homomorphic additions. This leads to a total of $O(dn)$ homomorphic operations for computing the DFT. But this is not the best we can do. Using the *Fast Fourier Transform* we show how to reduce the total cost to $O(n \log n)$ homomorphic operations.

The Fast Fourier Transform, or FFT for short, can be used to compute the DFT quickly. The algorithm was (re)discovered by Cooley and Tukey [5], and is one of the most important algorithms in practice. The high level idea is to divide f by $x^{n/2} - 1$ and $x^{n/2} + 1$. Computing the DFT of f is then recursively reduced to computing the DFT of the remainder polynomials of the two divisions. We review the FFT algorithm in Appendix A and here only describe its adaptation to work on encrypted polynomials.

Our main observation is that the FFT algorithm lends itself quite nicely to (additive) homomorphic properties of the encryption scheme, and hence can be computed *non-interactively* and efficiently on encrypted data.

Encrypted FFT Algorithm

$\text{EncFFT}_{w,n}(E_{pk}(\mathbf{f}))$

Input: $n = 2^k \in \mathbb{N}$ for $k \geq 1$; powers of a primitive n th root of unity $w \in R$; the public key pk for an additively homomorphic encryption scheme E with the plaintext domain R , and the encrypted vector $E_{pk}(\mathbf{f}) = \langle E_{pk}(f_0), E_{pk}(f_1), \dots, E_{pk}(f_{n-1}) \rangle$ where $f(x) = \sum_{0 \leq i < n} f_i x^i$.

Output: Encrypted vector $\langle E_{pk}(f(1)), E_{pk}(f(w)), \dots, E_{pk}(f(w^{n-1})) \rangle$

1. If $n = 2$ then return $E_{pk}(f_0) + E_{pk}(f_1) \times_h w$.
2. For $0 \leq j < n/2$ compute $E_{pk}(r_{0,j}) = E_{pk}(f_j) +_h E_{pk}(f_{j+n/2})$.
3. For $0 \leq j < n/2$ compute $E_{pk}(r_{1,j}) = E_{pk}(f_j) -_h E_{pk}(f_{j+n/2})$.
4. For $0 \leq j < n/2$ compute $E_{pk}(r_{1,j}^*) = E_{pk}(r_{1,j}) \times_h w^j$.
5. Let $\mathbf{r}_0 = \langle r_{0,0}, r_{0,1}, \dots, r_{0,n/2-1} \rangle$ and $\mathbf{r}_1^* = \langle r_{1,0}^*, r_{1,1}^*, \dots, r_{1,n/2-1}^* \rangle$.
6. Compute the two encrypted vectors $\mathbf{O}_0 = \langle o_{0,0}, \dots, o_{0,n/2-1} \rangle$ and $\mathbf{O}_1 = \langle o'_{1,0}, \dots, o'_{1,n/2-1} \rangle$ by letting $\mathbf{O}_0 \leftarrow \text{EncFFT}_{w^2, n/2}(E_{pk}(\mathbf{r}_0))$ and $\mathbf{O}_1 \leftarrow \text{EncFFT}_{w^2, n/2}(E_{pk}(\mathbf{r}_1^*))$.
7. Return $\langle o_{0,0}, o_{1,0}, o_{0,1}, o_{1,1}, \dots, o_{0,n-1}, o_{1,n-1} \rangle$.

Efficiency. It is easy to verify that the above protocol requires $n \log n$ homomorphic additions/subtractions and $n/2 \log n$ homomorphic multiplications by w^i . As noted earlier, each homomorphic addition/subtraction translates to a group multiplication (inversion) while each homomorphic multiplication requires an exponentiation. On the other hand, if w is chosen to be small, some of these exponentiations (which are homomorphic multiplications with powers of w) become more efficient. In particular, based on Lemma 2, for a Fourier prime $p = r2^\ell + 1$, we know that $w = a^r$ is primitive root of unity for any non-square a . Hence, as long as we make sure r is a small integer, w can also be chosen to be fairly small.

Encrypted Interpolation on Powers of Roots of Unity. In the interpolation problem, given the vector $\langle f(1), f(w), f(w^2), \dots, f(w^{n-1}) \rangle$, our goal is to

compute the coefficients of the corresponding polynomial f . As discussed in detail in Appendix A.3, interpolation at powers of w can essentially be reduced to computing DFT of a polynomial of degree n , and hence be computed efficiently using the EncFFT algorithm. Next, we briefly describe this algorithm.

Encrypted Interpolation on Powers of n th Root of Unity
 $\text{EncInterpol}_{w,n}(\mathbf{v})$

Input: $n = 2^k \in \mathbb{N}$ with $k \geq 1$, a primitive n th root of unity $w \in R$, public key pk for an additively homomorphic encryption scheme E and an encrypted vector $E_{pk}(\mathbf{v}) = \langle E_{pk}(v_0), \dots, E_{pk}(v_{n-1}) \rangle$ where $w \in R^n$.

Output: The encrypted polynomial $E_{pk}(f)$ of degree n where $f(w^i) = v_i$ for $0 \leq i < n$.

1. Compute $w^{-1}, w^{-2}, \dots, w^{-(n-1)}$.
2. Compute and return $1/n \times_h \text{EncFFT}_{w^{-1},n}(E_{pk}(\mathbf{v}))$

The algorithm's efficiency is almost identical to that of the EncFFT algorithm as the bulk of the computation is one invocation of that algorithm.

Multiplying Encrypted Polynomials. Next we show how to use the EncFFT and EncInterpol algorithms described above to efficiently multiply encrypted polynomials. We are mostly interested in the variant of the multiplication where one polynomial is encrypted and the other polynomial is in plaintext. The idea for the fast polynomial multiplication is to compute the DFT of both polynomials, multiply the components of the DFT individually to obtain the DFT of the product polynomial and then interpolate to recover the product polynomial itself.

As we show next, given an additively homomorphic encryption scheme, this variant can be computed non-interactively and without the need for decrypting any ciphertexts. The algorithm only requires computation that is linear in the degree of the polynomials.

Encrypted Polynomial Multiplication Algorithm
 $\text{EncPolyMult}_{w,n}(E_{pk}(\mathbf{f}), g)$

Input: Encryption of the polynomial $f(x) = \sum_{0 \leq i < d_1} f_i x^i$, i.e. $E_{pk}(\mathbf{f}) = \langle E_{pk}(f_0), \dots, E_{pk}(f_{d_1}) \rangle$ and the plaintext polynomial $g = \sum_{0 \leq i < d_2} g_i x^i$, and a primitive n th root of unity w where $n = 2^k$ and $d_1 + d_2 < n$.

Output: Encryption of the product polynomial $h = fg$.

1. Compute $\langle E_{pk}(f(1)), \dots, E_{pk}(f(w^{n-1})) \rangle \leftarrow \text{EncFFT}_{w,n}(f)$.
2. Compute $\langle g(1), \dots, g(w^{n-1}) \rangle \leftarrow \text{FFT}_{w,n}(g)$.
3. For $0 \leq i < n$, compute $E_{pk}(h(i)) = g(i) \times_h E_{pk}(f(i))$.
4. Let $\mathbf{v}_h \leftarrow \langle h(1), \dots, h(n-1) \rangle$.
5. Compute and return $\text{EncInterpol}_{w,n}(E_{pk}(\mathbf{v}_h))$.

Efficiency. The algorithm requires $2n \log n$ homomorphic additions/subtractions and $n \log n$ homomorphic multiplications, since the EncFFT and the EncInterpol are each invoked exactly once.

Encrypted Polynomial Division. Here, we describe a protocol for performing the division with remainder on encrypted polynomials. We focus on the version of the division protocol where polynomial a of degree n is encrypted, a *monic* polynomial b of degree $m < n$ is in plaintext and we want to compute encryptions of two polynomials q and r such that $a = qb + r$ and r is of degree less than m . Next we review simple algebraic tricks that allow us to reduce the encrypted polynomial division algorithm to the encrypted polynomial multiplication algorithm we described earlier. We note that similar tricks were used in [22] in the context of secure computation but in a different setting and with different applications in mind.

We define *reversal* of a polynomial a as $rev_k(a) = x^k a(1/x)$. When $k = n$, this is the polynomial with the coefficients of a reversed, that is, if $a = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$, then $rev(a) = rev_n(a) = a_0 x^n + \dots + a_{n-1} x + a_n$.

We can now rewrite the division with remainder expression as

$$rev_n(a) = rev_{n-m}(q) rev_m(b) + x^{n-m+1} rev_{m-1}(r)$$

and therefore, $rev_n(a) = rev_{n-m}(q) rev_m(b) \pmod{x^{n-m+1}}$

Note that since we assume b is a monic polynomial, $rev_m(b)$ has the constant coefficient 1 and thus is invertible modulo x^{n-m+1} . Hence we have that

$$rev_{n-m}(q) \equiv rev_n(a) rev_m(b)^{-1} \pmod{x^{n-m+1}},$$

and can obtain $q = rev_{n-m}(rev_{n-m}(q))$ and $r = a - qb$.

In other words, performing the polynomial division with remainder is reduced to inverting the polynomial b modulo x^{n-m+1} , two polynomial multiplications and one polynomial subtraction. Since in our variant of the algorithm b is in plaintext, we can use standard computer algebra algorithms for inverting b which requires $O(n \log n)$ ring operations. Since in all our applications we are only interested in the remainder polynomial, we define the output of the protocol to be only r . The algorithm follows.

Encrypted Polynomial Division Algorithm
 $\text{EncPolyDiv}_{w,n'}(E_{pk}(\mathbf{a}), b)$

Input: Encryption of the polynomial $a(x) = \sum_{0 \leq i \leq n} a_i x^i$, i.e. $E_{pk}(\mathbf{a}) = \langle E_{pk}(a_0), \dots, E_{pk}(a_n) \rangle$ and the plaintext polynomial $b(x) = \sum_{0 \leq i \leq m} b_i x^i$, and a primitive n' th root of unity w where $n' = 2^k$ and $n' > 2n - m + 1$.

Output: Encryption of the remainder polynomial r of degree less than m where $a = qb + r$.

1. Compute w^2, \dots, w^{n-1} .
2. Let $a' = rev_n(a)$. By reversing the order of coefficients of $E_{pk}(a)$, we arrive at the encrypted version of a' denoted by $E_{pk}(\mathbf{a}')$.
3. Compute the polynomial $b' = rev_{n-m}(b)^{-1} \bmod x^{n-m+1}$ using standard computer algebra techniques.
4. Compute $E_{pk}(q_1) = \text{EncPolyMult}_{w,n'}(b', E_{pk}(\mathbf{a}'))$.
5. Compute $E_{pk}(q_2) = E_{pk}(q_1) \bmod x^{n-m+1}$. This is a simple operation that can be performed non-interactively given the additive homomorphic property of the encryption scheme.
6. Compute $E_{pk}(q) = rev_{n-m}(E_{pk}(q_2))$ by reversing the coefficients.
7. Compute $E_{pk}(r) = E_{pk}(a) -_h \text{EncPolyMult}_{w,n'}(E_{pk}(q), b)$.
8. Output $E_{pk}(r)$.

Efficiency. The protocol invokes the EncPolyMult protocol twice, and requires m and $n - m$ homomorphic additions/subtractions in steps 5 and 7, respectively. This leads to a total of $2n' \log n'$ homomorphic multiplication and $4n' \log n' + n'$ homomorphic additions.

Encrypted Multipoint Polynomial Evaluation. Given an encrypted polynomial f of degree n and n points $u_0, \dots, u_{n-1} \in R$, our goal is to compute the encrypted vector $\langle f(u_0), f(u_1), \dots, f(u_{n-1}) \rangle$.

Through the use of the Horner's rule and the additive homomorphic properties of the encryption scheme, it is possible to perform this task with $O(n^2)$ homomorphic operations. However, we are interested in a significantly more efficient algorithm. We have already seen that in the special case when $u_i = w^i$ where w is a primitive n th root of unity, the EncFFT algorithm performs the same task with $O(n \log n)$ homomorphic operations. Our goal is to design an efficient algorithm for the general case of the problem.

Let $n = 2^k$ and $m_i = x - u_i$ for $0 \leq i < n$. We first compute the following sequence of polynomials:

$$M_{i,j} = m_{j2^i} m_{j2^i+1} \cdots m_{j2^i+(2^i-1)} = \prod_{0 \leq \ell < 2^i} m_{j2^i+\ell}$$

for $0 \leq i \leq k = \log n$ and $0 \leq j < 2^{k-i}$. In other words, each $M_{i,j}$ is a sub-product with 2^i factors from $M_{k,0} = \prod_{0 \leq \ell < n} m_\ell$. There exist a simple recursive algorithm for computing the polynomials $M_{i,j}$ for $0 \leq i \leq k$ and $0 \leq j < 2^{k-i}$,

with $O(n(\log n)^2)$ ring operations. Note that since u_i 's are in plaintext, the ring additions and multiplications are significantly cheaper than say encryption or homomorphic multiplication both of which require exponentiation.

The algorithm for multipoint evaluation uses these subproducts in a recursive way. The idea is to divide the polynomial we want to evaluate by two of these subproduct polynomials (see Step 3 of algorithm) and recursively run the multipoint evaluation algorithm on the remainder polynomials. Evaluating the remainder polynomials gives the same result as evaluating the original polynomial itself. We describe the detailed algorithm next:

Encrypted Multipoint Polynomial Evaluation Algorithm
 $\text{EncMultiEval}_n(E_{pk}(f), \mathbf{u})$

Input: Encryption of the polynomial $f(x) = \sum_{0 \leq i < n} f_i x^i$ over R , i.e. $E_{pk}(\mathbf{f}) = \langle E_{pk}(f_0), \dots, E_{pk}(f_n) \rangle$, and the plaintext vector $\mathbf{u} = \langle u_0, u_1, \dots, u_{n-1} \rangle$. Let $n = 2^k$, for $k \in \mathbb{N}$, and w the a primitive n th root of unity.

Output: The encrypted vector $\langle E_{pk}(f(u_1)), \dots, E_{pk}(f(u_{n-1})) \rangle$.

1. Compute the subproduct polynomials $M_{i,j}$ for $0 \leq i \leq k$, and $0 \leq j < 2^{k-i}$, as described above.
2. If $n = 1$ then return f . f is a constant in this case.
3. Compute $E_{pk}(r_0) \leftarrow \text{EncPolyDiv}_{w,n}(E_{pk}(f), M_{k-1,0})$, and $E_{pk}(r_1) \leftarrow \text{EncPolyDiv}_{w,n}(E_{pk}(f), M_{k-1,1})$. Note that r_0 and r_1 are of degree less than $n/2$.
4. Let $\mathbf{u}^0 = \langle u_0, \dots, u_{n/2-1} \rangle$ and $\mathbf{u}^1 = \langle u_{n/2}, \dots, u_n \rangle$. Recursively call the algorithm twice
 - (a) $\langle E_{pk}(r_0(u_0)), \dots, E_{pk}(r_0(u_{n/2-1})) \rangle \leftarrow \text{EncMultiEval}_{w,n/2}(E_{pk}(r_0), \mathbf{u}^0)$
 - (b) $\langle E_{pk}(r_1(u_{n/2})), \dots, E_{pk}(r_1(u_{n-1})) \rangle \leftarrow \text{EncMultiEval}_{w,n/2}(E_{pk}(r_1), \mathbf{u}^1)$
5. Output $E_{pk}(r_0(u_0)), \dots, E_{pk}(r_0(u_{n/2-1})), E_{pk}(r_1(u_{n/2})), \dots, E_{pk}(r_1(u_{n-1}))$.

Efficiency. A careful calculation we omit here (See Chapter 10 of [14]) shows that the above algorithm requires at most $D(n) \log n$ operations where $D(n)$ is the number of operations needed for dividing a polynomial of degree less than $2n$ by a monic polynomial of degree n . Given the complexity of our division algorithm, this leads to at most $6n(\log n)^2$ homomorphic multiplications and $12n(\log n)^2 + 3n \log n$ homomorphic additions/subtractions.

4 Applications

4.1 Batch Oblivious Polynomial Evaluation

In the Oblivious Polynomial Evaluation (OPE) problem a sender holds a polynomial f of degree n over some ring R . A receiver holding an input $u \in R$ wants

to learn $f(u)$ without learning anything else about the polynomial f and without revealing to the sender any information about u .

OPE was originally studied in [24], and can be seen as a generalization of a number of problems studied in the literature such as the 1-out-of- n oblivious transfer (e.g. see [23]), and private keyword search [12]. In case of 1-out-of- n oblivious transfer with a database $D = \{d_1, \dots, d_n\}$, the sender can choose f such that $f(i) = d_i$. In case of private keyword (PKS) search where the database is $D = \{(w_1, d_1), \dots, (w_n, d_n)\}$, the sender can choose a polynomial f where $f(w_i) = d_i$ and w_i is the keyword associated to d_i for $1 \leq i \leq n$.²

Given an additively homomorphic encryption scheme over the ring R , there exist a simple protocol for the OPE problem which requires $O(n)$ encryption/homomorphic operations when implemented using the Horner's rule. However, in most applications one is interested in evaluating the polynomial on many points. Given k evaluation points, the naive solution (and the only solution we are aware of) is to repeat the OPE protocol k times. This leads to $O(kn)$ encryption/homomorphic operations. For large values of k this is inefficient. Next, we use the techniques developed in previous sections to design a protocol for batch OPE that only requires $O(n(\log n) + k(\log k)^2)$ homomorphic operations. The protocol is a natural composition of the `EncPolyDiv` algorithm and the `EncMultiEval` algorithms we have designed. More specifically, to evaluate the polynomial f at points u_1, \dots, u_k , we first divide f by the polynomial $(x - u_1) \cdots (x - u_k)$. Denote the resulting polynomial by r . It is easy to see that $r(u_i) = f(u_i)$ for $1 \leq i \leq k$. Therefore we can use the `EncMultiEval` protocol to evaluate r at u_1, \dots, u_k . The protocol follows:

Batch Oblivious Polynomial Evaluation Protocol
BatchOPE(f, \mathbf{u})

Sender's Input: A polynomial f of degree n with coefficients in R .

Receiver's Input: The vector $\mathbf{u} = \langle u_1, u_2, \dots, u_k \rangle$ in the ring R^n .

Receiver's Output: $\langle f(u_1), \dots, f(u_k) \rangle$

1. Sender generates a key pair (pk, sk) for the public key encryption scheme E , and sends pk to the receiver.
2. Sender sends $E_{pk}(f)$ to the receiver.
3. Receiver computes the polynomial $g = (x - u_1)(x - u_2) \cdots (x - u_k)$.
4. Receiver computes the encrypted polynomial $E_{pk}(r) \leftarrow \text{EncPolyDiv}(E_{pk}(f), g)$ of degree k .
5. Receiver computes $E_{pk}(\mathbf{o}) \leftarrow \text{EncMultiEval}_k(E_{pk}(r), \mathbf{u})$.
6. Receiver computes and sends $E_{pk}(\mathbf{o} + \mathbf{o}_r) = E_{pk}(\mathbf{o}) +_h E_{pk}(\mathbf{o}_r)$ for a random vector $\mathbf{o}_r \in R^k$ to the sender.
7. Sender decrypts the encrypted vector and sends $\mathbf{o} + \mathbf{o}_r$ back to the receiver.
8. Receiver computes and outputs $\mathbf{o} = \mathbf{o} + \mathbf{o}_r - \mathbf{o}_r$.

² A standard assumption made in PKS protocols is that the keywords are unique.

Efficiency: The protocol executes the `EncPolyDiv` protocol on a polynomial of degree n and the `EncMultiEval` protocol on a polynomial of degree k . This adds to a total of $O(n \log n + k(\log k)^2)$ homomorphic operations.

Claim. The `BatchOPE` protocol is secure against semihonest adversaries, if the encryption scheme E is semantically secure.

Proof sketch: The proof of security of the protocol against semi-honest adversaries follows naturally from the semantic security of the encryption scheme, and the randomization steps that take place in the protocol. For completeness we include a sketch of the proof here. This also serves as a good example, since the proofs for all the other protocols follow the same pattern. The full proof of our protocol (deferred to the full version) follows the ideal/real simulation paradigm. In this extended abstract, however, we only give a proof sketch. In case of semi-honest adversaries, it is sufficient to simulate the view of the corrupted party given only his input/output. We have the following two cases:

Sender is corrupted. The Sender’s view is only the message he receives in step 6 of the protocol. This message is an encryption of a uniformly random message vector. This is because the sender does not know the random vector \mathbf{o}_r , which masks the output vector \mathbf{o} . Hence the simulator can simulate the sender’s view by computing an encryption of a randomly chosen message vector.

Receiver is corrupted. The simulator knows the receiver’s input and randomness and the output vector \mathbf{o} and wants to simulate the receiver’s view in the real protocol. He first generates a dummy polynomial f' of appropriate degree (arbitrary coefficients) to use in place of the sender’s input polynomial f . The simulator encrypts f' using the encryption scheme and performs all the computation on the encrypted f' instead to get an encrypted vector $E_{pk}(\mathbf{o}')$. Note that due to the semantic security of the encryption, the view generated using f' is computationally indistinguishable from the one generated using the real polynomial f .

To simulate the only remaining part of receiver’s view (i.e. the message received in step 7), given \mathbf{o}_r generated by the receiver, the simulator computes $\mathbf{o} + \mathbf{o}_r$. The generated vector is identical to the vector in receiver’s view. This completes the proof sketch of security for the above scheme.

4.2 Private Set Intersection via OPE

The set intersection problem involves two or more parties each with their own private data sets who want to learn which data items they share without revealing anything more. As mentioned earlier, several recent works have focused on designing protocols with linear computation and communication complexity.

Interestingly, our batch oblivious polynomial evaluation protocol can be used to solve the private set intersection problem with linear complexity. In order to find the intersection of two datasets A and B , it is sufficient to represent A via a polynomial f_A (where elements of A are roots of f_A), and then obviously evaluate all elements of B at f_A . Those values evaluated to zero are in the intersection while the rest are not. Simple randomization techniques can be added

to avoid leaking any information about those elements that are not in the set. The protocol follows.

Private Set Intersection Protocol (via OPE)

Inputs: Alice holds the dataset A of size n_a , and Bob holds the dataset B of size n_b with elements in R . Without loss of generality we assume that $n_a > n_b$.

Output: Alice learns the intersection of A and B .

1. Alice computes the polynomial f_A of degree n_a by letting the roots of f_A be the elements in A .
2. Bob randomly permutes and arranges the elements of B in a vector $\mathbf{b} \in R^{n_b}$.
3. Alice and Bob run the Steps 1 to 5 of the **BatchOPE**(f_A, \mathbf{b}) protocol. At this point Bob holds the encrypted vector $E_{pk}(\mathbf{o})$ which contains the evaluation of elements of B at polynomial f_A .
4. Bob generates two random vectors $\mathbf{r}_1, \mathbf{r}_2 \in R^{n_b}$. He then computes and sends $E_{pk}(\mathbf{o}_1) = \mathbf{r}_1 \times_h E_{pk}(\mathbf{o}) +_h \mathbf{b}$ and $E_{pk}(\mathbf{o}_2) = \mathbf{r}_2 \times_h E_{pk}(\mathbf{o})$ where the vector multiplications are component-wise multiplications. Note that \mathbf{o}_2 is zero in components corresponding to the elements in the intersection, and random otherwise. For indices corresponding to elements in the intersection, \mathbf{o}_1 holds the actual values.
5. Alice decrypts \mathbf{o}_2 to learn the locations of the elements in the intersection (they are the ones with 0). She marks the indices for those locations, decrypts \mathbf{o}_1 and outputs the values in the marked indices as the final output.

The above protocol can be easily modified to compute the size of the intersection set instead. Particularly, if in the final stage we only compute the vector \mathbf{o}_2 and count the number of zeros, we have a protocol that computes the size of the intersection. *Efficiency.* The bulk of computation consists of running the **BatchOPE** protocol once, and hence the computational complexity of the scheme is $O(n_a \log n_a + n_b (\log n_b)^2)$ homomorphic operations.

Claim. the above protocol is secure against semihonest adversaries if the encryption scheme E is semantically secure.

Similar to proof of Claim 4.1, the security follows from the semantic security of the encryption scheme in a standard way. Details are deferred to the full version.

4.3 Private Set Intersection Via Polynomial Multiplication

Kissner and Song [20] designed a simple and elegant protocol for the private set intersection problem. Given two sets A and B of equal size n , the idea is to represent the sets using polynomials f_A and f_B , respectively. Let r and s be two uniformly random polynomials of degree greater or equal to n over R . The polynomial $o = rf_A + sf_B = \gcd(f_A, f_B)u$, where the polynomial u has

coefficients uniformly distributed in R . Note that an element $a \in R$ is a root of $\gcd(f_A, f_B)$ if and only if a appears in $A \cap B$. Furthermore, if R is large, the fact that u is uniformly distributed implies that with overwhelming probability, the roots of u do not represent any elements in A or B (see [20] for more detail). Hence, one can determine if an element is in the intersection set by testing whether the element evaluates to zero at polynomial o .

This construction easily extends to work for computing the intersection of many sets (held by many users). As discussed in the literature, the main drawback of the construction is that it requires computational complexity that is quadratic in the size of the datasets. However, in light of the algorithms we have designed for computing on encrypted polynomials, this can be improved.

Private Set Intersection Protocol (via EncPolyMult)

Inputs: Alice holds the dataset A of size n_a , and Bob holds the dataset B of size n_b with elements in R . Without loss of generality we assume that $n_a > n_b$.

Output: Bob learns the intersection of A and B .

1. Bob generates a key pair (pk, sk) for the public key encryption scheme E , and sends pk to Alice.
2. Alice and Bob represent their data sets using polynomials f_A and f_B of degree n_a and n_b respectively.
3. Bob encrypts his polynomial and sends $E_{pk}(f_B)$ to Alice.
4. Alice generates two uniformly random polynomials r and s over R of degree n_a . She then computes the encrypted polynomial $E_{pk}(o) = \text{EncPolyMult}(r, E_{pk}(f_B)) +_h \text{PolyMult}(s, f_A)$ and sends it to Bob.
5. Bob then evaluates the encrypted polynomial $E_{pk}(o)$ at his input set using the homomorphic properties of the encryption and for each ciphertext that is an encryption of 0, he outputs the corresponding input as part of the intersection.

Efficiency. Note that the protocol requires one invocation of the EncPolyMult algorithm. Hence the computation consists of n_b encryptions, $n_a + n_b$ decryptions, $2n_a \log n_a$ homomorphic additions and $n_a \log n_a$ homomorphic multiplications.

Claim. The above private set intersection protocol is secure against semi-honest adversaries if the encryption scheme E is semantically secure.

References

1. G. Ateniese, E. De Cristofaro, and G. Tsudik. (if) size matters: size-hiding private set intersection. pages 156–173, 2011.
2. A. Beimel, Y. Ishai, E. Kushilevitz, and J. F. Raymond. Breaking the $O(n^{\frac{1}{2k-1}})$ barrier for information-theoretic private information retrieval. In *FOCS 2002*, pages 261–270, 2002.
3. Y.C. Chang and C.J. Lu. Oblivious polynomial evaluation and oblivious neural learning. *ASIACRYPT*, pages 369–384, 2001.

4. J. H. Cheon, S. Jarecki, and J. H. Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. Cryptology ePrint Archive, Report 2010/512, 2010.
5. J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, pages 297–301, 1965.
6. D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient Robust Private Set Intersection. In *ACNS*, page 142, 2009.
7. D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Secure efficient multiparty computing of multivariate polynomials and applications. In *ACNS*, pages 130–146, 2011.
8. E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. *ASIACRYPT*, pages 213–231, 2010.
9. E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. *FC*, pages 143–159, 2010.
10. M. Franklin and P. Mohassel. Efficient and secure evaluation of multivariate polynomials and applications. In *ACNS*, pages 236–254, 2010.
11. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.
12. M.J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. *TCC*, pages 303–324, 2005.
13. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1985.
14. J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 1999.
15. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
16. C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standard smartcards. In *ACM CCS*, pages 491–500, 2008.
17. C. Hazay and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. *PKC*, pages 312–331, 2010.
18. S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. *TCC*, pages 577–594, 2009.
19. S. Jarecki and X. Liu. Fast secure computation of set intersection. *SCN*, pages 418–435, 2010.
20. L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.
21. Y. Lindell and B. Pinkas. Privacy preserving data mining. In *CRYPTO*, page 36, 2000.
22. P. Mohassel and M. Franklin. Efficient polynomial operations in the shared-coefficient setting. In *PKC*, pages 44–57, 2006.
23. M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, pages 245–254, 1999.
24. M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM Journal on Computing*, pages 12–54, 2006.
25. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUCROCRYPT*, pages 223–238, 1999.
26. A. Shamir. How to share a secret. *Communications of the ACM*, pages 612–613, 1979.

A Computer Algebra Techniques

A.1 Roots of Unity

Definition 3. Let R be a ring, $n \in \mathbb{N}$, and $w \in R$.

- w is an n th root of unity if $w^n = 1$.
- w is a primitive n th root of unity, if it is an n th root of unity, $n \in R$ is a unit in R , and $w^{n/t} - 1$ is not a zero divisor for any prime divisor t of n .

Here n has two meanings: in w it is an integer used as a counter to express the n -fold product of w with itself, and in $n \in R$ it stands for the ring element $n \cdot 1_R \in R$, the n -fold sum of 1_R with itself.

A.2 Computing the Discrete Fourier Transform

Definition 4. Let $f \in R[x]$ be a polynomial of degree $d < n$. The Discrete Fourier Transform (DFT) mapping $DFT_w : R^n \rightarrow R^n$ denotes the evaluation of the polynomial f at the powers of w , i.e., $DFT_w(f) = \langle f(1), f(w), f(w^2), \dots, f(w^{n-1}) \rangle$.

The Discrete Fourier Transform can be seen as a special case of multipoint evaluation, at the powers of n th root of unity w . Next, we introduce the Fast Fourier Transform, or FFT for short, that computes *DFT* quickly. The algorithm was (re)discovered by Cooley and Tukey [5], and is one of the most important algorithms in practice. Let $n \in \mathbb{N}$ be even, $w \in R$ a primitive n th root of unity, and $f \in R[x]$ of degree less than n . To evaluate f at the powers $1, w, w^2, \dots, w^{n-1}$, we divide f by $x^{n/2} - 1$ and $x^{n/2} + 1$ with remainder:

$$f = q_0(x^{n/2} - 1) + r_0 = q_1(x^{n/2} + 1) + r_1$$

for some $q_0, r_0, q_1, r_1 \in R[x]$ of degree less than $n/2$. Due to the special form of the divisor polynomials, the computation of the remainders r_0 and r_1 can be done by adding the upper $n/2$ coefficients of f to, respectively subtracting them from, the lower $n/2$ coefficients. In other words, if $f = F_1 x^{n/2} + F_0$ with $\deg(F_0), \deg(F_1) < n/2$, then $x^{n/2} - 1$ divides $f - F_0 - F_1$, and hence $r_0 = F_0 + F_1$ and $r_1 = F_0 - F_1$. If we plug in a power of w for x we have:

$$f(w^{2\ell}) = q_0(w^{2\ell})(w^{n\ell} - 1) + r_0(w^{2\ell}) = r_0(w^{2\ell}) \quad (1)$$

$$f(w^{2\ell+1}) = q_1(w^{2\ell+1})(w^{n\ell} w^{n/2} + 1) + r_1(w^{2\ell+1}) = r_1(w^{2\ell+1}) \quad (2)$$

for all $0 \leq \ell < n/2$. In the above, we use the facts that $w^{n\ell} = 1$ and $w^{n/2} = -1$, since

$$0 = w^n - 1 = (w^{n/2} - 1)(w^{n/2} + 1)$$

and $w^{n/2} - 1$ is not a zero divisor. It remains to evaluate r_0 at the even powers of w and r_1 at the odd powers. Now, w^2 is a primitive $(n/2)$ th root of unity. It is easy to see that the evaluation of r_0 reduces to a DFT of order $n/2$. The evaluation

of $r_1(w^{2\ell+1}) = r_1^*(w^{2\ell})$ where $r_1^*(x) = r_1(wx)$, reduces to the computation of the coefficients of r_1^* which uses $n/2$ multiplications by powers of w , and a DFT of order $n/2$ for r_1^* . If n is a power of 2, we can proceed recursively to evaluate r_0 and r_1^* at the power of w^2 , which leads to the following FFT algorithm:

Fast Fourier Transform
 $\text{FFT}_{w,n}(f)$

Input: $n = 2^k \in \mathbb{N}$ with $k \in \mathbb{N}$, $f = \sum_{0 \leq j \leq n} f_j x^j \in R[x]$, and the powers w, w^2, \dots, w^{n-1} of a primitive n th root of unity $w \in R$.

Output: $\text{DFT}_{w,n}(f) = \langle f(1), f(w), \dots, f(w^{n-1}) \rangle \in R^n$

1. If $n = 1$ then return f_0 .
2. Compute $r_0 \leftarrow \sum_{0 \leq j < n/2} (f_j + f_{j+n/2})x^j$, and $r_1^* \leftarrow \sum_{0 \leq j < n/2} (f_j - f_{j+n/2})w^j x^j$
3. Call the algorithms $\text{FFT}_{w^2, n/2}(r_0)$ and $\text{FFT}_{w^2, n/2}(r_1^*)$ to compute r_0, r_1^* at the powers of w^2 .
4. Return $r_0(1), r_1^*(1), r_0(w^2), r_1^*(w^2), \dots, r_0(w^{n-2}), r_1^*(w^{n-2})$.

Efficiency. The above algorithm computes $\text{DFT}_{w,n}(f)$ using $n \log n$ additions in R and $(n/2) \log n$ multiplications by powers of w .

A.3 Interpolation on Roots of Unity

It turns out that interpolation at powers of w is again essentially a Discrete Fourier Transform, and can be computed efficiently using the above algorithm. In the interpolation problem, given the vector $\langle f(1), f(w), f(w^2), \dots, f(w^{n-1}) \rangle$, our goal is to compute the coefficients of f . Let V_w be the Vandermonde matrix:

$$V_w = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & \dots & w^{n-1} \\ 1 & w^2 & w^4 & \dots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \dots & w^{(n-1)^2} \end{pmatrix}$$

Then, we can compute the coefficients of f via the following matrix-vector multiplication:

$$\begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix} = (V_w)^{-1} \begin{pmatrix} f(1) \\ f(w) \\ \vdots \\ f(w^{n-1}) \end{pmatrix}$$

The following theorem shows how we can compute inverse of V_w via DFT computation.

Theorem 1 ([14]). *Let R be a ring (commutative, with 1), $n \in \mathbb{N}$, and $w \in R$ be a primitive n th root of unity. Then w^{-1} is a primitive n th root of unity and $(V_w)^{-1} = 1/nV_{w^{-1}}$.*

Based on above theorem we can interpolate on powers of an n th root of unity using the following algorithm:

Interpolation on powers of n th Root of Unity
Interpol $_{w,n}(\mathbf{V})$

Input: $n = 2^k \in \mathbb{N}$ with $k \in \mathbb{N}$, a primitive n th root of unity $w \in R$ and a vector $\mathbf{V} = \langle v_0, v_2, \dots, v_{n-1} \rangle \in R^n$.

Output: Polynomial f of degree n where $f(w^i) = v_i$ for $0 \leq i < n$.

1. Compute $w^{-1}, w^{-2}, \dots, w^{-(n-1)}$.
2. Let the components of \mathbf{V} represent the coefficients of a polynomial denoted by $v(x)$. Compute and return $1/n$ FFT $_{w^{-1},n}(v)$.

The efficiency of the algorithm is similar to that of the DFT algorithm. The only additional cost is to compute powers of w^{-1} . This requires n additional multiplications by w .

A.4 Polynomial Multiplication via FFT

The idea for the polynomial multiplication is to compute the DFT of both polynomials, multiply the components of the DFT individually to obtain the DFT of the product polynomial and then interpolate to recover the product polynomial itself.

Polynomial Multiplication via FFT
PolyMult (f, g)

Input: $n = 2^k \in \mathbb{N}$ with $k \in \mathbb{N}$, $f = \sum_{0 \leq j \leq d_f} f_j x^j$ and $g = \sum_{0 \leq j \leq d_g} g_j x^j$ in $R[x]$ where $d_f + d_g < n$. The algorithm also takes input powers w, w^2, \dots, w^{n-1} of a primitive n th root of unity $w \in R$.

1. Compute $\langle f(1), \dots, f(w^{n-1}) \rangle \leftarrow \text{FFT}_{w,n}(f)$.
2. Compute $\langle g(1), \dots, g(w^{n-1}) \rangle \leftarrow \text{FFT}_{w,n}(g)$.
3. For $0 \leq i < n$ compute $S(i) = f(i)g(i)$.
4. Let $\mathbf{V}_s = \langle S(1), \dots, S(n-1) \rangle$.
5. Computes $S \leftarrow \text{Interpol}_{w,n}(\mathbf{V}_s)$.