Fast Computation on Encrypted Polynomials and Applications

Payman Mohassel
Computer Science Department, University of Calgary
`pmohasse@cpsc.ucalgary.ca`

Abstract

In this paper we design fast algorithms for computing on encrypted polynomials. More specifically, we design efficient algorithms for *encrypted* FFT, polynomial multiplication, division, and multipoint evaluation. The encryption scheme we need only has to be *additively* homomorphic.

The above set of algorithms are useful building blocks for many applications in secure computation. We explore a few of them in this paper. First, we use the new algorithms to design a protocol for *batch oblivious polynomial evaluation* (OPE). Batch OPE can be seen as a generalization of k-out-n oblivious transfer and batch private keyword search. The computational complexity of our protocol is nearly linear in the degree of the polynomial and does not grow with the number of points evaluated at the polynomial. Second, we design two different protocols for the *private set intersection* problem both of which roughly require linear computation and communication. One is based on the batch OPE protocol we design, and the other is an adaptation of the ideas of [Kissner and Song, CRYPTO 2005] using the more efficient encrypted polynomial multiplication we introduce in this paper.

While we mostly focus on security against *semi-honest* adversaries, our algorithmic ideas, in essence, yield nearly *linear size arithmetic circuits* for the set intersection and the batch OPE problems. Implementing these circuits via the new and efficient compiler of [Damgard and Orlandi, CRYPTO 2010] leads to protocols with linear complexity for the private set intersection and batch OPE with security against *malicious* adversaries based on standard and general assumptions.

1 Introduction

Polynomials are powerful and elegant 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 constructions. 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], and secret sharing [22], as well as more recent applications such as private set intersection [8], privacy-preserving set operations [17], and private keyword search [9].

In almost all such cases the data is first represented using one or more polynomials and then various operations such as addition, multiplication, and point evaluation on the polynomials are performed. Furthermore, in many cases due to the security requirements, the polynomials are encrypted (or committed to) and the computation is performed on the encrypted version. Naturally, faster algorithms for computing on encrypted polynomials lead to more efficient protocols for such cryptographic applications.

One such application is the *private set intersection* (PSI) problem. A PSI protocol allows two parties -a server and a client- to interact on their respective input sets such that one or both of the parties learn the intersection of the two sets, but nothing else. A wide range of organizations dealing with sensitive data need to perform such an operation on their data. Examples include healthcare providers, insurance companies, law enforcement agencies, and aviation security. A large body of work has been studying the design of PSI protocols [8, 17, 14, 16, 4, 15, 6]. This line of research is mostly focused on designing protocols with low communication and computation while striving for the strongest security guarantees possible. The problem has proven fairly challenging, and only recently protocols with security against malicious adversaries and roughly linear complexity in the standard model were designed [16, 15].

1.1 Our Contribution

In this paper we study the problem of computing on encrypted polynomials, and its applications.

Computing on Encrypted Polynomials. We design efficient algorithms for performing different computational tasks on encrypted polynomials. One immediate observation is that with the recent developments in designing fully-homomorphic encryption schemes [11], it is possible to run any plaintext algorithm on encrypted data without the need for decryption (or interaction) and with efficiency that is asymptotically similar to the complexity of the original algorithm. However, the existing fully homomorphic schemes are far from practical, and hence it is desirable to rely on more efficient schemes but with limited homomorphic properties.

Particularly, given polynomials that are encrypted using an *additively* homomorphic encryption scheme, we design algorithms for computing the encrypted Discrete Fourier Transform, polynomial multiplication, division[1], and multipoint evaluation all with computational complexity that is linear in the size of the polynomials (upto a logarithmic factor). The constat 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 demonstrate the usefulness of our encrypted polynomial toolkit by applying it to several cryptographic problems mentioned above.

Oblivious Polynomial Evaluation. We first look at the oblivious polynomial evaluation problem [20]. In the OPE problem a sender holds a polynomial f of degree n over some ring R. A receiver holding an input

[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.

$u \in R$ 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 other client-server scenarios which can be reduced to oblivious evaluation of polynomials. In many such applications, the polynomial evaluation needs to be repeated multiple times for different values. The naive solution of rerunning the protocol for each instance is costly and the cost grows multiplicatively with the number of points being evaluated.

We show how our techniques 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 then move onto the private set intersection 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. We use our algorithmic ideas to design two different PSI 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 obliviously 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 [17] 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 and $n \log n$ homomorphic multiplications.

The asymptotic efficiency of both of the above PSI protocols can be improved further via the application of the hashing-to-bin technique (e.g. see [8]). This reduces their computational costs to $O(n \log \log \log n)$ which slightly improves on the complexity of the scheme of Freedman *et al.* [8].

Security Against Malicious Adversaries. So far the protocols we talked about only provide security against semihonest adversaries. When considering semi-honest adversaries, the use of an additively homomorphic encryption scheme allows us to keep the amount of interaction low. However, the underlying algorithmic ideas for batch OPE and PSI are not limited to this form of implementation. More specifically, the ideas can be interpreted as presenting efficient polynomial-based algorithms for computing intersection of two sets and batch polynomial evaluation, where the algorithms can be represented as arithmetic circuits of roughly linear size.

Recently, Damgard and Orlandi [5] designed a protocol for securely computing arithmetic circuits with security against malicious adversaries. The main components of their scheme are a trapdoor homomorphic commitment scheme, and a semihonest multiplication protocol. If the number of multiplication gates in the arithmetic circuit is M and the security parameter is s, the computational complexity of their scheme is $O(M + s)$. This leads to batch OPE and PSI protocols with security against *malicious* adversaries (in the standard model) with computational complexity $O(n \log n + s)$. The protocol can be instantiated using a wide range of assumptions such as DDH, QR, and DCR assumption (see [5] for more detail).

The only other PSI protocols with security against malicious adversaries in the standard model and with roughly linear complexity are the works [16] and [15]. However the protocol we get has some advantages over both. The work of [16] requires the elements in the input sets to be chosen from a small domain since an exhaustive search over the domain is needed. The work of [15] has an additional factor of m in its complexity where m is the number of bits needed to represent the set elements. For large domains, this can be costly since the complexities are counting the number of exponentiations needed. Our protocol does not have these limitations. Finally, the compiler of [5] divides the computation into an offline and an online phase, where the offline phase is independent of the inputs, and the online phase is quite fast (small constant factors). Our protocol inherits the same useful properties.

2 Preliminaries

2.1 Homomorphic Encryption

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 encryptions $E(m_1)$ 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 $\vec{v}$ we denote by $E(\vec{v})$ 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 and multiplications separately, since homomorphic addition tends to be significantly faster for all existing encryption schemes.

There are a number of homomorphic encryption schemes each with their own special properties. Our protocols work with any encryption scheme that is additively homomorphic and where 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 nth root of unity for an appropriate choice of n that is a power of two.

In case of the additive variant of the El Gamal encryption scheme (messages are in the exponent), the plaintext domain is the ring $\mathbb{Z}_p$ where p is a prime. Based on Lemma A.2 from Appendix A, if we make sure that $p = 2^\ell + 1$ for a positive integer ℓ, then $\mathbb{Z}_p$ has nth root of unity for any $n = 2^s$ for which $s \leq \ell$. Hence, we only need to modify the key generation step in order to sample from the space of primes of the form $2^\ell + 1$. Note that this does not seem to have any security side effects since p is publicly known, and the structure of p does not seem to help in solving the discrete-log problem.

We did not explore the possiblity of providing similar guarantees of existence of primitive roots of unity for other additively homomorphic encryption schemes such as the Paillier's encryption scheme [21] or the extended Goldwasser-Micali scheme [13][2]. But as discussed in more detail in Appendix A.5 using some additional computer algebra techniques, one can remove the condition of existence of roots of unity at a small computational cost. For ease of composition however, we describe the protocols for the simpler case where such roots of unity are available.

[2]The GM encryption scheme is originally defined for messages over GF(2). But it is possible to extend the scheme to work on larger plaintext domains (for example see [7]).

2.2 Security Definitions

The security definitions we use to prove our protocols secure follow the ideal-world/real-world simulation paradigm. Roughly speaking, in this framework, a protocol is secure if anything that an adversary can do in the real protocol can be simulated by a simulator in an ideal world where participants send their inputs to a trusted party who performs the computation on their behalf and sends back their corresponding outputs. We give a brief description of the security definitions for the two-party case in Appendix B. Please see [12] for more detail.

3 Algorithms on Encrypted Polynomials

First we show 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 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. This collection of efficient algorithms on encrypted data turns out to be a very useful tool in designing efficient protocols for private computation.

For simplicity, we assume that the plaintext domain of the encryption scheme is a commutative ring that contains a primitive nth root of unity, for an appropriate value of n. As briefly discussed in Section A.5, it is possible to remove this condition at the cost of increasing the complexity by a small factor.

3.1 Computing 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 nth 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 $\langle f(w), f(w^2), \ldots, f(w^{n-1})\rangle$.

Using Horner's rule and the homomorphic properties of an additive encryption scheme, we 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 FFT transform we show how to reduce the total cost to $O(n \log n)$ homomorphic operations.

The 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
$\mathsf{EncFFT}_{w,n}(E_{pk}(\vec{f}))$

```
Input: $n = 2^k \in \mathbb{N}$ with $k \in \mathbb{N}$; 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 $E_{pk}(\vec{f}) = \langle E_{pk}(f_0), E_{pk}(f_1), \ldots, E_{pk}(f_{n-1})\rangle$ where $f(x) = \sum_{0 \le i < n} f_i x^i$.
Output: Encrypted vector $\langle E_{pk}(f(1)), E_{pk}(f(w)), \ldots, E_{pk}(f(w^{n-1}))\rangle$

  1. If $n = 1$ then return $E_{pk}(f_0)$.
  2. For $0 \le j < n/2$ compute $E_{pk}(r_{0,j}) = E_{pk}(f_j) +_h E_{pk}(f_{j+n/2})$.
  3. For $0 \le j < n/2$ compute $E_{pk}(r_{1,j}) = E_{pk}(f_j) -_h E_{pk}(f_{j+n/2})$.
  4. For $0 \le j < n/2$ compute $E_{pk}(r^*_{1,j}) = E_{pk}(r_{1,j}) \times_h w^j$.
  5. Let $\vec{r_0} = \langle r_{0,0}, r_{0,1}, \ldots, r_{0,n/2-1}\rangle$ and $\vec{r^*_1} = \langle r^*_{1,0}, r^*_{1,1}, \ldots, r^*_{1,n/2-1}\rangle$.
  6. Compute the two encrypted vectors $\vec{O} = \langle o_0, \ldots, o_{n/2-1}\rangle$ and $O' = \langle o'_0, \ldots, o'_{n/2-1}\rangle$ where $\vec{O} \leftarrow \mathsf{EncFFT}_{w^2,n/2}(E_{pk}(\vec{r_0}))$ and $\vec{O'} \leftarrow \mathsf{EncFFT}_{w^2,n/2}(E_{pk}(\vec{r^*_1}))$.
  7. Return $\langle o_0, o'_0, o_1, o'_1, \ldots, o_{n-1}, o'_{n-1}\rangle$.
```

Efficiency. It is easy to verify that the above protocol requires $n \log n$ homomorphic additions and $n/2 \log n$ homomorphic multiplications by w^i. As noted earlier, each homomorphic addition translates to a group multiplication 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.

3.2 Encrypted Interpolation on Powers of Roots of Unity

Encrypted Interpolation on Powers of nth Root of Unity
$\mathsf{EncInterpol}_{w,n}(\vec{v})$

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

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

  1. Compute $w^{-1}, w^{-2}, \ldots, w^{-(n-1)}$.
  2. Compute and return $1/n \times_h \mathsf{EncFFT}_{w^{-1},n}(E_{pk}(\vec{v}))$
```

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.

3.3 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 a variant of the problem where one polynomial is encrypted and the other polynomial is in plaintext.

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
$\mathsf{EncPolyMult}_{w,n}(E_{pk}(\vec{f}), g)$

Input: Encryption of the polynomial $f(x) = \sum_{0 \le i < d_1} f_i x^i$, i.e. $E_{pk}(\vec{f}) = \langle E_{pk}(f_0), \ldots, E_{pk}(f_{d_1}) \rangle$ and the plaintext polynomial $g = \sum_{0 \le i < d_2} g_i x^i$, and a primitive nth root of unity w where $n = 2^k$ and $d_1 + d_2 < n$.
Output: Encryption of the product polynomial $h = fg$.

1. Compute $\langle E_{pk}(f(1)), \ldots, E_{pk}(f(w^{n-1})) \rangle \leftarrow \mathsf{EncFFT}_{w,n}(f)$.
2. Compute $\langle g(1), \ldots, g(w^{n-1}) \rangle \leftarrow \mathsf{FFT}_{w,n}(g)$.
3. For $0 \le i < n$, compute $E_{pk}(h(i)) = g(i) \times_h E_{pk}(f(i))$.
4. Let $\vec{v_h} = \langle h(1), \ldots, h(n-1) \rangle$.
5. Compute and return $\mathsf{EncInterpol}_{w,n}(E_{pk}(\vec{v_h}))$.

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

3.4 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 [18] 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 $a = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, then

$$rev(a) = rev_n(a) = a_0 x^n + \cdots + a_{n-1} x + a_0$$

We can now rewrite the division with remainder expression as

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

an therefore,

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

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

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

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
$\mathsf{EncPolyDiv}_{w,n'}(E_{pk}(\vec{a}), b)$

Input: Encryption of the polynomial $a(x) = \sum_{0 \le i \le n} a_i x^i$, i.e. $E_{pk}(\vec{a}) = \langle E_p k(a_0), \ldots, E_{pk}(a_n) \rangle$ and the plaintext polynomial $b(x) = \sum_{0 \le i \le m} b_i x^i$, 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 $w^2, \ldots, w^{n-1}$.
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 $E_{pk}(\vec{a'})$.
3. Compute the polynomial $b' = rev_{n-m}(b)^{-1} \bmod x^{n-m+1}$ using standard computer algebra techniques.
4. Compute $E_{pk}(q_1) = \mathsf{EncPolyMult}_{w,n'}(b', E_{pk}(\vec{a'}))$.
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 $E_{pk}(r) = E_{pk}(a) -_h \mathsf{EncPolyMult}_{w,n'}(E_{pk}(q), b)$.
8. Output $E_{pk}(r)$.

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

3.5 Encrypted Multipoint Polynomial Evaluation

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

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 nth 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 $0 \le i < n$. We first compute the following sequence of polynomials:

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

for $0 \le i \le k = \log n$ and $0 \le j < 2^{k-i}$. In other words, each $M_{i,j}$ is a subproduct with 2^i factors from $M_{k,0} = \prod_{0 \le \ell < n} m_\ell$. There exist a simple recursive algorithm for computing the polynomials $M_{i,j}$ for $0 \le i \le k$ and $0 \le j < 2^{k-i}$, 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 with remainder the polynomial we want to evaluate by two of these subproduct polynomials and recursively run the 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
$\mathsf{EncMultiEval}_n(E_{pk}(f), \vec{u})$

Input: Encryption of the polynomial $f(x) = \sum_{0 \le i < n} f_i x^i$ over R, i.e. $E_{pk}(\vec{f}) = \langle E_{pk}(f_0), \ldots, E_{pk}(f_n) \rangle$, and the plaintext vector $\vec{u} = \langle u_0, u_1, \cdots, u_n \rangle$. Let $n = 2^k$, for $k \in \mathbb{N}$, and w the a primitive nth root of unity.
Output: The encrypted vector $\langle E_{pk}(f(u_1)), \cdots, E_{pk}(f(u_1)) \rangle$.

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

Efficiency. A careful calculation we omit here (See Chapter 10 of [10]) 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.

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. The precise security definitions are those of secure multiparty computation described in Section 2.2.

OPE was originally studied in [20], 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 [19]), and private keyword search [9]. In case of 1-out-of-n oblivious transfer with a database $D = \{d_1, \ldots, d_n\}$, the sender can choose f such that $f(i) = d_i$. In case of private keyword (PKS) search where the database is $D = \{(w_1, d_1), \ldots, (w_n, d_n)\}$, the sender can choose a polynomial f where $f(w_i) = d_i$ and w_i is the keyword associated to d_i for $1 \le i \le n$. [3]

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

[3]A standard assumption made in PKS protocols is that the keywords are unique

Horner's rule. However, in most applications one is interested in evaluating the polynomial on many points. Given k evaluation points, this can be seen as a generalization of the k-out-of-n oblivious transfer problem. The naive solution is to repeat the OPE protocol k times. This leads to $O(kn)$ encryption/homomorphic operations. For large values of k this is rather 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 $\mathsf{EncPolyDiv}$ algorithm and the $\mathsf{EncMultiEval}$ algorithms we have designed. More specifically, to evaluate the polynomial f at points $u_1, \cdots, u_k$, 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 $\mathsf{EncMultiEval}$ protocol to evaluate r at $u_1, \ldots, u_k$. The protocol follows:

Batch Oblivious Polynomial Evaluation Protocol
$\mathsf{BatchOPE}(f, \vec{u})$

Sender's Input: A polynomial f of degree n with coefficients in R.
Receiver's Input: The vector $\vec{u} = \langle u_1, u_2, \cdots, u_k\rangle$ in the ring R^n.
Receiver's Output: $\langle f(u_1), \ldots, f(u_k)\rangle$

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 $E_{pk}(r) \leftarrow \mathsf{EncPolyDiv}(E_{pk}(f), g)$ of degree k.
5. Receiver computes $\mathbf{E}_{pk}(\vec{o}) \leftarrow \mathsf{EncMultiEval}_k(E_{pk}(r), \vec{u})$.
6. Receiver computes and sends $E_{pk}(o \vec{+} o_r) = E_{pk}(\vec{o}) +_h E_{pk}(\vec{o_r})$ for a random vector $o_r \in R^k$ to the sender.
7. Sender decrypts the encrypted vector and sends $\vec{o} + \vec{o_r}$ back to the receiver.
8. Receiver computes and outputs $\vec{o} = \vec{o} + \vec{o_r} - \vec{o_r}$.

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

Claim 4.1 The $\mathsf{BatchOPE}$ 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. We prove the security using the ideal/real simulation paradigm. 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 $\vec{o_r}$ which masks the output vector $\vec{o}$. 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 $\vec{o}$ 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 $E_{pk}(\vec{o'})$. 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 $\vec{o_r}$ generated by the receiver, the simulator computes $\vec{o} + \vec{o_r}$. 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 about their private data. 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 obliviously 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: 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 $\vec{b} \in R^{n_b}$.
3. Alice and Bob run the Steps 2 to 5 of the $\mathsf{BatchOPE}(f_A, \vec{b})$ protocol. At this point Bob holds the encrypted vector $E_{pk}(\vec{o})$ which contains the evaluation of elements of B at polynomial f_A.
4. Bob generates two random vectors $\vec{r_1}, \vec{r_2} \in R^{n_b}$. He then computes and sends $E_{pk}(\vec{o_1}) = \vec{r_1} \times_h E_{pk}(\vec{o}) +_h \vec{b}$ and $E_{pk}(\vec{o_2}) = \vec{r_2} \times_h E_{pk}(\vec{o})$ where the vector multiplications are component-wise multiplications. Note that $\vec{o_2}$ is zero in components corresponding to the elements in the intersection, and random otherwise. For indices corresponding to elements in the intersection, $\vec{o_1}$ holds the actual values.
5. Alice decrypts $\vec{o_2}$ to learn the locations of the elements in the intersection (they are the ones with 0). She marks the indices for those locations, decrypts $\vec{o_1}$ 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 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 $\mathsf{BatchOPE}$ 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 4.2 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 [17] 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 [17] 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: The intersection of A and B.

1.
2. Bob generates a key pair (pk, sk) for the public key encryption scheme E, and sends pk to Alice.
3. Alice and Bob represent their data sets using polynomials f_A and f_B of degree n_a and n_b respectively.
4. Bob encrypts his polynomial and sends $E_{pk}(f_B)$ to Alice.
5. Alice generates two uniformly random polynomials r and s over R of degree n_a. She then computes the encrypted polynomial $E_{pk}(o) = \mathsf{EncPolyMult}(r, E_{pk}(f_B)) +_h \mathsf{PolyMult}(s, f_A)$ and sends it to Bob.
6. Bob decrypts to recover o. He then outputs those elements in his set that evaluate to 0 at polynomial o.

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

Further Improvement in Efficiency. It is possible to further improve the efficiency of the above two protocols for set intersection using the hash-to-bin technique used in [8]. Particularly, before representing her dataset with a polynomial, Alice first hashes the elements in her database to ℓ bins using a public hash function H. Using hashing techniques introduced in [1], it is possible to guarantee that with high probability the maximum number of items mapped to each bin is $M = O(n/\ell + \log\log \ell)$. By letting $\ell = n/\log\log n$, we have $M = O(\log\log n)$.

For every element y in his dataset, Bob puts y in the bins into which it could be mapped. Alice and Bob then engage in a series of private set intersection protocols one for each bin. With a little bit of care, the output of the small PSI protocols can be combined to result in a private set intersection protocol for the original sets. Given the efficiency of our PSI protocols, running them for each bin requires $O(M \log M) = O(\log\log n \log\log\log n)$ computation, and since there are a total of $n/\log\log n$ bins, this leads to total of $O(n \log\log\log n)$ computational complexity. While the idea is very similar to the previous hashing-to-bin protocol of [8], we get a slightly better complexity due to the new algorithms we introduced. In other words the $\log\log n$ factor is replaced by a $\log\log\log n$ factor.

4.4 Security Against Malicious Adversaries

So far we have focused on the security of our protocols against semi-honest adversaries. When considering semi-honest adversaries, the use of an additively homomorphic encryption scheme allows us to keep the amount of interaction low. However, the underlying algorithmic ideas for batch OPE and private set intersection are not limited to this form of implementation. More specifically, the ideas can be interpreted as presenting efficient polynomial-based algorithms for computing set intersection and batch polynomial evaluation, where the algorithms only require linear computation, and all the computation are ring addition and multiplications. Hence the designed algorithms can be seen as arithmetic circuits for the set intersection or the batch OPE.

Recently, Damgard and Orlandi [5] designed a protocol for securely computing arithmetic circuits with security against malicious adversaries. The main components of there scheme is a trapdoor homomorphic commitment scheme, and a semi-honest multiplication protocol combined with cut-and-choose techniques. If the number of multiplication gates in the arithmetic circuit is M and the security parameter is s, the computational complexity of their scheme is $O(t(M + s))$ in the t-party case. We do not outline the details of the protocol since it is a straightforward application of the result of [5] to our algorithms, but summarize the results via the following two theorems:

Theorem 4.4 Assuming homomorphic trapdoor commitment schemes and a semi-honest multiplication protocol, there exist a secure two-party protocol for k-Batch OPE, with security against malicious adversaries, and with computational complexity $O(n \log n + k(\log k)^2 + s)$ where n is the degree of the polynomial and s is the security parameter.

Theorem 4.5 Assuming homomorphic trapdoor commitment schemes and a semi-honest multiplication protocol, there exist a secure two-party private intersection protocol with security against malicious adversaries with computation complexity $O(n \log n + s)$ where n is size of the input sets and s is the security parameter.

The trapdoor homomorphic commitment can be instantiated based on assumptions such as the discrete logarithm, RSA, and others. The multiplication protocol requires slightly stronger hardness assumptions but can be instantiated using homomorphic encryption schemes, OT, and more.

References

[1] Y. Azar, A.Z. Broder, A.R. Karlin, and E. Upfal. Balanced allocations (extended abstract). In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 593–602. ACM, 1994. (Cited on page 13.)

[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 *Proc. of the 43rd IEEE Symp. on Foundations of Computer Science*, pages 261–270, 2002. (Cited on page 2.)

[3] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965. (Cited on page 16.)

[4] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient Robust Private Set Intersection. In *Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, page 142, 2009. (Cited on page 2.)

[5] I. Damgård and C. Orlandi. Multiparty Computation for Dishonest Majority: from Passive to Active Security at Low Cost. *Advances in Cryptology–CRYPTO 2010*, pages 558–576, 2010. (Cited on page 3, 4, 13.)

[6] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. *Financial Cryptography and Data Security*, pages 143–159, 2010. (Cited on page 2.)

[7] M. Franklin and P. Mohassel. Efficient and Secure Evaluation of Multivariate Polynomials and Applications. In *Applied Cryptography and Network Security*, pages 236–254. Springer, 2010. (Cited on page 4.)

[8] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2004. (Cited on page 2, 3, 12, 13.)

[9] M.J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. *Theory of Cryptography*, pages 303–324, 2005. (Cited on page 2, 9.)

[10] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 1999. (Cited on page 9, 17.)

[11] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009. (Cited on page 2.)

[12] O. Goldreich. *Foundations of Cryptography, Voume II Basic Applications*. Cambridge University Press, 2004. (Cited on page 5.)

[13] S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 365–377, New York, NY, USA, 1982. ACM Press. (Cited on page 4.)

[14] C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standardsmartcards. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 491–500. ACM, 2008. (Cited on page 2.)

[15] C. Hazay and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. *Public Key Cryptography–PKC 2010*, pages 312–331, 2010. (Cited on page 2, 4.)

[16] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. *Theory of Cryptography*, pages 577–594, 2009. (Cited on page 2, 4.)

[17] L. Kissner and D. Song. Privacy-preserving set operations. In *Advances in Cryptology–CRYPTO 2005*, pages 241–257. Springer, 2005. (Cited on page 2, 3, 12.)

[18] P. Mohassel and M. Franklin. Efficient polynomial operations in the shared-coefficient setting. In *Proc. of Public Key Cryptography coference, PKC*, volume 3958, pages 44–57. Springer, 2006. (Cited on page 7.)

[19] M. Naor and B. Pinkas. Oblivious transfer and polynomial evalutation. In *Proc. of the 31st ACM Symp. on the Theory of Computing*, pages 245–254, 1999. (Cited on page 9.)

[20] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM Journal on Computing*, 35(5):1254, 2006. (Cited on page 2, 9.)

[21] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT '99*, pages 223–238, 1999. (Cited on page 4.)

[22] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979. (Cited on page 2.)

A Computer Algebra Techniques

A.1 Roots of Unity

Definition A.1 Let R be a ring, $n \in \mathbb{N}$, and $w \in R$.

- w is an nth root of unity if $w^n = 1$.
- w is a primitive nth root of unity, if it is an nth 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.

The following is a useful lemma regarding the existence of roots of unity in finite fields. We make use of the lemma to ensure that the additive variant of the elgamal encryption scheme has the primitive roots of unity we need (see Section 2.1).

Lemma A.2 For a prime power q and $n \in \mathbb{N}$, a finite field $\mathbb{F}_q$ contains primitive nth root of unity if and only if n divides $q - 1$.

A.2 Computing the Discrete Fourier Transform

Definition A.3 Let $f \in R[x]$ be a polynomial of degree $d < n$. The Discrete Fourier Transform (DFT) mapping $DFT_w : R^n \to R^n$ denotes the evaluation of the polynomial f at the powers of w, i.e., $DFT_w(f) = \langle f(1), f(w), f(w^2), \ldots, f(w^{n-1})\rangle$.

The Discrete Fourier Transform can be seen as a special case of multipoint evaluation, at the powers of nth 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 [3], and is one of the most important algorithms in practice. Let $n \in \mathbb{N}$ be even, $w \in R$ a primitive nth root of unity, and $f \in R[x]$ of degree less than n. To evaluate f at the powers $1, w, w^2, \ldots, w^{n-1}$, 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:

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

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

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 ot 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
$\mathsf{FFT}_{w,n}(f)$

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 $w, w^2, \ldots, w^{n-1}$ of a primitive nth root of unity $w \in R$.

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

```
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 $\mathsf{FFT}_{w^2,n/2}(r_0)$ and $\mathsf{FFT}_{w^2,n/2}(r_1^*)$ to compute $r_0, r_1^*$ at the powers of $w^2$.
4. Return $r_0(1), r_1^*(1), r_0(w^2), r_1^*(w^2), \ldots, r_0(w^{n-2}), r_1^*(w^{n-2})$.
```

Efficiency. The above algorithm computes $\mathrm{DFT}_{w,n}(f)$ 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

$\langle f(1), f(w), f(w^2), \ldots, f(w^{n-1})\rangle$, our goal is to compute the coefficients of f. Let V_w be the Vandermonde matrix:

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

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 A.4 [[10]] Let R be a ring (commutative, with 1), $n \in \mathbb{N}$, and $w \in R$ be a primitive nth root of unity. Then w^{-1} is a primitive nth root of unity and $(V_w)^{-1} = 1/nV_{w^{-1}}$.

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

Interpolation on powers of nth Root of Unity
$\mathsf{Interpol}_{w,n}(\vec{V})$

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

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

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

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
$\mathsf{PolyMult}(f, g)$

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

```
1. Compute $\langle f(1), \ldots, f(w^{n-1}) \rangle \leftarrow \mathsf{FFT}_{w,n}(f)$.
2. Compute $\langle g(1), \ldots, g(w^{n-1}) \rangle \leftarrow \mathsf{FFT}_{w,n}(g)$.
3. For $0 \le i < n$ compute $S(i) = f(i)g(i)$.
4. Let $\vec{V_s} = \langle S(1), \ldots, S(n-1) \rangle$.
5. Computes $S \leftarrow \mathsf{Interpol}_{w,n}(\vec{V_s})$.
```

A.5 Working over More General Rings

In the above algorithm for polynomial multiplication, we assumed that the underlying ring R contains certain primitive roots of unity. The described algorithm would not work directly if we instead work over an arbitrary ring. However, there are computer algebra techniques that extend the above algorithm to work over more general rings. This generalization is important for us when discussing the encrypted variants of these algorithms since each of the existing homomorphic encryption schemes work over a different plaintext domain (ring).

We give a high level description of the techniques here, but refer the reader to [Chapter 8,] for a detailed discussion.

Let R be a ring such that 2 is a unit in R,[4] $n = 2^k$ for some $k \in \mathbb{N}$, and $D = R[x]/\langle x^n + 1 \rangle$. Note that D is generally not a field. Then we have that

$$x^n \equiv -1 \bmod (x^n + 1), x^{2n} = (x^n)^2 \equiv 1 \bmod (x^n + 1)$$

This implies that $w = x \bmod (x^n + 1) \in D$ is a $2n$th root of unity. Moreover, $w^n - 1 - 1 = -2$ is a unit in R and hence w is in fact a primitive $2n$th root of unity.

To multiply two polynomials $f, g \in R[x]$ with $deg(fg) < n = 2^k$, it is sufficient to compute $fg \bmod (x^n + 1)$. This is called the *negative wrapped convolution of f and g*. Let $m = 2^{\lfloor k/2 \rfloor}$, $t = n/m = 2^{\lceil k/2 \rceil}$, and partition the coefficients of f and g into t blocks of size m:

$$f = \sum_{0 \le j \le t} f_j x^{mj}, g = \sum_{0 \le j \le t} g_j x^{mj}$$

with $f_j, g_j \in R[x]$ of degree less than m for $0 \le j \le t$. With $f' = \sum_{0 \le j \le t} f_j y^j, g' = \sum_{0 \le j < t} \in R[x, y]$ we have that $f = f'(x, x^m)$ and $g = g'(x, x^m)$. In order to compute fg it turns out that it is sufficient to compute $f'g' \bmod y^t$. The latter can be seen as polynomial multiplication where the coefficients of the polynomials are in $D[y]$ where $D = R[x]/x^{2m} + 1$. The advantage of this is that as we discussed earlier D contains the primitive roots of unity we need, and hence we can use PolyMult algorithm of the previous section for this purpose. This leads to $O(t \log t)$ operations in D (since f and g are of degree t in y). Each multiplication operation in D can itself be computed using the same algorithm in a recursive manner.

A careful analysis of the complexity of the recursive algorithm shows that the polynomial multiplication uses $9/2 n \log n \log \log n + O(n \log n)$ operations in R.

[4]This is the only restriction on the ring and is satisfied by all existing additively homomorphic encryption schemes.

B Security Definitions

A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-pair is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x, y)$ and the second party (with input y) wishes to obtain $f_2(x, y)$.

The security of a protocol is analyzed by comparing what an adversary can do in the protocol to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an ideal computation involving an incorruptible trusted third party to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the ideal computation.

Execution in the ideal model. An ideal execution proceeds as follows:
Inputs: Each party obtains an input, denoted w ($w = x$ for Alice, and $w = y$ for Bob).
Send inputs to trusted party: An honest party always sends w to the trusted party. A malicious party may, depending on w, either abort or send some $w' \in \{0,1\}^{|w|}$ to the trusted party.
Trusted party answers first party: In case it has obtained an input pair (x, y), the trusted party first replies to the first party with $f_1(x, y)$. Otherwise (i.e., in case it receives only one valid input), the trusted party replies to both parties with a special symbol $\perp$.
Trusted party answers second party: In case the first party is malicious it may, depending on its input and the trusted partys answer, decide to stop the trusted party by sending it $\perp$ after receiving its output. In this case the trusted party sends $\perp$ to the second party. Otherwise (i.e., if not stopped), the trusted party sends $f_2(x, y)$ to Bob.
Outputs: An honest party always outputs the message it has obtained from the trusted party. A malicious party may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the message obtained from the trusted party.

Execution in the real model: Next consider the real model in which a real (two-party) protocol is executed (and there exists no trusted third party). In this case, a semihonest party follows the steps of the real protocol but may output an arbitrary output. A malicious party, on the other hand, may follow an arbitrary feasible strategy; that is, any strategy implementable by non-uniform probabilistic polynomial-time machines. In particular, the malicious party may abort the execution at any point in time (and when this happens prematurely, the other party is left with no output).

Let f be as above and let π be a two-party protocol for computing f. Furthermore, let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic polynomial-time machines (representing parties in the real model). Such a pair is admissible if for at least one $i \in \{1, 2\}$ we have that M_i is honest (i.e., follows the strategy specified by π). Then, the joint execution of π under $\overline{M}$ in the real model (on input pair (x, y)), denoted $REAL_{\pi,\overline{M}}(x, y)$, is defined as the output pair of M_1 and M_2 resulting from the protocol interaction. The joint execution of f under $\overline{M}$ in the ideal model (on input pair (x, y)), denoted $IDEAL_{f,\overline{M}}(x, y)$, is defined as the output pair of M_1 and M_2 from the above ideal execution.

Definition B.1 [secure two-party computation]: Let f and π be as above. Protocol π is said to securely compute f (in the malicious model) if for every pair of admissible non-uniform probabilistic polynomial-time machines $\overline{A} = (A_1, A_2)$ for the real model, there exists a pair of admissible nonuniform probabilistic expected polynomial-time machines $\overline{B} = (B_1, B_2)$ for the ideal model, such that

$$\{IDEAL_{f,\overline{B}}(x,y)\}_{x,ys.t.|x|=|y|} \overset{c}{\equiv} \{REAL_{\pi,\overline{A}}(x,y)\}_{x,y\ s.t.\ |x|=|y|}$$

Namely, the two distributions are computationally indistinguishable