# Efficient Polynomial Operations in the Shared-Coefficients Setting

Payman Mohassel and Matthew Franklin

Department of Computer Science, University of California, Davis CA 95616
mohassel@cs.ucdavis.edu, franklin@cs.ucdavis.edu

**Abstract.** We study the design of efficient and private protocols for polynomial operations in the shared-coefficients setting. We propose efficient protocols for *polynomial multiplication*, *division with remainder*, *polynomial interpolation*, *polynomial gcd*, and a few other operations. All the protocols introduced in this paper are *constant-round*, and more efficient than the *general* MPC. The protocols are all composable, and can be combined to perform more complicated functionalities. We focus on using a *threshold additively homomorphic public key scheme* due to the applications of our protocols. But, our protocols can also be securely computed in the *information-theoretic* setting. Finally, we mention some applications of our protocols to *privacy-preserving set-operations*.

**Keywords**: secure multi-party computation, passive adversary, polynomial operations, threshold homomorphic encryption, privacy-preserving set operations.

## 1 Introduction

Secure multiparty computation (MPC) is an important and classic problem in the realm of *cryptography* and *distributed computing*. In this problem, a group of parties want to compute a function of their inputs while keeping their inputs private. The special case of two-party computation was first studied by Yao [Yao82,Yao86]. Classic works such as [GMW87], [BGW88], and [CCD88] give solutions for the more general case of multiparty computation. These works solve the problem of *general* multiparty computation by performing gate by gate secure computation of a *circuit* (boolean or arithmetic) that implements the desired function.

As the function being computed becomes more complicated, so does the circuit that computes such a function. This, in turn, makes the *general* MPC solutions inefficient. Therefore, researchers have turned to designing special-purpose protocols for specific functions in order to improve on the complexity of general MPC solutions.

Polynomials have turned out to be useful tools for designing efficient and secure distributed protocols for specific functionalities. [FNP04], [FIPR05], and [KS05] use polynomials to design efficient multiparty protocols. In these papers, coefficients of polynomials are encrypted using an additively homomorphic cryptosystem. Then, different operations on polynomials such as *polynomial evaluation*, and *polynomial multiplication* are performed.

These works motivated us to take a closer look at different operations on polynomials. Furthermore, polynomials have appeared (and will continue to appear) in many schemes, or algorithms. Some of these schemes might have privacy concerns, and require a set of parties to securely perform some operations on the polynomials. Anywhere that such operations on polynomials are being performed and privacy is a concern, our protocols can be useful.

### 1.1 Our Contribution

We propose efficient protocols for *polynomial multiplication*, *division with remainder*, *polynomial interpolation*, *polynomial gcd*, and other polynomial operations. These protocols are composable and can be combined to perform more complicated functionalities. In this paper, we are concerned with both *communication* and *round* complexity of our protocols. Particularly, all protocols introduced in this paper have constant number of rounds, while we try to optimize their communication complexity wherever possible.

We propose several applications of our methods to *privacy-preserving set-operations*. In this setting, parties hold sets of data, and want to perform joint operations on their sets. These operations could be union, intersection, subtraction, or any other operation on sets. In many cases, sets are represented by polynomials. Different operations on polynomials lead to different set-operations. The coefficients of such polynomials are often encrypted using a *threshold additively-homomorphic public-key cryptosystem*. That is why we also use such a cryptosystem to distribute secrets among the parties and to operate on those secrets.

But, we would like to note that our protocols can be implemented using tools other than a threshold additively-homomorphic cryptosystem. For example, any *threshold linear secret-sharing scheme* can be used to replace such a cryptosystem.

For simplicity, and with slight abuse of terminology, we will use the single term *shared* throughout this paper to refer to both distribution methods: threshold additively-homomorphic encryption, and threshold linear secret-sharing. See Section 2.1 for more detail.

Our protocols are secure against a semi-honest (passive) adversary. Such an adversary will follow the steps of protocol but will try to learn extra information from the messages it receives during every round of the protocol. The security of our protocols are guaranteed as along as the underlying protocols (multiplication, addition, sharing random values, ...) that are discussed in section 2, can be performed securely. In the *computational setting*, we require that the threshold homomorphic cryptosystem used be *semantically* secure. This leads to secure implementation of the multiplication protocol described in section 2.1. The threshold version of Paillier's cryptosystem is semantically secure and is suited for this purpose. Similarly, in the *information-theoretic setting*, [BGW88] and [CCD88] provide such secure protocols.

### 1.2 Organization

In section 2, we describe all the tools we need for our protocols. In section 3, we describe an efficient protocol for multiplying two shared polynomials. A protocol for Division with remainder is described in section 4. Multiplying many shared polynomials in constant-round is explained in section 5. An efficient protocol for polynomial interpolation is give in section 6. In section 7, we design constant-round protocols for gcd of two and many shared polynomials. We also describe a protocol for determining if two shared polynomials are coprime or not.

## 2 Preliminaries

Throughout this paper, $F$ is a finite field of size $q$. $F[x]$ is the ring of polynomials over $F$. $F[x]/f$ is the extension field where $f$ is an irreducible polynomial.

In this paper, by a *shared* value, we mean a value that has been distributed among the parties using either (1) a threshold additively-homomorphic public key encryption scheme or (2) a threshold linear secret sharing scheme. The techniques described in this paper will not depend on which method is used. See Section 2.1 for more details.

Throughout this paper, by a *shared polynomial*, we mean a polynomial whose coefficients have been individually encrypted or shared using one of the methods mentioned in the preceding paragraph. A shared polynomial leaks an upper bound on the degree of the polynomial. For some of our protocols, the exact degrees of some shared polynomials may be leaked.

In this paper, by a *shared matrix*, we mean a matrix whose elements have been individually encrypted or shared using one of the methods described above.

We measure the complexity of our protocols by the number of multiplications of shared values that are necessary. This is a natural measure due to the fact that addition of shared values (for either of the two aforementioned methods) can be performed non-interactively. When the context is clear, we may refer to this complexity measure as simply *multiplications*.

### 2.1 Shared Values

In this paper, we assume that we either have a threshold additively-homomorphic public key encryption scheme or a threshold linear secret-sharing scheme. As noted above, we may use the term *shared value* to refer to a value that has been distributed according to either method.

**Threshold Additively-Homomorphic Encryption** We use a threshold cryptosystem with homomorphic properties such as Paillier's cryptosystem [Pai00]. Paillier's cryptosystem is additively homomorphic, and supports threshold decryption [FP00]. Such a cryptosystem has the following four properties:

1. To share a value between the parties, a party can encrypt the value using the public key to the cryptosystem, and broadcast the ciphertext.

2. Parties can jointly reveal an encrypted value using the threshold decryption.
3. Given the ciphertexts, $E_{pk}(a)$, and $E_{pk}(b)$, and a public plaintext $c$, parties can compute $E_{pk}(a+b)$, and $E_{pk}(ca)$ non-interactively.
4. Given the ciphertexts $E_{pk}(a)$ and $E_{pk}(b)$ parties can securely compute $E_{pk}(ab)$ in constant number of rounds.

The first three properties are automatically satisfied by any threshold additively homomorphic cryptosystem. We give a simple constant-round protocol to satisfy the fourth property here[1]. Let's assume that parties are holding the ciphertexts $E_{pk}(a)$, and $E_{pk}(b)$. They want to compute $E_{pk}(ab)$. The protocol follows:

1. Party $i$ broadcasts $E_{pk}(r_i)$ to all the parties, where $r_i$ is a randomly chosen plaintext.
2. Parties compute $E_{pk}(a+\sum_i r_i)$, and decrypt the result to get $a' = a+\sum_i r_i$.
3. Each party computes $a'E_{pk}(b) = E_{pk}(a'b)$.
4. Party $i$ computes $r_i E_{pk}(b) = E_{pk}(r_i b)$, and broadcasts it to other parties.
5. Parties compute $E_{pk}(a'b) - \sum_i E_{pk}(r_i b) = E_{pk}(ab)$.

One minor issue is that the domain of Paillier's cryptosystem is the ring $Z_n$, where $n$ is the product of two large and secret primes. Note that $Z_n$ has all of the properties of a finite field except that some of the non-zero elements in $Z_n$ are not invertible. However, an extended gcd algorithm on $x$ and $n$ either finds the inverse of $x \bmod n$, or finds a non-trivial factor of $n$. So in practice we can describe computations in $Z_n$ as if it were a finite field.

**Threshold Linear Secret Sharing** We require a threshold linear secret-sharing scheme over a field with the following properties:

1. Parties can share a value in constant number of rounds.
2. Parties can reveal their shares in a constant number of rounds.
3. Given shares of values $a$ and $b$, and a publicly known value $c$, parties can compute shares of $(a+b)$, and $ca$ without any interaction.
4. Given shares of values $a$, and $b$, parties can compute shares of $ab$ in constant number of rounds.

The secret-sharing scheme can be unconditionally or computationally secure. Shamir's polynomial-based threshold linear secret sharing scheme [Sha79] is unconditionally secure. One example of a secret-sharing scheme for the computational setting is given in [GRR98].

### 2.2 Existing Constant-Round Protocols

In this section, we review some of the existing protocols with constant number of rounds. We would like to remind the reader that the term *shared value* (and

[1] see [CDN01] for a similar protocol.

*shared polynomial* and *shared matrix*) is used throughout the remainder of this paper regardless of whether the underlying distribution method is by threshold additively-homomorphic encryption or by threshold linear secret sharing.

Shares of a polynomial $P$ in $F[x]$, are simply the collection of shares of all of $P$'s coefficients. In a similar way, shares of a matrix $M$ over $F$ are the collection of shares of all the elements in $M$.

We will use or refer to the following techniques throughout the paper. Most of these protocols have appeared in [BB89], and [CD01].

**Sharing a Secret Random Field Element, Polynomial, or Matrix** A protocol in which parties generate shares of a random and secret value $r \in F$. This can be done by letting each party share a random value between the parties. Then, parties take the sum of all of those values as $r$. We can extend this protocol to share random polynomials in $F[x]$ or $F[x]/f$. In a similar way, we can also share random matrices over $F$.

**Constant-Round Multiplication and Division** Consider the polynomials $P_A$ and $P_B$ in $F[x]$. If $P_A$, and $P_B$ are both secret and shared among the parties, then the parties can use the basic polynomial multiplication to compute shares of $P_A P_B$ in constant-round. We will describe a more communication-efficient protocol for this task in Section 3. If at least one of the polynomials is publicly known, the multiplication does not need any interaction. The case of matrices is similar.

If $P_A$ is shared, and $P_B$ is publicly known, parties can compute shares of $Q$ and $R$ such that $P_A = QP_B + R$ and $deg(R) \leq P_B$. This can be done using the synthetic division, and does not require any interaction between the parties. In Section 4, we describe an efficient division protocol for the case where both $P_A$ and $P_B$ are secret and shared.

**Sharing Secret Invertible Field Elements and Matrices** This is a protocol that generates a sharing of a secret, random non-zero field element, or an invertible matrix. The protocol securely generates two random elements (matrices), securely multiplies them, and reveals the result. If this is non-zero (invertible), one of the secret elements(matrices) is taken as the desired output of the protocol. The probability that a random $n$ by $n$ matrix is invertible is greater than $\frac{1}{4}$, and at least $1 - \frac{n}{q}$.

**Constant-Round Inversion of Matrices and Field Elements** In this protocol, given shares of a field element (matrix) $X$, parties compute shares of the inverse of that element (matrix). As it is described in [BB89], parties first generate shares of a random non-zero (invertible) field element (matrix) $R$. Then, they compute shares of $RX$, and reveal the result. Parties compute $(RX)^{-1} = X^{-1}R^{-1}$ non-interactively. Finally, they compute $X^{-1}R^{-1} * R = X^{-1}$ non-interactively.

**Unbounded Fan-In Multiplication in Constant-Round** Given shares of polynomially many field elements (matrices) $X_1, \ldots, X_l$, parties want to compute shares of their product. Parties generate shared random non-zero (invertible) field elements (matrices) $R_1, \ldots, R_l$. They compute $P_1 = X_1 R_1$, and $P_i = R_{i-1}^{-1} X_i R_i$ for $i \geq 2$. They publicly announce all the $P_i$ values, compute $\prod_i P_i$, and multiply the result by $R_l$, all non-interactively. This gives them shares of $\prod_i X_i$.

**Linear Algebra in Constant-Round** consider the following protocols: (1) Given shares of a matrix $A$, parties want to compute shares of $det(A)$ (2) Given shares of a matrix $A$, and shares of vector $b$, parties want to compute shares of solution(s) to the linear system $Ax = b$. [CD01] proposes efficient and constant-round protocols for these two problems, and others. These protocols are more elaborate, and we will not describe them here.

### 2.3 Privacy-Preserving Set-Operations

[KS05] uses polynomials to represent sets of data. The polynomial representation of a set $S$ of elements in $F$ is the polynomial $P = \prod_i (x - s_i)$, where $s_i \in S$. Then, different operations on polynomials lead to different operations on the underlying set. For instance, to compute the union of two disjoint sets, one can simply multiply the two polynomials.

By performing different operations on the polynomials, [KS05] designs privacy preserving operations on sets. These operations include, *set-intersection, set-union, element reduction*, and others. For these protocols to be private, coefficients of polynomials can be shared using either a threshold additively-homomorphic encryption scheme or a threshold linear secret-sharing scheme. In fact, the results in [KS05] are presented using only threshold additively-homomorphic encryption, but it is easy to translate their results to the threshold linear secret-sharing setting. For consistency we will use the single term *shared* (as discussed in Section 2.1) throughout this section to describe their results.

Their protocols gain efficiency compared to general multiparty solutions, due to the fact that the following operations can be performed without any interaction:

1. If a party knows the polynomial $P_A$, and is given the shared polynomial $P_B$, then he can compute shared polynomial $P_A P_B$ without any interaction.
2. A party can compute the derivative of a shared polynomial $P$ without any interaction with other parties.

In case of polynomial multiplication, if both polynomials are shared, the most efficient solution is the general multiparty computation. An appropriate general MPC for this case is [CDN01]. For instance, the classic polynomial multiplication algorithm (polynomials of degree $O(n)$), gives us a circuit with $O(n^2)$ multiplication gates over a ring, each of which requires interaction between the parties. The most efficient polynomial multiplication algorithm has a circuit with $O(npolylog(n))$ gates.

Another interesting operation on polynomials that has *not* been considered in the privacy-preserving setting, is the division with remainder of polynomials. Such a protocol leads to operations for set deletion/subtraction (when the element(s) is known to be in the set).

One can verify that if a party $A$ knows polynomial $P_A$ and is given the shared polynomial $P_B$, he can perform the synthetic division algorithm to compute the shared polynomials $Q$ and $R$ without any interaction with other parties, where $P_B = QP_A + R$, and $deg(R) \leq P_A$. But, if both polynomials are shared, the most efficient protocols are the general MPC protocols.

In later sections, we will propose protocols for these, and other tasks. Our protocols will be constant-round , and more efficient than the general MPC.

## 3 Multiplying Two Polynomials

Consider two shared polynomials $f(x), g(x) \in F[x]$, where $F$ is a finite field, and $deg(f) = deg(g) = n$. Parties want to compute shares of the polynomial $h(x) = f(x) * g(x)$. The following is a simple and efficient constant-round protocol for computing the shared product polynomial, with communication complexity of $O(n)$ multiplications. To the best of our knowledge, this protocol has not been published previously.

1. Each party computes his/her share of $f(i)$ and $g(i)$ for all $0 \leq i \leq 2n$.
2. Parties engage in $2n$ multiplications to get their shares of $h(i) = f(i) * g(i)$.
3. Each party can perform the Lagrange Interpolation on its own to get his/her share of coefficients of $h(x)$.

In step 1, no interaction is necessary. All the $i$'s are public, and therefore, parties are computing a linear function of shared coefficients. In step 2, parties perform $2n$ multiplications of shared elements in the field. Step 3 is also performed without any interaction between the parties. This leads to the communication complexity of $O(n)$ multiplications. This also provides an efficient privacy-preserving set union protocol for the setting of [KS05].

## 4 Division with Remainder

Let $f(x)$ and $g(x)$ be two polynomials in $F[x]$ where $deg(f) = n$, $deg(g) = m$, and $m \leq n$. Given shares of $f$ and $g$, parties want to compute shares of $q$ and $r$ such that $f(x) = g(x)q(x) + r(x)$, and $deg(r) \leq deg(g)$. The synthetic polynomial division is sequential and does not directly lead to a constant-round protocol for polynomial division. Furthermore, it requires $O(n^2)$ multiplication of elements of a field. The fastest division algorithm still requires $O(npolylog(n))$ such multiplications.

Next, we give a constant-round protocol for the division with remainder of two shared polynomials. The communication complexity of our protocol is $O(n)$ multiplications. The idea is borrowed from a division algorithm using Newton's iteration (See chapter 9 of [GG03] for more information).

Consider the equation

$$f(x) = g(x)q(x) + r(x) \tag{1}$$

By substituting $\frac{1}{x}$ for the variable $x$ in the polynomials, we have the following equation:

$$x^n f(1/x) = [x^{n-m} q(1/x)] * [x^m g(1/x)] + (x^{n-m+1}) * [x^{m-1} r(1/x)] \tag{2}$$

Let $rev_k(a) = x^k a(1/x)$ for an arbitrary polynomial $a(x)$ . When $deg(a) = k$, this operation simply reverses the order of coefficients of $a(x)$. We can rewrite the above equation as:

$$\begin{aligned} rev_n(f) &= rev_{n-m}(q) * rev_m(b) + x^{n-m+1} * rev_{m-1}(r) \Rightarrow \\ rev_n(f) &= \qquad rev_{n-m}(q) * rev_m(g) \ mod \ x^{n-m+1} \qquad \Rightarrow \\ rev_{n-m}(q) &= \qquad rev_n(f) * rev_m(g)^{-1} \ mod \ x^{n-m+1} \end{aligned} \tag{3}$$

If parties can compute shares of $rev_{n-m}(q)$ efficiently and in constant-round, they can also compute shares of $q(x)$, and $r(x) = f(x) - g(x)q(x)$ efficiently. Note that we already have an efficient protocol for multiplying two polynomials from section 3. The division algorithm follows:

1. Parties compute shares of $rev_n(f)$ and $rev_m(g)$ without any interaction.
2. Now, parties need to compute shares of $rev_m(g)^{-1} \mod x^{n-m+1}$. We will give an efficient sub protocol for this operation later in this section.
3. They can use the protocol for multiplying two polynomials to compute shares of $rev_v(f) * rev_m(g)^{-1}$.
4. Parties reduce the result mod $x^{n-m+1}$ and reverse the coefficients to get $q(x)$ shared without any interaction.
5. By performing another polynomial multiplication, parties compute shares of $g(x)q(x)$.
6. Parties compute shares of $r(x) = f(x) - g(x)q(x)$ without any interaction.

Note that division with remainder is reduced to two polynomial multiplications (step 3, 5) and one polynomial inversion (step 2). We already know how to do the polynomial multiplication in an efficient way. It suffices to give an efficient protocol for inverting an invertible polynomial $f$ mod $x^t$, where $t$ is known to all parties. It is important to note that $rev_m(g)$ is invertible mod $x^t$ for any $t \geq 1$. To see that, note that $deg(g) = m$. This means that the leftmost coefficient of $g(x)$ is non-zero. Therefore, the free coefficient (constant term) of $rev_m(g)$ is also non-zero. This implies that $rev_m(g)$ is not divisible by $x$, and is invertible mod $x^t$ for any $t \geq 1$. This observation implies that for our division algorithm to work properly, we need to guarantee that the given (maximum) degrees for the polynomials are in fact the exact degrees of those polynomials.

Consider the ring $F[x]/x^t$. Consider the multiplicative subgroup $(F[x]/x^t)^*$ that contains all the invertible polynomials in $F[x]/x^t$. To invert a polynomial $f \in (F[x]/x^t)^*$, we adapt the matrix inversion technique of [BB89].

1. Parties compute shares of a uniformly random polynomial $s$ in $F[x]/x^t$.
2. Parties compute shares of $f(x) * s(x)\ mod\ x^t$, and publicly announce their shares.
3. Parties compute their shares of $g(x) = (f(x) * s(x))^{-1}\ mod\ x^t = s(x)^{-1} * f(x)^{-1}\ mod\ x^t$. We expect $s$ to be invertible in $F[x]/x^t$, because $gcd(s, x^t) = 1$ with high probability $(1 - 1/q)$.
4. Parties compute their shares of $f(x)^{-1}\ mod\ x^t = s(x) * g(x)\ mod\ x^t$.

Step 2 requires a polynomial multiplication. Steps 3 and 4 do not require any interaction between the parties. Therefore, the communication complexity of this inversion is $O(n)$ multiplications.

Our division with remainder protocol also provides an efficient privacy preserving subset deletion protocol for the setting of [KS05].

# 5 Multiplying Many Polynomials

Let $f_1, f_2, \ldots, f_l$ be polynomials in $F[x]$, where $deg(f_i) = n_i$. Given shares of $f_i$ for all $1 \leq i \leq l$, parties want to compute shares of $h(x) = \prod_{i=1}^{l} f_i(x)$. In this section, we will give an efficient constant-round protocol for this task.

We would like to use the technique of [BB89] for unbounded fan-in multiplication of elements in a field. But, note that F[x] is not a field. An appropriate field that contains all the $f_i$'s and their product $h(x)$ is the extension field $F[x]/f$ where $f$ is an irreducible polynomial of degree $n_1 + n_2 + \ldots + n_l + 1$. To multiply two polynomials in $F[x]/f$, parties can multiply the polynomials using the protocol in section 3, and compute the result mod $f$. Since $f$ is a publicly known polynomial, the second step doesn't require any interaction. The protocol for multiplying many polynomials follows:

1. One party computes an irreducible polynomial $f$ of degree $n_1 + \ldots + n_l + 1$, and announces it to other parties.
2. Parties share $l$ random polynomials $r_1, \ldots, r_l$ in the field $F[x]/f$.
3. Parties compute shares of $r_i(x)^{-1}$ for all $i \in \{0..l\}$.
4. Parties compute and publicly announce $(f_1(x)r_1(x)\ mod\ f)$, $(r_1(x)^{-1}f_2(x)r_2(x)\ mod\ f)$ , $(r_2(x)^{-1}f_3(x)r_3(x)\ mod\ f)$, $\ldots$, $(r_{l-1}(x)^{-1}f_l(x)r_l(x)\ mod\ f)$.
5. Parties compute the product of the $l$ public polynomials, and multiply the result by $r_l(x)^{-1}$.
6. Parties reduce the result mod $f$ to obtain shares of $h(x)$.

The above protocol is constant-round. The communication complexity is dominated by step 4, which requires $O(l(\sum_{i=1}^{l} n_i))$ multiplications.

## 6 Polynomial Interpolation

Let $x_i$ and $y_i$ be shared elements of the field $F$ for $i \in \{1..n\}$. Parties would like to compute shares of the polynomial $f \in F[x]$ such that $f(x_i) = y_i$ for all $i \in \{1..n\}$.

A simple and constant-round protocol for this problem is possible based on the existing techniques. Consider the Vandermonde matrix:

$$V = \begin{bmatrix} 1 & x_1 & x_1^2 & ... & x_1^n \\ 1 & x_2 & x_2^2 & ... & x_2^n \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & x_n & x_n^2 & ... & x_n^n \end{bmatrix} \tag{4}$$

Parties can compute the matrix $V$, and solve the linear system $VX = [y_1, y_2, ..., y_n]^T$. The components of the solution to the linear system are the coefficients of $f$. If $x_i$'s are distinct, the system will be non-singular, and solving the linear system is reduced to matrix inversion. [BB89] gives an efficient and constant-round algorithm for inverting matrices. The communication complexity of this method will be $O(n^3)$ multiplications.

Using the protocol from section 5 for multiplying many polynomials, we can improve on that, and achieve a constant-round protocol with communication complexity of $O(n^2)$ multiplications. The protocol follows:

1. Parties compute shares of the polynomial $P = (x - x_1)(x - x_2)...(x - x_n)$.
2. Parties compute shares of $n$ polynomials $P_i = P/(x - x_i)$ for all $i \in \{1..n\}$.
3. Parties compute shares of $P_i(x_i)$ and $P_i(x_i)^{-1}$ for all $i \in \{1..n\}$.
4. Parties compute shares of the Lagrange coefficients, $\gamma_i = P_i(x) * P_i(x_i)^{-1}$ for all $i \in \{1..n\}$.
5. Parties compute shares of $f = \sum_{i=1}^{n} \gamma_i * y_i$.

The above protocol is constant round. For step 1, we will use the protocol for multiplying many polynomials which requires $O(n^2)$ multiplications. Step 2 requires $n$ runs of polynomial division protocol, and also requires $O(n^2)$ multiplications. Step 3 consists of $n$ polynomial evaluation (or equivalently, $n$ runs of unbounded fan-in multiplication of $n$ field elements), and $n$ inversion of field elements. Therefore, it requires a total of $O(n^2)$ multiplications. It is easy to see that steps 4 and 5 also require $O(n^2)$ multiplications. Therefore, the above protocol has a total communication complexity of $O(n^2)$ multiplications.

## 7 Computing GCD of Polynomials

Given shares of $f(x)$, and $g(x)$ in $F[x]$, where $deg(f) = n$, $deg(g) = m$, and $n \geq m$, parties want to compute shares of $d(x) = gcd(f, g)$. Here, we assume that we leak the degree of the gcd (so that everyone learns number of coefficients of the gcd).

Euclid's Algorithm for computing the gcd of two polynomials is sequential in nature. Particularly, it requires the parties to perform $O(n)$ division protocols in a sequential manner. Furthermore, we cannot use the division algorithm we introduced in Section 4, to improve the communication complexity of a Euclid-based protocol. The reason is that, as we mentioned earlier, the division protocol assumes that the polynomials have non-zero coefficients for their highest degree. This might not be true during all the steps of Euclid's Algorithm. Therefore, we take a different approach to solving the polynomial gcd problem efficiently and in constant number of rounds.

### 7.1 Extended Euclidean Algorithm and Subresultants

Consider the Extended Euclidean Algorithm (EEA) below:

$$
\begin{array}{rll}
r_0 = f & s_0 = 1 & t_0 = 0 \\
r_1 = g & s_1 = 0 & t_1 = 1 \\
r_2 = r_0 - r_1 * q_1 & s_2 = s_0 - s_1 * q_1 & t_2 = t_0 - t_1 * q_1 \\
. & . & . \\
. & . & . \\
. & . & . \\
r_{i+1} = r_{i-1} - r_i * q_i & s_{i+1} = s_{i-1} - s_i * q_i & t_{i+1} = t_{i-1} - t_i * q_i \\
. & . & . \\
. & . & . \\
. & . & . \\
0 = r_{l-1} - q_l * r_l & s_{l+1} = s_{l-1} - q_l * r_l & t_{l+1} = t_{l-1} - q_l * r_l
\end{array} \tag{5}
$$

One invariant of the EEA is that $r_i = s_i * f + t_i * g$ for all $i \in \{0..l\}$. Note that $d(x) = r_l(x)$, and $s_l(x)$, and $t_l(x)$ are the *Bezout coefficients*. We denote the degree sequence of EEA by $(n_0, n_1, n_2, \ldots, n_l)$, where $n_i = deg(r_i)$ for all $i \in \{0..l\}$.

Next, we review some of the properties of subresultants of two polynomials. These properties will help us design a constant-round protocol for polynomial gcd.

**Some Properties of the Subresultants** Consider the polynomials $f(x) = \sum_{j=0}^{n} a_j x^j$ and $g(x) = \sum_{j=0}^{m} b_j x^j$. The subresultant matrix $S_i$ (for $i \in \{0..m\}$) is a $(n+m-2i) * (n+m-2i)$ matrix of the form:

$$S_i = \begin{bmatrix} a_n & & & & b_m & & & & \\ a_{n-1} & a_n & & & b_{m-1} & b_m & & & \\ \vdots & & \ddots & & \vdots & & \ddots & & \\ a_{n-m+i+1} & \cdots & \cdots & a_n & b_{i+1} & \cdots & \cdots & b_m & \\ \vdots & & & \vdots & \vdots & & & & \ddots \\ a_{i+1} & \cdots & \cdots & a_m & b_{m-n+i+1} & \cdots & \cdots & \cdots & \cdots \; b_m \\ \vdots & & & \vdots & \vdots & & & & \vdots \\ \vdots & & & \vdots & \vdots & & & & \vdots \\ a_{2i-m+1} & \cdots & \cdots & a_i & b_{2i-n+1} & \cdots & \cdots & \cdots & \cdots \; b_i \end{bmatrix} \tag{6}$$

All the entries in the $S_i$'s are coefficients of $f$ and $g$. The following two theorems about the subresultant matrices will be useful (please refer to [GG03] for proof details).

**Theorem 1.** *Integer $k$ appears in the degree-sequence of EEA, iff $det(S_k) \neq 0$.*

**Theorem 2.** *If $k = n_i$, where $n_i$ is the ith element in the degree sequence of EEA, the linear system $S_k * x = [0, ..., 0, 1]^T$ has a unique solution $x$ such that $s_i = x[1...(m-k)]$, and $t_i = x[(m-k+1)...(m+n-2k)]$.*

### 7.2 Shared GCD of Two Polynomials

Here is the intuition behind our protocol for computing shares of the gcd of two shared polynomials. First, parties compute $p = deg(d(x)) = deg(r_l(x))$ without learning anything else. Note that based on Theorem 1, $p$ is the only element in the degree sequence with the property that:

$$det(S_p) \neq 0, \; and \; det(S_i) = 0 \; for \; all \; 0 \leq i < p. \tag{7}$$

Parties then jointly solve a linear system (Theorem 2) to compute shares of the *Bezout* coefficients $s_l(x)$ and $t_l(x)$ from which shares of the gcd can be derived. The polynomial gcd protocol follows:

#### The Protocol

1. Parties compute shares of $det(S_i)$ for all $i \in \{0..m\}$.
2. Parties compute shares of non-zero random field elements $h_1, ..., h_m$.
3. Parties compute shares of $b_0, ..., b_m$ such that $b_k = \sum_{i=0}^{k} h_i * det(S_i)$ for all $k \in \{0..m\}$. (Note: $b_i = 0$ for all $0 \leq i < p$. $b_i \neq 0$ for all $p \leq i \leq m$, with high probability).
4. Parties generate shares of non-zero random elements $r_1, ..., r_m$ of the field.
5. Parties compute and announce shares of values $c_i = r_i * b_i$.
6. Note that $c_i = 0$ for all $0 \leq i < p$, and $c_i \neq 0$ for all $p \leq i \leq m$ . By counting the non-zero $c_i$'s, parties learn $p = deg(d(x))$ (Nothing else is learned, since all the non-zero $c_i$'s are random).

7. Parties compute shares of the solution to the linear system $S_p * X = [0, ..., 1]^T$, and extract shares of the $s_l(x)$ and $t_l(x)$ from the unique shared solution (based on Theorem 2).
8. Parties compute shares of $d = f * s_l + g * t_l$. (Parties only consider the first $p+1$ coefficients of the result).

All the steps of the protocol can be performed in constant number of rounds. [CD01] introduces an efficient, and constant-round algorithm for computing determinant of a shared matrix (step 1). Since $S_p$ is always invertible, the linear system of step 7 can be solved using the matrix inversion protocol of [BB89].

The communication complexity of the protocol is dominated by step 1, in which determinants of $O(n)$ matrices are computed, and each matrix is $O(n)$ by $O(n)$. In the computational setting, our protocol is not very appealing. In particular, the general MPC protocol of [BMR90] can compute the gcd of two polynomials in constant number of rounds and with communication complexity of $O(n^2)$ multiplications. Our protocol is more interesting in the *information-theoretic* setting. In the information-theoretic setting, we only know of general constant-round protocols for problems in $NL$ (please see [FKN94],[IK97],[IK00]). It is unlikely that these general techniques would lead to the same communication-efficiency as our protocol for polynomial gcd.

### 7.3 Are Two Shared Polynomials Coprime?

Given shares of $f(x)$ and $g(x)$ in $F[x]$, parties want to compute shares of the bit $b$ such that $b = 0$ if $gcd(f, g) = 1$, and $b = 1$ otherwise.

Let us consider the *Syslvester* matrix ($S_0$) of the two polynomials. The determinant of this matrix is also called the *resultant* of two polynomials. The following is a corollary of theorem 1:

**Corollary 1.** *$gcd(f, g) = 1$ iff $det(S_0) \neq 0$.*

This leads to the following protocol:

1. Parties compute shares of the determinant of $S_0$.
2. Parties compute shares of the bit $b$ such that $b = 0$ if $det(S_0) = 0$ and $b = 1$ otherwise.

This reduces the problem to a protocol for testing equality of a shared value with zero. One can use the protocol given in [DFNT05] to implement such a functionality in constant-round.

### 7.4 Shared GCD of Many Polynomials

Given shares of polynomials $f_1, f_2, \ldots, f_t$ in $F[x]$, parties want to compute shares of $gcd(f_1, ..., f_t)$.

Let $g = f_2 + \sum_{3 \leq i \leq t} r_i f_i$, where $r_i$'s are chosen independently at random from $F$. The following theorem shows that $gcd(f_1, \ldots, f_t) = gcd(g, f_1)$ with very high probability (see Chapter 6 of [GG03] for proof details).

**Theorem 3.** *The probability that $gcd(f_1, \ldots, f_t) \neq gcd(g, f_1)$ is less than* $(\max_{1 \leq i \leq t} deg(f_i))/q$.

This leads to the following protocol for computing the shared gcd of many shared polynomials:

1. Parties generate shares of random field elements $r_i$ for all $i \in \{1..t\}$.
2. Parties compute shares of $g = f_2 + \sum_{3 \leq i \leq t} r_i f_i$.
3. Parties compute shares of $d = gcd(g, f_1)$ using the given constant-round polynomial gcd.

## References


[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of ACM PODC, pp. 201-209*, 1989.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of ACM STOC, pages 1-10*, 1988.

[BMR90] D. Beaver, S. Micali, P. Rogaway. The Round Complexity of Secure Protocols. In *Proceedings of 22nd ACM STOC, pp. 503-513*, 1990.

[CCD88] D. Chaum, C. Crepeau, and I. Damgard. Multi-party unconditionally secure protocols. In *Proceedings of ACM STOC, pages 11-19*, 1988.

[CD01] R. Cramer and I. Damgard. Secure distributed linear algebra in a constant number of rounds. In *Proceedings of Crypto, pages 119-136*, August 2001.

[CDN01] R. Cramer, I. Damgard, and J. Nielsen. Multiparty computation from homomorphic encryption. In *Proceedings of Eurocrypt, pages 280-300*, 2001.

[DFNT05] I. Damgård, M. Fitzi, J. Buus Nielsen, and T. Toft. How to split a shared secret into shared bits in constant-round. Cryptology ePrint Archive, Report 2005/140, 2005. http://eprint.iacr.org/.

[FIPR05] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Proceedings of Theory of Cryptography Conference*, 2005.

[FKN94] U. Feige, J. Kilian, M. Naor. A Minimal Model for Secure Computation. In *Proceedings of ACM STOC '94, pp. 554-563*, 1994.

[FNP04] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of Eurocrypt*, 2004.

[FP00] P. Fouque and D. Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In *Proceedings of Asiacrypt, pages 573-84*, 2000.

[GG03] J. Von Zur Gathen and J. Gerhard. *Modern Computer Algebra*. University Press, Cambridge, 2nd edition, 2003.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. *Proceedings of the 19th Annual ACM symposium on Theory of Computing, pages 218-229*, 1987.

[GRR98] R. Gennaro, M. Rabin, and T. Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of ACM PODC, pages 101-111*, 1998.

[IK97] Y. Ishai and E. Kushilevitz. Private Simultaneous Messages Protocols with Applications. In *Proceedings of 5th Israel Symposium on Theoretical Comp. Sc., pp. 174-183*, 1997.

[IK00] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A New Paradigm for Round-efficient Secure Computation. In *Proceedings of FOCS*, 2000.

[KS05] Lea Kissner and Dawn Song. Privacy preserving set operations. In *Proceedings of CRYPTO '05*, August 2005.

[Pai00] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of Asiacrypt, pages 573-84*, 2000.

[Sha79] A. Shamir. How to share a secret. In *CACM, pages 612-613*, 1979.

[Yao82] A. C. Yao. Protocols for secure computation. In *Proceedings of Focs, pp. 160-164*, 1982.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Proceedings of 27th FOCS, pages 162-167*, 1986.