Private Computation Based On Polynomial Operations

Shuailiang Hu[1][0000−0003−3934−9093]

Huazhong University of Science and Technology
HSL_03299319@126.com

Abstract. Privacy computing is a collection of a series of technical systems that intersect and integrate many disciplines such as cryptography, statistics, artificial intelligence, and computer hardware. On the premise of not exposing the original data, it can realize the fusion, sharing, circulation and calculation of data and its value in a manageable, controllable and measurable way. In the case of ensuring that the data is not leaked, it can achieve the purpose of making the data available and invisible, as well as the conversion and release of data value.
We propose a privacy computing algorithm based on polynomial operations based on the unsolvable problem of high-degree polynomial modulo n. Encryptors can encrypt their own private information to generate ciphertext that can be calculated. This ciphertext can support any integer calculation including addition, multiplication, power calculation, etc. Different ciphertexts generated by the same key are fully homomorphic, and addition and multiplication operations can be performed between these ciphertexts. All calculations in our encryption system are carried out with polynomials as the medium, so the calculation efficiency is guaranteed. Our ciphertext can be provided to a third party for processing without revealing the encryption party's key and secret information.

Keywords: privacy computing · Lagrangian Interpolation Polynomials · Secure Multiparty Computation.

1 Introduction

With the rapid development of the mobile Internet and the advent of the era of big data, private information has become an important part of big data. At the same time, users also enjoy personalized services provided by different applications, which brings great convenience to life. However, the information collected by big data includes sensitive information such as identity, hobbies, geographical location, and personal income. Once this personal privacy information is leaked, it will bring great security risks. Therefore, the emergence of privacy computing provides a solution for our data security. To put it simply, privacy computing realizes the purpose of "data is available but not visible" and the transformation and release of data value to ensure that the data itself is not leaked to the outside world.

Compared with traditional data usage methods, the privacy computing encryption mechanism can enhance data protection and reduce the risk of data leakage. Traditional data security methods, such as data desensitization or anonymization, must sacrifice part of the data At the expense of dimensionality, data information cannot be effectively used, while privacy computing provides another solution to ensure that the value of data is maximized as much as possible under the premise of security.

Therefore, we design a privacy computing mechanism based on polynomial operations by combining the unsolvable properties of high-degree polynomial modulo n (n=p*q, p, q are both private primes). Our security is based on the fact that the plaintext information corresponding to the ciphertext is unknowable. Assuming that for attackers, they can only obtain the encrypted ciphertext without knowing the corresponding plaintext, then our encryption mechanism is safe.

1.1 Our Result

- We propose a privacy computing algorithm based on polynomial operations based on the unsolvable problem of high-degree polynomial modulo n. Encryptors can encrypt plaintext information to obtain a ciphertext that can be used for calculation. Computation of fully homomorphic properties is supported between different ciphertexts generated by the same key.
- We present a private computation that supports batch operations. We simultaneously encrypt multiple plaintext messages to generate an integrated ciphertext. The operation on the ciphertext can be fed back to multiple plaintexts at the same time.

2 Preliminaries

2.1 Finding Roots of Polynomial Modulo n

We define the assumption of Finding Roots of Polynomial Modulon (FROP-MN) as follows:

Assumption 1 ***FROP-MN.*** *In the case of modulo n(n=p*q, p, q are both private primes), the polynomial with a degree of at least 2 cannot be solved if neither p nor q is known. That means that given a polynomial with a degree of at least 2 such as the cubic polynomial, or quartic polynomial equation satisfying $P(x) = c$, we cannot find even a root of P.*

Through the Rabin encryption algorithm[**?**], we have known that quadratic polynomials cannot be solved modulo n (n=p*q, p, q are both private large primes). Then, we have the following theorem:

Theorem 1. *In the case of modulo n(n=p*q, p, q are both private primes), a quadratic equation P satisfying $P(x) = c$ cannot be solved if neither p nor q is known.*

Proof. Suppose p, q are two large prime numbers satisfying n=p*q, and c is an element in Z_n. We want to solve the following equation:

$$x^2 \equiv c(mod \quad n)$$

This is a quadratic equation about the unknown element x in Z_n. Decryption requires finding the square root modulo n, equivalent to solving the following congruence equations.

$$\begin{cases} x^2 \equiv c(mod \quad p) \\ x^2 \equiv c(mod \quad q) \end{cases}$$

Because p, q are unknowns, solving $x^2 \equiv$ c(mod n) is as difficult as factoring a large integer n to get p, q and it is impossible as p and q are large enough. As shown above, the quadratic equation can be transformed into a quadratic congruence equation, so a quadratic equation modulus n cannot be solved.

Theorem 2. *If two independent polynomials P_1 and P_2 are given at the same time and $P_1(x_1) = P_2(x_1) = 0$, then the minimum order of solving equations can be reduced by 1. But given two non-independent polynomials P_1 and P_2 satisfying $P_1(x_1) = P_2(x_1) = 0$ at the same time, no effective information about x_1 can be obtained.*

Proof. Here we take quadratic polynomials and cubic polynomials as an example. Suppose there are two polynomials satisfying $P_2(x_1) \equiv 0, P_3(x_1) \equiv 0$(mod n, n=p*q, and p, q are unknown) and P_2, P_3 are independent for each other, where P_2 and P_3 are quadratic polynomial and cubic polynomial respectively.

Suppose we have $P_2 = x^2 + ax + b$ and $P_3 = x^3 + a'x^2 + b'x + c'$ satisfy:

$$\begin{aligned} P_3 \equiv & 0(mod \quad n) \\ & and \\ P_2 \equiv & 0(mod \quad n) \end{aligned}$$

To find intersection coordinate point x_1, let us combine two polynomials to get the following system of equations:

$$\begin{cases} P_3 = x^3 + a'x^2 + b'x + c' \equiv 0(mod \quad n) \\ P_2 = x^2 + ax + b \equiv 0(mod \quad n) \end{cases}$$

Then, we can change the solution of the above system of equations into the solution of the following system of equations(convert P_3 to the quadratic equation)

$$\begin{cases} P_3 = x^3 + a'x^2 + b'x + c' \equiv 0(mod \quad n) \\ x^2 = -ax - b \equiv 0(mod \quad n) \end{cases}$$

Finally, we can compute the above system of equations to obtain a first-order equation for x(convert P_3 to a first-order equation by using x^2). On the contrary, if P_2 and P_3 are not independent of each other, P_3 can not be calculated through P_2, and the result obtained through the above equation will be 0=0 instead of a usable first-degree polynomial. This method is also applicable to higher-degree equations.

Proof of Assumption 1. We perform a generous condition to this assumption. We use the polynomial intersection problem to prove it: Given multiple polynomials passing through the same point at the same time, we can reduce the intersection problem to the problem of solving polynomials with a lower degree. Suppose we have a polynomial P_1 of degree s-1(s is an integer) that satisfies $P_1(x_1) = 0$. In order to solve intersection point x_1, we generously give another s-3 polynomials satisfying $P_2(x_1) = 0, P_3(x_1) = 0, ..., P_{s-2}(x_1) = 0$.

We know that given two polynomials passing through the same point, the minimum degree of polynomials to be solved can be reduced by 1(Reference to Theorem 2). In this proof process, we have generously given s-2 polynomials $P_1(x_1) = 0, P_2(x_1) = 0, P_3(x_1) = 0, ..., P_{s-2}(x_1) = 0$ and we can combine these polynomials to get a quadratic polynomial P through point x_1 satisfying $P(x_1) = 0$. For example, we can first combine P_1, P_2 through Theorem 2 to obtain a polynomial of degree s-2, and then combine the new polynomial with P_3 to obtain a polynomial of degree s-3. In this way, we can finally get a quadratic polynomial P through the point x_1 satisfying $P(x_1) = 0$ by combing these s-2 polynomials. Then, according to Theorem 1, we have known that the second-degree polynomial modulo n is unsolvable, so we say higher-degree polynomials are also unsolvable.

In other words, assuming an algorithm F exists that can solve the roots of high-degree polynomials modulo n in polynomial time, then the theorem 1 is incorrect and the quadratic polynomial can also be solved. It is said that given a polynomial f(x) of quadratic or greater, F(f) can output a value x_1 satisfying f(x_1)=0. Then, according to the Rabin algorithm, we already know that the quadratic congruence equation is unsolvable in the case of modulo n since factoring is intractable. Therefore, a quadratic polynomial cannot be solved when modulo n (n=p*q, p, q are both private primes). However, if the algorithm F exists, the quadratic polynomial can be solved and the Rabin encryption algorithm can be cracked. But we all know Rabin's algorithm is safe if the large integer is not decomposed, so F does not exist and Assumption 1 holds.

2.2 Lagrangian Interpolation Polynomial

The Lagrange interpolation formula refers to a node basis function given on the nodes of a two-dimensional coordinate system. A polynomial function with the degree of s-1 can be determined by s coordinate points (x_i, y_i)($1 \leq i \leq s$) in a two-dimensional rectangular coordinate system. As shown in Fig. 1, a curve can be determined according to s (s is an integer and s≥2) points that are

different from each other in the rectangular coordinate system. For this curve, there is only one definite polynomial corresponding to it. Similarly, if a curve's polynomial function expression (polynomial coefficient) is known, as long as any abscissa value x_i can be given, its ordinate value y_i can be obtained. Therefore, if we can give s coordinate points, we can also construct a polynomial with polynomial degree s-1.

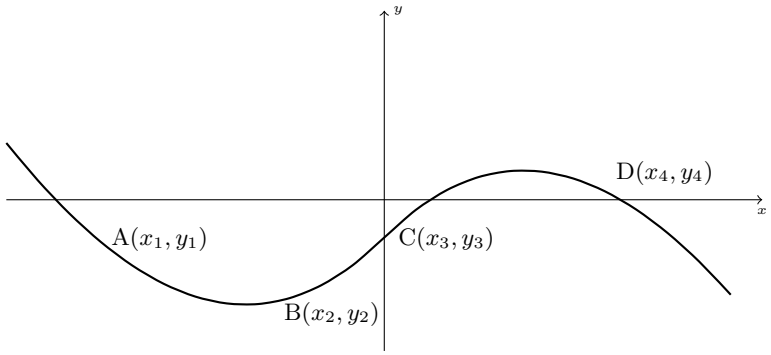

Fig. 1. Lagrangian interpolation polynomial

Suppose there are s pairs of coordinate points, the generalized definition of the Lagrangian interpolation formula is shown in equation (1).

$$\begin{aligned} p(x) =& \sum_{i=1}^{s} \prod_{j \neq i}^{s} \frac{(x - x_j)}{(x_i - x_j)} * y_i \\ =& \frac{(x - x_2)\dots(x - x_s)}{(x_1 - x_2)\dots(x_1 - x_s)} * y_1 + \\ & \frac{(x - x_1)(x - x_3)\dots(x - x_s)}{(x_2 - x_1)\dots(x_2 - x_s)} * y_2 + \\ & \quad \dots \quad + \\ & \frac{(x - x_1)\dots(x - x_{s-1})}{(x_s - x_1)\dots(x_s - x_{s-1})} * y_s \\ =& P_{s-1} * x^{s-1} + P_{s-2} * x^{s-2} + \dots + P_0 * x^0 \end{aligned} \tag{1}$$

We extract the coefficients of each term and give the following definition:

$$
\begin{aligned}
& R_i = \prod_{j \neq i}^{s} \frac{(x - x_j)}{(x_i - x_j)} \\
then \quad & R_i(x_i) = 1 \quad and \quad R_i(x_j) = 0 \quad j \neq i \quad satisfy \quad 1 \leq i, j \leq s \\
& R_i(x_l) = else \quad s < l
\end{aligned} \tag{2}
$$

(We ignore the case where values of two coordinate points are equal.)

To facilitate the construction of our encryption scheme, we denote x above as k, y as m, and approximately denote R_i as a candidate encryption key, k_i as a decryption key. Therefore, according to equation (2) we have $R_i = \prod_{j \neq i}^{s} \frac{(k-k_j)}{(k_i-k_j)}$ and $R_i(k_i) = 1, R_i(k_j) = 0, R_i(k_l) = else(1 \leq i, j \leq s < l, j \neq i)$. Then we have $R_1 = \prod_{j \neq 1}^{s} \frac{(k-k_j)}{(k_1-k_j)}$ and $R_1(k_1) = 1, R_1(k_j) = 0, R_1(k_l) = else(1 < j \leq s < l, j \neq i)$. We can find that for users with k_1, he can calculate $R_1 = 1$ and $R_j = 0(1 \leq j \leq s < l, j \neq 1)$. But other users without k_1 get nothing from these polynomials. We already know that high-order congruence equations are unsolvable modulo n (p, q are unknown) according to Assumption 1. Then, according to the particularity of the Lagrange interpolation polynomials, we know that the s-coefficient polynomials $R_1, R_2, ..., R_s$ are independent of each other since we have the following equations:

$$
\begin{cases}
R_1[k_1, k_2, ..., k_s] = [1, 0, ..., 0, ..., 0] \\
R_2[k_1, k_2, ..., k_s] = [0, 1, ..., 0, ..., 0] \\
\qquad\qquad ..... \\
R_i[k_1, k_2, ..., k_s] = [0, 0, ..., 1, ..., 0] \\
R_s[k_1, k_2, ..., k_s] = [0, 0, ..., 0, ..., 1]
\end{cases}
$$

3 Private Computation Based On Polynomial Operations

3.1 Construction of P-PC

A complete Private computation based on polynomial operations(P-PC) is a 4-tuple of ppt algorithms(Gen, Enc, Com, Dec) such that:

Gen*($\mathbf{1^\lambda}, s$)**: The key generation algorithm *Gen* takes a security parameter $\mathbf{1^\lambda}$ and a lagrangian interpolation parameter s=5 as input. ***Gen does the following:

1. Generate two large primes p, q and compute n=p*q.
2. Randomly select k_1, k_2, k_3, k_4, k_5.
3. Use k_1, k_2, k_3, k_4, k_5 to generate :R_1, R_2, R_3 according to equation (2).

Output: dk=k_1, ek=[R_1, R_2], inf={n, R_3}

Enc*(ek, m, inf)**: The encryption algorithm *Enc* takes ek, a message $m \leftarrow Z_n$, and supplementary information inf as input. Then, ***Enc does the following:

1. Sample random element $r_1, r_2 \leftarrow Z_n$.

2. Use ek, r_1, r_1, $\{m_1, m_2, ...\}$ and inf to generate c=[{c_{m1}, c_{m2}, ...}, inf]:
$c_{mi} = \boldsymbol{Enc(ek, m_i, inf)} = R_1 * m_i + (R_1 - 1) * r_1 + R_2 * r_2$ mod n

$\boldsymbol{Com(c)}$: Com takes a ciphertext and a privacy algorithm $\mathcal{C}$ as input. $\boldsymbol{Dec}$ outputs $c' = \mathcal{C}(c)$.

$\boldsymbol{Dec(dk, c')}$: The decryption algorithm Dec takes c' and the decryption key dk as input. $\boldsymbol{Dec}$ does the following:
$\mathcal{C}(m) = \sum_{i=0}^{4} c'[i] * k_1^i$

3.2 Composition of Algorithm $\mathcal{C}$

In the above computing system, the privacy computing algorithm C is a set of addition and multiplication algorithms on Z_n. In fact, our calculations support the four arithmetic operations of addition, subtraction, multiplication, and division. However, existing literature tells us that addition and multiplication are already Turing complete, so we only discuss addition and multiplication. Therefore, in this section, we discuss several situations that may arise in $\boldsymbol{Com(c)}$.

$c_m \Delta$ t. $c_m + t$ is to add a t to the constant term of the ciphertext polynomial c_m. Suppose $c_m = [a, b, c, d, e]$, then $c_m + t = [a, b, c, d, e + t]$. The calculation is equivalent to $\boldsymbol{Enc(ek, m + t, inf)}$. $c_m + t$ is to add a t to the constant term of the ciphertext polynomial c_m. $c_m * t$ is to multiply the ciphertext polynomial c_m by an integer t. This calculation is equivalent to $\boldsymbol{Enc(ek, m + t, inf)} = c_m * t = [a * t, b * t, c * t, d * t, e * t]$.

We can find that the above two operations are relatively simple, and the information of $\boldsymbol{\mathcal{C}}$ may be leaked when facing an algorithm $\boldsymbol{\mathcal{C}}$ that only involves integers in the operation. Therefore, when facing simple $\boldsymbol{\mathcal{C}}$, we can get $\boldsymbol{Enc(ek, 1, inf)}$ advance, and then perform calculations based on addition and multiplication between polynomials. But for complex algorithms $\boldsymbol{\mathcal{C}}$, $\boldsymbol{Enc(ek, 1, inf)}$ is not needed and we use $c_m \Delta$ t.

$c_{m1} \Delta$ c_{m2}. In our system, the ciphertexts generated by $Enc(ek, m, inf)$ are polynomials of degree s-1 represented by s-dimensional vectors and s is equal to 5 here. Therefore, $c_{m1} + c_{m2}$ represents the addition between two quartic polynomials, and $c_{m1} * c_{m2}$ represents the multiplication modulus between two quartic polynomials $R_3 * k$ in inf. Therefore, $c_{m1} + c_{m2}$ represents the addition between two quartic polynomials, and $c_{m1} * c_{m2}$ represents the multiplication between two quartic polynomials. In addition, $c_{m1} * c_{m2}$ needs to modulo $R_3 * k$ in inf to ensure that the length of the new ciphertext does not change. For example, $c_{m1} = [a, b, c, d, e], c_{m2} = [a', b', c', d', e']$. We have $c_{m1} + c_{m2} = [a + a', b + b', c + c', d + d', e + e']$ and $c_{m1} * c_{m2} = (ak^4 + bk^3 + ck^2 + dk + e) * (a'k^4 + b'k^3 + c'k^2 + d'k + e')$ mod R_3. This calculation can make the $\boldsymbol{Com(c)}$ algorithm support the fully homomorphic property among the same type of ciphertexts and make $\boldsymbol{Com(c)}$ support non-linear operations such as power operations.

3.3 Correctness

First, for the ciphertext $c = R_1 * m + (R_1 - 1) * r_1 + R_2 * r_2$ generated by $\boldsymbol{Enc(ek, m, inf)}$, we have $R_1(k_1) = 1, R_2(k_1) = R_3(k_1) = 0$. Thus we have $\boldsymbol{Dec(dk, Enc(ek, m, inf))} = m$. Then, for several cases that appear in $\boldsymbol{Com(c)}$, we have: 1) $\boldsymbol{Dec(dk, c_m \Delta t)} = m\Delta t$; 2)$\boldsymbol{Dec(dk, c_{m1} \Delta c_{m2})} = m_1 \Delta m_2$. Therefore, the calculation information of c in $\boldsymbol{Com(c)}$ will be directly reflected in the plaintext information. Thus, we have $\boldsymbol{Dec(dk, Com(c))} = \mathcal{C}(m)$ and the above privacy calculation process is correct.

3.4 Security

The security proof of the above calculation process is divided into two aspects: the security of k_1 and the security of m. The security requirement of k_1 is based on the fact that the attacker cannot solve the encryption party's private key $dk = k_1$. The security of m requires that the attacker cannot obtain its plaintext information m through the parameter inf and ciphertext c.

First, in the above encryption and calculation system, we set s=5 and only use R_1, R_2, R_3 as encryption parameters. We know that R_1, R_2, R_3 are all quartic polynomials, and we can only get a quadratic equation about k_1 by combining them. According to the Assumption 1 and the polynomial intersection problem, even if the attacker obtains the ciphertext corresponding to a certain amount of plaintext, he cannot solve k_1.

For the ciphertext $c = R_1 * m + (R_1 - 1) * r_1 + R_2 * r_2$ generated by the $\boldsymbol{Enc}$ algorithm, we only have n and R_3 in inf for the calculation side, while R_1 and R_2 are all unknown. Because R_1, R_2, R_3 are linearly independent, given R_3 will not affect the encryption of plaintext information by R_1, R_2. Therefore, it is impossible for an attacker to construct an equation to solve the plaintext information m through the ciphertext. However, it is worth noting that if the attacker obtains a certain amount of ciphertext corresponding to the plaintext, it is possible to crack the ciphertext to obtain the plaintext m. Therefore, we only give the ciphertext of the message m without revealing the corresponding plaintext in the system.

In addition, we need to carefully consider whether to reuse ek to encrypt the same plaintext information m, because this may reveal some information about R_1, R_2 in actual use. For example, we have two ciphertexts c_{m1}, c_{m2} and $m_1 = m_2$. Then we have $c_{m1} - c_{m2} = R_2 * (r_1 - r_2)$. Therefore, we do not recommend reusing ek to encrypt the same plaintext information but we can use the same ciphertext corresponding to the same message m.

4 Batched Private Computation Based On Polynomial Operations

4.1 Construction of BP-PC

beginConstruction Let P-PC = (Gen, Enc, Com, Dec) be a Private Computation Based On Polynomial Operations. We construct b-Batched Private Computation

Based On Polynomial Operations(b is the size of a batch), BP-PC, as follows:

$\boldsymbol{Gen(1^{\lambda}, s)}$: The key generation algorithm Gen takes a security parameter $\mathbf{1^{\lambda}}$, a batch size b, and a lagrangian interpolation parameter s=b+3 as input. $\boldsymbol{Gen}$ does the following:

1. Generate two large primes p, q and compute n=p*q.
2. Randomly select k_1, k_2, k_3, ..., k_{b+3}.
3. Use k_1, k_2, k_3, ..., k_{b+3} to generate :R_1, R_2, R_3, ..., R_{b+1} according to equation (2).

Output: dk=$\{k_1, k_2, k_3, ..., k_b\}$, ek=$[R_1, R_2, R_3, ..., R_{b+1}]$, inf=$\{n, R_{b+1}\}$

$\boldsymbol{Enc(ek, m, inf)}$: The encryption algorithm Enc takes ek, a message $m \leftarrow Z_n$, and supplementary information inf as input. Then, $\boldsymbol{Enc}$ does the following:

1. Sample random element $r_1, r_2 \leftarrow Z_n$.
2. Use ek, r, $\{[m_{11}, m_{12}, ..., m_{1b}], [m_{21}, m_{22}, ..., m_{2b}], ...\}$ and inf to generate c=$[\{c_{m1}, c_{m2}, ...\}, inf]$:

$c_{mi} = \boldsymbol{Enc(ek, m_i, inf)} = \sum_{j=1}^{b} R_1 * (m_{ij} + r_1) + R_{b+1} * r_2 - r_1 \bmod n \ (1 \leq i)$

$\boldsymbol{Com(c)}$: Com takes a ciphertext and a privacy algorithm $\mathcal{C}$ as input. $\boldsymbol{Dec}$ outputs $c' = \mathcal{C}(c) \bmod R_{b+1} * k$.

$\boldsymbol{Dec(dk, c')}$: The decryption algorithm Dec takes c' and the decryption key dk as input. $\boldsymbol{Dec}$ does the following:

$\mathcal{C}(m_l) = \sum_{j=0}^{4} c'[j] * k_l^j \ (1 \leq l \leq b)$

4.2 Correctness

According to Equation (2), we know $R_i(k_i) = 1, R_i(k_j) = 0, R_i(k_l) = else (1 \leq i, j \leq s < l, i \neq j)$, so we have $R_i^2(k_i) = 1, R_i * R_j(k_i) = 0$. Therefore, for the ciphertext c_m generated by $Enc(ek, m, inf)$ we have $\boldsymbol{Dec(dk, Enc(ek, m, inf))} = m = [m_1, m_2, m_3, ..., m_b]$. For the calculated ciphertext $c' = c_m \Delta t$, obviously $\boldsymbol{Dec(dk, c')} = m\Delta t = [m_1 \Delta t, m_2 \Delta t, m_3 \Delta t, ..., m_b \Delta t]$ is correct. For $c' = c_{m1} \Delta c_{m2}$, we have $\boldsymbol{Dec(dk, c')} = \boldsymbol{Dec(dk, c_{m1} \Delta c_{m2})} = \boldsymbol{Dec(dk, c_{m1}) \Delta Dec(c_{m2})} = m_1 \Delta t m_2 = [m_{11} \Delta m_{21}, m_{12} \Delta m_{22}, m_{13} \Delta m_{23}, ..., m_{1b} \Delta m_{2b}]$. The result of the operation between polynomial ciphertexts will not be affected by $R_{b+1} * k$, because any k_i in dk has $R_{b+1} * k(k_i) = 0 (1 \ lei \leq b)$.

4.3 Security

We set s=b+3 and only use b+1 independent R as encryption parameters in the encryption process. According to the Assumption 1 and the polynomial intersection problem, the attacker cannot crack any Decryption key $k_i (1 \leq i \leq b)$. In addition, compared to $c = R_1 * m + (R_1 - 1) * r_1 + R_2 * r_2$, $c = \sum_{j=1}^{b} R_1 * (m_j + r_1) + R_{b+1} * r_2 - r_1$ contains more plaintext information. Because multiple plaintexts are sufficiently confused in one ciphertext, it will be more difficult

for an attacker to obtain the plaintext information in the ciphertext. Because $R_1, R_2, ..., R_b$ are all unknown, it is impossible for an attacker to obtain a certain plaintext when only having the ciphertext.

5 Conclusion

We propose a privacy computing algorithm based on polynomial operations. This algorithm enables the encryption party to hide its private information and provide it to other users for calculation. In our scheme, the encryption party encrypts the plaintext information and provides it to an algorithm mechanism for any integer calculation (including addition, multiplication, power, and calculation between ciphertexts). In addition, we provide a batch of encryption algorithms for privacy calculations. We encrypt multiple plaintext information and generate an integrated ciphertext, and the operation of this ciphertext will be fed back to multiple plaintext information at the same time. There is no doubt about the efficiency of our scheme because of the efficiency of polynomials. When building an encryption system, we can use a sufficiently large modulus n to make our encryption scheme more secure. In addition, it is not difficult to see that those ciphertexts in our scheme can be directly used for addition, subtraction, multiplication, and division calculations without any key. Based on this property, our schemes can be used in various blind computing application scenarios. For example, we can try them in the following application scenarios:

- The stock market. Assuming that the stock index of shareholder A has risen by a certain percentage, he does not want others to know how much capital he has invested. He can encrypt his wallet, and then send the encrypted ciphertext wallet to the stock center for new stock calculation directly. After receiving the encrypted Wallet over evaluation, shareholder A can decrypt it to obtain the final stock. But users except A cannot know how much amount A owns because they do not have A's decryption key.
- Encrypted digital wallet. When conducting a transaction, user A encrypts his wallet and sends the encrypted wallet to the transaction partner. Then transaction partner directly performs operations such as deduction and payment on the received encrypted wallet and sends it to A in the form of ciphertext. A can get his balance after decrypting the encrypted wallet as the transaction closes. This idea is generally used in public domains such as digital wallets. We can perform encrypted homomorphic calculations on users' wallets. Then users can use their encrypted wallets to perform any transaction, but only the wallet's owner can know the balance during the transaction.

To facilitate the improvement of our scheme and propose a better fully homomorphic encryption scheme, we give the prospect of future work:

1) Further analyze the feasibility and security of our scheme, and hope to improve our scheme to a fully homomorphic scheme. Our encryption scheme

uses Lagrange interpolation polynomials to generate ciphertexts. We know that there are many polynomials available in the Lagrange interpolation theorem, and it can be considered whether our scheme can be extended to realize the fully homomorphic encryption based on the polynomial operation.

2) Find other techniques that are more suitable for constructing our scheme to replace the Lagrangian interpolation polynomials, and try more efficient methods to improve our scheme.

References