Fully Homomorphic Encryption Based On Polynomial Operation

Shuailiang Hu[1][0000−0003−3934−9093]

Huazhong University of Science and Technology
HSL_03299319@126.com

Abstract. Homomorphic encryption requires the homomorphism of encrypted ciphertext, and the operation between ciphertexts can be reflected in plaintexts. Fully homomorphic encryption requires that the encryption algorithm can satisfy additive homomorphism and multiplicative homomorphism at the same time. At present, there are many fully homomorphic encryption schemes, such as fully homomorphic encryption based on ideal lattices, AGCD problem, LWE problem, RLWE problem, and so on. But the improvement of efficiency, length of ciphertext, and calculation limit of the fully homomorphic encryption scheme are still problems that need further study.
Based on Lagrangian interpolation polynomials, we propose a fully homomorphic encryption scheme according to the difficulty of finding roots of a polynomial with the degree of at least two(mod n=p*q, p, q are both private large primes). We reasonably construct polynomials $trap_1$ and $trap_0$ to generate the ciphertext of message m, so that calculation between ciphertexts can directly act on plaintexts. Our scheme does not involve noise in the whole encryption process and does not require any related techniques such as bootstrapping techniques or rescaling techniques during the ciphertext evaluation process. Our decryption algorithm always outputs accurate results when decryption and our scheme is safe as long as the Rabin encryption algorithm cannot be cracked.

Keywords: Fully Homomorphic Encryption · Lagrangian Interpolation Polynomial · Secure Multiparty Computation.

1 Introduction

Homomorphic encryption is a cryptographic technique based on the computational complexity theory of mathematical puzzles. Using homomorphic encryption technology, a user performs operations on ciphertexts and then decrypts the result of ciphertext to be consistent with the result obtained by directly operating on plaintexts. This feature allows an untrusted third party to directly perform operations on ciphertexts without a private key, avoiding the leakage of sensitive information caused by a third party. Fully homomorphic encryption technology is a trending technology that can be applied to outsourcing computing, privacy-preserving machine learning, secure multi-party computing, joint learning, data exchange, sharing, etc.[1]

In 1978, Rivest et al. proposed the first public key encryption scheme: the RSA encryption scheme. Then, they pointed out the multiplicative homomorphic property of the RSA cryptographic system. The effective ciphertext of the plaintext $m_1 * m_2$ can be calculated using the ciphertext $c_1 * c_2$ "homomorphism" without knowing the private key information. Therefore, they proposed the definition of "Fully Homomorphic Encryption" (FHE)[2]. Considering the powerful capabilities of fully homomorphic encryption, it has become an open issue in the cryptographic community once it is proposed[3].

Cryptographic assumptions that can construct fully homomorphic encryption mainly include: Ideal Coset Problem called ICP based on ideal lattice[4], Approximate Greatest Common Devisor (AGCD) on integers[5], Learning with Errors (LWE)[6], Approximate eigenvectors[7], circuit control technology[8] and so on. However, many restrictions still need to improve, such as inefficient information processing, complex ciphertext and keys, the difficulty of implementing encryption, etc. These restrictions make fully homomorphic encryption difficult to use in practice.

To improve the efficiency of FHE, we propose a scheme based on the difficulty of finding roots of a polynomial with the degree of at least two(mod n=p*q, p, q are both private large primes), and it is secure as long as Rabin encryption is secure. We call it a new high-efficiency polynomial-based fully homomorphic encryption scheme, $trap_1 \times m$ ($trap_1$ is a polynomial of degree seven or higher-degree polynomial and $m \leftarrow Z_n$). Our encryption algorithm does not target only one bit of information at a time but for an integer plaintext message m ($m \leftarrow Z_n$).

In our fully homomorphic encryption scheme, calculations between ciphertexts are all calculated by seven or higher degree polynomials. For example, if a message is encrypted with polynomials of degree seven in our scheme, the ciphertext length is only seven times the size of n. In addition, to make our scheme more effective, we also propose a method to fix the ciphertext length of our scheme. No matter how many ciphertexts are involved in the calculation, we still get a polynomial ciphertext of degree seven by modulo an octave polynomial. Because of the unsolvable problem of the quadratic polynomial or higher-degree polynomial of $Z_n(x)$ (n=p*q, p, q are both private large primes), the security of our scheme is guaranteed.

1.1 Contribution

The contributions of this paper are divided into the following aspects:

– Different from existing fully homomorphic encryption schemes, we construct an efficient fully homomorphic encryption based on Lagrangian interpolation polynomials and operations between polynomials. Our scheme does not involve noise in the whole encryption process and does not require any related techniques such as bootstrapping techniques or rescaling techniques during the ciphertexts evaluation process. The decryption algorithm is always able to decrypt the ciphertext exactly. Our scheme satisfies the unsolvable property of polynomials modulo n(n=p*q, p, q are both large private primes), making our scheme feasible.

- We mainly give two implementations of our scheme. They are encryption using the same encryption key and using different encryption keys respectively. Compared with existing schemes, our scheme is simpler to implement and only a few polynomials of degree seven modulo n are required to implement an efficient fully homomorphic encryption scheme. We prove that our algorithm is safe with the safety of polynomial computation and unsolvable polynomial modulo n. The security of our encryption algorithm is guaranteed as long as the Rabin algorithm is safe.

- We also give a method to make the ciphertext size constant for our scheme. In constant size polynomial-based fully homomorphic encryption(Construction 3), our scheme has no restriction on calculation and the length of ciphertexts in our encryption remains unchanged during the evaluation, which always keeps the s size of n. The performance of our algorithm will not decrease as the calculation between ciphertexts is always equivalent to the calculation of polynomials of degree seven.

It is worth noting that our encryption scheme is arithmetically homomorphic because our scheme not only supports addition and multiplication operations but also supports subtraction and division operations. However, relevant literature indicates that only addition and multiplication are needed to satisfy Turing completeness. Therefore, the main contribution of this paper will be around the fully homomorphic encryption scheme with addition and multiplication operations.

1.2 Related Works

In 1978 [2], the concept of homomorphic encryption was proposed by three researchers, Rivest, Adleman, and Dertouzos. The earliest public key cryptosystem RSA was introduced and it is also the earliest encryption scheme with multiplicative homomorphism. Then, Fully Homomorphic Encryption(FHE) came into being.

The first fully homomorphic encryption scheme was proposed in 2009 and it was proposed by Gentry. This fully homomorphic encryption is constructed based on ideal lattice[4] and its security is based on two assumptions: some Worst-case problems, and sparse (or low-weight) subset-sum problems. The proposal of this scheme has caused an upsurge in the research of fully homomorphic encryption. In 2010, Dijk, Gentry, Halevi, and Vaikuntanathan[5] proposed a fully homomorphic encryption scheme based on integers(DGHV), and the design is based on the approximate greatest common factor problem. This scheme uses many tools of Gentry's construction but does not require ideal lattices. As a result, their scheme is conceptually simpler than Gentry's ideal lattice scheme, but operational efficiency has not been improved and the original scheme only supports low-order polynomial operations. In 2011, Brakerski and Vaikuntanathan introduced two FHE schemes based on LWE[9] and RLWE[10] problems using bootstrapping technique and circular security assumption. They also introduce two new techniques called "re-linearization" and dimension-modulus reduction to reduce the multiplication ciphertext size. In 2012, Brakerski, Gentry, and Vaikunthanathan[11] present a method for defining a leveled fully homomorphic scheme that avoids computationally expensive bootstrapping techniques. Their scheme of RLWE problem was implemented and optimized by Fan and Vercauteren[12]. In 2013, Gentry, Sahai, and Waters (GSW)[13] proposed a new technique for constructing FHE scheme that avoids the expensive "re-linearization" step in homomorphic multiplication. Brakerski and Vaikuntanathan observed that for certain types of circuits, GSW cryptosystems have slower noise growth, and are more efficient and secure. These techniques were further refined to develop efficient loop variants of the GSW cryptosystem: FHEW[14] and TFHE[15]. In 2017, Cheon proposed a new fully homomorphic encryption scheme, CKKS[7]. This scheme supports the homomorphic operations of addition and multiplication of floating-point numbers for real or complex numbers. The calculation results obtained by it are approximate values, which are suitable for scenarios that do not require accurate results, such as machine learning model training. However, these FHE schemes still have great defects, such as low encryption efficiency, only supporting bit operations, and requiring the assistance of control circuits, etc.

An article from a long time ago gave us the inspiration for our encryption scheme. In 1979, the Rabin encryption algorithm was released by Michael O. Rabin[16]. It is an asymmetric encryption algorithm based on the modular square root and its security is based on the difficulty of finding the modular square root of composite number n(n=p*q, p, q are both private large primes)[17]. We know Rabin's algorithm is secure as long as the factorization of large numbers remains practically intractable. It is said that finding the modular square root of an equation when modulo a composite number n is difficult. Through the Rabin algorithm, we know that polynomials with the second degree and above are not rootable in the case of modulo composite number n(n=p*q, p, q are both private large primes). At the same time, we have found that fully homomorphic encryption has high feasibility in the polynomial field, and it is challenging to construct a fully homomorphic encryption scheme using the polynomial-solving problem. In this paper, we hope to construct trapdoor polynomials $trap_1$ and $trap_0$ through high-degree polynomials and then construct a fully homomorphic encryption scheme about $trap_1 \times m + trap_0$. Our scheme is safe as long as the quadratic congruence equation in Rabin's algorithm is not successfully cracked.

2 Preliminaries

In this section, we define $trap_1$, $trap_0$, and give the basic definition of $trap_1 \times m$. We state that they are the basis of our fully homomorphic encryption, and they are actually some polynomial mechanisms. In addition, we give the assumption of our scheme, finding roots of a polynomial modulo n(n=p*q, p, q are both private large primes), and prove it is difficult using the security of Rabin algorithm[16]. We also give a simple method about how to realize $trap_1$, $trap_0$ by using Lagrangian interpolation polynomial[18] and give the general expression of $trap_1 \times m$.

2.1 Overview of Our Scheme

Our fully homomorphic encryption scheme is called Polynomial-based Fully Homomorphic Encryption, as our encryption is performed with polynomial operations. During the encryption process, we let the Enc algorithm be expressed as $trap_1 \times m + trap_0$, and the Dec algorithm directly outputs the result in the form of plaintext. We explain that $trap_1$ and $trap_0$ are the key encryption mechanism of our scheme, and $trap_0$ serves as an auxiliary item to assist $trap_1$ for encryption. Here we give the simple definition of $trap_1$ and $trap_0$:

$\boldsymbol{trap_1}$ **:** A public cryptographic mechanism that allows the mechanism to output 1 after bringing in secret information k. It satisfies $trap_1 \times trap_1 = trap_1$. This mechanism guarantees the security of secret information k. It means that the cryptographic mechanism $trap_1$ can be obtained through k and output number 1, but the secret information k cannot be obtained through $trap_1$. In this paper, a polynomial with a high degree modulo n is used to realize $trap_1$.

$\boldsymbol{trap_0}$ **:** This mechanism is similar to $trap_1$, except that number 0 is output after secret information k is brought in. It satisfies $trap_0 \times trap_1 = trap_0$, $trap_0 + trap_1 = trap_1$, $trap_0 \times else = trap_0$ and $trap_1 - 1 = trap_0$. We denote that $trap_{0_i}$ is a component of $trap_0$ and it has the same property as $trap_0$, which means that $trap_{0_i}$(i is an index) can also exist as a $trap_0$. The main role of mechanism $trap_0$ is to assist $trap_1$ to ensure the security of plaintext message m and secret information k.

We explain here that $\times$ is used to represent the multiplication calculation of the mechanism $trap_1$ and $trap_0$, while $*$ represents the calculation of the implementation method of $trap_1$ and $trap_0$. Since $trap_0$ outputs 0 when it is correctly decrypted, we use $trap_1 \times m$ to represent our encryption scheme for the convenience of expression. Then, we give a general representation method of fully homomorphic encryption, $trap_1 \times m$(Setup, Encrypt, Evaluate, Decryption) here:

Setup($\boldsymbol{1^\lambda}$): On input the security parameter λ, output the master public encryption key mpk(mainly include $trap_1$ and $trap_0$), and secret key sk.

Encrypt(mpk,m): On input public encryption key mpk and message m, output c=$trap_1 \times m + trap_0$.

Evaluate(mpk, $\mathcal{C}$,[$\boldsymbol{c_1, c_2, c_3 ...}$]): On input public encryption key mpk, a batch of ciphertexts $[c_1, c_2, c_3 ...]$, and an algorithm $\mathcal{C}$ that supports multiplication and addition operations, output c'=Encrypt(mpk, $\mathcal{C}(m_1, m_2, m_3, ...)$)=$\mathcal{C}(c_1, c_2, c_3 ...)$.

Decryption(sk, $\boldsymbol{c'}$): On input secret key sk and ciphertext c', then output: $\mathcal{C}(m_1, m_2, m_3, ...)$.

Among them, the Setup algorithm generates mpk represented as $trap_1$ and $trap_0$, and the Enc algorithm adds elements $trap_1$ and $trap_0$ to m for encryption. Then a ciphertext c containing $trap_1$ and $trap_0$ is generated and can calculate with other ciphertexts. Finally, the Dec algorithm removes $trap_1$ and $trap_0$ in the final ciphertext c' after evaluation and outputs the result of plaintext m'.

We find that the existence of $trap_1$ and $trap_0$ in the whole process of encryption and evaluation makes operations between different ciphertexts possible. When decrypting, we can bring the decryption key sk in ciphertext to eliminate $trap_1$ and $trap_0$ carried in ciphertext to get the final arithmetic plaintext because $trap_1$, $trap_0$, and sk exist together. As long as the Setup party does not publish sk, calculations in the entire process are performed in the form of ciphertext. Anyone who doesn't have the decryption key sk can't get any useful information about calculations and plaintexts. Therefore, the key point to realizing the encryption scheme is how to realize $trap_1$ and $trap_0$, and the calculations between ciphertexts.

The calculations between polynomials give us the idea to realize the scheme we desire. We say if there is polynomial f(x) satisfying f(x_0)=1 and f(x) is guaranteed to be complex enough that x_0 cannot be solved, we can get the $trap_1$ we want to construct $trap_1\times$ m. The same is true for $trap_0$, but the polynomial needs to output 0 after bringing in x_0.

2.2 Finding Roots of Polynomial Modulo n

Finding roots of polynomial modulo n(FROP-MN) was introduced 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 higher-degree polynomial equation satisfying $P(x) = c$, we cannot find even a root of P.*

Through the Rabin encryption algorithm[16], 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. The quadratic equation can be transformed into a quadratic congruence equation as shown above, 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:

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

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 that there exists an algorithm F 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. We already know that the security of the Rabin algorithm is based on the quadratic congruence equation under modulo n. If F exists, then Rabin's security is compromised. Therefore, we have successfully reduced the security of higher-degree equations to quadratic congruence equations. Therefore, as long as the Rabin algorithm is still safe, the higher-order congruence equations modulo n are unsolvable and Assumption 1 is correct.

2.3 Lagrangian Interpolation Polynomial and $trap_1 \times m$

According to Assumption 1, if we can construct sufficiently complex polynomials of at least second-degree modulo n, we can successfully implement $trap_1$ and $trap_0$. Therefore, we intend to use polynomials with high-degree to represent $trap_1$ and $trap_0$ and the encryption scheme $trap_1 \times m$. As for how to construct complex polynomials of more than two-degree, the realization of the Lagrangian interpolation polynomial effectively solves this problem.

The Lagrange interpolation formula refers to a node basis function given on the nodes of a two-dimensional coordinate system. Then a linear combination of this basis function is made, and the combination coefficient is an interpolation polynomial of the node function value. In simple words, 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$\geq$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 polynomial function expression (polynomial coefficient) of this curve 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.

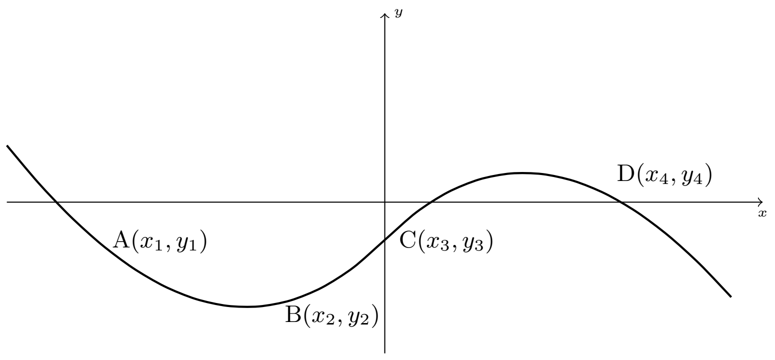

Fig. 1. Lagrangian interpolation polynomial

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. So, we put R_1 modulo n as our desired $trap_1$, $R_j(1 < j \leq s)$ modulo n as the components of $trap_0$, and k_1 as the secret information to let $R_1 = 1$ and $R_j = 0$. If there are no special instructions, we use R_1 as the master encryption key $trap_1$ and k_1 as the decryption key later. Then, according to the particularity of the Lagrange interpolation polynomials, we can get the following theorem:

Theorem 3. *According to equation (1) and (2), given s k, the s-coefficient polynomials $R_1, R_2, ..., R_s$ composed of Lagrange interpolation polynomial are independent of each other. In other words, $\{R_1, R_2, ..., R_s\}$ is a set of linearly independent vectors.*

Proof. According to equation (2), we know that polynomial $R_i(1 \leq i \leq s)$ satisfies following property:

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

According to the above equations, we can get the matrix:

$$\begin{bmatrix} 1\ 0\ ...\ 0\ ...\ 0 \\ 0\ 1\ ...\ 0\ ...\ 0 \\ ...\quad ... \\ 0\ 0\ ...\ 1\ ...\ 0 \\ 0\ 0\ ...\ 0\ ...\ 1 \end{bmatrix}$$

According to the matrix, we know that $R_1, R_2...R_s$ is a set of linearly independent vectors, and the polynomials they represent are also linearly independent.

Through the implementation of Rabin encryption algorithm[16] and Assumption 1, we know that k_1 cannot be obtained according to polynomial R_1 in equation (2) if s is equal to at least 3 when modulo n(n = p*q, p,q are both large private primes). If no special instruction exists, the following R_i in equation (2) are all calculated modulo n. Therefore, we can use R_1 as the public encryption key and k_1 as the decryption key to construct the fully homomorphic encryption we want. Then, we can get the following algorithm to represent $trap_1 \times m$:

$$\begin{aligned}&Enc(ek = trap_1 = R_1, m) = trap_1 \times m + trap_0 = R_1 * m + trap_0\\&Dec(sk = k_1, c) = R_1(k_1) * m + 0 = m\end{aligned} \tag{3}$$

Since the existence of $trap_1(R_1)$ and $trap_0$, we find that ciphertext obtained by equation (3) is fully homomorphic. Because for users who have decryption key k_1, $trap_1(k_1) = R_1(k_1)$ is equal to 1, and $trap_0(k_1)$ is equal to 0. So during decryption, all R_1 and $trap_0$ elements of the operation between ciphertexts can be removed, and the result of m' can be obtained from the final ciphertext. For example, $c' = ((c_1+c_2)*c_3+c_4) = (trap_1)^2 \times (m_1+m_2)*m_3+trap_1 \times m_4+trap_0 = R_1^2 * (m_1+m_2)*m_3+R_1*m_4+trap_0$, and then $Dec(sk = k_1) = R_1(k_1)^2*(m_1+m_2)*m_3+R_1(k_1)*m_4+0 = 1^2*(m_1+m_2)*m_3+1*m_4 = (m_1+m_2)*m_3+m_4)$. The problem now is how to construct a safe and reasonable fully homomorphic encryption scheme through these contents.

Through Assumption 1, we already know that a high-degree equation, i.e. quadratic polynomial is unsolvable modulo n(n=p*q, p, q are both private large primes). It shows that it is impossible to find a root of polynomials of degree seven modulo n(n=p*q, p, q are all both large primes). We say that our constructions of the FHE scheme covered in this paper are all computed in $Z_n[x]$(n=p*q, p, q are both private large primes). The calculation of ciphertexts is the calculation of polynomials with a degree of at least seven. It means that s is at least eight and there are at least $R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8$ in equation (3)(a portion is used in our scheme). The ciphertext obtained using $trap_1 \times m$ is often represented by a polynomial of degree seven, such as $c = a * k^7 + bk^6 + ck^5 + dk^4 + ek^3 + fk_2 + gk + h = [a, b, c, d, e, f, g, h]$.

In the following sections, we give a construction of the sing-key polynomial-based fully homomorphic encryption(P-FHE), Construction 1, and we also give the corresponding security proof. In this encryption, different encryption parties always use the same encryption public parameters to encrypt ciphertext, which is suitable for large public domains such as voting systems. Then, we give a construction of multi-key polynomial-based fully homomorphic encryption(P-FHEs) Construction 2. Different from Construction 1, this encryption scheme allows different encryption parties to have their own unique encryption keys for encryption. Finally, to maximize efficiency, we proposed a method for fixing the length of ciphertext by assigning a safe octave polynomial to restrict all computations to polynomials of degree seven. We say these two encryption constructions are all implementations of our $trap_1 \times m$ encryption scheme since their encryption and decryption processes all rely on $trap_1 \times m + trap_0$.

3 Single-Key Polynomial-based Fully Homomorphic Encryption

Through the introduction in the previous part, we know that ciphertext constructed by equation (3) satisfies the property of full homomorphism. We have known that a quadratic polynomial is unsolvable when modulo n(n=p*q, p, q are both private large primes), and the same is true for a polynomial of higher-degree. So we use polynomials of degree seven to build a general fully homomorphic encryption scheme $trap_1 \times m$(also called $trap_1 \times m + trap_0$). We donate the polynomials in equation 2 except R_1 as the components of $trap_0$ because the values of these polynomials are all 0 after the secret information k_1 is brought in. In addition, because $trap_1 - 1 = trap_0$, we also use $R_1 - 1$ as a composition of $trap_0$, where $R_1 - 1$ is expressed as the constant term of R_1 minus 1. It should be noted that the composition of $trap_0$ has the same properties as $trap_0$, and the composition of $trap_0$ can also exist as $trap_0$.

3.1 Construction of P-FHE

Construction 1 *Let P-FHE = (Setup, Encrypt, Evaluate, Decrypt) be a highly efficient Single-Key Polynomial-based Fully Homomorphic Encryption. We construct P-FHE as follows:*

Setup(1^λ): On input security parameter λ and set s=8, the setup algorithm does following process:

1.Generate large prime numbers p, q according to λ and compute n=p*q.

2.Randomly select k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, $\hat{r_1}$, $\hat{r_2}$, $\hat{r_3}$, $\hat{r_4}$, $\hat{r_5}$, $\hat{r_6}$, $\hat{r_7}$, $\hat{r_8}$ $\leftarrow Z_n$.

3.Use k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, to generate R_1, R_2, R_3, R_4, R_5, R_6, according to equation (2) and check whether R_1, R_2, R_3, R_4, R_5, R_6 are linear independent. If not, come back to process 2, else continue.

4.Use R_2, R_3, R_4, R_5, R_6, $\hat{r_1}$, $\hat{r_2}$, $\hat{r_3}$, $\hat{r_4}$, $\hat{r_5}$, $\hat{r_6}$, $\hat{r_7}$, $\hat{r_8}$ and p, q to generate $R'_1 = R_2{*}p{+}R_6{*}\hat{r_1}$, $R'_2 = R_2{*}q{+}R_6{*}\hat{r_2}$, $R'_3 = R_3{*}p{+}R_6{*}\hat{r_3}$, $R'_4 = R_3{*}q{+}R_6{*}\hat{r_4}$, $R'_5 = R_4{*}q{+}R_6{*}\hat{r_5}$, $R'_6 = R_4{*}q{+}R_6{*}\hat{r_6}$, $R'_7 = R_5{*}q{+}R_6{*}\hat{r_7}$, $R'_8 = R_5{*}q{+}R_6{*}\hat{r_8}$.

Output sk= msk= k_1, mpk=$\{n, R_1, R'_1, R'_2, R'_3, R'_4, R'_5, R'_6, R'_7, R'_8\}$.

Encrypt(m, ek=mpk): On input message m and ek=mpk, sample random element $\vec{r} = [r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8 \leftarrow Z_n]$ and output:

$c = trap_1 \times m + trap_0 \times \vec{r} = R_1 * m + (R_1 - 1) * r_0 + \sum_{i=1}^{8} R'_i * r_i$ mod n

Evaluate(mpk, $\mathcal{C}$, $(c_1, c_2, ...)$): On input public key mpk, an algorithm $\mathcal{C}$ that supports multiply and adds operations(calculations of polynomial multiplication and addition), a set of input ciphertext $(c_1, c_2, ...)$ and then output:

$c' = \mathcal{C}(c_1, c_2, ...)$ mod n, which is equal to Encrypt($\mathcal{C}(m_1, m_2, ...), mpk$)

Decrypt(c', sk=k_1): On input secret key sk, and a ciphertext c'=$\mathcal{C}$(c_1, c_2,...), output:

$m' = c'(sk) = c'(k_1) = 1 * \mathcal{C}(m_1, m_2, ...) = \mathcal{C}(m_1, m_2, ...)$

Because $R_1(k_1)$ is equal to 1 but $(R_1 - 1)(k_1)$, $R_2(k_1)$, $R_3(k_1)$, $R_4(k_1)$, $R_5(k_1)$ and $R_6(k_1)$ are all equal to 0.

Correctness. We show that the correctness of the above fully homomorphic encryption holds. Ciphertext in this encryption is encrypted in the form of c=$trap_1 \times m + trap_0 \times \vec{r} = trap_1 \times m + \sum_{i=0}^{8} trap_{0i} \times r_i = R_1 * m + (R_1 - 1) * r_0 + R'_1 * r_1 + R'_2 * r_2 + R'_3 * r_3 + R'_4 * r_4 + R'_5 * r_5 + R'_6 * r_6 + R'_7 * r_7 + R'_8 * r_8$(we express R_1 as $trap_1$, $(R_1 - 1)$ as $trap_{0_0}$ and R'_j as $trap_{0_j}(1 \le j \le 8)$ respectively). When decrypting, $trap_1$=R_1 $\rightarrow$1, $trap_0 = \sum_{i=0}^{8} trap_{0i} \times r_i \rightarrow \sum_{i=0}^{8} 0 * r_i = 0$ can be made to get message m=m+0=m from ciphertext c.

According to equation (2) and equation (3), we can know that the ciphertext of the entire encryption process exists in the form of $trap_1 \times m + trap_0$. Because users without k_1 have no secret key k_1, they can only get a ciphertext that participated in the operation but get nothing about message m. However, for a user who has the decryption key k_1, R_1 in the ciphertext is equal to 1, and $(R_1 - 1)$ and $R'_i(1 \le i \le 8)$ are all equal to 0. He can easily use the key to remove $R_1, (R_1 - 1), R'_i$ and other polynomials in the final ciphertext to obtain the calculated plaintext m'. For example, we say for $c' = (c_1 + c_2) * c_3 + c_4$, we have $c' = R_1^2 * (m_1 + m_2) * m_3 + R_1 * m_4 + trap_0$. With the help of sk=$k_1$ we can get $Dec(sk = k_1) = R_1(k_1)^2 * (m_1 + m_2) * m_3 + R_1(k_1) * m_4 + trap_0(k_1) = 1^2 * (m_1 + m_2) * m_3 + 1 * m_4 + 0 = (m_1 + m_2) * m_3 + m_4)$. But for others who have no sk=k_1, since the polynomials of ciphertext cannot be removed, they cannot successfully decrypt. Therefore, in the process of calculating different ciphertexts, it still satisfies the form of $trap_1 \times m + trap_0$ to ensure the correctness of the decryption process.

3.2 Fully Homomorphic Operations of Ciphertexts

Let's review the ciphertext generated by the encryption algorithm. During encrypting, the ciphertext is generated in the form of $c = R_1 * m + (R_1 - 1) * r_0 + R'_1 * r_1 + R'_2 * r_2 + R'_3 * r_3 + R'_4 * r_4 + R'_5 * r_5 + R'_6 * r_6 + R'_7 * r_7 + R'_8 * r_8$, which is equal to $trap_1 \times m + trap_0$ since $R_1(k_1) = 1$ and other polynomials is equal to 0 during decryption. When two ciphertexts are added, the coefficients of the same polynomial term between the ciphertexts c_0 and c_1 will be combined. The newly generated ciphertext is $c' = R_1 * (m_1 + m_2) + (R_1 - 1) * (r_{00} + r_{10}) + \sum_{i=1}^{8} R'_i(r_{0i} + r_{1i})$, which is still equal to $c = trap_1 \times (m_0 + m_1) + trap_0 = Encrypt(m_0 + m_1, ek = mpk)$. Therefore, the additive homomorphic property of computation between ciphertexts is guaranteed.

When performing multiplication between ciphertexts, we have $trap_0 \times trap_1 = trap_0, trap_0 \times trap_0 = trap_0, trap_1 \times trap_1 = trap_1$ since $trap_1(k_1) = 1, trap_0(k_1) = 0$. Suppose there are two ciphertexts c_0, c_1 for multiplication calculation, where $c_0 = trap_1 \times m_0 + trap_0$, $c_1 = trap_1 \times m_1 + trap_0$($trap_1$ and $trap_0$ may be different between different ciphertexts). We have $c_0 * c_1 = {trap_1}^2 \times m_1 * m_2 + trap_0 = trap_1 \times m_1 * m_2 + trap_0$, where $trap_1$ and $trap_0$ have changed. Therefore, we say that our encryption scheme satisfies the fully homomorphic property.

3.3 IND-CPA Security

We indicate that our encryption algorithm satisfies IND-CPA security. To prove this, we only need to prove the security of the Encrypt algorithm in Construction 1 since the Evaluate algorithm can be expressed as a computation of ciphertexts generated by Encrypt algorithm.

Assuming that an adversary wants to recover the plaintext information m, he needs to solve the following equation:

$$\begin{aligned} c =& R_1 * m + (R_1 - 1) * r_0 + R'_1 * r_1 + R'_2 * r_2 + R'_3 * r_3 + R'_4 * r_4 + \\ & R'_5 * r_5 + R'_6 * r_6 + R'_7 * r_7 + R'_8 * r_8 \end{aligned}$$

We know that the purpose of $(R_1 - 1)$, $R'_j (1 \leq j \leq 8)$ is to introduce more random unknowns in the encryption process, so as to protect message m from being leaked. In this way, the number of unknowns is s+2(s=8 here) and it is greater than s for the attacker in the process of encryption, where unknowns include $(R_1 - 1)$, coefficients of R'_j, and m. Because R'_j cannot be mutually calculated(prove later), any of them cannot be represented by other polynomials including $R_1 - 1$. So it is a solving equations problem with 10 unknowns and 8 equations if an adversary wants to solve m from c. Normally, the adversary cannot know any of the unknowns of $m, r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8$, and then he cannot solve the above equation to get m. But in the IND-CPA game, the attacker can choose m as he wishes. In this case, the attacker can know an unknown(m) in the ciphertext equations. However, there are still 9 (s+1) unknowns that make it impossible for the attacker to make further attacks.

Definition 1 ***IND-CPA.*** *Consider the following game between a challenger C and a stateful adversary A.*

Game Definition	**Oracle Definition**
1.(mpk, msk)$\leftarrow$**Setup**(1^λ);	$O_G(*)$:
2.$(m_0, m_1)\leftarrow A^{O_G(*),O_E(*)}(mpk)$;	1.Output $k \leftarrow Z_n$;
3.$b \leftarrow \{0,1\}$;	
4.$c \leftarrow \mathbf{Enc}(mpk, m_b)$;	$O_E(*)$:
5.$b' \leftarrow A^{O_G(*),O_E(*)}(c)$;	1.Output $c \leftarrow \mathbf{Enc}(mpk, m)$;

We say that A wins IND-CPA game if $b = b'$, $|m_0| = |m_1|$ and the following holds:

For all queries to $O_G(*)$ with k, it holds that:

$$k \notin \textbf{\textit{Setup}}$$

This definition restricts the adversary from obtaining the k used in the Setup algorithm, thus ensuring the security of the encryption process.

We state that the encryption proposed above is secure if for any PPT adversary A, it holds that:

$$Pr[adv^A]= |Pr[A\ wins\ the\ IND\text{-}CPA\ Game] - \tfrac{1}{2}| \le negl(\kappa)$$

Proof. We define the general IND-CPA adversary-challenger game. The challenger C initializes the encryption system in Construction 1. Then he sends the public parameters of the system to adversary A. We assume that A is polynomially conditional, and he can choose the plaintext pair (m_0, m_1) to be encrypted at will. At the same time, A also has access to encryption oracle and key oracle.

During the process of Encrypt algorithm, the composition of ciphertext c is $R_1 * m + (R_1 - 1) * r_0 + \sum_{i=1}^{8} R_i' * r_i$. Suppose adversary A chooses m_0, m_1 and sends them to challenger C. C generates different ciphertexts c_0, c_1 satisfying $c_0 = R_1 * m_0 + (R_1 - 1) * r_{00} + \sum_{i=1}^{8} R_i' * r_{0i}$, $c_1 = R_1 * m_1 + (R_1 - 1) * r_{10} + \sum_{i=1}^{8} R_i' * r_{1i}$. Then C randomly selects $b \leftarrow 0, 1$ and gives c_b to A. Since $R_1 - 1$, $R_i'(1 \le i \le 8)$ cannot be eliminated as A doesn't know k_1 and different unknowns in ciphertext cannot be converted to each other, the random factor $\vec{r_b}$ in c_b cannot be eliminated. Therefore, A cannot distinguish ciphertext c_b from c_0 and c_1 and our scheme satisfies the IND-CPA security.

3.4 Security

To prove the security of Construction 1, we give the following definitions:

Theorem 4. *Suppose we have two independent polynomials P_1, P_2 and two large primes p, q. It is impossible to solve P_1, p or q only given $P_1 * p + P_2 * r_1$ and $P_1 * q + P_2 * r_2$.*

Proof. Suppose we solve P_1 or p by $P_1 * p + P_2 * r_1$, we need to get the value of $P_2 * r_1$ in advance. But we cannot get $P_2 * r_1$ in polynomial time when we only know $P_1 * p + P_2 * r_1$. The same is true for $P_1 * q + P_2 * r_2$. If we want to solve P_1, p or q by $P_1 * p + P_2 * r_1$ and $P_1 * q + P_2 * r_2$ together, we need to know at least the value of P_2. We strictly restrict the condition that P_2 will not be leaked, so P_1, p or q cannot be solved. Therefore, Theorem 4 is correct.

Proposition 1 *According to polynomials $R_1, R_i'(1 \le i \le 8)$, sk=k_1 cannot be solved. In other words, users other than **Setup** cannot get sk=k_1.*

Proposition 2 *If a polynomial R_i cannot be solved for point (k_i, 1) corresponding to k_i, then the new point corresponding to k_i cannot be solved after R_i is multiplied or added by a random element r. In other words, given an unsolvable equation, it is still unsolvable after multiplying or adding a certain value. It means that the equation is still unsolvable after calculations such as addition and multiplication.*

Proposition 3 *If a polynomial R_i cannot solve the point corresponding to k_i, then the point corresponding to k_i cannot be solved after multiplying or adding an unsolvable polynomial R_j. Obtaining information about k_i points of two polynomials before the calculation is impossible.*

Proposition 4 *Given polynomials $R_1, R'_1, R'_2, R'_3, R'_4, R'_5, R'_6, R'_7, R'_8$ in Construction 1, we cannot solve for any of R_2, R_3, R_4, R_5, R_6.*

Proposition 5 *In Construction 1, $R_1, R_2, R_3, R_4, R_5, R_6$ are linearly independent, and cannot calculate each other, so is it to $R_1 - 1, R'_i(1 \leq i \leq 8)$. That means we cannot use a part of them to calculate others, such as $R'_2 \neq R'_1 * a + (R_1 - 1) * b$ and $R'_3 \neq R'_2 * a + R'_1 * b + (R_1 - 1) * c + R'_4 * d$. For example, although R'_1 and R'_2 are all calculated by the same polynomial, R'_1 and R'_2 are linearly independent since R_6 is concealed.*

Proof. According to the given theorems, we prove that the above propositions are correct, and correspondingly prove that our scheme is safe and efficient.

First, we prove that Proposition 1 is correct. We say that if an adversary wants to crack $sk = k_1$, he must attack from $R_1, R'_1, R'_2, R'_3, R'_4, R'_5, R'_6, R'_7, R'_8$. We think it is impossible because the length of n we set is generally at least 2048 bits and longer.

Through Assumption 1, we know that it is difficult to crack a polynomial of the second degree or more in the case of modulo n. So we say it is impossible to get sk by cracking R_1, such as letting $R_1 = 1$ get sk=k_1. The same is true for $R'_i(1 \leq i \leq 8)$ respectively. Of course, if we want to combine $R_1, R'_1, R'_2, R'_3, R'_4, R'_5, R'_6, R'_7, R'_8$ to solve k_1, we say it is impossible. We express that the combination of $R_1, R'_1, R'_2, R'_3, R'_4, R'_5, R'_6, R'_7, R'_8$ is equivalent to the combination of $R_1, R_2, R_3, R_4, R_5, R_6$, because these polynomials are all Calculated by $R_1, R_2, R_3, R_4, R_5, R_6$. In Construction 1, because s is equal to 8, R_1, R_2, R_3, and other polynomials are all polynomials of degree seven. According to the polynomial intersection problem, we can get a quadratic polynomial R satisfying $R(k_1) = 0$ by combining the above polynomials, and we know it is unsolvable according to 1. Therefore, we say that sk=k_1 is unsolvable, meaning Proposition 1 is correct. But it needs to be noted that if we give a new polynomial R_7 satisfying equation (2) in Construction 1, then k_1 can be solved by combining existing polynomials because the first-degree polynomial can be solved. This also means that we need to control the number of given independent polynomials to be less than the degree of polynomials of Construction 1 to ensure the scheme we built is safe.

Then we prove that a polynomial multiplied by a number or adding a number is still unsolvable, meaning Proposition 2 is correct. We know that for a polynomial, no matter how many times it is expanded or numbers added, the solution to its equation remains unchanged. So if the polynomial in Proposition 1 cannot be solved, Proposition 2 is correct.

Likewise, we prove that Proposition 3 is correct. Suppose there are three polynomials P_1, P_2, P_3 satisfying $P_1 = P_2 * P_3$, and P_2, P_3 are unsolvable polynomials. If P_1 is solvable, then at least one of P_2 and P_3 participating in the calculation is solvable according to the principle of polynomial calculation. But P_2, P_3 are all unsolvable polynomials, so P_1 is also an unsolvable polynomial. The main purpose of this proposition is to prove that the calculation between ciphertexts in the encryption process is legal and safe. Because ciphertexts in our scheme are all in the form of polynomials, we need to ensure the calculation security of polynomial ciphertext.

For Proposition 4, we have known that R_1, R_2, R_3, R_4, R_5, R_6 are linearly independent according to Theorem 3. Then according to Theorem 4, we know that R_2 cannot be obtained according to R_1' and R_2'. If we want to combine them with other polynomials such as R_1, R_3', and so on, we know it is impossible because the newly introduced polynomials are independent of them. The purpose of this proposition is to prevent the adversary from obtaining plaintext by calculating equations after obtaining R_2, R_3, R_4, R_5, R_6. Let's look at the form of ciphertext $c = R_1 * m + (R_1 - 1) * r_0 + R_1' * r_1 + R_2' * r_2 + R_3' * r_3 + R_4' * r_4 + \ldots$. It including 10 unknowns, and they are m, r_0, r_1, r_2, r_3, ..., r_8 respectively. Because the calculated ciphertext is related to s, if we solve polynomials through equation problems, then we can get 8 equations since s=8. However, we have 10 unknowns in the ciphertext include m. We mean that the adversary cannot solve the equations to obtain m since they cannot reduce the number of unknowns to less than s. But if R_2, R_3, R_4, R_5 can be calculated, then R_1', R_2', R_3', R_4' and other polynomials can be replaced by $R_2.R_3, R_4, R_5, R_6$, and the number of unknowns will be reduced so that equations can be solved to get message m finally.

For Proposition 5, we first know that R_1, R_2, R_3, R_4, R_5, R_6 are linearly independent according to Theorem 3. For R_1' and R_2', we cannot find a such that $R_1' = R_2' * a$ because R_2 and R_6 are independent and we don't know R_2, R_6. In addition, $R_i'(1 \leq i \leq 8)$ derived from different polynomials is also independent of each other because R_2, R_3, R_4, R_5, R_6 are not related to each other. For example, R_1' and R_3' are independent of each other because R_2, R_3, and R_6 are independent of each other. Because $R_1 - 1(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8) = [0, -1, -1, -1, -1, -1, -1, -1]$, $R_1 - 1$, R_2, R_3, R_4, R_5, R_6 are also linearly independent. Therefore, $R_1 - 1, R_i'(1 \leq i \leq 8)$ are linearly independent and cannot calculate by each other. The purpose of Proposition 5 is to prevent $R_1 - 1$, $R_i'(1 \leq)$ from being interchangeable, so that the attacker cannot reduce the number of unknowns to solve polynomial equations of ciphertext to obtain m. In other words, if any of $R_1 - 1, R_1', R_2', R_3', R_4', R_5', R_6', R_7', R_8'$ can be represented by other polynomials, one or more unknowns in the ciphertext can be ignored for the attacker. In this case, the plaintext information m contained in the ciphertext may be deciphered by the attacker. In addition, because $R_1, R_1 - 1, R_2, R_3, R_4, R_5, R_6$ are all independent of each other, different unknowns in the ciphertext cannot be converted to each other. Therefore, if the attacker wants to crack the plaintext information m through the ciphertext, it is impossible because the equations are unsolvable if some of the unknowns cannot be obtained (knowing the value of some unknowns can reduce the number of unknowns).

In summary, the above theorems and propositions are all satisfied in Construction 1 proposed in this section. For example, Theorems 1, 2 and Propositions 1, 2, 3 guarantee the security of the encryption system key, so that the attacker cannot crack the decryption key k_1 through public parameters or other information. Theorems 3, 4 and Propositions 4, 5 guarantee the security of the encryption process. Thus, combined with the above security definition, we prove that Construction 1 is secure.

4 Multi-Key Polynomial-based Fully Homomorphic Encryption

We have presented homomorphic encryption based on the same encryption key and given the corresponding security proof. But considering real-life application scenarios, we need to consider the use of different encryption keys for different users as much as possible[19,20,21,22]. We hope that ciphertexts generated by different users using their keys can perform various operations and still get correct decryption information. Therefore, we improve Construction 1 to get a new construction, so that each user has a different encryption key ek, and the ciphertexts of different users can participate in the operation together.

4.1 Construction of P-FHEs

Construction 2 *Let P-FHE = (Setup, Encrypt, Evaluate, Decrypt) be a highly efficient Polynomial-based Fully Homomorphic Encryption in Construction 1. We construct Polynomial-based Fully Homomorphic Encryption with different ek, P-FHEs(Setup, Encrypt, Key-generate, Evaluate, Decrypt), as follows (Note that s=8 here, R_i $(1 \leq i \leq 6)$ and ciphertext exist as polynomials with the degree of 7):*

Setup(1^λ): On input security parameter λ and set s=8, the setup algorithm does following process:

1.Generate large prime numbers p, q according to λ and compute n=p*q.

2.Randomly select k_1, k_2, k_3, k_4, k_5, k_6, k_7, $k_8 \leftarrow Z_n$.

3.Use k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, to generate R_1, R_2, R_3, R_4, R_5, R_6, according to equation (2) and check whether R_1, R_2, R_3, R_4, R_5, R_6 are linear independent. If not, come back to process 2, else continue.

4.Use R_2, R_3, R_4, R_5, R_6, and p, q to generate $\hat{R_1} = R_2 * p$, $\hat{R_2} = R_2 * q$, $\hat{R_3} = R_3 * p$, $\hat{R_4} = R_3 * q$, $\hat{R_5} = R_4 * p$, $\hat{R_6} = R_4 * q$, $\hat{R_7} = R_5 * p$, $\hat{R_8} = R_5 * q$.

Output sk=k_1, msk=$\{R_6, \hat{R_1}, \hat{R_2}, \hat{R_3}, \hat{R_4}, \hat{R_5}, \hat{R_6}, \hat{R_7}, \hat{R_8}\}$, mpk=$\{n, R_1\}$.

KeyGen(msk, mpk): On input master secret key msk, master public key mpk, sample random elements $\hat{r_1}$, $\hat{r_2}$, $\hat{r_3}$, $\hat{r_4}$, $\hat{r_5}$, $\hat{r_6}$, $\hat{r_7}$, $\hat{r_8} \leftarrow Z_n$, output $ek = \{R'_1, R'_2, R'_3, R'_4, R'_5, R'_6, R'_7, R'_8\}$:

$R'_1 = \hat{R_1} + R_6 * \hat{r_1}$, $R'_2 = \hat{R_2} + R_6 * \hat{r_2}$, $R'_3 = \hat{R_3} + R_6 * \hat{r_3}$, $R'_4 = \hat{R_4} + R_6 * \hat{r_4}$, $R'_5 = \hat{R_5} + R_6 * \hat{r_5}$, $R'_6 = \hat{R_6} + R_6 * \hat{r_6}$, $R'_7 = \hat{R_7} + R_6 * \hat{r_7}$, $R'_8 = \hat{R_8} + R_8 * \hat{r_8}$

Encrypt(m, ek, mpk): On input message m, encryption key ek, master public key mpk, sample random element $\vec{r} = [r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8 \leftarrow Z_n]$ and output:

$c = trap_1 \times m + trap_0 \times \vec{r} = R_1 * m + (R_1 - 1) * r_0 + \sum_{i=1}^{8} R'_i * r_i$ mod n

Evaluate(mpk, $\mathcal{C}$, $(c_1, c_2, ...)$): On input master public key mpk, an algorithm $\mathcal{C}$ that supports multiply and adds operations(calculations of polynomial multiplication and addition), a set of input ciphertext $(c_1, c_2, ...)$ and then output:

$c' = \mathcal{C}(c_1, c_2, ...)$ mod n, which is equal to Encrypt($\mathcal{C}(m_1, m_2, ...), ek, mpk$)

Decrypt(c', sk=k_1, mpk): On input secret key sk, a cipher-text c'=$\mathcal{C}$(c_1, c_2,...), and master public key mpk, output:

$m' = c'(sk) = c'(k_1) = 1 * \mathcal{C}(m_1, m_2, ...) = \mathcal{C}(m_1, m_2, ...)$

Because $R_1(k_1)$ is equal to 1 but $(R_1 - 1)(k_1)$, $R_2(k_1)$, $R_3(k_1)$, $R_4(k_1)$, $R_5(k_1)$ and $R_6(k_1)$ are all equal to 0.

4.2 Correctness and Security

Correctness: We say that the correctness of the above encryption is established. We see that Construction 2 is a variant of Construction 1. We just convert the random element R'_i(1≤ i ≤8) in the public parameter of Construction 1 into private elements and none of $trap_1$ and $trap_0$ elements have changed. After bringing $sk = k_1$, $R_1 = 1$, $R_2 = R_3 = R_4 = R_5 = R_6 = 0$ is still satisfied. Therefore, the encryption and decryption process is still correct.

Security. Compared with Construction 1, Construction 2 converts some public parameters into private parameters. However, the encryption algorithm $c = R_1 * m + (R_1 - 1) * r_0 + \sum_{i=1}^{8} R'_i * r_i$ is still unchanged during encryption. Although everyone gets a different encryption key ek, they face the same public parameter mpk=$\{R_1, n\}$, so they can generate ciphertexts in the same homomorphic environment.

Because different user gets different ek, it is more difficult for an attacker to use ciphertext to crack the user's plaintext. For example, there is a user encrypts m by his ek to generate a ciphertext c. If an attacker wants to crack m through c, he can only try to crack ciphertext c through his ek. However, we know that the encryption keys of different users are different, so the attacker must first find a way to obtain the encryption key of the encrypting party to crack m. We say that the difficulty of cracking is more difficult than that of Construction 1. In Construction 1, ek is directly told to the attacker, and everyone uses the same encryption key to encrypt. But in Construction 2, the attacker has less information because he doesn't even know the honest user's encryption key. Therefore, compared with Construction 1, Construction 2 proposed in this part is more secure.

5 Making Ciphertexts of Our Scheme Constant Size

We have given the construction of polynomial-based fully homomorphic encryption with single-key and multi-key, and the ciphertexts of them are all based on polynomials of degree seven. There is still a serious problem that we need to pay attention to. Because the calculation between ciphertexts is polynomials calculation, the length of new ciphertext changes as multiplication occurs. For example, if two ciphertexts are multiplied, the length of the calculated ciphertext is the sum of two ciphertexts minus 1. However, if two ciphertexts are added, the resulting ciphertext length is the longest of them. In this way, an attacker can easily infer whether there is a multiplication calculation involved and the calculation efficiency between ciphertexts will also decrease with the multiplication. Therefore, in this section, we will give a method to improve the fully homomorphic encryption scheme given above based on the modulus calculation of polynomials to make the length of ciphertexts fixed.

5.1 Overview

We know that the encryption constructions given above are constructed on basis elements $R_1, R_2, R_3, R_4, R_5, R_6$, and $s = 8$ according to Equation 2. Therefore, ciphertexts generated by Encrypt are all ciphertexts represented by polynomials of degree seven. However, when different ciphertexts are multiplied, the new ciphertext appears as a polynomial with a higher degree. For example, multiplying two ciphertexts in the form of polynomials of degree seven will produce a ciphertext in the form of polynomial of degree 14.

In order to not limit the calculation of ciphertexts and make encryption more efficient, we need to keep the calculation result of ciphertexts to polynomials of degree seven and keep the length of ciphertext at $s * |n|$(s=8 here). We use a high-degree polynomial to limit the calculation of low-degree polynomials so that the result of the calculation of low-degree polynomials can be kept at a low level. Therefore, we can use a octave polynomial P satisfying $P(k_1) = 0$ to limit the generation of high-degree polynomials generated between ciphertexts during evaluation. However, when we introduce P, we need to ensure that the newly introduced polynomial and random element satisfy the theorems and propositions proposed before. Therefore, we give the idea of how to improve the fully homomorphic encryption constructions given above(P-FHE and P-FHEs) based on the modulus calculation of polynomials to make the length of ciphertexts fixed.

To ensure the normal progress of encryption, we cannot destroy the security of the original scheme when introducing a new polynomial P. For example, P must remain non-independent from the public polynomials that have been used, and not affect the theorems and propositions that the encryption scheme satisfies. Therefore, the simplest improvement is using $trap_0$, $(R_1 - 1)$, used in the encryption process, and combining it with the basic polynomial k to construct P. We give a typical encryption with the constant size of ciphertext according to Construction 1.

We already know that Construction 2 is a variant of Construction 1, so we only use Construction 1 as an example to explain the method of ciphertext fixing. But we declare that the method for improving Construction 1 is also valid for Construction 2 and we can construct a new construction with constant-size ciphertexts in the same way.

5.2 Constant-Size Ciphertexts of Polynomial-based Fully Homomorphic Encryption

To make the ciphertexts of our encryption constant size, we use R_1 in Construction 1 and basic polynomial k to construct a octave polynomial $P(k) = k*(R_1-1)$ that satisfies $P(k_1)$=0(mod n) during the **setup** process. In this way, we can use the high-order terms of P to convert terms with the degree over s-1 generated by ciphertexts into low-order terms during polynomial multiplication.

Based on Construction 1 and P, we can limit the computation of ciphertexts to the polynomials of degree seven(shown in equation 4). Because in the process of ciphertext calculation, once the result of calculations exceeds the polynomial of degree seven, we can call $P(k_1)$=0 to convert polynomials of more than the degree of seven into a polynomial of less than seven(including seven). For example, multiplying two ciphertexts of polynomials of degree seven can generate a new polynomial of degree 14, and we can convert coefficients of high-degree over seven into coefficients of degree less than or including seven. With the help of $P = k*(R_1-1)$, we give the following equations.

$$\begin{aligned}
&Let\\
&\qquad R_1 \equiv k^7 + bk^6 + ck^5 + dk^4 + ek^3 + fk^2 + gk + h + 1(mod \quad n)\\
&\qquad P = k*(R_1-1)\\
&\qquad\quad = k^8 + bk^7 + ck^6 + dk^5 + ek^4 + fk^3 + gk^2 + hk\\
&\qquad R_2 = a'k^7 + b'k^6 + c'k^5 + d'k^4 + e'k^3 + f'k^2 + g'k + h'\\
&\qquad R_3 = \hat{a}k^7 + \hat{b}k^6 + \hat{c}k^5 + \hat{d}k^4 + \hat{e}k^3 + \hat{f}k^2 + \hat{g}k + \hat{h}\\
&Then, \quad we \quad have\\
&\qquad k^8 = -bk^7 - ck^6 - dk^5 - ek^4 - fk^3 - gk^2 - hk\\
&\qquad R_1 * R_1 = R_1 + (R_1-1)*R_1 = R_1 + (R_1-1)*h\\
&\qquad R_1 * R_2 = R_2 + (R_1-1)*R_2 = R_2 + (R_1-1)*h'\\
&\qquad R_1 * R_3 = R_3 + (R_1-1)*R_3 = R_3 + (R_1-1)*\hat{h}\\
&\qquad R_2 * R_3 = else \quad but \quad R_2 * R_3(k_1) = 0
\end{aligned} \tag{4}$$

Through appropriate construction of P such as the second half of equation (4), we can get the formula $R_1^l = R_1 + C*(R_1-1) = trap_1 + trap_0 = trap_1 (l \geq 2$, C is a number decided by l and the constant term of R_1), making $R_1^l(k_1) = R_1(k_1) = 1$. Other multiplication operations between polynomials contain $trap_0$ output 0. For easier explanation, we take quadratic polynomial ciphertext and cubic polynomial $P(k)$ as an example. Just like the calculation process shown in equation (4), we give the following example:

Suppose $R_1 = k^2 - 3k + 3, R_1 - 1 = (k-1)(k-2) = k^2 - 3k + 2$, and $sk = k_1 = 1$.

We make $P(k) = k * (R_1 - 1) = k^3 - 3k^2 + 2k$ and we have:

$$\begin{aligned} R_1^2 &= (k^2 - 3k + 3)^2 \\ &= k^4 - 6k^3 + 15k^2 - 18k + 9 \end{aligned}$$

According to $P(k_1) = 0$, we have $k^3 = 3k^2 - 2k$. Then, we can get the following result:

$$\begin{aligned} R_1^2 &= 4k^2 - 12k + 9 \\ &= (k^2 - 3k + 3) + (3k^2 - 9k + 6) \\ &= (k^2 - 3k + 3) + 3[(k^2 - 3k + 3) - \\ &= R_1 + 3(R_1 - 1) \end{aligned}$$

The same is true if calculations of higher-order polynomials are involved and so it is to polynomials of degree seven and octave polynomials. In this way, we can limit the result of computation between ciphertexts to polynomials of seven. We know $R_1 = 1$ is the problem of solving a polynomial in Z_n(k), while $P(k) = 0$ is equal to $k * (R_1) = 0$. Therefore, we say that it has the same difficulty as solving k_1 by $R_1 = 1$. Then, Assumption 1 tells us that it is difficult to solve a polynomial of at least second-degree modulo n. The length of ciphertext in our fully homomorphic operation is the size of a polynomial and our security parameter n can be set long enough such as 2048bit, 4096bit, and so on. Therefore, the safety of $P(k)$ is guaranteed and we construct a constant size polynomial-based fully homomorphic encryption, Construction 3 as follows.

Construction 3 *Let P-FHE = (Setup, Encrypt, Evaluate, Decrypt) is a highly efficient Polynomial-based Fully Homomorphic Encryption in Construction 1. We construct constant-size P-FHE=(Setup, Enc, Evaluate, Dec) as follows:*

Setup(1^λ): On input security parameter λ and set s=8, the setup algorithm does following process:

1.Generate large prime numbers p, q according to λ and compute n=p*q.

2.Randomly select k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, $\hat{r_1}$, $\hat{r_2}$, $\hat{r_3}$, $\hat{r_4}$, $\hat{r_5}$, $\hat{r_6}$, $\hat{r_7}$, $\hat{r_8}$ $\leftarrow Z_n$.

3.Use k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, to generate R_1, R_2, R_3, R_4, R_5, R_6, according to equation (2) and check whether R_1, R_2, R_3, R_4, R_5, R_6 are linear independent. If not, come back to process 2, else continue.

4.Use R_2, R_3, R_4, R_5, R_6, $\hat{r_1}$, $\hat{r_2}$, $\hat{r_3}$, $\hat{r_4}$, $\hat{r_5}$, $\hat{r_6}$, $\hat{r_7}$, $\hat{r_8}$ and p, q to generate $R'_1 = R_2{*}p{+}R_6{*}\hat{r_1}$, $R'_2 = R_2{*}q{+}R_6{*}\hat{r_2}$, $R'_3 = R_3{*}p{+}R_6{*}\hat{r_3}$, $R'_4 = R_3{*}q{+}R_6{*}\hat{r_4}$, $R'_5 = R_4{*}q{+}R_6{*}\hat{r_5}$, $R'_6 = R_4{*}q{+}R_6{*}\hat{r_6}$, $R'_7 = R_5{*}q{+}R_6{*}\hat{r_7}$, $R'_8 = R_5{*}q{+}R_6{*}\hat{r_8}$.

5.Use R_1 to generate $P = P(k) = (R_1 - 1) * k$ according to equation (4).

Output sk= msk= k_1, mpk=$\{n, P, R_1, R'_1, R'_2, R'_3, R'_4, R'_5, R'_6, R'_7, R'_8\}$.

Encrypt(m, ek=mpk): On input message m and ek=mpk, sample random element $\vec{r} = [r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8 \leftarrow Z_n]$ and output:

$c = trap_1 \times m + trap_0 \times \vec{r} = R_1 * m + (R_1 - 1) * r_0 + \sum_{i=1}^{8} R'_i * r_i$ mod n

Evaluate(mpk, $\mathcal{C}$, $(c_1, c_2, ...)$): On input public key mpk, an algorithm $\mathcal{C}$ that supports multiply and adds operations(calculations of polynomial multiplication and addition), a set of input ciphertext $(c_1, c_2, ...)$ and then output:

$c' = \mathcal{C}(c_1, c_2, ...)$ mod P mod n, which is equal to Encrypt$(\mathcal{C}(m_1, m_2, ...), mpk)$. With the help of P, c' is a polynomial of degree seven modulo n.

Decrypt(c', sk=k_1): On input secret key sk, and a cipher-text c'=$\mathcal{C}$(c_1, c_2,...), output:

$m' = c'(sk) = c'(k_1) = 1 * \mathcal{C}(m_1, m_2, ...) = \mathcal{C}(m_1, m_2, ...)$

Because $R_1(k_1)$ is equal to 1 but $P(k_1)$, $(R_1-1)(k_1)$, $R_2(k_1)$, $R_3(k_1)$, $R_4(k_1)$, $R_5(k_1)$ and $R_6(k_1)$ are all equal to 0.

Correctness. At the beginning of this section we have explained why P allows the computation of polynomials to be restricted to polynomials of degree seven (see equation (4)). Because $P(k_1)$=0, it is feasible to use P to replace higher-order polynomials with polynomials of seven. Other calculations including encryption, ciphertext, and decryption calculation are the same as Construction 1. Therefore, Construction 3 is also correct.

Security. Same as R_1, the security of $P(k)$ is also based on the difficulty of finding roots of polynomial modulo large integer n since $P(k) = k * (R_1 - 1)$. Therefore, we denote that $P(k)$ is secure, and an attacker cannot solve any useful information about k_1 through $P(k)$. Considering the attacker may combine $P(k)$ and R_1 to try to crack k_1, we give the definition of Proposition 6.

Proposition 6 *Suppose there are two polynomials satisfying $P_7(k_1) = 0$, $P_8(k_1) = 0$ and P_7, P_8 are not independent. For example, $P_8 = P_7 * k$ or $P_5 = P_7 * (k - k')$, where P_7 and P_8 are polynomials of degree seven and eight respectively. If P_7 cannot be solved for k_1 in the case of modulo n (n=p*q, and p, q are unknown), then P_8 cannot be solved for k_1 and we can get nothing useful by combining them.*

Proof. Proposition 6 is actually another case of Theorem 2. Suppose we have P_7 and P_8 satisfy:

$$\begin{aligned} P_7 \equiv& 0(mod \quad n) \\ &and \\ P_8 \equiv& P_7 * k \equiv 0(mod \quad n) \end{aligned}$$

Then, in order to find the intersection coordinate point k_1, let us combine two polynomials to get a system of equations. Finally, we can get 0=0. Because P_8 is just constructed by polynomials P_7 and k, we cannot solve P_8 by existing polynomial P_7. According to Proposition 1, P_7 is unsolvable, we think that P_8

is unsolvable too. In addition, if we want to combine P_8 with other polynomials in k_1, it is equivalent to combining P_7 with them. This is why in Construction 3 we construct polynomial $P(k)$ like this.

Construction 3 provides encryption of P-FHE based on $P{=}k*(R_1-1)$, and the purpose of P is to limit the change of ciphertext length during the calculation process between ciphertexts. Through Proposition 6, we know that the attacker cannot solve useful information of k_1 through P and R_1. The security of other polynomials computing is held through the definitions of Construction 1. Therefore we show that the encryption process of Construction 3 is secure. The reason why $P{=}k*(R_1-1)$ is because theorems and propositions must be satisfied to ensure the security of encryption are given in Construction 1. For example, Theorem 2, Proposition 3, and Proposition 4 need to ensure that polynomials used cannot disclose decryption key k_1 by collusion and message m cannot be obtained by solving equations. Before introducing P, we need to carefully consider whether P will cause trouble to other polynomials, invalidate previous theorems and propositions, and destroy the security of the encryption scheme. The introduction of $P{=}k*(R_1-1)$ perfectly conforms to these conditions, and we can restrict polynomials to polynomials of degree seven by modular multiplication without introducing a new independent polynomial. In other words, we limit the length of ciphertext to s=8, and it perfectly conforms to the encryption in Construction 1. In addition, we have shown that Construction 2 is a variant of Construction 1. Therefore, the above method of fixing the length of the ciphertext is also valid for Construction 2.

Many construction methods meet the requirements that can be used to construct the polynomial P as long as the encryption security and efficiency can be guaranteed. We explain here that the construction of P is not limited to R_1-1 and k. We show that $R'_i(1 \leq i \leq 8)$ in Construction 1 can all be used to construct P, and the encryption constructed is the same as Construction 3. We say that any complex polynomial of degree 8 that satisfies $P(k_1)=0$ used in Construction 1 will do, such as $(k-k')$*(R_1-1), $(k-k')*R'_1$, $(k-k')*R_2$, $(k-k')*R_3$ $(k' \leftarrow Z_n)$ and other polynomials satisfied $P(k_1) \equiv 0(\text{mod n})$. But when introducing the polynomial P, we must satisfy that the introduced polynomial cannot affect the security of Construction 1. If we use high-degree polynomials other than $k*(R_1-1)$, we need to consider more complicated situations. (1) Constructed P cannot be combined with previous polynomials to recover the decryption key or message m; (2) The constructed polynomial P needs to ensure that the encryption scheme satisfies theorems and propositions in Construction 1; (3) R_2, R_3, R_4, R_5, R_6 cannot be leaked. If we use $R'_i(1 \leq i \leq 8)$ to construct P, obviously it meets the above conditions, and the encryption construction is similar to Construction 3. But if we use R_2, R_3, R_4, R_5 to construct P, we must select a proper element k' to protect them from disclosure. Here we give an example of the encryption system **Setup** with P constructed by R_2.

Setup($\mathbf{1^\lambda}$): On input security parameter λ and set s=8, the setup algorithm does following process:

1.Generate large prime numbers p, q according to λ and compute n=p*q.

2.Randomly select k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, $\hat{r_1}$, $\hat{r_2}$, $\hat{r_3}$, $\hat{r_4}$, $\hat{r_5}$, $\hat{r_6}$, $\hat{r_7}$, $\hat{r_8}$ $\leftarrow Z_n$.

3.Use k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, to generate R_1, R_2, R_3, R_4, R_5, R_6, according to equation (2) and check whether R_1, R_2, R_3, R_4, R_5, R_6 are linear independent. If not, come back to process 2, else continue.

4.Use R_2, R_3, R_4, R_5, R_6, $\hat{r_1}$, $\hat{r_2}$, $\hat{r_3}$, $\hat{r_4}$, $\hat{r_5}$, $\hat{r_6}$, $\hat{r_7}$, $\hat{r_8}$ and p, q to generate $R_1' = R_2{*}p{+}R_6{*}\hat{r_1}$, $R_2' = R_2{*}q{+}R_6{*}\hat{r_2}$, $R_3' = R_3{*}p{+}R_6{*}\hat{r_3}$, $R_4' = R_3{*}q{+}R_6{*}\hat{r_4}$, $R_5' = R_4{*}q{+}R_6{*}\hat{r_5}$, $R_6' = R_4{*}q{+}R_6{*}\hat{r_6}$, $R_7' = R_5{*}q{+}R_6{*}\hat{r_7}$, $R_8' = R_5{*}q{+}R_6{*}\hat{r_8}$.

5.Randomly select k'. Use R_2 and k' to generate $P = P(k) = (R_1{-}1){*}(k{-}k')$ according to equation (4).

Output sk= msk= k_1, mpk=$\{n, P, R_1, R_1', R_2', R_3', R_4', R_5', R_6', R_7', R_8'\}$.

Security. We know that $P = P(k) = (k - k') * R_2$ satisfies $P(k_1) = 0$. The security of the encryption system cannot be destroyed through P, because P is composed of R_2 and we have not introduced new independent polynomials. It can be seen that compared with Construction 3, the second step of **Setup** generates one more random element k', and in the fifth step, P=(k-k')* is generated by R_2 and k' together. We said that if k' is selected properly, at least similar to k_1, k_2, k_3, k_4, k_5, then it can be guaranteed that R_2 cannot be solved through P, satisfying Proposition 4 and 5. So the security can be guaranteed as P constructed by $(k - k') * R_2$ doesn't leak R_2.

6 Conclusion

In this work, we innovatively use the polynomial-based operations to construct a fully homomorphic encryption scheme, which we called $trap_1 \times m$. The key point of our scheme is to use high-order polynomials that are difficult to find roots to realize the encryption parameter $trap_1$ and $trap_0$ we want. We have explained how to realize $trap_1$ and $trap_0$ through the Lagrangian interpolation theorem, making it easier for us to build the encryption scheme.

We mainly give two constructions of our scheme in this paper: using the single encryption key(P-FHE) and using different encryption keys(P-FHEs). Compared with P-FHEs, the P-FHE scheme is more suitable for public domains such as voting systems and certificate systems because it is oriented to the same encrypted public parameters. Because the encryption key ek of each user is different, P-FHEs are more suitable for personalization fields where different users are different. We say that the construction of fully homomorphic encryption using different encryption parameters(P-FHEs) has the highest security as it not only conforms to theorems and propositions of the first construction P-FHE, but also gives less information to an attacker when he wants to break the encryption system. We also provide a method to constant the size of their ciphertext and give details about the construction of P and precautions that need to be observed.

We also show the relationship between s and polynomials used in our scheme so that we can build the encryption process more conveniently. For example, when constructing an encryption system, we need to ensure that there are no more than s-2 independent polynomials. Otherwise, the secret information k_1 of the encryption system will be leaked.

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[23]. 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[24]. 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 give a more efficient fully homomorphic encryption scheme $trap_1 \times m$.
2) Our encryption scheme uses Lagrange interpolation polynomials to realize $trap_1$. 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 operation of batch processing.
3) Find other techniques that are more suitable for constructing $trap_1$ and $trap_0$ to replace the Lagrangian interpolation polynomials, and try more efficient methods to improve our scheme.

References

1. P. Martins, L. Sousa, and A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–33, 2017.
2. R. L. Rivest, L. Adleman, M. L. Dertouzos, *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.
3. C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.
4. C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.
5. M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pp. 24–43, Springer, 2010.
6. M. Li, "Leveled certificateless fully homomorphic encryption schemes from learning with errors," *IEEE Access*, vol. 8, pp. 26749–26763, 2020.
7. J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pp. 409–437, Springer, 2017.
8. C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the aes circuit," in *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pp. 850–867, Springer, 2012.
9. Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on computing*, vol. 43, no. 2, pp. 831–871, 2014.
10. Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Advances in Cryptology–CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings 31*, pp. 505–524, Springer, 2011.
11. Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.
12. J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.
13. C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pp. 75–92, Springer, 2013.
14. L. Ducas and D. Micciancio, "Fhew: bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*, pp. 617–640, Springer, 2015.

15. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Tfhe: fast fully homomorphic encryption over the torus,” *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.
16. M. O. Rabin, “Digitalized signatures and public-key functions as intractable as factorization,” tech. rep., Massachusetts Inst of Tech Cambridge Lab for Computer Science, 1979.
17. W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, “Rsa and rabin functions: Certain parts are as hard as the whole,” *SIAM Journal on Computing*, vol. 17, no. 2, pp. 194–209, 1988.
18. T. Sauer and Y. Xu, “On multivariate lagrange interpolation,” *Mathematics of computation*, vol. 64, no. 211, pp. 1147–1170, 1995.
19. H. Chen, I. Chillotti, and Y. Song, “Multi-key homomorphic encryption from tfhe,” in *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part II 25*, pp. 446–472, Springer, 2019.
20. Z. Brakerski and R. Perlman, “Lattice-based fully dynamic multi-key fhe with short ciphertexts,” in *Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pp. 190–213, Springer, 2016.
21. H. Chen, W. Dai, M. Kim, and Y. Song, “Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 395–412, 2019.
22. H. Chen, I. Chillotti, and Y. Song, “Improved bootstrapping for approximate homomorphic encryption,” in *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part II*, pp. 34–54, Springer, 2019.
23. G. Spanos and L. Angelis, “The impact of information security events to the stock market: A systematic literature review,” *Computers & Security*, vol. 58, pp. 216–229, 2016.
24. S. Jokić, A. S. Cvetković, S. Adamović, N. Ristić, and P. Spalević, “Comparative analysis of cryptocurrency wallets vs traditional wallets,” *ekonomika*, vol. 65, no. 3, pp. 65–75, 2019.