Hindawi
Security and Communication Networks
Volume 2021, Article ID 9962575, 14 pages
https://doi.org/10.1155/2021/9962575

Research Article

Blockchain-Based Secure Outsourcing of Polynomial Multiplication and Its Application in Fully Homomorphic Encryption

Mingyang Song, Yingpeng Sang, Yuying Zeng, and Shunchao Luo

School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou, China

Correspondence should be addressed to Yingpeng Sang; sangyp@mail.sysu.edu.cn

Received 15 March 2021; Revised 26 May 2021; Accepted 7 June 2021; Published 25 June 2021

Academic Editor: Yinghui Zhang

The efficiency of fully homomorphic encryption has always affected its practicality. With the dawn of Internet of things, the demand for computation and encryption on resource-constrained devices is increasing. Complex cryptographic computing is a major burden for those devices, while outsourcing can provide great convenience for them. In this paper, we firstly propose a generic blockchain-based framework for secure computation outsourcing and then propose an algorithm for secure outsourcing of polynomial multiplication into the blockchain. Our algorithm for polynomial multiplication can reduce the local computation cost to $O(n)$. Previous work based on Fast Fourier Transform can only achieve $O(n\log(n))$ for the local cost. Finally, we integrate the two secure outsourcing schemes for polynomial multiplication and modular exponentiation into the fully homomorphic encryption using hidden ideal lattice and get an outsourcing scheme of fully homomorphic encryption. Through security analysis, our schemes achieve the goals of privacy protection against passive attackers and cheating detection against active attackers. Experiments also demonstrate our schemes are more efficient in comparisons with the corresponding nonoutsourcing schemes.

1. Introduction

As the development of the big data era, there is an increasing demand for large-scale time-consuming computations. Fortunately, with the emergence of cloud computing, computation outsourcing brings convenience to resource-constrained users. They can outsource complex computing tasks into the cloud by paying a fee and avoiding buying expensive high-performance hardware. It not only improves the resource utilization in cloud but also brings economic benefits to resource-constrained users. Nevertheless, the attractive computing scheme also causes security issues. A passive attacker in the cloud may be only curious about the privacy contained in the user's outsourced data, while an active attacker may make malicious damage or forge the results to sabotage the computation. Even if there is no attacker, computing errors caused by cloud hardware failure and software errors, etc. should also be considered. Furthermore, it should not be a great burden for the user to check the correctness of the returned results from the cloud; otherwise, the efficiency benefit of outsourcing will be nullified. Therefore, the secure and efficient outsourcing of computations that can not only protect the privacy of users but also ensure the correct results has become a hot research topic.

Gentry proposed a homomorphic encryption algorithm based on ideal lattice [1] for the first time, providing us with a direction to solve the privacy issues in computation outsourcing. The direction is a secure computation outsourcing mode: encryption-outsourcing-decryption (EOD). Even if we use the common EOD model, the device should also undertake the computations of secret key generation, encryption, verification, decryption, and so on, locally. These computations are also great burden for the resource-constrained devices (such as mobile phones and IoT nodes).

Blockchain has attractive features such as transparency, traceability, decentralization, and immutability, which make it an optimal approach for applications intrinsically with untrusted natures, such as computation outsourcing. A central trusted entity is not required for the computation outsourcing based on blockchain. Information about the whole data exchange process, computations, users, and computational nodes is recorded and is traceable in blockchain. Besides, smart contract can be utilized to digitally facilitate the implementation of whole transaction, which greatly improves the speed of building applications on blockchain. However, privacy is still an issue in the computation outsourcing based on blockchain.

Owing to the low efficiency of fully homomorphic encryption algorithms, the general computation outsourcing mode based on EOD is impractical on the resource-constrained devices. In this paper, we will outsource some complex computations in the fully homomorphic encryption using hidden ideal lattice (FHEHIL) [2] into a blockchain framework. The contributions of this paper can be summarized as follows:

(1) We propose a framework of blockchain-based computation outsourcing, in which we can implement secure outsourcing for FHEHIL. The framework has a credit-based task allocation strategy, which will significantly reduce the probability of malicious nodes participating in computing.

(2) We propose a secure outsourcing algorithm for polynomial multiplication, which reduces the local computation cost (including the cost on result verification) to $O(n)$. Previous work based on the Fast Fourier Transform (FFT) can only achieve $O(n\log(n))$ for the local cost. Besides, the algorithm can not only detect cheating but also identify cheating nodes combining with blockchain. The result verification in the outsourcing algorithm does not cause extra burden.

(3) We also extend the secure outsourcing algorithm of modular exponentiation, in [3], in our blockchain-based framework. The two algorithms for polynomial multiplication and modular exponentiation are employed in FHEHIL as basic operations, and the FHEHIL implementation on the blockchain-based framework can have higher efficiency compared with previous work.

2. Related Work

At present, research studies on secure outsourcing can be roughly divided into two directions. In one direction, a general outsourcing mechanism is studied. In this mechanism, a fully homomorphic encryption algorithm is designed and the EOD model is used to outsource any computations. After the work of Gennaro et al. [1], great progress has been made in the field of fully homomorphic encryption [4–6]. However, the fully homomorphic encryption algorithms have high computational complexity. Recently, there are many researches to reduce the computation cost of homomorphic encryption algorithm. For example, Su et al. accelerated the leveled Ring-LWE fully homomorphic encryption [7]. These research studies mainly focus on the efficiency of hardware. However, secure outsourcing complex computations of fully homomorphic encryption is a better way to improve efficiency for resource-constrained devices.

In the other direction, specific outsourcing algorithms are designed for various scientific computations, e g., modular exponentiation, solution of large-scale linear equations [8], bilinear pairings, and extend Euclidean. The Wei pairing and Tate pairing in algebraic curves are commonly used in key establishment and signature schemes in the field of cryptography. However, the computation of bilinear pairings is time-consuming in resource-constrained devices. Thus, many outsourcing schemes have been proposed [9–11]. To our knowledge, the scheme in [9] is the most efficient and secure till now. Because of the wide application in cryptography, the study on modular exponentiation is also a hot topic of research. Hohenberger and Lysyanskaya [12] proposed a modular exponentiation secure outsourcing scheme. Chen et al. [13] further improved its efficiency and verifiability. Ren et al. [14] proposed a scheme that only protects the privacy of exponent. Recently, Fu et al. [3] proposed a secure outsourcing scheme of modular exponentiation with hidden exponent and base. It has a stronger checkability. In cryptography, extended Euclidean algorithm is usually used to calculate modular inverse, which is widely used in RSA encryption algorithm. Similarly, Euclidean algorithm can be used to find the greatest common factor of two polynomials, which is commonly used in encryption algorithm based on Lattice. Zhou et al. [15] proposed the secure outsourcing algorithm of extended Euclidean algorithm.

Polynomial multiplication is likewise a commonly used operation in cryptography schemes, error correcting codes, and computer algebra. The complexity of polynomial multiplication is still a major open problem. Using the FFT, the local computation of polynomial multiplication can achieve the complexity of $O(n\log(n))$. Recently, some efficient polynomial multiplication methods based on the FFT are proposed. Harvey et al. proposed a faster method over finite fields Z_p when the degree of polynomials is less than p [16]. The efficiency has been further improved in [17]. For the hardware utilization, Liu et al. designed a high hardware efficiency polynomial multiplication on field-programmable gate array (FPGA) platform [18]. Hsu and Shieh proposed a method with less addition and multiplication [19] in 2020. Besides, there are also some research studies on reducing the space complexity of polynomial multiplication [20, 21]. However, the complexity of some research studies based on the acceleration of FFT remains at $O(n\log(n))$. Other methods using distributed computing to improve hardware utilization are not applied to the resource-constrained device. Till now, there are few research studies on the secure outsourcing of polynomial multiplication.

Due to the characteristics of the blockchain and Bitcoin [22], there are lots of research studies and applications on blockchain in recent years including the secure outsourcing. Lin et al. studied the secure outsourcing for bilinear pairings

based on blockchain [23]. Zheng et al. [24] proposed a secure outsourcing scheme for attribute-based encryption on blockchain. There are also some schemes [25, 26] of outsourced data integrity verification. The fairness problem in blockchain-based secure multiparty computation was also solved by multiple efforts. For example, Gao et al. [27] proposed a scheme which realized fairness by maintaining an open reputation system. This type of general scheme for secure multiparty computation can be cumbersome for the problems in secure outsourcing computation.

Andrychowicz et al. [28] utilized only scripts in Bitcoin currency to construct a fair protocol for secure multiparty lotteries, without relying on a trusted third party. Zhang et al. proposed BCPay in [29] and BPay in [30] to achieve fair payment for blockchain-based outsourcing services, which are compatible with the Bitcoin and Ethereum platforms. However, these frameworks can only provide fairness between the client and a single server. They are applicable to the outsourcing scenarios where the task is outsourced to a single server from the client. In the problem of our work, the computation task needs to be outsourced to multiple computational nodes simultaneously. Therefore, we propose a new one, considering the penalty on the cheating nodes, compensation on the honest nodes, and application of a credit-based scheme.

3. Notations and Background

3.1. Notations. We use upper case bold letters for matrices and $\det(M)$ for the determinant of matrix M. Lower case bold letters represent vectors (eg., $v = [v_1, \ldots, v_n]$), where v_i is the i^{th} element in v. We denote polynomial by lower case italics (eg., $f(x)$). For a rational number r, $\text{round}(r)$ represents the nearest integer to r. The rational vector v can also be rounded to $\text{round}(v) = [\text{round}(v_1), \ldots, \text{round}(v_n)]$. We use $v(x)$ for the polynomial form of the vector v. We use $v_1 \times v_2$ for polynomial multiplication ($v_1 \times v_2 = (v_1(x) \times v_2(x)) \bmod f(x)$) on the ring. We use $|v|$ for the norm of v and $|S|$ for the base of set S. We use $\mathrm{v} \circ \mathrm{w}$ for correlation ($v \circ w = [v_1 * w_1, \ldots, w_n]$). We use $R(v, f)$ for the rotation matrix of v whose i^{th} row is the coefficients of $v(x) \times x^{(i-1)} \bmod f(x)$. We use $x\gcd(a(x), b(x))$ for the extended Euclidean algorithm on $a(x)$ and $b(x)$. $\deg(f(x))$ represents the degree of $f(x)$. We use $\mathbf{F}_v$ to denote the coefficient set of Discrete Fourier transform (DFT) of v and $\mathbf{F}_v^-$ for the coefficient set of inverse DFT of $\mathbf{v}$.

3.2. Fully Homomorphic Encryption Using Hidden Ideal Lattice. The FHEHIL scheme [2] is described in Algorithm 1, including the components of key generation, encryption, and decryption. The related parameters are shown in Table 1.

As shown in Algorithm 1, it is obvious that polynomial multiplication is the primary computation in encryption and decryption. Therefore, our proposed algorithm for the secure outsourcing of polynomial multiplication can be directly applied. As for the key generation, the computing burden is from Step 7, 9, and 11. The main computational cost of Step 7 is on computing the determinant of matrix V. The most time-consuming operation in Step 9 is polynomial multiplication. Step 11 computes the inverse of polynomial. Below, we will analyze the detailed computations in Step 7 and 11 and demonstrate that polynomial multiplication and modular exponentiation are the main types of computations, which are the two improvement aims of this paper.

3.2.1. The Method of Computing d in Algorithm 1. Because of the characteristic of V, computing the determinant of matrix V needs only $\log(n)$ times of polynomial multiplication using the method in [31]. d is the free item of $g(z) = \prod_{i-0}^{n-1}(v(p_i) - z)$. Thus, $d = \prod_{i=0}^{n-1} v(p_i)$, where p_i is the root of $f(x) = 0$ in the complex domain and they satisfy equation (2). Due to equation (2), we have $d = \prod_{i=0}^{n-1} v(p_i) = \prod_{i=0}^{n/2-1} v(p_i)v(-p_i) = \prod_{i=0}^{n/2-1} a(p_i)$. Thus, the computation of d mainly involves polynomial multiplications and modular exponentiations.

3.2.2. The Method of Computing w in Algorithm 1 When d Is a Prime. In the algorithm of FHEHIL, if d is a prime, we can obtain w by performing $x\gcd(v, f)$ once. The extended Euclidean algorithm computes $x\gcd(a(x), b(x))$ to get $u(x)$, $v(x)$, and $d(x)$, satisfying $u(x) \times a(x) + v(x) \times b(x) = d(x)$. The specific procedures of secure outsourcing for extended Euclidean algorithm [15] are summarized in Algorithm 2. It can be seen that the local computations of this algorithm consist of mostly modular exponentiation and polynomial multiplications. Detailed analysis of Algorithm 2 can be found in [15].

3.2.3. The Method of Computing w in Algorithm 1 When d Is Not a Prime. When d is not a prime, the fastest method to calculate polynomial inverse at present is Gentry's method in [31]. The method is based on fast Fourier transform and halves the number of terms in each step to offset the doubling of the bit length of the coefficients. This method relays on $f(x) = x^{n+1}$, where n is a power of 2. The method is analyzed as follows.

Firstly, the second coefficient (g_1) of polynomial $g(z) = \prod_{i=0}^{n-1}(v(p_i) - z)$ can be computed, where p_i is the ith root of $f(x) = 0$ in the complex domain. We can get $w_0 = (g_1/n)$. Secondly, $v'(x) = x \times v(x) \bmod f(x)$ can be constructed. Then, the second coefficient g_1') of polynomial $g'(z) = \prod_{i=0}^{n-1}(v'(p_i) - z)$ can be computed. We can get $w_1 = (g_1'/n)$. Finally, other coefficients of w can be computed by

$$\frac{w_1}{w_0} = \frac{w_2}{w_1} = \cdots = \frac{w_{n-1}}{w_{n-2}}, \tag{1}$$

$$\left(p_i + \frac{n_j}{2}\right)^{2^i} = -\left(p_i^{2^i}\right)\left(n_j = \frac{n}{2^j}\right). \tag{2}$$

Since n is a power of 2 in $f(x)$, the roots satisfy equation (2). In the process of computing coefficients g_1 and g_1', the major computations are also polynomial multiplications and modular exponentiations.

Input: ζ, γ, η, t, n
Output: SK = $\{d, w\}$, PK = $[p_1, \ldots, p_t]$
(1) **function** KEY GENERARTION $(\zeta, \gamma, \eta, t, n)$
(2) generate a random vector v satisfying $\left\{v \in Z^n, 2^{\gamma-1} < |v| < 2^{\gamma}, \sum_{i=0}^{n-1} v_i \bmod 2 = 1\right\}$
(3) generate $t-1$ random vectors $[g_1, \ldots g_{t-1}]$ satisfying $\{g_i \in Z^n, 2^{n-1} < [g_i] < 2^n\}$
(4) generate a random vector $\mathbf{g_t}$ satisfying $\left\{g_t \in Z^n, [g_i] < 2^n, \sum_{i-0}^{n-1} g_t|i| \bmod 2 = 1\right\}$
(5) generate $t-1$ random vectors $[r_1, \ldots, r_{t-1}]$ satisfying $\{r_i \in \{0, 1, -1\}^n, |r_i| < \zeta\}$
(6) generate a random vector $\mathbf{r_t}$ satisfying $\left\{r_t \in \{0, 1, -1\}^n, |r_t| < \zeta, \sum_{i=0}^{n-1} r_t|i| \bmod 2 = 1\right\}$
(7) $f(x) \longleftarrow x^n + 1$; $V \longleftarrow R(v, f)$; $d \longleftarrow |\det(V)|$
(8) **for** $i = 1 \longrightarrow t$ do
(9) $p_i \longleftarrow g_i \times v + r_i$
(10) **end for**
(11) Get **w** satisfying $w \times v = d \bmod f$
(12) **return** SK = $\{d, w\}$, PK = $[p_1, \ldots, p_t]$
(13) **end return**
(14) **function** ENCRYPTION (PK = $[p_1, \ldots, p_t]$, plaintext)
(15) generate $t-1$ random integer vectors $[s_1, \ldots, s_{t-1}]$ satisfying $\sum_{j=0}^{n-1} s_i[j] \bmod 2 = 0$
(16) generate a random vector s_t satisfying $\sum_{i=0}^{n-1} s_t[j] \bmod 2 =$ plaintext
(17) generate a random vector s_{t+1} satisfying $\sum_{j=0}^{n-1} s_{t+1}[j] \bmod 2 = 0$
(18) $\psi \longleftarrow \sum_{i=1}^{t} s_i \times p_i + s_{t+1}$
(19) **return** ψ
(20) **end function**
(21) **function** DECRYPTION (SK = (d, w), ψ)
(22) $\psi' \longleftarrow$ round$(\psi \times w/d)$
(23) plaintext $\longleftarrow \psi'(1) \bmod 2$
(24) **return plaintext**
(25) **end function**

Algorithm 1: Fully homomorphic encryption using hidden ideal lattice.

Table 1: Parameters in the FHEHIL algorithm.

Parameters	Implication
ζ	The norm of random noise vector
γ	The bit length of norm of generating polynomial
η	The bit length of norm of the random multiplier vector
t	The number of vectors contained in the public key
n	The dimension of the hidden lattice (power of 2)

4. The Framework of Blockchain-Based Computation Outsourcing

This section introduces a blockchain-based computation outsourcing framework. The overall system model is illustrated in Figure 1, and the related symbols are described in Table 2. In Figure 1, we assume that, at least, one trusted third party is available to implement the smart contract. This is a generic model on which a variety of computational tasks can be implemented, including the tasks of secure outsourcing of polynomial multiplication and modular exponentiation. The two specific tasks will be given in details in the rest of this paper.

4.1. Registration. Users and computational nodes need to register before joining the network. They need to pay deposits in advance, the amount of which needs to be greater than a specified threshold or they will be rejected. The smart contract initializes the same credit score to all new nodes and users. After registration, information of users and computational nodes is written to the smart contract. The specific function is shown as Algorithm 3. In this algorithm, addr[p] is the deposit account of node p in the smart contract. Node p's asset privacy is protected because it is unnecessary for him to expose his total asset to the smart contract.

4.2. Computational Service. User p posts computing tasks, data, and rewards of each task to the smart contract. If the account balance of p is insufficient for the computations, smart contract refuses this service and reduces the credit score of p, to prevent malicious users from attacking the smart contract by constantly sending tasks that they cannot afford to. After the smart contract accepts the tasks of p, the user's computing tasks are stored in the task queue and wait for the selection of computational nodes.

To achieve a high benefit, the computational nodes will be active to undertake computing tasks. If multiple computational nodes select the same task at the same time, the node with the highest credit score will win the task. Nodes that undertake the computing task should submit the results after completing. If any computational node cannot finish on time, it will be added to the dishonest set. If all computational nodes can submit the results on time, the smart contract sends the results to user and initiates the period of dispute resolving. During this period, the user needs to verify the results locally and notify the smart contract whether the

Input: $a(x)$, $b(x)$
Output: $v(x)$, $v(x)$, $d(x)$
(1) generate $r(x)$ and $a \in Z$ randomly
(2) $f(x) \longleftarrow a(\alpha x), g(\alpha x) \longleftarrow b(x)$
(3) $a'(x) \longleftarrow r(x) \times f(x), b'(x) \longleftarrow r(x) \times g(x)$
(4) $a''(x) \longleftarrow u_{11}(x) \times a'(x) + u_{12}(x) \times b'(x), b''(x) \longleftarrow u_{21}(x) \times a'(x) + u_{22}(x) b'(x)$
(5) send $a''(x)$ and $b''(x)$ to computational node
(6) get $u''(x), v''(x)$ and $d''(x)$ from computational mode
(7) verify $a''(x) \times u''(x) + b''(x) \times v''(x) = d''(x)$; $\deg(u''(x)) < \deg(b''(x))/d''(x)$; $\deg(v''(x)) < \deg(a''(x))/d''(x)$;
(8) $u(x) \longleftarrow \alpha^{\deg(d''(x)) - \deg(r(x))}(u_{11}(\alpha^{-1}x) \times u''(\alpha^{-1}x) + u_{21}(\alpha^{-1}x) \times v''(\alpha^{-1}x))$
(9) $v(x) \longleftarrow \alpha^{\deg(d''(x)) - \deg(r(x))}(u_{12}(\alpha^{-1}x) \times u''(\alpha^{-1}x) + u_{22}(\alpha^{-1}x) \times v''(\alpha^{-1}x))$
(10) $d(x) \longleftarrow \alpha^{\deg(d''(x)) - \deg(r(x))} d''(\alpha^{-1}x)/r(\alpha^{-1}x)$

Algorithm 2: Securely outsource the extended Euclidean algorithm.

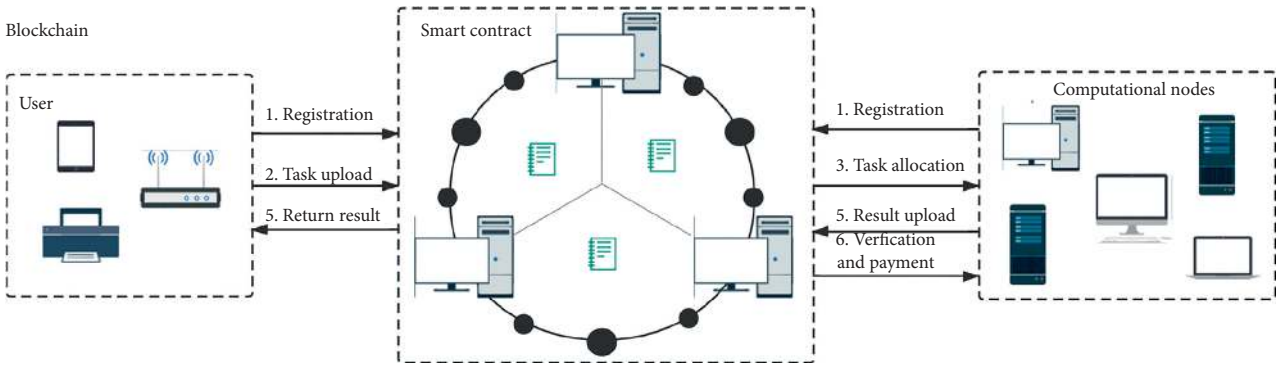

Figure 1: The framework of blockchain-based computation outsourcing.

Table 2: Symbols in the framework of blockchain-based computation outsourcing.

Symbol	Implication
addr[p]	The account address of p
User/node	The set of users/computational nodes
Rep[p]	The reputation of p
Task	The queue of published tasks
Allocated	The queue of allocated tasks
Data	The queue of published data
Time	The set of the latest result return time
Dishonest/honest	The set of dishonest/honest nodes
Result	The set of results
Δb	The amount of economic punishment
Δc	The amount of credit punishment/reward
Δt	The longest computing time consumption

results are accepted or not. If the period of dispute ends and there is no feedback received from the user, the smart contract assumes that the computations succeed and performs the reward and charge operations. If the user does not accept the results in the feedback, the smart contract will verify the results by itself. The specific function is shown as Algorithm 4.

4.3. Verification and Payment. If the user does not accept the computing results, the smart contract will perform verification operations to find dishonest nodes or users. In the verification, he will simulate all computations required by the user on the encrypted data uploaded by the latter. This means he will repeat exactly every step the computational nodes have carried out, so as to find which step is not correct and who is cheating. Decryptions are not required in the simulation, and thus, the data privacy of the user is protected.

To complete the verification, the smart contract should be equipped with the same function modules as those of the computational nodes. For example, in the secure outsourcing of polynomial multiplication, a function should be

```
(1) function Registration(p, role)
(2)   if the balance of addr[p] ≥ the threshold then
(3)     if role = user then
(4)       put addr[p] into User
(5)     else
(6)       put addr[p] into Node
(7)     end if
(8)       Rep[p] ⟵ Default credit value; state ⟵ true
(9)     else
(10)      state ⟵ false
(11)    end if
(12)    return false
(13) end function
```

Algorithm 3: Registration.

```
(1) function TASK UPLOAD (user, tasks, data, price)
(2)   if the balance of addr[user] ≥ |tasks| * price then
(3)     put tasks into Task; put data into Data
(4)   else
(5)     Rep[user]⟵Rep[user] − c
(6)   end if
(7) end function
(8) function TASK ALLOCATION (nodes, task)
(9)     if task ≠ ALLOCATED then
(10)      sort the nodes with their credit descending; send task and Data[task] to nodes [0]
(11)      put task into ALLOCATED; Time[task]⟵current time + t
(12)    end if
(13) end function
(14) function RESULT UPLOAD (node, task, result)
(15)    if current time < Time[task] then
(16)      put node into Honest; Put result into result
(17)    else
(18)      put node into Dinhonest
(19)    end if
(20) end function
(21) function RESULT UPLOAD(node, task, result)
(22)    if Dishonest = 0 and |Result| = |Task| then
(23)      send Result to user; Wait for the feedback from user
(24)    end if
(25)    call verification and payment
(26) end function
```

Algorithm 4: Computational service.

added into the smart contract to simulate the FFT/IFFT operations on the data uploaded by the user in case of disputes.

The dishonest nodes and users will be put into the dishonest set. If no one is put into the dishonest set, the user will pay the reward to all participating nodes. Otherwise, cheating nodes will be penalized and the honest nodes will be compensated. The credit score of the participating nodes that have correctly completed their tasks will increase, while the credit score of the malicious nodes will decrease. When the account balance of a node is lower than the threshold value or its credit score is reduced to zero, the system will remove it. The specific function is shown as Algorithm 5.

A malicious user with enough balance may constantly initiate transactions, aiming to increase the burden of the smart contract. However, he cannot refuse to pay because his deposit account is managed by the smart contract. By setting up a suitable threshold in the registration, sooner or later his balance will be used up by his attack.

```
(1) function VERIFICATION AND PAYMENT (feedback, Result, Honest, user, price)
(2)   if feedback ≥ user or (feedback = NULL and |Result| = |Task|) then
(3)     addr[user] ⟵ addr[user]-[Honest] ∗ price
(4)     for i = 1⟶|Honest| do
(5)       Rep[node_i] ⟵ Rep[node_i] + c; addr[node_i]⟵addr[node_i] + price
(6)     end for
(7)   else
(8)     stimulate all computations and put dishonest user or nodes into Dishonest
(9)     t ⟵ b ∗ [IDishonest]
(10)    for i = 1⟶|Dishonest| do
(11)      Rep[node_i]⟵Rep[node_i] + c; addr[node_i] ⟵ addr[node_i]− b
(12)    end for
(13)    for i = 1⟶|Honest|do
(14)      Rep[node_i] ⟵ Rep[node_i] + c; addr[node_i] ⟵ addr[node_i] + t/|Honest|
(15)    end for
(16)  end if
(17) end function
```

Algorithm 5: Verification and payment.

4.4. Security Analysis. Both the malicious user nodes and computational nodes can launch attacks to the framework, but their strategies are different. The malicious user nodes could launch a DDoS attack. A malicious computational node could destroy the computation by returning forged results or not returning any result.

The user nodes could employ two ways to launch a DDoS attack. One is to continuously publish the tasks that the user actually cannot afford to; the other is to maliciously inform the smart contract that the results are not accepted during the dispute resolving period. For the first attack, the smart contract will refuse to add the computing tasks and data to the queue and reduce the credit score of the user. Moreover, when the node's credit score drops below 0, the node will be removed. We can increase Δc in the function of TASK UPLOAD to remove malicious users as soon as possible. For the second attack, the smart contract has to simulate the computations of all nodes participating in the outsourcing according to the data and records. This attack has a greater impact on the smart contract, but it brings more loss to the attackers (including the financial punishment). Similarly, we can increase Δb in Algorithm 5 to mitigate the impact on smart contracts.

The computational nodes also have two ways to deploy attack: returning forged results or not returning any result. The cost of both attacks is the same (in financial and credit punishment). Since forged results render the smart contract to simulate the computations of all nodes, rational computational nodes prefer to attack by returning forged results, rather than return nothing. Similarly, we can increase Δb and Δc in Algorithm 5 to mitigate the impact on smart contracts. The proposed framework adopts a task allocation strategy based on credit scores. When malicious nodes are found, their credit score is reduced, and their probability of obtaining computing tasks in the next time is also reduced. We assume there are enough computational nodes which are willing to return correct results to fulfil the requirement of outsourcing, under the incentives of achieving financial and credit reward.

4.5. Compatibility Analysis. We know that the Bitcoin script is not Turing-complete, and Ethereum has a complete programming language on the blockchain to execute more complex smart contracts. It is easy to see that our framework is compatible with opcodes allowed by the Ethereum blockchain. Since the function of Verification and Payment (Algorithm 5) involves loops, which are not allowed by the Bitcoin script, our framework is not compatible with opcodes of the Bitcoin blockchain.

5. Polynomial Multiplication and Modular Exponentiation Secure Outsourcing Algorithm

5.1. Polynomial Multiplication Secure Outsourcing Algorithm. The computational complexity of traditional polynomial multiplication is $O(n^2)$, which is reduced to $O(n\log(n))$ by the FFT. In this section, we employ secure outsourcing to further reduce the local computational complexity to $O(n)$. The outsourcing is implemented in our proposed framework of blockchain-based secure computation outsourcing. The main idea of our algorithm is as follows. Firstly, the Fourier transform of the polynomial coefficients are securely outsourced. Secondly, correlation operation on the results of the Fourier transform is locally performed. Finally, the inverse Fourier transform on result of the correlation operation are securely outsourced. The specific process of our algorithm is shown as Algorithm 6.

5.1.1. Description. In Algorithm 6, the input polynomials are $f(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$ and $g(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1}$. The output is $t(x) = f(x) \times g(x) = c_0 + c_1x + \cdots + c_{2n-1}x^{2n-1}$. For convenience, polynomials are replaced with vectors of polynomial coefficients ($\mathbf{a} = [a_0, a_1, \ldots, a_{n-1}]$, $\mathbf{b} = [b_0, b_1, \ldots, b_{n-1}]$, and $\mathbf{c} = [c_0, c_1, \ldots, c_{2n-2}]$). Besides, $W_n^m = e^{-2\pi jm/n}$, in this section. We use $6p$ computational nodes in this algorithm, where p is a parameter associated with the

Input: a,b
Output: $\mathbf{c} = \mathbf{a} \times \mathbf{b}$
(1) **function** DISCRETE FOURIER TRANSFORM FOR RESERVED VECTOR (DFTRV) ($\mathbf{r}$, i)
(2) $n \longleftarrow$ the length of $\mathbf{r}$
(3) base $\longleftarrow W_n^i$
(4) $F_r[0] \longleftarrow r_i$
(5) **for** $j = 1 \longrightarrow n-1$ **do**
(6) $F_r[j] \longleftarrow F_r[j-1] *$ base
(7) **end for**
(8) **return** $\mathbf{F_r}$
(9) **end function**
(10) **function** INVERSE DISCRETE FOURIER TRANSFORM FOR RESERVED VECTOR(IDFTRV) ($\mathbf{r}$, i)
(11) $n \longleftarrow$ the length or $\mathbf{r}$
(12) base $\longleftarrow W_n^{-i}$
(13) $F_r^-[0] \longleftarrow r_i$
(14) **for** $j = 1 \longrightarrow n-1$ **do**
(15) $F_r^-[j] \longleftarrow F_r^-[j-1] *$ base
(16) **end for**
(17) **return** $\mathbf{F_r^-}$
(18) **end function**
(19) The user picks random parameters $i, j, \beta, k_1, k_2, k_3$, and generates
(20) $\mathbf{r_1} \longleftarrow L(i, k_1, n), \mathbf{r_2} \longleftarrow L(j, k_2, n), \mathbf{r_3} \longleftarrow L(\beta, k_3, 2n)$,
(21) $\mathbf{V} \longleftarrow T(\mathbf{a}, \mathbf{r_1}), \mathbf{U} \longleftarrow (\mathbf{a}, \mathbf{r_1}), \mathbf{Z} \longleftarrow T(\mathbf{b}, \mathbf{r_2}), \mathbf{S} \longleftarrow T(\mathbf{b}, \mathbf{r_2})$;
(22) The user calls TASK UPLOAD locally and uploads $\mathbf{U}, \mathbf{V}, \mathbf{Z}, \mathbf{S}$to the smart contract;
(23) The smart contract calls TASK ALLOCATION and sends vectors in $\mathbf{U}, \mathbf{V}, \mathbf{Z}, \mathbf{S}$to $4p$ nodes;
(24) The computational nodes compute $[\mathbf{F_{v_1}}, \ldots, \mathbf{F_{v_p}}]$, $[\mathbf{F_{u_1}}, \ldots, \mathbf{F_{u_p}}]$, $[\mathbf{F_{z_1}}, \ldots, \mathbf{F_{z_p}}]$, $[\mathbf{F_{s_1}}, \ldots, \mathbf{F_{s_p}}]$, and call RESULT UPLOAD;
(25) The smart contract calls RETURN RESULT;
(26) The user computes $\mathbf{F_{r_1}} \longleftarrow$ DFTRV$(\mathbf{r_1}, i), \mathbf{F_{r_2}} \longleftarrow$ DFTRV$(\mathbf{r_2}, j), \mathbf{F_{r_3}^-} \longleftarrow$ IDFTRV$(\mathbf{r_3}, \beta)$, and verifies equations (3) and (4) locally;
(27) The user computes $\mathbf{F_a} \longleftarrow \sum_{i=1}^{p} \mathbf{F_{v_i}} + \mathbf{F_{r_1}}, \mathbf{F_b} \longleftarrow \sum_{i=1}^{p} \mathbf{F_{z_i}} + \mathbf{F_{r_2}}, \mathbf{F_c} \longleftarrow \mathbf{F_a} \circ \mathbf{F_b}$;
(28) The user generates $\mathbf{D} \longleftarrow T(\mathbf{F_c}, \mathbf{r_3}), \mathbf{E} \longleftarrow T(\mathbf{F_c}, \mathbf{r_3})$, calls TASK UPLOAD locally and uploads $\mathbf{D}, \mathbf{E}$ to the smart contract;
(29) The smart contract calls TASK ALLOCATION and sends vectors in $\mathbf{D}, \mathbf{E}$ to $2p$nodes;
(30) The computational nodes compute $[\mathbf{F_{d_1}^-}, \ldots, \mathbf{F_{d_p}^-}]$, $[\mathbf{F_{e_1}^-}, \ldots, \mathbf{F_{e_p}^-}]$, and call RESULT UPLOAD;
(31) The smart contract calls RETURN RESULT;
(32) The user computes $\mathbf{c} \longleftarrow \sum_{i=1}^{p} \mathbf{F_{d_i}^-} + \mathbf{F_{r_3}^-}$, and verifies equations (5)–(7) locally;
(33) The user sends feedback to the smart contract;
(34) The smart contract calls VERIFICATION AND PAYMENT.

Algorithm 6: Secure outsourcing of polynomial multiplication.

number of computational nodes. There are six steps in the algorithm:

(1) Six parameters are picked randomly, three of which are $i, j, \beta, s.t. 0 \le n-1, 0 \le j \le n-1$ and $0 \le \beta \le 2n-2$. The other three are $k_1, k_2, k_3 \in_R Z$. We define that $L(i, k, n)$: $Z^3 \longrightarrow Z^n$ can generate one n-dimensional vector in which the i^{th} element is k, and all the other elements are zeros. We define that $T(\mathbf{v}, \mathbf{r})$: $z^{(n*2)} \longrightarrow Z^{(n*p)}$ can generate a random matrix $\mathbf{W} = [\mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_p}]$ satisfying $\mathbf{v} = \sum_{i=1}^{p} \mathbf{w_i} + \mathbf{r}$. Then, the user generates $\mathbf{r_1} = L(i, k_1, n)$, $\mathbf{r_2} = L(j, k_2, n)$, $\mathbf{r_3} = L(\beta, k_3, 2n)$, $\mathbf{V} = T(\mathbf{a}, \mathbf{r_1})$, $\mathbf{U} = T(\mathbf{a}, \mathbf{r_1})$, $\mathbf{Z} = T(\mathbf{b}, \mathbf{r_2})$, and $\mathbf{S} = T(\mathbf{b}, \mathbf{r_2})$. In this way, one must know $\mathbf{r_1}$ to recover a from $\mathbf{V}$ or $\mathbf{U}$ and know $\mathbf{r_2}$ to recover $\mathbf{b}$ from $\mathbf{Z}$ or $\mathbf{S}$.

(2) The user uploads the data ($\mathbf{V}$, $\mathbf{U}$, $\mathbf{Z}$, and $\mathbf{S}$) and task requirement (i.e., FFT) to smart contract, by the function TASK UPLOAD. The smart contract calls the function TASK ALLOCATION and distributes the vectors in $\mathbf{V}$, $\mathbf{U}$, $\mathbf{Z}$, and $\mathbf{S}$ to $\mathbf{Z}$ and 4p computational nodes. After receiving the vectors, the computational nodes perform the Fourier transform on the received vectors and call the function RESULT UPLOAD. The smart contract collects the results and returns $[\mathbf{F_{v1}}, \ldots, \mathbf{F_{vp}}]$, $[\mathbf{F_{u1}}, \ldots, \mathbf{F_{up}}]$, $[\mathbf{F_{z1}}, \ldots, \mathbf{F_{zp}}]$, and $[\mathbf{F_{s1}}, \ldots, \mathbf{F_{sp}}]$ to the user by calling the function RETURN RESULT. Meanwhile, the user locally computes $\mathbf{F_{r1}}$ $\mathbf{F_{r2}}$ and $\mathbf{F_{r1}^-}$ which are Fourier transform of $\mathbf{r_1}$ and $\mathbf{r_2}$ and inverse Fourier transform of $\mathbf{r_3}$, respectively. Because there is only one nonzero coefficient in $\mathbf{r_1}$, as well as $\mathbf{r_2}$and $\mathbf{r_3}$, we simplify the DFT/IDFT, as shown in the DFTRV/IDFTRV functions in Algorithm 6.

(3) The user verifies equations (3) and (4). If they are valid, the user computes $\mathbf{F_a} = \sum_{i-1}^{p} \mathbf{F_a} + \mathbf{F_{r1}} + \mathbf{F_b}$, $\mathbf{F_b} = \sum_{i-1}^{p} \mathbf{F_{zi}} + \mathbf{F_{r2}}$, and $\mathbf{F_c} = \mathbf{F_a} \circ \mathbf{F_b}$:

$$\sum_{i-1}^{p} \mathbf{F}_{\mathbf{v_i}} \stackrel{?}{=} \sum_{i-1}^{p} \mathbf{F}_{\mathbf{u_i}}, \tag{3}$$

$$\sum_{i-1}^{p} \mathbf{F}_{\mathbf{z_i}} \stackrel{?}{=} \sum_{i-1}^{p} \mathbf{F}_{\mathbf{s_i}}. \tag{4}$$

(4) The user generates $\mathbf{D} = T(\mathbf{F}_c, \mathbf{r}_3)$ and $\mathbf{E} = T(\mathbf{F_c}, \mathbf{r}_3)$ and uploads them with the task requirement (i.e., IFFT) to smart contract, by the function TASK UPLOAD. Calling the function TASK ALLOCATION, the smart contract distributes the vectors in $\mathbf{D}$ and $\mathbf{E}$ to $2p$ computational nodes.

(5) After receiving the vectors, the computational nodes perform the inverse Fourier transform on the received vectors and return $[\mathbf{F^-_{d1}}, \ldots, \mathbf{F^-_{dp}}]$ and $[\mathbf{F^-_{e1}}, \ldots, \mathbf{F^-_{ep}}]$ to smart contract by the function TASK UPLOAD. Smart contract returns them to the user by calling the function RETURN RESULT.

(6) The user computes$\mathbf{c} = \mathbf{F^-_{rs}} + \sum_{i=1}^{p} \mathbf{F^-_{d_i}}$, selects two integers $m, l \in_R \{0, \ldots, 2n-2\}$ randomly, and verifies equations (5)–(7). If they are valid, the computing succeeds; otherwise, the computing fails. The user sends a message to the smart contract. Then, the latter calls the function VERIFICATION AND PAYMENT:

$$\sum_{i=1}^{p} \mathbf{F^-_{d_i}} \stackrel{?}{=} \sum_{i=1}^{p} \mathbf{F^-_{e_i}}, \tag{5}$$

$$\sum_{i=0}^{p} a_i b_{l-i} \stackrel{?}{=} \sum_{i=0}^{2n-2} W^{-li}_{2n-1} \mathbf{F_c}[i], \tag{6}$$

$$\sum_{i=0}^{2n-2} W^{mi}_{2n-1} c_i \stackrel{?}{=} \mathbf{F_c}[m]. \tag{7}$$

In Algorithm 6, if any verification fails, the user will report a cheating and the algorithm will come to an end.

Figure 2 demonstrates the procedures and data communications in the six steps of Algorithm 6.

5.1.2. Correctness and Complexity. Because $\sum_{i=0}^{p} \mathbf{v_i} + \mathbf{r}_1 = \mathbf{a}$ and $\sum_{i=0}^{p} \mathbf{z_i} + \mathbf{r}_2 = \mathbf{b}$ and the Fourier transform is a linear transform, we can have $\sum_{i=0}^{p} \mathbf{F}_{\mathbf{v}_1} + \mathbf{F}_{\mathbf{r}_1} = \mathbf{F_a}$ and $\sum_{i=1}^{p} \mathbf{F}_{\mathbf{z}_1} + \mathbf{F}_{\mathbf{r}_2} = \mathbf{F_b}$. We get $\mathbf{F_c} = \mathbf{F_a} \circ \mathbf{F_b}$. Because of the convolution theorem, $\mathbf{F^-_{F_c}} = \mathbf{F^-_{F_a \circ F_b}} = \mathbf{a} \times \mathbf{b} = \mathbf{c}$.

Using the DFTRV/IDFTRV in Algorithm 6, computing Fourier transform of r_1, r_2 and inverse Fourier transform of r_s needs 4_n multiplications. Because of the characteristics of $\mathbf{r}_1$,$\mathbf{r}_2$, and $\mathbf{r}_3$, only one multiplication is needed to compute each term in $\mathbf{F}_{\mathbf{r}_1}$, $\mathbf{F}_{\mathbf{r}_2}$, and $\mathbf{F^-_{rs}}$. Computing $\mathrm{F_a}$ and $\mathrm{F_b}$ needs $2pn$ additions. Computing $\mathbf{F_c}$ needs $2n$ multiplications. The verification of equations (3)–(5) takes $6(p-1)n$ additions. $6(p-1)n$ additions are needed to compute c. The final verification (equations (6) and (7)) needs $l + 2n$ multiplications. To sum up, we need $l + 8n$ multiplications and $10pn - 6n$ additions. The local complexity of multiplication in this algorithm is $O(n)$, and the local complexity of addition is $O(n)$. Therefore, the local complexity of this algorithm is $O(n)$.

5.1.3. Security against Passive Attackers. A participant may be a passive attacker. Passive attackers will follow the scripts of the algorithm while exploiting the intermediate information to breach the privacy of polynomials. In the following, we analyze the security of our algorithm against passive attackers.

The algorithm should protect the privacy of $f(x)$, $g(x)$, $\mathbf{c}$, and $\mathbf{F_c}$. As it is known to all, when all nodes collude, the passive attackers can get the most information, and the security of privacy is the lowest.

Since the operations on $f(x)$ and $g(x)$ are consistent, the risks of privacy leakage of them are the same. We analyze the security of $f(x)$ in the worst case, i e., collusion of all nodes. When all the computational nodes collude, they can guess a set of values $[a_0', a_1', \ldots, a_{n-1}']$, in which $n-1$ values are consistent with the true coefficients of $f(x)$, while one value is not. Because of $\mathbf{r}_1$, they even do not know the position of the false value. They still have to make a brute-force guessing. If $a_i \in D$, where the base of D is m, the attackers should traverse all possibilities by taking m different values for each coefficient. In this case, the attackers have to make m^n attempts to get $f(x)$. However, $a_i \in Z$ in FHEHIL. Then, $m \longrightarrow \infty$, and the attackers cannot get $f(x)$.

$\mathbf{F_c}$ and $\mathbf{c}$ are also privacy-protected. For the security of $\mathbf{F_c}$, when all the computational nodes collude, the passive attackers can guess a set of values $F_c^1[0]$, $F_c^1[1]$, $\ldots, F_c^1[2n-2]$, in which $2n-2$ values are consistent with the true coefficients of $\mathbf{F_c}$, while one value is not. However, the existence of $\mathbf{r}_3$ shows that passive attackers do not know the position of the false value. The only way to attack is by making a brute-force guessing. Same as $\mathbf{a}$ and $\mathbf{b}$, the domain of the coefficients of $\mathbf{F_c}$ is infinite. Therefore, the attackers cannot get $\mathbf{F_c}$. For the security of $\mathbf{c}$, on the one hand, the attackers cannot compute $\mathbf{c}$ using inverse Fourier transform without knowing $\mathbf{F_c}$. On the other hand, it is easy to see that when lacking $\mathbf{F^-_{ra}}$, attackers cannot get $\mathbf{c}$ which is equal to $\sum_{i=1}^{p} \mathbf{F^-_{di}} + \mathbf{F^-_{rs}}$.

5.1.4. Security against Active Attackers. A participant may also be an active attacker. Active attackers will inject false computations into the algorithm to tamper with the whole process. In the following, we analyze the security of our algorithm against active attackers.

Active attackers may return forged values to damage computing. To damage computing without being detected, attackers prefer to make minimal changes on results. In Algorithm 6, it is easy to see that the lowest risk way for computational nodes to cheat is to tamper with only one item of the results returned to the user, while the other items are correct.

There is one way to cheat in the process of securely outsourcing Fourier transform. For example, the nodes of

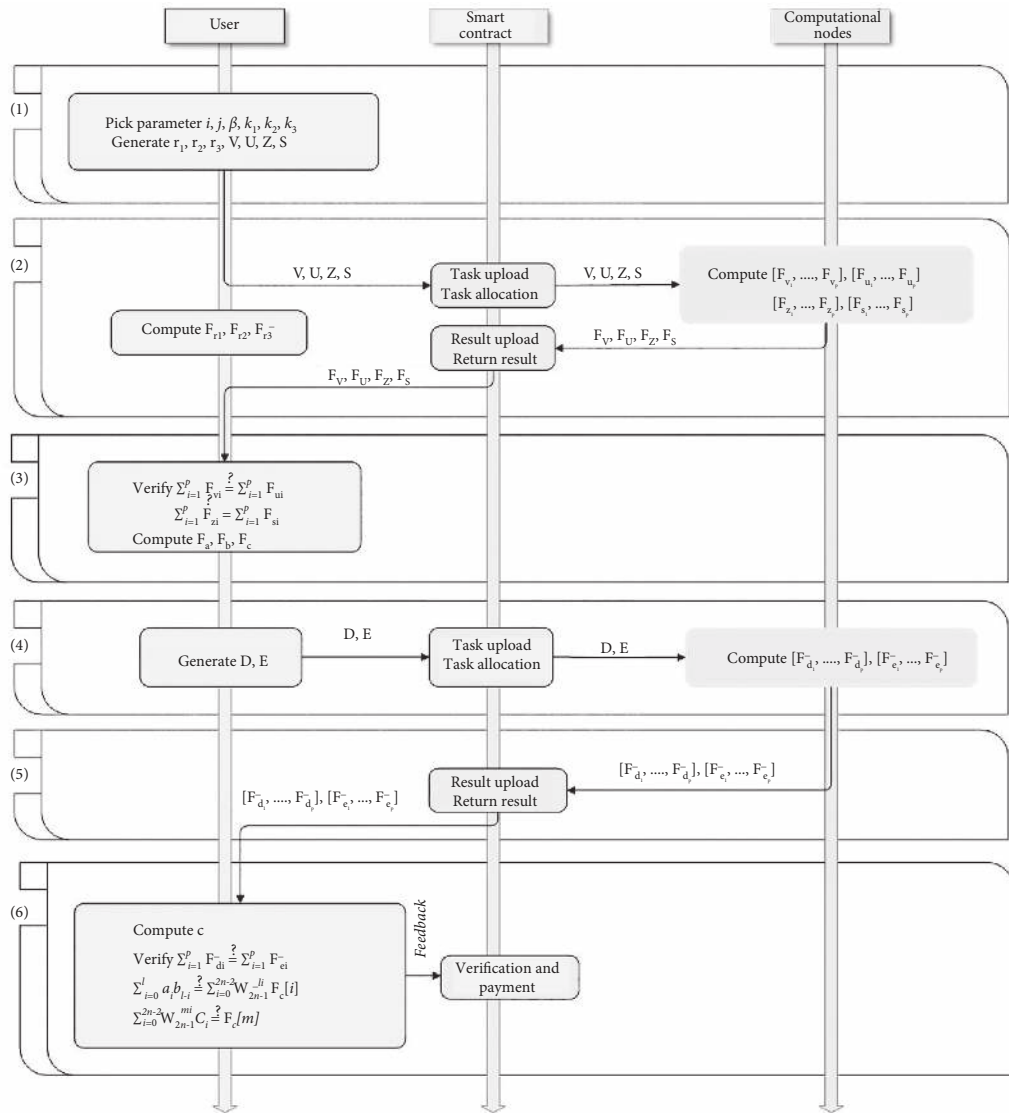

Figure 2: Blockchain-based secure outsourcing of polynomial multiplication.

computing the DFT of $f(x)$ perform honestly, while the nodes of computing the DFT of $g(x)$ do not. One node n_j changed the i^{th} term in $\mathbf{F}_{\mathbf{z}_j}$ and another node n_k' also changed the i^{th} term in $\mathbf{F}_{\mathbf{s}_k}$. This way of cheating can nullify the verification at equations (3) and (4) and damage the result of $\mathbf{F_b}$, whereas the error in the i^{th} term of $\mathbf{F_b}$ will be propagated to $F_c[i]$ and every term of c. Equation (6) computes the correct $c_1 = \sum_{i=0}^{l} a_i b_{l-i}$ and the false $c_l' = \sum_{i=0}^{2n-2} W_{2n-1}^{-li} \mathbf{F_c}[i]$. Since $\mathbf{F_c}[i]$ is false, c_l' is not equal to c_l for any random l. The verification at equation (6) can certainly detect this cheating.

There is the other way to cheat in the process of securely outsourcing the inverse DFT of $\mathbf{F_c}$. One node n_j changed the i^{th} term in $\mathbf{F}^-_{\mathbf{d}_j}$ and another node n_k' also changed the i^{th} term in $\mathbf{F}^-_{\mathbf{e}_k}$. In this way, we can nullify the verification at equation (9) and return a false item c_i. The false item c_i causes that $F_c'[m] = \sum_{i=0}^{2n-2} W_{2n-1}^{mi} c_i$ is also a false value in equation (7). $\mathbf{F_c'}[\mathbf{m}]$ will not be equal to $\mathbf{F_c}[m]$, for all $0 \le m \le 2n-2$. The verification at equation (7) can certainly detect this way of cheating.

5.2. Secure Outsourcing of Modular Exponentiation. To the secure outsourcing of modular exponentiation, we extend the algorithm in [3] and apply it to the blockchain. In our extension, six modular exponentiation pairs are outsourced to six computational nodes, instead of a single cloud, aiming to protect against possible attacks on small discrete logarithms. The process is shown as Algorithm 7. The input is two integers. The output is u^d, where u is the base and d is the exponent. In a similar way to Figure 2, Algorithm 7 can be implemented in the framework of Blockchain-based computation outsourcing, but we omit it due to limitation of space.

5.2.1. Correctness and Complexity. It is easy to prove that the algorithm is correct from equations (8) and (9). In the process of parameter generation, there are two exponentiations, two divisions, and two multiplications. Two exponentiations and six multiplications are involved during the verification. Compared with the exponentiation, the

Input: u, d
Output: u^d
(1) The user generates random parameters $g_1, g_2, e, k_1, k_2 \in Z$,
(2) and computes $v_1 \longleftarrow g_i^e, v_2 \longleftarrow g_2^e, w_1 \longleftarrow (u/g_1), w_2 \longleftarrow (u/g_2)$,
(3) $t_1 \longleftarrow d - k_1 e, l_1 \longleftarrow d - k_2 t_1$;
(4) The user uploads $(k_1, v_1), (k_1, v_2), (l_1, w_1), (k_2, w_1), (l_1, w_2), (k_2, w_2)$ to the smart contract;
(5) The smart contract distributes $(k_1, v_1), (k_1, v_2), (l_1, w_1), (k_2, w_1), (l_1, w_2), (k_2, w_2)$ to 6 computational nodes;
(6) The computational nodes compute b^a after receiving (a, b) and return results to the smart contract;
(7) The user gets $v_1^{k_1}, v_2^{k_1}, w_1^{l_1}, w_1^{k_2}, w_2^{l_1}, w_2^{k_2}$ from the smart contract;
(8) The user verifies $v_1^{k_1} w_1^{l_1} (g_1 w_1^{k_2})^{t_1} = v_2^{k_1} w_2^{l_1} (g_2 w_2^{k_2})^{t_1}$.
(9) If the verification is valid, $u^d \longleftarrow v_1^{k_1} w_1^{l_1} (g_1 w_1^{k_2})^{t_1}$

Algorithm 7: Secure outsourcing of modular exponentiation.

complexity of multiplication and division can be ignored. However, in the algorithm, the exponents are e and t_1 which are much smaller than the original exponent d through the transformation of $t_1 = d - k_1 e$. Therefore, local complexity will be greatly reduced:

$$v_1^{k_1} w_1^{l_1} \left(g_1 w_1^{k_2}\right)^{t_1} = v_1^{k_1} g_1^{t_1} w_1^{l_1} \left(w_1^{k_2}\right)^{t_1} = v_1^{k_2} g_1^{t_2} w_1^{l_1 + k_2 t_1} = g_1^d w_1^d = v^d, \tag{8}$$

$$v_2^{k_1} w_2^{l_1} \left(g_2 w_2^{k_2}\right)^{t_1} = v_2^{k_1} g_2^{t_1} w_2^{l_1} \left(w_2^{k_2}\right)^{t_1} = v_2^{k_2} g_2^{t_2} w_2^{l_1 + k_2 t_1} = g_2^d w_2^d = v^d. \tag{9}$$

5.2.2. Security. The only way to pass the verification is that the six computational nodes perform correctly. The forged results of active attackers cannot pass the verification in Step 8. The user only needs to know whether the results are correct or not, and the smart contract can detect the cheating nodes according to the records.

We analyze the security against passive attackers in the worst case, i e., the conspiring of six computational nodes. The exponents k_1, l_1, and k_2 are visible for attackers, while the other exponents t_1, d, and e are not. The bases v_1, v_2, w_1, and w_2 are visible for attackers, while g_1, g_2, and e are not. We discover that the privacy of u may leak in [3], which sends six pairs to a single node in the cloud. In [3], the base and exponent are about 1000 bit, while the parameters including g_1, g_2, e, k_1, and k_2 are only 64-bit long, to reduce the overhead of local computation. The shorter bit length of parameters may promote an easier attack on the small discrete logarithms. In this kind of attack, an attacker in the cloud can exhaust x so that $w_1^* * v_1 = w_2^* * v_2$. Then, e is breached. The attacker then exhausts g, satisfying $g^e = v_1$. Finally, the cloud can obtain u by $w_1 * g$.

We solve this attack by distributing six modular exponentiation pairs to six computational nodes, which increases the difficulty of the above attack.

6. Results and Discussion

In this section, we conduct three types of experiments. Firstly, we evaluate the efficiency of the secure outsourcing of polynomial multiplication in various numbers of polynomial multiplications and compare it with the traditional nonoutsourcing method using FFT. Secondly, we evaluate the efficiency of the secure outsourcing of polynomial multiplication by varying the numbers of items and bit length of coefficients and also compare it with the nonoutsourcing method. Finally, we complete the secure outsourcing for FHEHIL in blockchain, analyze the time consumption of each step, and compare it with the nonoutsourcing method. The experiments are simulated on two machines with Intel Core i7 processor running at 2.90 GHz and 16G memory as a cloud server and Intel Core i5 processor running at 1.80 GHz and 8G memory as a local user. The communication bandwidth is 20 Mbps.

6.1. The Evaluation of Secure Outsourcing of Polynomial Multiplication. We make experiments to evaluate the efficiency of the secure outsourcing algorithm for polynomial multiplication. We implement this experiment using Python3 language.

We compare the secure outsourcing of polynomial multiplication with the nonsourcing algorithm on time consumption with different numbers of polynomial multiplication, in which $n = 1024$, $p = 3$, and all the coefficients are 512-bit long. As demonstrated in Figure 3, it is easy to see that when the number of polynomial multiplications is less than 60, the efficiency of the outsourcing scheme is lower than the nonoutsourcing scheme due to the communication time consumption. However, when the number of polynomial multiplications increases above 60, the efficiency of the outsourcing scheme becomes higher than the nonoutsourcing scheme. When the number of polynomial multiplications is less than 300, the bottleneck of the outsourcing scheme is the time consumption on nodes' computations and interactions. When the number of polynomial

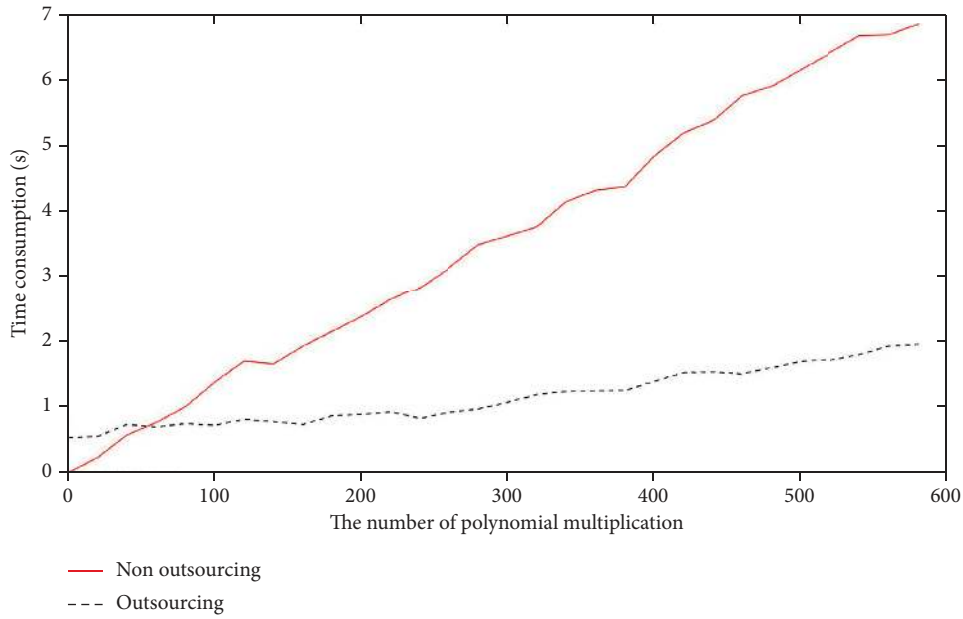

Figure 3: Comparison of time consumption on the outsourcing and nonoutsourcing scheme of polynomial multiplications.

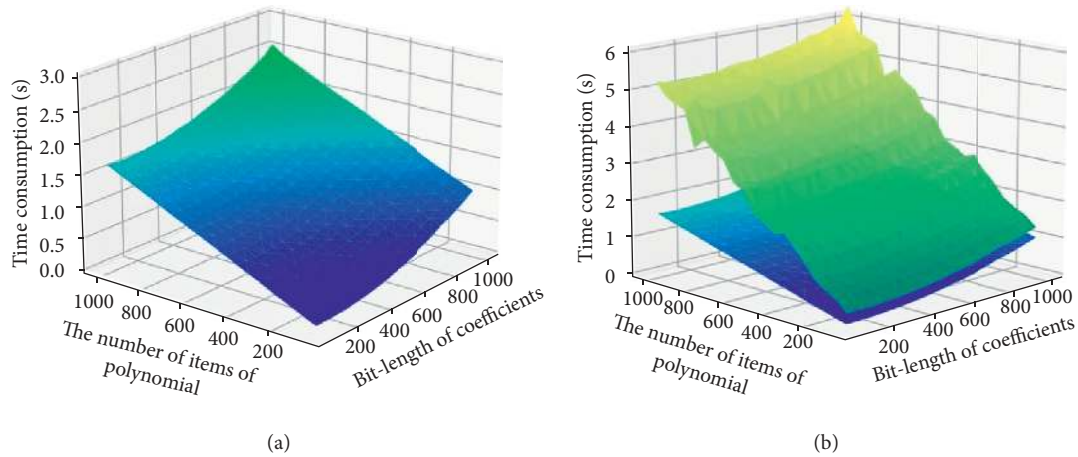

Figure 4: Comparison of time consumption on the outsourcing and nonoutsourcing scheme of polynomial multiplications.

multiplications becomes larger, the bottleneck is the time consumption on local computation.

We make another type of experiments to analyze the influence of number of terms and bit length of coefficients on the efficiency of secure outsourcing of polynomial multiplication. We count the time consumption of 400 random polynomial multiplications, with the number of polynomial items varying from 50 to 1000, and the item bit length varying from 50 to 1000. Figure 4(a) demonstrates that the time consumption increases with the increase of items of polynomials and bit length of coefficients. Moreover, the number of terms has a more obvious effect on time consumption. Besides, compared with the nonoutsourcing polynomial multiplication, our method always has a higher efficiency under all scales of data, as shown in Figure 4(b).

6.2. The Evaluation of Blockchain-Based Secure Outsourcing Scheme of Fully Homomorphic Encryption Using Hidden Ideal Lattice. We employ the relevant security parameters recommended in [2], i.e., $n = 1024$, $t = 310$, and $p = 3$. Our outsourcing scheme consists of the local user's program and the computational nodes' program. Our outsourcing scheme is compared with the nonoutsourcing scheme. The programs are written in Python3, and the smart contract based on the Ethereum platform is written in Solidity. The smart contract interacts with computational nodes' program and local program by the interface provided by Web3.

Figure 5 demonstrates the running time at all stages of the two schemes. This figure does not display the time consumption on generating parameters in the FHEHIL because that is not what we are improving. In the process of

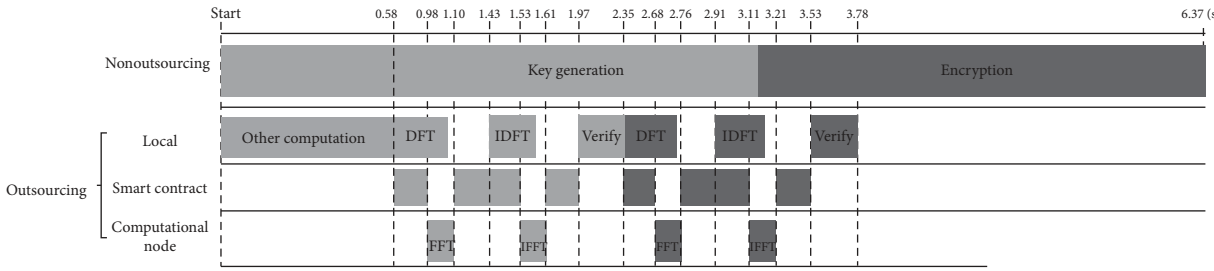

Figure 5: Time consumption on stages of the outsourcing and nonoutsourcing schemes.

Table 3: Details of time consumption in key generation.

	Verification (s)	DFTRV/IDFTRV (s)	Communication (s)	Others (s)	FFT/IFFT (s)
User	0.37	0.89	0.46	0.58	0
Smart contract	0	0	1.01	0	0
Computational nodes	0	0	0.52	0	0.19

Table 4: Details of time consumption in encryption.

	Verification (s)	DFTRV/IDFTRV (s)	Communication (s)	Others (s)	FFT/IFFT (s)
User	0.24	0.93	0.43	0.11	0
Smart contract	0	0	0.99	0	0
Computational nodes	0	0	0.37	0	0.20

computing w efficiency is slightly improved. Compared with the non-sourcing scheme, our scheme saves about 2.6 s. The overall time consumption is improved by about 40.7% (the unmarked areas in Figure 5 are the communication time consumption for interacting with the blockchain). Table 3 shows the detailed time consumption of different entities (user, smart contract, and computational nodes) in different stages (verification, communication, DFTRV/IDFTRV, FFT/IFFT, and other computations) for Key Generation. Table 4 shows the detailed time consumption of different entities in different stages for encryption.

The time consumption of decryption is not shown in Figure 5. Since there is only one polynomial multiplication, the time consumption of communication is dominant in the process of decryption, as illustrated in Figure 3. Therefore, the time consumption of outsourcing decryption (0.379 s) is larger than the nonoutsourcing decryption (0.103 s).

7. Conclusions

In this paper, we propose a secure outsourcing algorithm for polynomial multiplication that reduces the local complexity to $O(n)$. According to security analysis, our algorithm is secure against passive and active attackers. We also propose a framework for blockchain-based computation outsourcing. It has a credit-based task allocation strategy, which significantly reduces the probability of failed computations. Using this framework, we implement the secure outsourcing of FHEHIL, in which the basic computations including polynomial multiplication and modular exponentiation can be securely outsourced by our proposed algorithms. The security analysis and experimental results show that our proposed outsourcing schemes are secure and efficient. In the future, we will apply the secure outsourcing of FHEHIL into some practical secure computation problems, such as the millionaire problem, and set operation problems.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

The conference version of this paper has been published in the 21st International Conference on Parallel and Distributed Computing, Applications, and Technologies (PDCAT 2020).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Key-Area Research and Development Program of Guangdong Province (No. 2020B010164003), the Science and Technology Program of Guangzhou, China (No. 201904010209), and the Science and Technology Program of Guangdong Province, China (No. 2017A010101039).

References

[1] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: outsourcing computation to untrusted workers," in *Proceedings of the Advances in Cryptology-CRYPTO 2010*, pp. 465–482, Santa Barbara, CA, USA, August 2010.

[2] T. Plantard, W. Susilo, and Z. Zhang, "Fully homomorphic encryption using hidden ideal lattice," *IEEE transactions on information forensics and security*, vol. 8, no. 12, pp. 2127–2137, 2013.

[3] A. Fu, S. Li, S. Yu, Y. Zhang, and Y. Sun, "Privacy-preserving composite modular exponentiation outsourcing with optimal checkability in single untrusted cloud server," *Journal of Network and Computer Applications*, vol. 118, pp. 102–112, 2018.

[4] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Proceedings of the Annual Cryptology Conference*, pp. 868–886, Santa Barbara, CA, USA, August 2012.

[5] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[6] G. Craig, S. Amit, and B. Waters, "Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based," in *Proceedings of the Annual Cryptology Conference*, pp. 75–92, Santa Barbara, CA, USA, August 2013.

[7] Y. Su, B. Yang, C. Yang, and L. Tian, "Fpga-based hardware accelerator for leveled ring-lwe fully homomorphic encryption," *IEEE Access*, vol. 8, pp. 168008–168025, 2020.

[8] F. Chen, T. Xiang, and Y. Yang, "Privacy-preserving and verifiable protocols for scientific computation outsourcing to the cloud," *Journal of Parallel and Distributed Computing*, vol. 74, no. 3, pp. 2141–2151, 2014.

[9] X. Chen, W. Susilo, D. S. Wong, J. Ma, S. Tang, and Q. Tang, "Efficient algorithms for secure outsourcing of bilinear pairings," *Theoretical Computer Science*, vol. 562, pp. 112–121, 2015.

[10] B. Kang, M. Lee, and J. Park, "Efficient delegation of pairing computation," *International Association of Cryptologic Research*, vol. 259, 2005.

[11] Y. Ren, N. Ding, T.-Y. Wang, H. Lu, and D. Gu, "New algorithms for verifiable outsourcing of bilinear pairings," *Science China Information Sciences*. vol. 59, no. 9, Article ID 99103, 2016.

[12] S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations," in *Proceedings of the Theory of Cryptography Conference*, pp. 264–282, Cambridge, MA, USA, February 2005.

[13] X. Chen, L. Jin, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations,," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2386–2396, 2013.

[14] Y. Ren, N. Ding, X. Zhang, H. Lu, and D. Gu, "Verifiable outsourcing algorithms for modular exponentiations with-improved checkability," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 293–303, NewYork, NY, USA, May 2016.

[15] Q. Zhou, C. Tian, H. Zhang, Y. Jia, and F. Li, "How to securely outsource the extended euclidean algorithm for large-scale polynomials over finite fields," *Information Sciences*, vol. 512, pp. 641–660, 2020.

[16] D. Harvey, J. Van Der Hoeven, and G. Lecerf, "Faster polynomial multiplication over finite fields," *Journal of the Association for Computing Machinery*, vol. 63, no. 6, p. 52, 2016.

[17] D. Harvey and J. van der Hoeven, "Faster polynomial multiplication over finite fields using cyclotomic coefficient rings," *Journal of Complexity*, vol. 54, Article ID 101404, 2019.

[18] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 10, pp. 2459–2463, 2019.

[19] H. J. Hsu and M. D. Shieh, "Vlsi architecture of polynomial multiplication for bgv fully homomorphic encryption," in *Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, Monterey, CL, USA, October 2020.

[20] P. Giorgi, B. Grenet, and D. S. Roche, "Generic reductions for in-place polynomial multiplication," 2019.

[21] V. Nakos, "Nearly optimal sparse polynomial multiplication," 2019, https://arxiv.org/abs/1901.09355.

[22] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," Technical report, Springer, Berlin, Germany, 2019.

[23] C. Lin, D. He, X. Huang, X. Xie, and K. -Kwang Raymond Choo, "Blockchain-based system for secure outsourcing of bilinear pairings," *Information Sciences*, vol. 527, pp. 590–601, 2020.

[24] H. Zheng, J. Shao, and G. Wei, "Attribute-based encryption with outsourced decryption in blockchain," *Peer-to-Peer Networking and Applications*, vol. 13, no. 5, pp. 1643–1655, 2020.

[25] H. Kun, J. Xin, Z. Wang, and G. Wang, "Outsourced data integrity verification based on blockchain in untrusted environment," *World Wide Web*, vol. 43, pp. 1–24, 2020.

[26] H. Wang, X. A. Wang, W. Wang, and S. Xiao, "A basic framework of blockchain-based decentralized verifiable outsourcing," in *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems*, pp. 415–421, Oita, Japan, September 2019.

[27] H. Gao, Z. Ma, S. Luo, and Z. Wang, "BFR-MPC: a blockchain-based fair and robust multi-party computation scheme," *IEEE Access*, vol. 7, pp. 110439–110450, 2019.

[28] M. Andrychowicz, S. Dziembowski, D. Malinowski, and K. Mazurek, "Secure multiparty computations on Bitcoin," *Communications of the ACM*, vol. 59, no. 4, pp. 76–84, 2016.

[29] Y. Zhang, R. H. Deng, X. Liu, and Z. Dong, "Blockchain based efficient and robust fair payment for outsourcing services in cloud computing," *Information Sciences*, vol. 462, pp. 262–277, 2018.

[30] Y. Zhang, R. H. Deng, X. Liu, and Z. Dong, "Outsourcing service fair payment based on blockchain and its applications in cloud computing," *IEEE Transactions on Services Computing*, vol. 73, p. 1, 2018.

[31] G. Craig and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Proeedings of the Annual international conference on the theory and applications of cryptographic techniques*, pp. 129–148, Tallinn, Estonia, May 2011.