future internet

MDPI

Article

Protecting Function Privacy and Input Privacy in the Publicly Verifiable Outsourcing Computation of Polynomial Functions

Beibei Song [1], Dehua Zhou [2,*], Jiahe Wu [1], Xiaowei Yuan [1], Yiming Zhu [1] and Chuansheng Wang [2]

[1] College of Cyber Security, Jinan University, Guangzhou 511436, China
[2] College of Information Science and Technology, Jinan University, Guangzhou 510632, China
* Correspondence: tzhoudh@jnu.edu.cn

Abstract: With the prevalence of cloud computing, the outsourcing of computation has gained significant attention. Clients with limited computing power often outsource complex computing tasks to the cloud to save on computing resources and costs. In outsourcing the computation of functions, a function owner delegates a cloud server to perform the function's computation on the input received from the user. There are three primary security concerns associated with this process: protecting function privacy for the function owner, protecting input privacy for the user and guaranteeing that the cloud server performs the computation correctly. Existing works have only addressed privately verifiable outsourcing computation with privacy or publicly verifiable outsourcing computation without input privacy or function privacy. By using the technologies of homomorphic encryption, proxy re-encryption and verifiable computation, we propose the first publicly verifiable outsourcing computation scheme that achieves both input privacy and function privacy for matrix functions, which can be extended to arbitrary multivariate polynomial functions. We additionally provide a faster privately verifiable method. Moreover, the function owner retains control over the function.

Keywords: outsourcing computation; function privacy; input privacy; publicly verifiable computation; homomorphic encryption; proxy re-encryption

Citation: Song, B.; Zhou, D.; Wu, J.; Yuan, X.; Zhu, Y.; Wang, C. Protecting Function Privacy and Input Privacy in the Publicly Verifiable Outsourcing Computation of Polynomial Functions. *Future Internet* **2023**, *15*, 152. https://doi.org/10.3390/fi15040152

Academic Editors: Weizhi Meng and Christian D. Jensen

Received: 23 March 2023
Revised: 10 April 2023
Accepted: 20 April 2023
Published: 21 April 2023

1. Introduction

With the explosive growth of data, the demand for data processing among users is constantly growing, which makes cloud services increasingly popular. These cloud services can help weak clients with limited resources store large-scale data or complete expensive computational tasks at a low cost. One of the most common applications of cloud services is outsourcing computation.

Consider a setting with three entities in the process of outsourcing the computation of functions as shown in Figure 1: a function owner, who possesses the function model and wants to outsource its computation to a cloud server; a series of request users, who need to obtain the function's results on the private inputs; and a cloud server that performs the computation of the function based on the inputs and returns the results to the request users. In this process, the cloud server is untrusted. It should not learn anything about either the input or the function. Additionally, the correctness of the results should be verified. Thus, there are three requirements that need to be considered: function privacy, input privacy and public verifiability.

For example, in a medical predication scenario, a doctor, as the function owner, has developed an expensive disease prediction function model; in addition, some patients, as the request users, want to predict the probability of contracting this disease with their personal health data. It would be stressful for the doctor to perform the computation locally, so he wants to outsource the computation of the function to a cloud server. The cloud server would then compute the prediction function based on the inputs from the patients and return the predicted results to them.

The function parameters in this process are that private data owned by the doctor should remain confidential from the cloud server. Similarly, the health data of the patients should also be kept confidential. Since the cloud server is unreliable, it may compress the function model or simplify the computation for its own benefit, which can lead to a biased predicted result. Therefore, a fundamental requirement is that the patients can verify the correctness of the computed results. However, in many scenarios, individual verification is not sufficient. For example, patients may need to convince their insurance institutions to accept verification, which means that private verification cannot meet this requirement. Hence, we emphasize public verification, which allows anyone to verify the results of an outsourced function without learning anything about the sensitive data.

Figure 1. Outsourcing computation model of functions.

To address these requirements, researchers have proposed a series of solutions, such as Homomorphic Encryption (HE) [1–3], Secure Multi-Party Computation (SMPC) [4–6] and Publicly Verifiable Computation (PVC) [7–9]. Among them, homomorphic encryption can perform computations on ciphertexts. Secure multi-party computation can enable multiple parties to jointly complete a computing task without revealing their secret information. Publicly verifiable computation can verify the correctness of the computation results. It is important to combine privacy protection with verifiable computation.

Many works have been devoted to addressing one or more of the requirements in outsourcing the computation of functions [7–19], but none have achieved all of the above-mentioned requirements simultaneously. Our work aims to fill this gap, and the contributions of this study are listed below:

- We propose the first publicly verifiable outsourcing computation scheme that achieves both function privacy and input privacy for matrix functions, which can be extended to arbitrary polynomial functions (the extension method is provided).
- We additionally provide a faster privately verifiable method, which allows the request user to verify the results more efficiently than with public verification.
- The function owner retains control over the function. If they do not allow a user to use their function model, the cloud server cannot complete the computation.

The remainder of this paper is organized as follows. We present a brief overview of the related works in Section 2. We define the notation and review the necessary preliminaries in Section 3. We provide the system model and design goals in Section 4. In Section 5, we provide our publicly verifiable outsourcing computation scheme with privacy measures, and in Section 6, we analyze the performance of our scheme. Finally, Section 7 contains our conclusions and future works.

2. Related Works

There have been many works addressing one or more of the requirements mentioned above in outsourcing function computation. The schemes in [10–12] realize function privacy and input privacy based on FHE [2] or GCs [4]. However, they can only achieve private verification and not public verification. Additionally, the expensiveness of cryptographic primitives, such as GCs and FHE, limits their practical relevance. The schemes in [13–15] use lightweight algorithms for encryption and decryption instead of FHE to construct schemes with input privacy and function privacy. They reduce the system costs but still do not realize public verification.

The scheme in [16] shares the function model and input data among multiple servers using additive secret sharing [20] rather than encryption, and the results are obtained through the cooperation of these servers. Although it protects both function privacy and input privacy, it only provides private verification and not public verification. It also requires a strong assumption of non-collusion for the servers or the private data will be leaked.

Some works construct publicly verifiable computation schemes for specific functions. For instance, the works in [7,17–19] have constructed publicly verifiable computation schemes that are efficient for specific functions, such as univariate polynomial functions and matrix functions. In [8], a publicly verifiable computation scheme for Boolean functions was constructed based on key policy attribute-based encryption (KP-ABE) [21]. However, this scheme does not offer input or function privacy. Moreover, ref. [9] proposed a publicly verifiable scheme with privacy protection, but it can only protect one element between function privacy and input privacy, i.e., either a publicly verifiable computation scheme with function privacy or a publicly verifiable computation scheme with input privacy.

Table 1 summarizes the requirements achieved in the surveyed literature, which can inform the research findings of our proposed scheme.

Table 1. Comparison of the proposed scheme with previous works.

Study Ref./No	Function Privacy/Input Privacy	Private Verification	Public Verification
[10–12]	✓/✓	✓	×
[13–15]	✓/✓	✓	×
[16]	✓/✓	✓	×
[7,8,17–19]	×/×	✓	✓
[9]	✓/ × or ×/✓	✓	✓
Our scheme	✓/✓	✓	✓

Note: ✓—supported; and ×—not supported.

To improve efficiency, our work employs a partial homomorphic encryption scheme. The cloud server needs to perform a transformation on the ciphertexts from two different clients before computing. Several works have addressed this transformation on ciphertexts. For instance, Peter et al. [22] proposed a scheme that requires a participant to possess a strong private key that can decrypt any ciphertext in order to complete the transformation on ciphertexts. However, this approach poses a threat to data privacy and makes the data owner lose the right to control their data.

In contrast, our work ensures that the function owner retains control over their function, thereby strengthening data privacy. In [23], Ximeng Liu et al. split the strong private key in the BCP [24] scheme among multiple participants to decentralize the decryption power of the strong private key. However, their approach did not consider how to split the key securely. Additionally, their ciphertext transformation requires more cooperation among participants, which increases communication costs and decreases computing efficiency.

In contrast, our approach involves fewer participants, has lower communication costs and offers higher computing efficiency. In [25], Yutaka Kawai et al. proposed a homomorphic proxy re-encryption scheme that supports homomorphic operations for re-encrypted ciphertexts. However, this scheme requires a large number of computations of the discrete logarithm and group operations in bilinear groups, which increases the computational cost. Our approach is based on a lean homomorphic proxy re-encryption scheme, which makes computations more efficient.

3. Preliminaries

In this section, we review the basic notations and the primitives that are treated in this paper. The basic notations are shown in Table 2.

Table 2. Summary of notations.

Notations	Descriptions
$\boldsymbol{x}$	The lowercase bold letters denote vectors where x_i indexes the i-th element
$\mathbf{M}$	The uppercase bold letters denote matrices where $M_{i,j}$ indexes the (i,j)-th element
$\boldsymbol{x}^\top$	The transpose of vector $\boldsymbol{x}$
$\boldsymbol{x} \dots \boldsymbol{y}$	The dot product of two vectors, $\boldsymbol{x} = (x_1, \dots, x_n)$ and $\boldsymbol{y} = (y_1, \dots, y_n)$, where $\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i$
$\mathbf{M} \cdot \boldsymbol{x}$	The matrix–vector multiplication of $\mathbf{M}$ and $\boldsymbol{x}$
$x \xleftarrow{\$} S$	Uniformly sample x from a set S at random.
$\mathbb{N}$	The set of all natural numbers
$[n]$	The set $\{1, \dots, n\}$, where $n \in \mathbb{N}$
λ	The security parameter
p, q	The odd primes
$\mathbb{Z}_N$	A group under addition modulo N
$\mathbb{Z}_N^*$	A group satisfies $gcd(b, N) = 1$, where $b \in [N-1]$
$\mathcal{F} = \mathbb{Z}_q^{m \times d}$	The set of all $m \times d$ matrices over $\mathbb{Z}_q$
$\mathbb{G} = \langle g \rangle$	The cyclic group with generator g
$\Phi_m(X)$	The m-th cyclotomic polynomial, where $\Phi_m(X) = X^N + 1$
$R := \mathbb{Z}[X]/\Phi_m(X)$	The ring of integer polynomials modulo Φ_m
R_p	The ring R with coefficients modulo p
χ	The discrete Gaussian distribution over the ring
(pk, sk)	The key pair of public key and secret key

3.1. Additive Homomorphic Encryption (Additive HE)

Rivest et al. first proposed homomorphic encryption (HE) in 1978 [1], and this encryption performs homomorphic operations on ciphertexts without decryption. This is equivalent to performing the corresponding operations on plaintext, ensuring data privacy while also maintaining data availability.

Depending on the supported mathematical operations, HE can be classified into partial (additive or multiplicative) homomorphic encryption (PHE) [26,27], fully homomorphic encryption (FHE) [2] and somewhat homomorphic encryption (SWHE) [28–30]. PHE supports only one type of operation (either additive or multiplicative) on the ciphertext. FHE supports both additive and multiplicative operations but with high computational costs. SWHE supports limited multiplication and addition operations, with lower computational costs compared with FHE.

The following describes the additive homomorphic encryption scheme that we use based on [24], which contains five algorithms: $HE = (Setup, KeyGen, Enc, Dec, Eval)$. The details are as follows:

- $Setup(\lambda) \rightarrow pp$: The security parameter λ is taken as input, and two λ-bit large prime numbers, $p^{'}, q^{'}$, are chosen. Then, we have $N = pq$, where $p = 2p^{'} + 1$ and $q = 2q^{'} + 1$. A generator $g \in \mathbb{Z}_{N^2}^*$ is randomly sampled, and then the public parameters $pp = (N, g)$ are output.
- $KeyGen(pp) \rightarrow (pk, sk)$: The public parameters pp are taken as input. According to $pp = (N, g)$, $a \xleftarrow{\$} \mathbb{Z}_{N^2}$ is randomly sampled. Then, the public/secret key pair $(pk = g^a \ mod \ N^2, sk = a)$ is output.
- $Enc(m, pk) \rightarrow c$: For a message $m \in \mathbb{Z}_N$, $r \xleftarrow{\$} \mathbb{Z}_{N^2}$ is sampled. The message m is encrypted with public key $pk = g^a$ and outputs ciphertext $c = (A, B)$, where $A = g^{ar} \ mod \ N^2$, $B = g^r(1 + mN) \ mod \ N^2$.
- $Dec(c, sk) \rightarrow m$: For the ciphertext $c = (A, B)$, secret key $sk = a$ is used for decryption, and the output message is $m = \frac{B/(A^{\frac{1}{a}}) - 1 \ mod \ N^2}{N}$.

- $Eval(c_1, c_2) \rightarrow c_1 \oplus c_2$: For two ciphertexts c_1 and c_2, whose corresponding plaintexts are m_1, m_2, where

$$\begin{aligned} c_1 &= Enc(m_1) = (A_1, B_1) = (g^{ar_1} \ mod \ N^2, \ g^{r_1}(1 + m_1 N) \ mod \ N^2) \quad (1) \\ c_2 &= Enc(m_2) = (A_2, B_2) = (g^{ar_2} \ mod \ N^2, \ g^{r_2}(1 + m_2 N) \ mod \ N^2), \quad (2) \end{aligned}$$

the additive homomorphism property is described as follows:

$$\begin{aligned} c_1 \oplus c_2 &= Enc(m_1) \oplus Enc(m_2) \\ &= (g^{ar_1} \cdot g^{ar_2}, g^{r_1}(1 + m_1 N) \cdot g^{r_2}(1 + m_2 N)) \ mod \ N^2 \\ &= (g^{a(r_1 + r_2)}, g^{r_1 + r_2}(1 + (m_1 + m_2)N)) \ mod \ N^2 \\ &\rightarrow Enc(m_1 + m_2, pk). \quad (3) \end{aligned}$$

In particular, given a constant γ, we have $Eval(c_1, \gamma) \rightarrow c_1^{\gamma}$, where

$$\begin{aligned} c_1^{\gamma} &= (g^{a\gamma r_1}, \ g^{\gamma r_1}(1 + m_1 N)^{\gamma}) \ mod \ N^2 \\ &= (g^{a\gamma r_1}, g^{\gamma r_1}(1 + \gamma m_1 N)) mod \ N^2 \\ &\rightarrow Enc(\gamma m_1, pk). \quad (4) \end{aligned}$$

3.2. Linear Homomorphic Encryption (LHE)

Linear homomorphic encryption (LHE) [28–30] can support limited additive and multiplicative operations on ciphertexts. It is able to compute a linear combination of multiple ciphertexts, which has practical applications.

Given n plaintexts $(m_1, \ldots, m_n)$, the corresponding ciphertext is $(c_1, \ldots, c_n)$. Given n constant coefficients $(\gamma_1, \ldots, \gamma_n)$, the ciphertext of the linear combination $\sum_{i=1}^{n} \gamma_i \cdot m_i$ can be computed by $\{c_i\}_{i=1}^{n}$ and $\{\gamma_i\}_{i=1}^{n}$. Next, we describe the LHE algorithms based on the private key version by Brakerski and Vaikuntanathan [3]. It contains five algorithms: $LHE = (Setup, KeyGen, Enc, Dec, Eval)$. The details are as follows:

- $Setup(\lambda) \rightarrow pp$: The security parameter λ is taken as input. Two λ-bit large prime numbers q and $p \in \mathbb{Z}_q^*$ are chosen. Set $n = 2^{\lfloor \log \lambda \rfloor - 1}$, and there is a cyclotomic polynomial $f(x) = x^n + 1$. χ is the discrete Gaussian distribution over the ring $R_q = \mathbb{Z}_q[x]/\langle f(x) \rangle$. The public parameters pp={n, f, q, χ} are obtained as outputs.
- $KeyGen(pp) \rightarrow sk$: The public parameter pp is taken as input. A ring element $s \xleftarrow{\$} \chi$ is a sample, and the secret key $sk = s$ is obtained as an output.
- $Enc(sk, m) \rightarrow c$: For a message $m \in R_p$, $a \xleftarrow{\$} R_q, e \xleftarrow{\$} \chi$ is taken as a sample, and the output ciphertext is $c = (u, v) = (as + pe + m, -a)$.
- $Dec(sk, c) \rightarrow m$: The ciphertext $c = (u, v)$ is decrypted with the secret key $sk = s$, and the output $m = (u + v \cdot \text{sk}) \bmod p$ is obtained.
- $Eval((\gamma_1, \ldots, \gamma_d), ((c^{(1)}, \ldots, c^{(d)})) \rightarrow c$: Given d coefficients $\gamma_1, \ldots, \gamma_d \in \mathbb{Z}_q$ and d ciphertexts $c^{(1)}, \ldots, c^{(d)} \in (R_q)^2$, the linear homomorphism property is described as follows:

$$\begin{aligned} c &= \sum_{i=1}^{d} \gamma_i \cdot c^{(i)} = \sum_{i=1}^{d} \gamma_i \cdot (u, v)^{(i)} \\ &= (\sum_{i=1}^{d} \gamma_i \cdot u^{(i)}, \sum_{i=1}^{d} \gamma_i \cdot v^{(i)}) \\ &= (\sum_{i=1}^{d} \gamma_i a \cdot s + \sum_{i=1}^{d} \gamma_i pe + \sum_{i=1}^{d} \gamma_i m_i, -\sum_{i=1}^{d} \gamma_i a) \\ &\rightarrow Enc(sk, \sum_{i=1}^{d} \gamma_i m_i). \quad (5) \end{aligned}$$

3.3. Homomorphic Proxy Re-Encryption (HPRE)

Blaze, Bleumer and Strauss first proposed proxy re-encryption (PRE) [31] at EUROCRYPT 1998. The proxy can convert the ciphertext encrypted by the data owner into another ciphertext that can be decrypted by the specified data receiver. Homomorphic proxy re-encryption was introduced in [25], which combines the homomorphic property of HE and the "key-switching" property of PRE.

The following describes our homomorphic proxy re-encryption method, which consists of six algorithms: $PRE = (Setup, KeyGen, Enc, ReKeyGen, ReEnc, Dec)$. It is based on the additive homomorphic encryption above, and the details are as follows:

- $Setup(\lambda) \rightarrow pp$: The security parameter λ is taken as input. $HE.Setup(\lambda) \rightarrow pp$ is run to generate $pp = (N, g)$. Then, the output $pp = (N, g)$ is obtained.
- $KeyGen(pp) \rightarrow ((pk_i, sk_i), (pk_j, sk_j))$: The public parameter pp is taken as input. $HE.KeyGen(pp) \rightarrow (pk, sk)$ is run twice to generate $(pk_i = g^a \ mod \ N^2, sk_i = a)$, $(pk_j = g^b \ mod \ N^2, sk_j = b)$, which stands for the public/secret key pair of the data owner and data receiver, respectively. Then, the output $((pk_i, sk_i), (pk_j, sk_j))$ is obtained.
- $Enc(m, pk_i) \rightarrow c_i$: For a message $m \in \mathbb{Z}_N$, the data owner runs $HE.Enc(m, pk_i) \rightarrow c_i$ to output the ciphertext $c_i = (A_i, B_i)$, where $A_i = g^{ar} \ mod \ N^2$, $B_i = g^r(1 + mN) \ mod \ N^2$. Then, the output is c_i.
- $ReKeyGen(sk_i, sk_j) \rightarrow rk_{i \rightarrow j}$: The data owner's secret key sk_i and the data receiver's secret key sk_j are taken as input, and the re-encryption key $rk_{i \rightarrow j} = \frac{sk_j}{sk_i} = \frac{b}{a}$ is the output.
- $ReEnc(rk_{i \rightarrow j}, c_i) \rightarrow c_j$: The re-encryption key $rk_{i \rightarrow j}$ and the ciphertext c_i are taken as inputs, and the re-encryption ciphertext c_j is obtained as the output, where

$$\begin{aligned} c_j &= (A_j, B_j) = (A_i^{rk_{i \rightarrow j}}, B_i) \\ &= (g^{ar\frac{b}{a}}, g^r(1 + mN)) \ mod \ N^2 \\ &= (g^{br}, g^r(1 + mN)) \ mod \ N^2 \\ &\rightarrow Enc(m, pk_j). \end{aligned} \tag{6}$$

- $Dec(c_j, sk_j) \rightarrow m$: The ciphertext $c_j = (A_j, B_j) = (g^{br}, g^r(1 + mN))$ is decrypted with the data receiver's secret key $sk_j = b$, and the output is $m = \frac{B_j / (A_j{}^{\frac{1}{b}}) - 1 \ mod \ N^2}{N}$.

3.4. Verifiable Matrix-Vector Multiplication (VerM)

Verifiable computation is a computational model that allows the client to delegate computational tasks to a computational entity and then to verify the correctness of the computed result without re-executing the entire computation task. The schemes in [32,33] propose a critical observation of matrix–vector multiplication, and the scheme in [9] proposes a solution for publicly verifiable matrix–vector multiplication computation.

The following describes the publicly verifiable matrix–vector multiplication scheme that we use based on [9], which consists of four algorithms: $VerM = (Setup, Prepare, Compute, Verify)$. Table 3 shows the data size in the scheme, and the details are given below:

- $Setup(\lambda) \rightarrow pp$: The security parameter λ is taken as input, and a λ-bit large prime number q is chosen. Let $\mathbb{Z}_q$ be a finite field and $\mathbb{G} = \langle g \rangle$ be a cyclic group of generator g and order q. Let $m, d > 0$ be integers, and then output $pp = (q, m, d, g)$.
- $Prepare(\mathbf{F}) \rightarrow \boldsymbol{ver}$: The function owner takes their matrix $\mathbf{F} \in \mathcal{F}$, where $\mathcal{F} = \mathbb{Z}_q^{m \times d}$ is the set of all $m \times d$ matrices over $\mathbb{Z}_q$ as input to generate the public verification data $\boldsymbol{ver}$ (once and for all). A vector $\boldsymbol{t} = (t_1, \ldots, t_m) \xleftarrow{\$} \mathbb{Z}_q^m$ is uniformly sampled, and $\mathbf{s} = (s_1, \ldots, s_d) = \boldsymbol{t} \cdot \mathbf{F}$ is computed. The obtained output is $\boldsymbol{ver} = (\boldsymbol{T}, \boldsymbol{S})$, where $\boldsymbol{T} = g^{\boldsymbol{t}} = (g^{t_1}, \ldots, g^{t_m})$, $\boldsymbol{S} = g^{\boldsymbol{s}} = (g^{s_1}, \ldots, g^{s_d})$.

- $Compute(\mathbf{F}, \boldsymbol{x}) \rightarrow \boldsymbol{y}$: The function $\mathbf{F} \in \mathcal{F}$ and the input $\boldsymbol{x} = (x_1, \ldots, x_d)^\top \in \mathbb{Z}_q^d$ are taken as inputs. The result $\boldsymbol{y} = \mathbf{F} \cdot \boldsymbol{x} = (y_1, \cdots, y_m)^\top$ is obtained.
- $Verify(\mathbf{ver}, \boldsymbol{x}, \boldsymbol{y}) \rightarrow (0, 1)$: The public verification data $\boldsymbol{ver}$, the input $\boldsymbol{x}$ and the result $\boldsymbol{y}$ are taken as input. This convinces the user to accept the result if and only if

$$\prod_{i=1}^{d} g^{s_i \cdot x_i} = \prod_{i=1}^{m} g^{t_i \cdot y_i} \tag{7}$$

If the equation is valid, then the output is 1; otherwise, the output is 0.

The correctness of the verification is as follows:
The left side:

$$\prod_{i=1}^{d} g^{s_i \cdot x_i} = \prod_{i=1}^{d} g^{(\sum_{j=1}^{m} t_j \mathbf{F_{j,i}}) x_i} = g^{\sum_{i=1}^{d} \sum_{j=1}^{m} t_j \mathbf{F_{j,i}} x_i}. \tag{8}$$

The right side:

$$\prod_{i=1}^{m} g^{t_i \cdot y_i} = \prod_{i=1}^{m} g^{t_i \cdot (\sum_{j=1}^{d} \mathbf{F_{i,j}} x_j)} = g^{\sum_{i=1}^{m} \sum_{j=1}^{d} t_j \mathbf{F_{i,j}} x_j}. \tag{9}$$

If $\boldsymbol{y} = \mathbf{F} \cdot \boldsymbol{x}$, then it is easy to verify that Equation (7) holds.

Table 3. Data size.

Data	Size
Matrix $\mathbf{F}$	$m \times d$
Verification data $\boldsymbol{t}$	$1 \times m$
Verification data $\boldsymbol{s}$	$1 \times d$
Vector $\boldsymbol{x}$	$d \times 1$
The final result $\boldsymbol{y}$	$m \times 1$

4. System Model and Design Goals

4.1. System Model

Consider a setting in our system model with three entities: a function owner who wants to outsource the function's computation; two non-collusive cloud service providers CSP_1 and CSP_2 that collaborate to complete the computation; and a series of request users who want to obtain the computation results on their inputs. Figure 2 outlines a rough sketch of our system model. It consists of the following three phases: Preparation, Computation and Verification. The number before the text corresponds to Figure 2.

- Preparation:
  - (1) The function owner encrypts the function parameters and prepares the public verification data. Then, they upload the function's ciphertext to the cloud service provider CSP_1.
  - (2) A request user encrypts their input when they request CSP_1 to perform the computation.
  - (3) When CSP_1 receives the request from the user, it asks the function owner for permission.
  - (4) The function owner generates a temporary public/secret key pair and a re-encryption key when they permit the use of their function.
  - (5) The function owner sends the temporary public key and the re-encryption key to CSP_1 and sends the temporary secret key to CSP_2.

- Computation:
  - (6) CSP_1 re-encrypts the original function ciphertext with a mask. Then, the new ciphertext can be decrypted by CSP_2 with the temporary secret key. CSP_1 then collaborates with CSP_2 to complete the computation.
  - (7) CSP_1 returns the result ciphertext to the user.
- Verification:
  - (8) The public verification is performed using the public verification data, the input ciphertext and the result ciphertext. Simultaneously, the user decrypts the result ciphertext and verifies the result privately in a fast way with the public verification data and the plaintexts of their input and result.

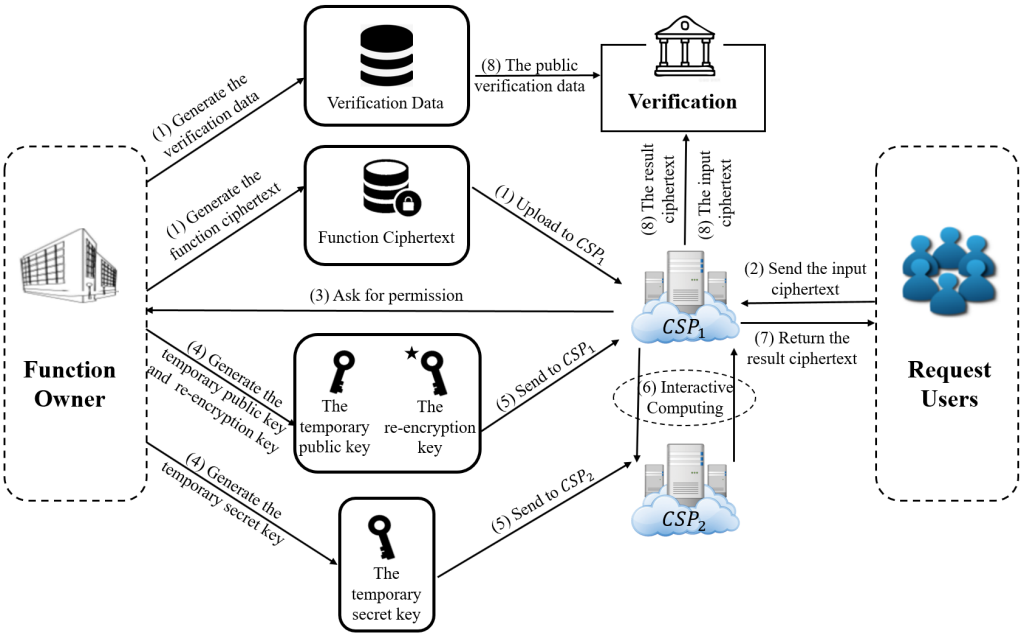

Figure 2. The publicly verifiable outsourcing computation model with privacy.

4.2. Design Goals

There are five goals that need to be achieved in our model:

- Maintain function privacy: The function ciphertext and public verification data cannot reveal any information about the function, so the encryption schemes that we use should be semantically secure so that no adversary can infer any information from the obtained ciphertext.
- Maintain input privacy. Similarly, the input ciphertext must not reveal information about the input. The encryption scheme for the input, thus, needs to be semantically secure.
- Maintain result privacy. No one should be able to obtain information about the result plaintext except for the request user. The result ciphertext cannot be distinguished from the random elements.
- Achieve public verification. Anyone can complete the verification with the publicly available information, and there is no malicious CSP that can persuade the public to accept an incorrect result with a non-negligible probability. We assume that the public has limited computational resources, and so verification must be efficient.
- Ensure the function owner's control over the function. CSPs cannot complete the computation if the function owner does not allow the user to use their function model.

5. The Proposed Solution

We propose a scheme for the publicly verifiable outsourcing of computation with both input privacy and function privacy for matrix–vector multiplication in this section. The scheme can also be extended to various polynomial functions.

We provide the extension method in Section 5.4. In our setting, the function owner has a matrix $\mathbf{F} \in \mathbb{Z}_q^{m \times d}$, and the request user has a vector $\boldsymbol{x}$. The user requests access from the cloud server for the result $\boldsymbol{y} = \mathbf{F} \cdot \boldsymbol{x}$. Our scheme consists of three phases: Preparation, Computation and Verification, which are specified below.

5.1. Preparation

1. The key pair of preparation steps for the function owner:
   - Run $HE.Setup(\lambda) \rightarrow pp_{he}$ to generate $pp_{he} = (N, g)$.
   - Run $HE.KeyGen(pp_{he}) \rightarrow (pk_f, sk_f)$ to generate (pk_f, sk_f)=(g^a, a).
2. The function owner encrypts the matrix and uploads it to CSP_1:
   - For each element f_i in the matrix $\mathbf{F} \in \mathbb{Z}_q^{m \times d}$, $0 < i \leq m \times d$, run $HE.Enc(pk_f, f_i) \rightarrow ct_i$ to generate $ct_i = (g^{ar_i}, g^{r_i}(1 + f_i N))$, where $r_i \xleftarrow{\$} \mathbb{Z}_{N^2}$.
   - Upload $\boldsymbol{ct} = (ct_1, \ldots, ct_{m \cdot d})$ to CSP_1.
3. The function owner generates the verification data $\boldsymbol{ver}$:
   - Run $VerM.Prepare(\mathbf{F}) \rightarrow \boldsymbol{ver}$ to generate $\boldsymbol{ver} = (\boldsymbol{T}, \boldsymbol{S})$. It holds that $\boldsymbol{T} = g^{\boldsymbol{t}} = (g^{t_1}, \ldots, g^{t_m})$, $\boldsymbol{S} = g^{\boldsymbol{s}} = (g^{s_1}, \ldots, g^{s_d})$, where $\boldsymbol{t} \xleftarrow{\$} \mathbb{Z}_q^m$ and $\boldsymbol{s} = (s_1, \ldots, s_d) = \boldsymbol{t} \cdot \mathbf{F}$.
4. The request user encrypts the input vector and sends it to CSP_1:
   - Run $LHE.Setup(\lambda) \rightarrow pp_{lhe}$ to generate the public parameters $pp_{lhe} = (n, f, q, \chi)$ for the request user.
   - Run $LHE.KeyGen(pp_{lhe}) \rightarrow sk$ to generate the secret key $sk = s$ for the request user.
   - For each x_i in the input vector $\boldsymbol{x} = (x_1, \ldots, x_d)^\top, i \in [d]$, $x_i \in \mathbb{Z}_p[x]/\langle f(x) \rangle$, run $LHE.Enc(sk, x_i) \rightarrow c_i$ to generate the ciphertext $c_i = (u_i, v_i) = (a_i s + pe_i + x_i, -a_i)$, where $a_i \xleftarrow{\$} R_q, e_i \xleftarrow{\$} \chi$ and $c_i \in R_q^2$.
   - The user sends the ciphertext $\boldsymbol{c} = (c_1, \ldots, c_d)^\top$ to CSP_1 and requests computation.
5. CSP_1 asks the function owner for permission.
6. The function owner will generate a temporary key pair and a re-encryption key if permission is granted:
   - The function owner runs $HE.KeyGen(pp_{he}) \rightarrow (pk_f^{'}, sk_f^{'})$ again to generate a temporary public/secret key pair $(pk_f^{'} = g^{a^{'}}, sk_f^{'} = a^{'})$.
   - Run $PRE.ReKeyGen(sk_f, sk_f^{'}) \rightarrow rk_{f \rightarrow f'}$ to generate a re-encryption key $rk_{f \rightarrow f'} = \frac{sk_f^{'}}{sk_f} = \frac{a^{'}}{a}$.
   - The function owner sends $pk_f^{'}$ and $rk_{f \rightarrow f'}$ to CSP_1 and sends $sk_f^{'}$ to CSP_2.

5.2. Computation

1. CSP_1 re-encrypts the matrix ciphertext:
   - For each element ct_i in the matrix ciphertext $\boldsymbol{ct}$, $0 < i \leq m \times d$, CSP_1 runs $PRE.ReEnc(rk_{f \rightarrow f'}, ct_i) \rightarrow ct_i^{'}$ to re-encrypt ct_i into $ct_i^{'}$, where $ct_i = (A, B) = (g^{ar_i}, g^{r_i}(1 + f_i N))$, $rk_{f \rightarrow f'} = \frac{a^{'}}{a}$. Then, $ct_i^{'} = (A^{rk_{f \rightarrow f'}}, B) = (g^{a^{'} r_i}, g^{r_i}(1 + f_i N)) \, mod \, N^2$.

2. CSP_1 sends the re-encryption ciphertext with a mask to CSP_2:
   - For each element $ct_i^{'}$ in $\boldsymbol{ct}^{'} = (ct_1^{'}, \ldots, ct_{m\cdot d}^{'})^{\top}$, $0 < i \leq m \cdot d$, CSP_1 samples $r_i^{'} \xleftarrow{\$} \mathbb{Z}_{N^2}$ as the mask.
   - For each mask $r_i^{'}$, run $HE.Enc(r_i^{'}, pk_f^{'}) \rightarrow rct_i^{'}$ with the temporary key $pk_f^{'}$ to generate $rct_i^{'} = (g^{a^{'}\hat{r}_i}, g^{\hat{r}_i}(1 + r_i^{'}N))mod\ N^2$, where $\hat{r}_i \xleftarrow{\$} \mathbb{Z}_{N^2}$.
   - For $i \in [m \cdot d]$, run $HE.Eval(ct_i^{'}, rct_i^{'}) \rightarrow ct_i^{'} \oplus rct_i^{'}$ to generate the masked ciphertext $mct_i^{'} = rct_i^{'} \oplus ct_i^{'} \rightarrow Enc(f_i + r_i^{'})$.
   - CSP_1 sends the masked ciphertext $\boldsymbol{mct}^{'} = \{mct_i^{'}\}_{i=1}^{m\times d}$ and the user's input ciphertext $\boldsymbol{c} = \{c_i\}_{i=1}^{d}$ to CSP_2.
3. CSP_1 computes the final result's first part $\boldsymbol{res^1}$:
   - Parse $\boldsymbol{r}^{'}$ as a $m \times d$-size matrix and write $r_{i,j}^{'}$ to index the (i,j)-th element.
   - For $i \in [m]$, run $LHE.Eval((r_{i,1}^{'}, \ldots, r_{i,d}^{'}), (c_1, \ldots, c_d)) \rightarrow res_i^1$ to generate $res_i^1 = \sum_{j=1}^{d} r_{i,j}^{'} c_j$. The first part of the final result is $\boldsymbol{res^1} = [res_1^1, ..., res_m^1]^T$.
4. CSP_2 performs the decryption for the masked ciphertext:
   - After receiving the masked ciphertext $\{mct_i^{'}\}_{i=1}^{m\times d}$, for $i \in [m \cdot d]$, CSP_2 runs $HE.Dec(mct_i^{'}, sk_f^{'}) \rightarrow f_i^{'}$ to generate the masked matrix parameters $\{f_i^{'} = f_i + r_i^{'}\}_{i=1}^{m\times d}$.
   - Parse $\boldsymbol{f}^{'} = (f_1^{'}, ..., f_{m\cdot d}^{'})$ as an $m \times d$-size matrix and write $f_{i,j}^{'}$ to index the (i,j)-th element. It holds that $f_{i,j}^{'}$=$f_{i,j} + r_{i,j}^{'}$.
5. CSP_2 computes the second part of the final result $\boldsymbol{res^2}$:
   - For $i \in [m]$, run $LHE.Eval((f_{i,1}^{'}, \ldots, f_{i,d}^{'}), (c_1, \ldots, c_d)) \rightarrow res_i^2$ to generate $res_i^2 = \sum_{j=1}^{d} f_{i,j}^{'} c_j$. Then, send the second part of the final result $\boldsymbol{res^2} = [res_1^2, ..., res_m^2]^T$ to CSP_1.
6. CSP_1 computes the final result $\boldsymbol{res}$ and returns it to the user:
   - For $i \in [m], CSP_1$ computes $res_i = res_i^2 - res_i^1 = \sum_{j=1}^{d} f_{i,j}^{'} c_j - \sum_{j=1}^{d} r_{i,j}^{'} c_j = \sum_{j=1}^{d} f_{i,j} c_j$. Then, the final result $\boldsymbol{res}$=$[res_1, \ldots, res_m]^T$ is returned to the user.

5.3. Verification

1. Public Verification:
   - The public runs $VerM.Verify(\boldsymbol{ver}, \boldsymbol{c}, \boldsymbol{res}) \rightarrow (0,1)$. It outputs 1 if and only if

$$\prod_{i=1}^{d} g^{s_i \cdot c_i} = \prod_{i=1}^{d} g^{t_i \cdot res_i}. \tag{10}$$

2. Private Verification:
   - After the user receives the result ciphertext $\boldsymbol{res}$=$(res_1, ..., res_m)^T$, for $i \in [m]$, they run $LHE.Dec(sk, res_i) \rightarrow y_i$ to generate the plaintext y_i, where $y_i = \sum_{j=1}^{d} f_{i,j} x_j$. The final result plaintext is $\boldsymbol{y} = (y_1, \cdots, y_m)^{\top} = \mathbf{F} \cdot \boldsymbol{x}$.
   - Run $VerM.Verify(\boldsymbol{ver}, \boldsymbol{x}, \boldsymbol{y}) \rightarrow (0,1)$. It outputs 1 if and only if

$$\prod_{j=1}^{d} g^{s_i \cdot x_i} = \prod_{j=1}^{d} g^{t_i \cdot y_i}. \tag{11}$$

5.4. The Extension Method for Arbitrary Polynomial Functions

In the previous subsection, we described the verifiable matrix–vector multiplication. However, the function may be arbitrary. The scheme in [9] gives a solution for outsourcing

these arbitrary polynomial functions based on the verifiable matrix–vector multiplication scheme.

This scheme decomposes the computation of arbitrary polynomials into a two-phase computation in which the heavy part is the matrix–vector multiplication that can be delegated to the cloud server, and the other part provides a substantially fast computation for the user. Then, we describe the decomposition scheme for univariate polynomials and arbitrary multivariate polynomials based on [9].

5.4.1. Univariate Polynomials

Let $\mathcal{F}$ be the set of all univariate polynomials of higher order over a finite field $\mathbb{Z}_q$. For any $f(x) \in \mathcal{F}$, suppose that $f(x) = f_0 + f_1 x + \ldots f_d x^d$. Let $m = \lceil \sqrt{d+1} \rceil$ and define

$$\mathbf{F} = \begin{pmatrix} f_0 & f_1 & \cdots & f_{m-1} \\ f_m & f_{m+1} & \cdots & f_{2m-1} \\ \vdots & \vdots & \cdots & \vdots \\ f_{m^2-m} & f_{m^2-m+1} & \cdots & f_{m^2-1} \end{pmatrix} \tag{12}$$

as a matrix of order m, where $f_i = 0$ for all $i > d$. Set $\boldsymbol{x} = (1, x, \ldots, x^{m-1})^\top$ and $\boldsymbol{y} = (1, x^m, \ldots, x^{m^2-m})$. Then, we have $f(x) = \boldsymbol{y} \cdot (\mathbf{F} \cdot \boldsymbol{x})$.

This is a two-phase computation: (i) the computation of $\boldsymbol{u} = \mathbf{F}\boldsymbol{x}$ and (ii) the computation of $f(x) = \boldsymbol{y} \cdot \boldsymbol{u}$. The first phase is a matrix–vector multiplication function that can be delegated to a cloud server using our verifiable computation scheme. It requires $O(m^2) = O(d)$ arithmetic operations. The second phase is a substantially fast computation for the user that requires only $O(m) = O(\sqrt{d})$ arithmetic operations locally.

5.4.2. Multivariate Polynomials

Let $\mathcal{F}$ be the set of all arbitrary multivariate polynomials over a finite field $\mathbb{Z}_q$. For any $f(x_1, \ldots, x_m) = \sum_{i_1,\ldots,i_m=1}^{d} f_{i_1,\ldots,i_m} \cdot x_1^{i_1} \cdots x_m^{i_m} \in \mathcal{F}$, $m \geq 2$, $l = \lfloor m/2 \rfloor$, parse the parameters as a $d^l \times d^{m-l}$ matrix

$$\mathbf{F} = \left(F_{(i_1,\ldots,i_l),(i_{l+1},\ldots,i_m)} \right) \tag{13}$$

where $F_{(i_1,\ldots,i_l),(i_{l+1},\ldots,i_m)} = f_{i_1,\ldots,i_m}$. Set $\boldsymbol{y} = \left(x_1^{i_1} \cdots x_l^{i_l} \right) \in \mathbb{Z}_q^{d^l}$ and $\boldsymbol{x} = \left(x_{l+1}^{i_{l+1}} \cdots x_m^{i_m} \right)^\top \in \mathbb{Z}_q^{d^{m-1}}$; then, we have $f(x_1, \ldots, x_m) = \boldsymbol{y} \cdot (\mathbf{F} \cdot \boldsymbol{x})$.

This is a two-phase computation: (i) the computation of $\boldsymbol{u} = \mathbf{F}\boldsymbol{x}$ and (ii) the computation of $f(x_1, \ldots, x_m) = \boldsymbol{y} \cdot \boldsymbol{u}$. The first phase is a matrix–vector multiplication that can be delegated to a cloud server using our verifiable computation scheme. It requires $O(d^m)$ arithmetic operations. The second phase is a substantially fast computation for the user that requires $O(d^{\lceil m/2 \rceil})$ arithmetic operations locally.

6. Results

6.1. Correctness Analysis

- Correctness of the result ciphertext ***res*** from CSP_1. Given the ciphertexts of the result ***res*** and the input vector $\boldsymbol{c}$, for $i \in [d]$, it holds that:

$$c_i = (u_i, v_i) = (a_i s + pe_i + x_i, -a_i) \tag{14}$$

$$res_i = \left(\sum_{j=1}^{d} f_{i,j} u_j, \sum_{j=1}^{d} f_{i,j} v_j \right)$$

$$= \left(\sum_{j=1}^{d} f_{i,j} a_i s + \sum_{j=1}^{d} f_{i,j} pe_i + \sum_{j=1}^{d} f_{i,j} x_i, -\sum_{j=1}^{d} f_{i,j} a_i \right) \tag{15}$$

Let $\sum_{j=1}^{d} f_{i,j} a_i = \hat{a}_i$ and $\sum_{j=1}^{d} f_{i,j} e_i = \hat{e}_i$, such that the following holds:

$$\begin{aligned} res_i &= \left(\hat{a}_i s + p\hat{e}_i + \sum_{j=1}^{d} f_{i,j} x_i, -\hat{a}_i \right) \\ &\rightarrow Enc\left(sk, \sum_{j=1}^{d} f_{i,j} x_i \right) \end{aligned} \tag{16}$$

We can see that the result ***res*** corresponds to the multiplication of matrix **F** and vector ***x*** in plaintext.

- Correctness of decryption for the result ciphertext ***res***. Given the ciphertext of the result ***res***, where

$$res_i = \left(\sum_{j=1}^{d} f_{i,j} u_j, \sum_{j=1}^{d} f_{i,j} v_j \right) = \left(\hat{a}_i s + p\hat{e}_i + \sum_{j=1}^{d} f_{i,j} x_i, -\hat{a}_i \right) \tag{17}$$

for $i \in [d]$, it holds that:

$$\begin{aligned} y_i &= \sum_{j=1}^{d} f_{i,j} u_j + \left(\sum_{j=1}^{d} f_{i,j} v_j \right) \cdot sk \ mod \ p \\ &= \hat{a}_i s + p\hat{e}_i + \sum_{j=1}^{d} f_{i,j} x_i - \hat{a}_i s \ \ mod \ p \\ &= \sum_{j=1}^{d} f_{i,j} x_i \end{aligned} \tag{18}$$

We can see that the decryption for the result ciphertext is correct.

- Correctness of public verification.
Given the input vector ciphertext ***c***, the result ciphertext ***res*** and the public verification data $\boldsymbol{ver} = (\boldsymbol{T}, \boldsymbol{S}) = ((g^{t_1}, \ldots, g^{t_m}), (g^{s_1}, \ldots, g^{s_d}))$, where

$$c_i = (u_i, v_i) \tag{19}$$

$$res_i = \left(\sum_{j=1}^{d} f_{i,j} u_j, \sum_{j=1}^{d} f_{i,j} v_j \right) \tag{20}$$

$$s_i = \sum_{j=1}^{m} t_j F_{\mathbf{j},\mathbf{i}} \tag{21}$$

for $i \in [d]$, it convinces the public to accept the result if and only if:

$$\prod_{i=1}^{d} g^{s_i \cdot c_i} = \prod_{i=1}^{d} g^{t_i \cdot res_i}. \tag{22}$$

The left side is:

$$\begin{aligned} \prod_{i=1}^{d} g^{s_i \cdot c_i} &= \left(\prod_{i=1}^{d} g^{s_i \cdot u_{\mathbf{i}}}, \prod_{i=1}^{d} g^{s_i \cdot v_i} \right) \\ &= \left(\prod_{i=1}^{d} g^{(\sum_{j=1}^{m} t_j f_{\mathbf{j},\mathbf{i}}) \cdot u_{\mathbf{i}}}, \prod_{i=1}^{d} g^{\sum_{j=1}^{m} t_j f_{\mathbf{j},\mathbf{i}} \cdot v_i} \right) \\ &= \left(g^{\sum_{i=1}^{d} \sum_{j=1}^{m} t_j f_{\mathbf{j},\mathbf{i}} u_i}, \ g^{\sum_{i=1}^{d} \sum_{j=1}^{m} t_j f_{\mathbf{j},\mathbf{i}} v_i} \right) \end{aligned} \tag{23}$$

and the right side is:

$$\begin{aligned}\prod_{i=1}^{d} g^{t_i \cdot res_i} &= \left(\prod_{i=1}^{d} g^{t_i \cdot \sum_{j=1}^{d} f_{i,j} u_j}, \prod_{i=1}^{d} g^{t_i \cdot \sum_{j=1}^{d} f_{i,j} v_j}\right) \\ &= \left(g^{\sum_{i=1}^{m} \sum_{j=1}^{d} t_j f_{i,j} u_j}, g^{\sum_{i=1}^{m} \sum_{j=1}^{d} t_j f_{i,j} v_j}\right) \end{aligned} \tag{24}$$

If $\boldsymbol{y} = \mathbf{F} \cdot \boldsymbol{x}$, then Equation (22) holds.

- Correctness of private verification.
  The correctness of private verification is consistent with the description provided in Section 3.4.

6.2. Security Analysis

In our two-server setup, we assume that CSP_1 and CSP_2 are non-collusive. Our security model allows for one or both servers to be malicious, as our verifiable computation scheme can detect the malicious behavior and reject the computation result. However, they cannot cooperate with each other. Under the non-collusion assumption, suppose one of the CSPs is malicious; then, the two CSPs cannot collaborate to perform the correct computation because the other CSP updates its data, causing the temporary ciphertext to not correspond to the temporary key. In this process, the malicious CSP does not learn any valuable information.

In our system, the function owner cannot obtain any additional information other than the function parameters. CSP_1 learns only the original ciphertext, the temporary ciphertext after re-encryption and the input ciphertext. CSP_2 learns only the temporary private key, the masked function parameters and the user's input ciphertext. The request user only learns its own input and the computation result. Any public verification party can only learn the input ciphertext, the result ciphertext and the public verification data.

Next, we present the security analysis of our encryption schemes.

- HE. The scheme is based on the hardness of the Decisional Composite Residuosity Assumption and Decisional Diffie–Hellman Assumption, which is stated below:

  Definition 1. *(Decisional Composite Residuosity Problem [26]) Given a composite n and an integer z, decide if z is an $n-$ residue modulo n^2 or not, namely, if there exists y such that $z = y^n (mod n^2)$.*

  Definition 2. *(Decisional Diffie–Hellman Problem [24]) Let g be an element of prime order in a cyclic group $\mathbb{G}$. Given $g^a, g^b, h \in \mathbb{G}$, decide whether or not $h = g^{ab}$.*

  Theorem 1. *The scheme is semantically secure if and only if the Decisional Composite Residuosity Assumption and Decisional Diffie–Hellman Assumption hold.*

- LHE. The security is established in [3], and we state it below:

  Theorem 2. *LHE is linearly homomorphic as long as* $\max\{|\gamma_1|, \ldots, |\gamma_d|\} \cdot prn^{1.5} < q/2$.

  Theorem 3. *Let* $r = \mathrm{poly}(n)$ *and* $q = 2^{n^\epsilon}$ *for some* $0 < \epsilon < 1$. *The scheme is semantically secure under the worst-case hardness of approximating the shortest vectors on ideal lattices to within a factor of* $O(2^{n^\epsilon})$.

- HPRE. It is based on the security of HE.
- VerM. It is based on the hardness of the Discrete Logarithm Assumption [9]. It will pass the verification by mistake with a probability of $1/q$ at most, which is negligible when q is a λ-bit prime. It is stated below:

Definition 3. *(Discrete Logarithm Problem) Let g be an element of prime order in a cyclic group $\mathbb{G}$. Given $h \in \mathbb{G}$, compute x such that $h = g^x$.*

Theorem 4. *The scheme is semantically secure if and only if the Discrete Logarithm Assumption holds.*

6.3. Experimental Results

To verify the efficiency, we performed our experiments on a personal computer with an AMD Ryzen 5 3600 3.6 GHz processor with 8 GB memory and running on the Ubuntu Desktop-20.04.1-LTS operating system using python language. We used the charm-crypto library to implement HE operations, PRE operations and VerM operations and used Microsoft SEAL to implement LHE operations. In the experiment, we set the security parameter of LHE to 4096 and that of HE to 1024. We tested the efficiency for the four entities when the matrix size was $(10 \times 10, 50 \times 50, 100 \times 100, 150 \times 150)$. The average time for each experiment was obtained by running it multiple times as shown in the graphs.

Figure 3 illustrates the time cost for the function owner, where the blue line presents the time for encrypting the matrix, the red line presents the time for preparing the generation of the verification data, and the gray line represents the time for the generation of the verification data. As shown in Figure 3, the function owner efficiently prepares the generation of verification data once for all parties. The primary overhead is the encryption of matrix parameters; however, since the function owner only needs to encrypt the function once for all parties, the computation cost can be amortized. Note that the vertical axis is on a logarithmic scale with a base of 10, and the same applies to Figures 4 and 5.

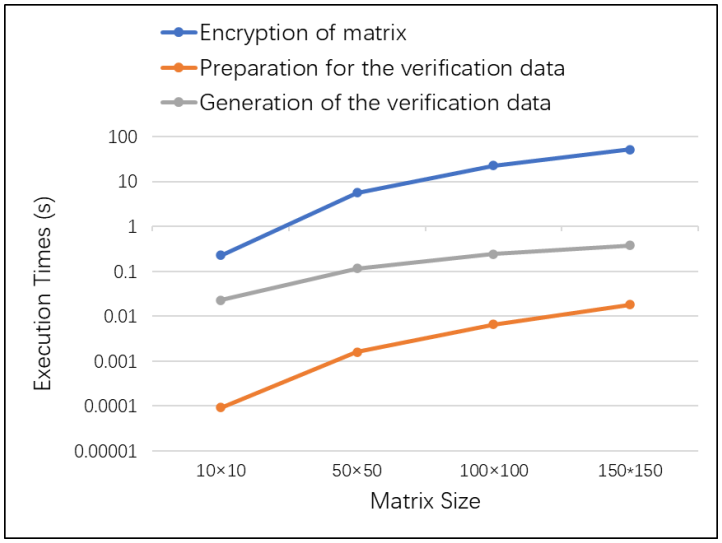

Figure 3. The time cost for the function owner.

Figure 4 illustrates the time cost for CSP_1. The blue line represents the time for re-encrypting the ciphertext, the red line represents the time for encrypting the mask, the gray line represents the time for generating the masked ciphertext, and the yellow line represents the time for computing the first part of the result ciphertext. As shown in Figure 4, the time for re-encryption and encrypting masks is nearly identical to that for the function owner to encrypt the matrix, and the homomorphic operations are highly efficient. The computation involving LHE ciphertexts is the most time-consuming, which would significantly burden the user if performed locally. Therefore, it was outsourced to the cloud server for computation.

Figure 5 illustrates the time cost for CSP_2, where the blue line represents the time for decrypting the masked ciphertext, and the red line represents the time computing the second part of the result ciphertext. As can be seen from Figure 5, the computation

with LHE ciphertexts is the most time-consuming, which is the same as CSP_1, so it was outsourced to the cloud server.

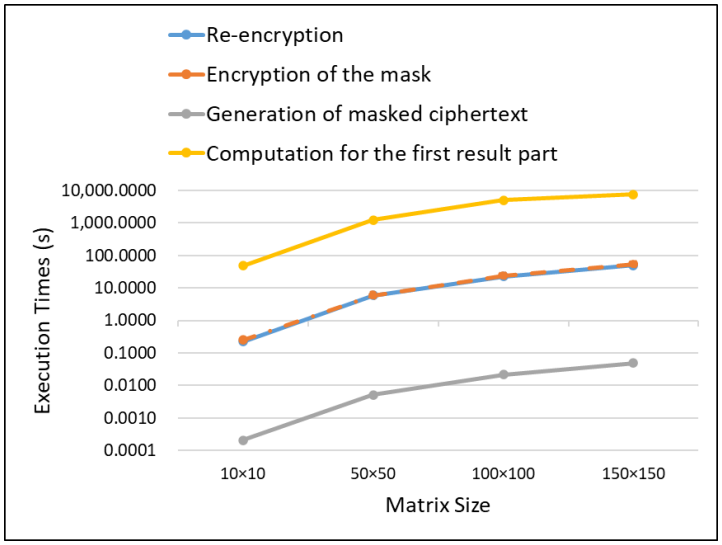

Figure 4. The time cost for CSP_1.

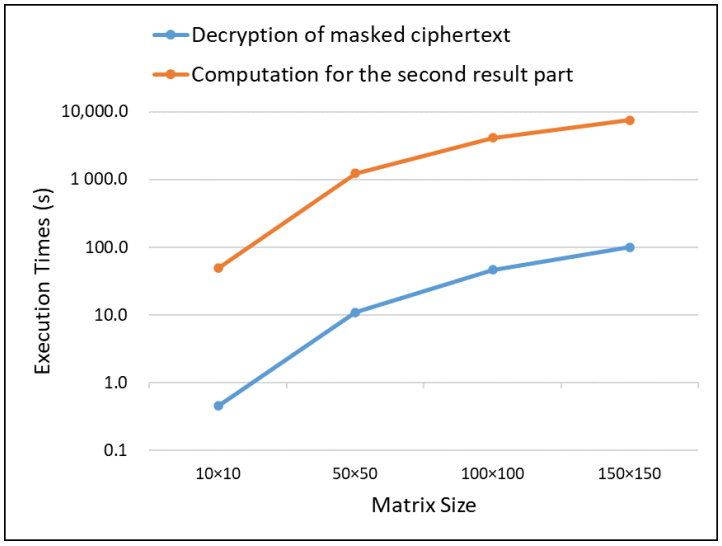

Figure 5. The time cost for CSP_2.

Figure 6 illustrates the time cost for the request user, where the blue line represents the time for encrypting their input vector and the red line represents the time for decrypting the final result ciphertext. As shown in Figure 6, the encryption and decryption for users are very efficient.

Figure 7 illustrates the time cost for verification, where the blue line represents the time for private verification, and the red line represents the time for public verification. As shown in Figure 7, the process of private verification is quick and efficient. Public verification, on the other hand, may take more time but is still accessible for the public.

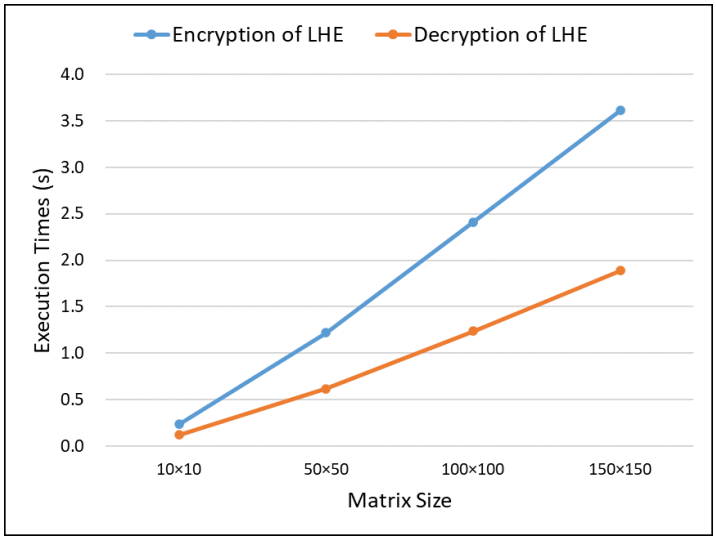

Figure 6. The time cost for the request user.

Figure 7. The time cost for verification.

Next, we present the communication costs in Table 4, where the matrix size is $m \times d$, N is the parameter in HE, and R_p is the ring in LHE.

Table 4. Communication Costs.

Sender	Receiver	Phase	Communication Cost
Function owner	CSP_1	Preparation	$2 \cdot (m \times d) \cdot \mathbb{Z}^*_{N^2}$ (Once for all)
Function owner	The public	Preparation	$(m + d) \cdot \mathbb{Z}_q$ (Once for all)
Function owner	CSP_1	Preparation	$1 \cdot \mathbb{Z}_{N^2} + 1 \cdot \mathbb{Z}^*_{N^2}$
Function owner	CSP_2	Preparation	$1 \cdot \mathbb{Z}_{N^2}$
Request User	CSP_1	Preparation	$d \cdot (R_q)^2$
CSP_1	Function owner	Preparation	Permission Message
CSP_1	CSP_2	Computation	$2 \cdot (m \times d) \cdot \mathbb{Z}^*_{N^2} + d \cdot (R_q)^2$
CSP_1	Request User	Computation	$m \cdot (R_q)^2$
CSP_1	The public	Verification	$(d + m) \cdot (R_q)^2$
CSP_2	CSP_1	Computation	$m \cdot (R_q)^2$

Compared with [9], our scheme achieves both function privacy and input privacy. In terms of computational complexity, our scheme is similar to their second scheme with only input privacy for the cloud server side, but we have two servers, which requires twice the computational overhead. At the same time, the two servers need to communicate with each other, which incurs additional communication overhead, but these additional costs provide privacy for the function.

7. Conclusions and Future Work

In this paper, we focused on the privacy-preserving and publicly verifiable outsourcing computation of matrix functions and polynomial functions. Our publicly verifiable computation scheme achieved both input privacy and function privacy for matrix functions, and it can be extended to arbitrary polynomial functions. Our scheme additionally provides a faster privately verifiable method and ensures the function owner's control over access to the function. Our solution may be of interest to applications in oblivious polynomial evaluation or polynomial prediction model inference.

In the future, we plan to construct a publicly verifiable outsourcing computation scheme for nonlinear functions, such as for activation functions in convolutional neural networks. Furthermore, we will address how to identify malicious cloud servers if the verification fails.

Author Contributions: Conceptualization, B.S. and D.Z.; methodology, B.S.; software, J.W. and X.Y.; validation, J.W., X.Y. and B.S.; formal analysis, B.S., J.W. and X.Y.; investigation, B.S. and Y.Z.; resources, B.S. and Y.Z.; data curation, J.W. and X.Y.; writing—original draft preparation, B.S.; writing—review and editing, Y.Z., D.Z. and C.W.; visualization, B.S. and J.W.; supervision, D.Z. and C.W.; project administration, D.Z. and C.W.; funding acquisition, D.Z. and C.W. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was funded by the National Natural Science Foundation of China (Nos. U2001205, 61732021,61932010), Guangdong Basic and Applied Basic Research Foundation (Nos. 2019B030302008, 2023B1515040020), and TESTBED2 (No. H2020-MSCA-RISE-2019).

Data Availability Statement: All data are presented in the main text.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rivest, R.L.; Adleman, L.; Dertouzos, M.L. On data banks and privacy homomorphisms. *Found. Secur. Comput.* **1978**, *4*, 169–180.
2. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, 31 May–2 June 2009; Mitzenmacher, M., Ed.; ACM: New York, NY, USA, 2009; pp. 169–178.
3. Brakerski, Z.; Vaikuntanathan, V. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In Proceedings of the Advances in Cryptology—CRYPTO 2011—31st Annual Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2011; Rogaway, P., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6841, pp. 505–524. [CrossRef]
4. Yao, C.C. How to generate and exchange secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science, Toronto, ON, Canada, 27–29 October 1986; IEEE Computer Society: Washington, DC, USA, 1986; pp. 162–167.
5. Demmler, D.; Schneider, T.; Zohner, M. ABY—A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In Proceedings of the 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, CA, USA, 8–11 February 2015; The Internet Society: Reston, VA, USA, 2015.
6. Feng, D.; Yang, K. Concretely efficient secure multi-party computation protocols: Survey and more. *Secur. Saf.* **2022**, *1*, 2021001. [CrossRef]
7. Papamanthou, C.; Shi, E.; Tamassia, R. Signatures of Correct Computation. In Proceedings of the Theory of Cryptography—10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, 3–6 March 2013; Volume 7785, pp. 222–242.
8. Parno, B.; Raykova, M.; Vaikuntanathan, V. How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption. In Proceedings of the Theory of Cryptography—9th Theory of Cryptography Conference, TCC 2012, Taormina, Italy, 19–21 March 2012; Volume 7194, pp. 422–439.
9. Zhang, L.F.; Safavi-Naini, R. Protecting data privacy in publicly verifiable delegation of matrix and polynomial functions. *Des. Codes Cryptogr.* **2020**, *88*, 677–709. [CrossRef]

10. Applebaum, B.; Ishai, Y.; Kushilevitz, E. From Secrecy to Soundness: Efficient Verification via Secure Computation. In Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming, ICALP 2010, Bordeaux, France, 6–10 July 2010; Springer: Berlin/Heidelberg, Germany, 2010; Part I; Volume 6198, pp. 152–163.
11. Barbosa, M.; Farshim, P. Delegatable Homomorphic Encryption with Applications to Secure Outsourcing of Computation. In Proceedings of the Topics in Cryptology—CT-RSA 2012—The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, 27 February–2 March 2012; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7178, pp. 296–312.
12. Chung, K.M.; Kalai, Y.T.; Vadhan, S.P. Improved Delegation of Computation using Fully Homomorphic Encryption. In *Advances in Cryptology—CRYPTO 2010, Proceedings of the 30th Annual Cryptology Conference, Santa Barbara, CA, USA, 15–19 August 2010*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6223, pp. 483–501.
13. Fiore, D.; Gennaro, R.; Pastro, V. Efficiently Verifiable Computation on Encrypted Data. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; ACM: New York, NY, USA, 2014; pp. 844–855.
14. Joo, C.; Yun, A. Homomorphic authenticated encryption secure against chosen-ciphertext attack. In Proceedings of the Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, 7–11 December 2014; Springer: Berlin/Heidelberg, Germany, 2014; Part II 20; pp. 173–192.
15. Libert, B.; Peters, T.; Joye, M.; Yung, M. Linearly Homomorphic Structure-Preserving Signatures and Their Applications. In Proceedings of the Advances in Cryptology—CRYPTO 2013—33rd Annual Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2013; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2013; Part II, Volume 8043, pp. 289–307.
16. Zhang, L.F. Multi-server verifiable delegation of computations: Unconditional security and practical efficiency. *Inf. Comput.* **2021**, *281*, 104740. [CrossRef]
17. Catalano, D.; Fiore, D.; Gennaro, R.; Vamvourellis, K. Algebraic (trapdoor) one-way functions and their applications. In Proceedings of the Theory of Cryptography: Tenth Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, 3–6 March 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 680–699.
18. Elkhiyaoui, K.; Önen, M.; Azraoui, M.; Molva, R. Efficient Techniques for Publicly Verifiable Delegation of Computation. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, 30 May–3 June 2016; ACM: New York, NY, USA, 2016; pp. 119–128.
19. Fiore, D.; Gennaro, R. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In Proceedings of the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, 16–18 October 2012; ACM: New York, NY, USA, 2012; pp. 501–512. [CrossRef]
20. Beimel, A. Secure Schemes for Secret Sharing and Key Distribution. Ph.D. Thesis, Technion—Israel Institute of Technology, Haifa, Israel, 1996.
21. Goyal, V.; Pandey, O.; Sahai, A.; Waters, B. Attribute-based encryption for fine-grained access control of encrypted data. In Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, 30 October–3 November 2006; ACM: New York, NY, USA, 2006; pp. 89–98.
22. Peter, A.; Tews, E.; Katzenbeisser, S. Efficiently Outsourcing Multiparty Computation Under Multiple Keys. *IEEE Trans. Inf. Forensics Secur.* **2013**, *8*, 2046–2058. [CrossRef]
23. Liu, X.; Robert, H.; Deng, K.K.R.C.; Weng, J. An Efficient Privacy-Preserving Outsourced Calculation Toolkit With Multiple Keys. *IEEE Trans. Comput.* **2016**, *65*, 3567–3579. [CrossRef]
24. Bresson, E.; Catalano, D.; Pointcheval, D. A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications. In Proceedings of the Advances in Cryptology—ASIACRYPT 2003, Ninth International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, 30 November–4 December 2003; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2894, pp. 37–54.
25. Kawai, Y.; Matsuda, T.; Hirano, T.; Koseki, Y.; Hanaoka, G. Proxy Re-Encryption That Supports Homomorphic Operations for Re-Encrypted Ciphertexts. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2019**, *102-A*, 81–98. [CrossRef]
26. Paillier, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Proceedings of the Advances in Cryptology—EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, 2–6 May 1999; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1592, pp. 223–238. [CrossRef]
27. Gamal, T.E. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472. [CrossRef]
28. Goldwasser, S.; Micali, S. Probabilistic Encryption. *J. Comput. Syst. Sci.* **1984**, *28*, 270–299. [CrossRef]
29. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory* **2014**, *6*, 1–36.
30. Zhou, H.; Wornell, G.W. Efficient homomorphic encryption on integer vectors and its applications. In Proceedings of the 2014 Information Theory and Applications Workshop, ITA 2014, San Diego, CA, USA, 9–14 February 2014; pp. 1–9.
31. Blaze, M.; Bleumer, G.; Strauss, M. Divertible Protocols and Atomic Proxy Cryptography. In Proceedings of the Advances in Cryptology—EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo,

Finland, 31 May–4 June 1998; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1998; Volume 1403, pp. 127–144.
32. Blum, M.; Luby, M.; Rubinfeld, R. Self-Testing/Correcting with Applications to Numerical Problems. In Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 13–17 May 1990; ACM: New York, NY, USA, 1990; pp. 73–83. [CrossRef]
33. Blum, M.; Wasserman, H. Program Result-Checking: A Theory of Testing Meets a Test of Theory. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; IEEE Computer Society: Washington, DC, USA, 1994; pp. 382–392. [CrossRef]