Journal of Information Security and Applications 74 (2023) 103463

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Privacy preserving via multi-key homomorphic encryption in cloud computing

Xuelian Li [a], Hui Li [a,*], Juntao Gao [b], Runsong Wang [a]

[a] School of Mathematics and Statistics, Xidian University, Xi'an, 710071, China
[b] School of Telecommunications Engineering, Xidian University, Xi'an, 710071, China

A R T I C L E I N F O

Keywords:
Multi-key
Approximate GCD
Integer factorization
Public key compression
Homomorphic encryption
Cloud computing

A B S T R A C T

As the world grapples with the COVID-19 and its variants, multi-user collaboration by means of cloud computing is ubiquitous. How to make better use of cloud resources while preventing user privacy leakage has become particularly important. Multi-key homomorphic encryption(MKHE) can effectively deal with the privacy disclosure issue during the multi-user collaboration in the cloud computing setting. Firstly, we improve the DGHV homomorphic scheme by modifying the selection of key and the coefficients in encryption, so as to eliminate the restriction on the parity of the ciphertext modulus in the public key. On this basis, we further propose a DGHV-type MKHE scheme based on the number theory. In our scheme, an extended key is introduced for ciphertext extension, and we prove that it is efficient in performance analysis. The semantic security of our schemes is proved under the assumption of error-free approximate greatest common divisor and the difficulty of large integer factorization. Furthermore, the simulation experiments show the availability and computational efficiency of our MKHE scheme. Therefore, our scheme is suitable for the multi-user scenario in cloud environment.

1. Introduction

In the post-pandemic era, telecommuting has become an indispensable working mode, and emerging technologies such as cloud computing have played an important role in epidemic monitoring, prevention and control and medical assistance, showing broader application prospects and growth potential [1,2]. According to Gartner, from 2015 to 2020, the penetration rate of global cloud computing market represented by IaaS, PaaS and SaaS increased year by year from 4.3% to 13.1%, and will rise to 15.3% in the next year. The market size is 208.3 billion dollars in 2020 and will exceed 600 billion dollars in 2025 [3].

Cloud services can eliminate the storage space constraints of personal devices and reduce local computing overhead. In practice, different companies or organizations store data in the cloud and use the cloud to share data with other members. Designing a one-to-many data sharing scheme based on attribute-based encryption(ABE) [4,5] can achieve effective access control, but ABE is not suitable for collaborative computing between multiple members in a cloud environment. At the same time, multiple clients want to use the computing power of the cloud server to perform machine learning and data mining, collaboratively solve optimization problems and obtain optimal results. However, uploading user data to the server and training the model on the server may lead to serious user privacy disclosure. For example, in 2021, the registration information of Alibaba Cloud users was leaked, in 2022, 37 GB source code of Microsoft was leaked, and 170 million data of super star learning APP was illegally sold. The losses caused by global cybercrime more than 6 trillion in 2021, about six times as much as in 2020. With frequent privacy disclosure, the public pays more attention to privacy protection. In order to make better use of cloud services, we need a method to process data safely in the cloud environment.

Traditional encryption such as searchable encryption(SE), order preserving encryption(OPE) and comparable encryption (CE) [6–9] can protect the security of data and realize data query according to the ciphertext. However, in these schemes, the data must be decrypted before any calculation. Homomorphic encryption can perform calculations on encrypted data without decryption, and is an effective algorithm to solve the security problems of cloud computing. In 2009, Gentry proposed the first fully homomorphic encryption(FHE) scheme [10]. Dijk et al. constructed the DGHV scheme [11] based on integers and modular arithmetic in 2010. Li et al. [12] optimized the scheme of Dijk et al. [11] and proposed a simple FHE suitable for cloud computing and achieve efficient ciphertext retrieval. Kocabas et al. [13] proposed a privacy-preserving medical cloud computing method using homomorphic encryption. Ren et al. [14] proposed an XOR-homomorphic

* Corresponding author.
E-mail addresses: xlli@mail.xidian.edu.cn (X. Li), hui0921@stu.xidian.edu.cn (H. Li), jtgao@mail.xidian.edu.cn (J. Gao).

https://doi.org/10.1016/j.jisa.2023.103463

Available online 9 March 2023

encryption scheme to perform keyword searches on encrypted data in the cloud.

More generally, cloud servers need to perform homomorphic evaluation of data encrypted by different owners. To be more suitable for multi-user scenario and reduce interaction with the cloud, there are two types of current research, one is based on multi-key homomorphic encryption(MKHE), and the other is based on multiparty homomorphic encryption(MHE). MKHE supports homomorphic calculation on ciphertexts encrypted by different keys, and the plaintext is obtained by the joint decryption of participants. MHE includes a key generation protocol and a decryption protocol. The former is designed to jointly generate a common public key and users encrypt private data under the same common public key. Each party has only one share of the secret key, so users perform the decryption protocol interactively when decrypting.

In 2012, López-Alt et al. [15] put forward the first MKHE scheme to solve the problem of joint computing of multi-user ciphertext data in the cloud environment. Chen et al. [16] proposed a MKHE scheme based on the ring learning with errors problem(RLWE) [17] and applied it to neural network in the ciphertext domain, but the ciphertext dimension of this scheme is linear with the number of users. Ma et al. [18] designed a new federated learning scheme with privacy protection using MKHE. In addition, MHE is proposed by Asharov et al. [19] using threshold homomorphic encryption to solve the problem of secure computing among multiple users. Following the construction principle of Asharov et al. [19], Mouchet et al. [20] proposed a MHE scheme based on RLWE, but it requires multiple rounds of interaction. Further, Park et al. [21]. proposed a scheme with less interaction that is more suitable for multi-user scenarios in cloud environment. However, the homomorphic multiplication on the extended ciphertext in the above schemes needs the evaluation key or relinearization key to perform the relinearization algorithm.

In terms of homomorphic evaluation and performance characteristics, both the integer-based DGHV scheme [11] and the lattice theory-based homomorphic scheme [17,22] maintain the compactness of the ciphertext length. However, DGHV scheme is constructed based on modular arithmetic and integers, which is more concise in concept and form than the schemes based on lattice theory. Dyer et al. [23] proposed a homomorphic encryption scheme based on integer arithmetic, which is mainly used for secure single-party computation in the cloud. Theoretically, any standard homomorphic scheme can be extended to multi-key to solve the multi-user collaborative computing problem [15]. Further, we hope to design a computationally and conceptually simpler MKHE scheme based on the DGHV scheme.

1.1. Our contribution

Based on the DGHV scheme [11], We design a novel and simple MKHE scheme to prevent privacy disclosure in the multi-user collaboration in the cloud computing. To this end, two main points must be solved on the basis of maintaining the randomness of the encryption: how to expand the ciphertext and the generation of the evaluation key. In addition, decryption essentially depends on eliminating large random elements in the ciphertext to keep the decryption correct. Therefore, we must ensure that it does not generate residual random elements when decrypting in the MKHE scheme. Our main contributions are as follows:

1. In order to select the appropriate modulus in a multi-key scheme without generating additional noise, we improve the DGHV scheme by modifying the encryption factor to remove the restriction on the parity of the largest element in the public key. The noise in the improved DGHV scheme is still within a small range relative to the threshold, which can ensure the validity of decryption. Furthermore, we analyze the operation circuit and prove that the scheme can perform homomorphic calculation in the permitted circuit and give the degree of the permitted polynomial.
2. On the basis of the improved DGHV scheme, we propose a DGHV-type MKHE scheme, where the joint secret key used for decryption is calculated from secret keys of participating users, rather than the concatenation of secret keys. It is difficult for an engaged user to decompose the joint key to get the secret keys of other users. Therefore, the joint secret key will not disclose the users' secret keys. Unlike previous MKHE schemes, an extended key is introduced for extending ciphertext in our multi-key scheme and all arithmetics just depend on three basic operations, namely addition, multiplication and modular operation. Compared with other MKHE schemes, our scheme is more concise in computational form and ciphertext composition.
3. We prove that our scheme is semantically secure under the error-free approximate GCD assumption and the large integer factoring problem. Besides, we illustrate the zero knowledge property of participating users and the cloud server. Therefore, complicated calculations on extended ciphertexts can be delegated to the cloud by sending the evaluation key to cloud server. Users only need to download ciphertexts after computation and decrypt with the joint secret key. In addition, the experiments show that the size of the public key can be decreased to $O(\lambda^3)$ by compressing in a higher dimension to further reduce the number of elements in the public key.

1.2. Related work

The concept of homomorphic encryption (HE) [24] was first proposed by Rivest et al. in 1978. It means that we have $f(\mathrm{Enc}(m_1, m_2)) = \mathrm{Enc}(f(m_1, m_2))$ for any operation f. After that, researchers put forward many partial homomorphic encryption schemes [25,26]. Until 2009, Gentry [10] proposed the first FHE scheme based on the ideal lattice. The principle is to design a homomorphic scheme with limited ciphertext calculation, and perform reencryption operation when the ciphertext noise reaches the threshold, so as to realize bootstrapping. Then, based on Gentry's idea, other homomorphic schemes are proposed [27,28].

In 2010, following Gentry's blueprint, Dijk et al. constructed the DGHV scheme [11] which is entirely on the basis of integer operation. This scheme does not use the ideal lattice on the polynomial ring, which not only brings a new construction method of homomorphic encryption scheme, but also makes the principle of the algorithm easier to understand. However, the public key size is $O(\lambda^{10})$, which is too large to implement, and this is also the main reason for low efficiency.

Subsequently, two different techniques were proposed to compress the public key of DGHV scheme. The first way is to reduce the number of elements in the public key. Coron et al. [29] used quadratic encryption $x_{i,j} = x_{i,0} \cdot x_{j,1} \bmod x_0$ rather than linear encryption for public key elements to reduce the public key size to $O(\lambda^7)$. This can be called the square compression technique. The second way is to optimize the length of the bits in the elements of the public key. Coron et al. [30] proposed the offset public key compression scheme by introducing a pseudo-random number generator f and a random seed, and the public key size is reduced to $O(\lambda^5)$. Chen et al. [31] combined the two techniques and proposed a quadratic offset compression scheme, which can be further reduced the size of the public key to $O(\lambda^{3.5})$ by reducing the number and the bit length of public key at the same time. Subsequent optimization [32,33] based on DGHV are put forward, but these schemes only have homomorphism for a single key.

López-Alt et al. [15] put forward the concept of multi-key homomorphic encryption in 2012 and constructed the first MKHE scheme based on NTRU cryptosystem. Subsequently, the scheme was continuously optimized and other NTRU-type schemes have been proposed, such as [34,35]. At present, the decryption complexity and communication of NTRU-type MKHE schemes are large, because it needs to be realized through complex interaction. In addition, the GSW-type MKHE schemes and the BGV-type MKHE schemes are common.

Table 1
Comparison of different types of MKHE schemes

Type	Assumption	Dimension	Key aggregation	Relinearization
NTRU	DSPR	Independent of k	Multiplication	Yes
GSW	LWE	$O(k^2)$	Concatenation	No
BGV	RLWE	$O(k)$	Concatenation	Yes
Our	AGCD	Independent of k	Multiplication	No

GSW-type MKHE schemes [36–38] are based on the learning with errors problem (LWE) [39]. In these schemes, the ciphertext sequence includes other auxiliary information used to complete ciphertext expansion. The ciphertext expansion of these GSW-type MKHE schemes is complex, and the ciphertext dimension is quadratic with respect to the number of participants.

Chen et al. [40] first put forward a BGV-type MKHE scheme by combining the ring-GSW. Based on the scheme of Chen et al. [40], Zhang et al. [41] proposed four unit protocols to construct a multiparty k-means clustering scheme and used a cloud server to implement outsourced computation of data when participants are offline. Chen et al. [16] proposed a MKHE scheme relied on the ring learning with errors problem (RLWE) [17]. Kim et al. [42] redesigned the multiplication of Chen et al. [16] using a gadget decomposition with homomorphic properties. Instead of following the traditional procedure of performing tensor product and relinearization sequentially, the calculation performs both operations simultaneously, but causes an increase in noise. The ciphertext dimension of these schemes is linear with the number of users. Compared with the GSW-type MKHE, BGV-type MKHE has simpler ciphertext expansion, but the dimension of ciphertext increases at an exponential rate after homomorphic multiplication. In order to control the dimension of the ciphertext, the nonlinear entries in the ciphertext need to be relinearized. We compare different types of MKHE in Table 1 and give the characteristics of the DGHV-type MKHE we constructed in this paper.

1.3. Our framework

The roadmap of this article is shown below. Section 2 introduces some preliminary concepts and the system model. In Section 3 and Section 4, we describe in detail the construction of the improved DGHV scheme and the multi-key scheme respectively. The security of our schemes is proved in Section 5 and the performance comparison, time, and memory consumption about our algorithm are given in Section 6. Finally, Section 7 briefly summarizes the general idea and main content of this paper.

2. Preliminaries

In this part, we introduce some basic notations and definitions and give the system model of our scheme. We also review the homomorphic encryption scheme proposed by Dijk, Gentry, Halevi and Vaikuntanathan in [11].

2.1. Notations

We denote parameters by Greek letters ($\eta, \rho, \gamma, \tau, \lambda$, etc.). In particular, let λ represent the security parameter. Lowercase English letters(p, q, x, y, etc.) denote real numbers and integers. For a real number x, we denote by $\lceil x \rceil$, $\lfloor x \rfloor$, $\lfloor x \rceil$ the rounding of x up, down, or to the nearest integer. For integers z and p, $q_p(z)$ and $r_p(z)$ denote the quotient and remainder of z with respect to p, that is, $q_p(z) = \lfloor z/p \rceil$ and $r_p(z) = z - p \cdot q_p(z)$. The remainder can also be expressed as $[z]_p$. Unless otherwise specified, all logarithms in this paper are base-2.

2.2. Approximate GCD

For a specific η-bit odd integer p, the distribution $\mathcal{D}_{\gamma,\rho}(p)$ over γ-bit integers is defined as:

$$\mathcal{D}_{\gamma,\rho}(p) = \{ \text{ Choose } q \leftarrow \mathbb{Z} \cap [0, 2^\gamma/p), r \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho) : \\ \text{Output } x = q \cdot p + r \}$$

The (ρ, η, γ)-approximate-GCD problem is: Given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$ for a randomly chosen η-bit odd integer p, output p.

2.3. The DGHV scheme over the integers

2.3.1. Parameters

Given the security parameter λ, the following parameters are defined to control the number of elements in the public key and the bit-length of integers to ensure the security of the scheme.

- γ is the bit-length of the public key elements x_i.
- η is the bit-length of the secret key p.
- ρ is the bit-length of the noise r_i.
- τ is the number of the public key elements x_i.

2.3.2. The description of DGHV scheme

1. *KeyGen*(λ): Select a random η-bit odd integer p, namely $p \leftarrow (2\mathbb{Z}+1) \cap [2^{\eta-1}, 2^\eta)$. Sample x_i from distribution $\mathcal{D}_{\gamma,\rho}(p)$ for $i = 0, 1, \ldots, \tau$. Rearrange these integers so that x_0 is the largest and satisfies the condition that x_0 is odd and $[x_0]_p$ is even, otherwise restart. The secret key is $sk = p$ and the public key is $pk = (x_0, x_1, \ldots, x_\tau)$.
2. *Encrypt*(pk, m): For $m \in \{0, 1\}$, select a random integer r and a random subset S of the public key, where $S \subseteq \{1, 2, \ldots, \tau\}$, $r \in (-2^{\rho'}, 2^{\rho'})$, ρ' is the secondary noise parameter. Then calculate and output the following ciphertext:
$$c = \left[m + 2r + 2\sum_{i \in S} x_i \right]_{x_0}$$
3. *Evaluate*$(pk, C, c_1, \ldots, c_t)$: Given the public key pk, the circuit C belonging to the permitted circuit set C_ε with t inputs, and t ciphertexts c_i, perform the following *Add* and *Mult* step by step and output the result $C(c_1, \ldots, c_t)$.
$$Add(pk, c_1, c_2) : c_{add} = c_1 + c_2 \bmod x_0$$
$$Mult(pk, c_1, c_2) : c_{mult} = c_1 \times c_2 \bmod x_0$$
4. *Decrypt*(sk, c): As defined above, $sk = p$. Hence, the decryption process is as follows:
$$m' = (c \bmod p) \bmod 2$$

The above section completes the description of the DGHV scheme. There are some restrictions on the selection of x_0 in the public key. On the one hand, to eliminate the connection between the ciphertext parity and the plaintext, on the other hand, to ensure that the noise items can be eliminated during decryption.

2.4. Multi-key homomorphic encryption

A multi-key homomorphic encryption scheme $\mathcal{E}$ consists of five algorithms (KeyGen, Enc, Extend, Eval, Dec), the characteristics of each algorithm are as follows:

1. $\mathcal{E}$.*KeyGen*$(1^\lambda, 1^K, 1^L)$: Take the security parameter λ, the number of different participants K and the circuit depth L as input, generate public and secret keys (pk_i, sk_i) for participants, the extended key ek_i required for ciphertext extension and the evaluation key required for evaluation algorithm.

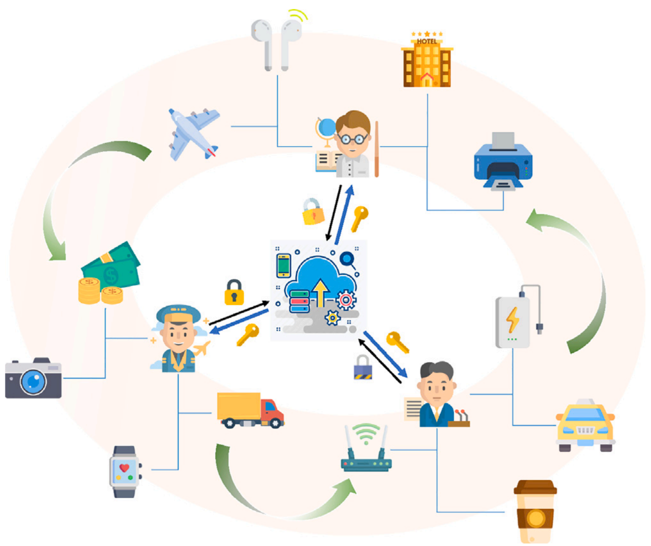


Fig. 1. System Model.

2. $\mathcal{E}.Enc(pk_i, m_i)$: For example, given the public key pk_i and message m_i to be encrypted, output a ciphertext c_i.
3. $\mathcal{E}.Extend(c_i, ek_i)$: Given the ciphertext c_i and the extended key ek_i, output the extended ciphertext $\hat{c_i}$.
4. $\mathcal{E}.Eval(C, (\hat{c_1}, \ldots, \hat{c_\ell}), evk)$: Given the circuit C with ℓ inputs, ℓ extended ciphertext and evaluation key evk, output a ciphertext c under the joint secret key sk_S.
5. $\mathcal{E}.Dec(c, sk_S)$: Given the ciphertext c and the joint secret key sk_S which is constructed from the participating users' secret keys, output the message m.

For a MKHE scheme, we say it is semantically secure if the following distributions are computationally indistinguishable:

$$(pp, \mathcal{E}.Enc(0, pk_i)) \overset{comp}{\approx} (pp, \mathcal{E}.Enc(1, pk_i))$$

2.5. System model

As shown in Fig. 1, the proposed multi-key homomorphic encryption scheme is mainly composed of three parts: data owner (DO), cloud server (CS) and data user (DU). The information is possibly collected from various domains, such as education, medical treatment, transportation, etc. Through our scheme, the data from different parties can be shared to make better use of resources and construct a large information network. It is assumed that a trusted party deals with the distribution of keys in the system, which is not shown in the figure. The responsibilities of each participant are as follows.

1. DO: DO encrypts the privacy data with his public key and uploads it to the cloud after extension. In particular, there are multiple DOs in this system, at this time, each DO only has his own information.
2. CS: Firstly, CS can provide data storage for DOs. In addition, CS can use the evaluation key perform ciphertext calculations and share the data when a participant in the system initiates an inquiry request.
3. DU: DU is the user who has the joint secret key. In particular, DO can also be DU in the multi-key scenario. After cloud computing, DU can decrypt the data from CS by using the joint secret key to obtain the information in the whole system.

We assume that the cloud server in the system model is semi-honest. It performs the calculation as required, does not quit halfway, and does not enter fake data. However, it may be curious about the original data, retain the results of the calculation, and try to obtain sensitive information by analyzing what it has. This semi-honest assumption makes sense in practice. In this paper, MKHE is used to protect the privacy of participants and enable the server to efficiently complete the calculation.

3. Our variant of the DGHV scheme

In order to construct a MKHE scheme based on the DGHV scheme, we improve the DGHV scheme in this section.

3.1. The construction

1. *KeyGen*(λ): Select a random, large and η-bit prime $p \in [2^{\eta-1}, 2^\eta)$. Choose an integer q_0 in $[0, 2^\gamma/p)$, where q_0 is three times that of an integer which is square free and without prime factors less than 2^λ. Then, let $x_0 = q_0 \cdot p$. Generate integers $x_i = q_i \cdot p + r_i$ for $i = 1, \ldots, \tau$, where $q_i \leftarrow [0, q_0)$, $r_i \leftarrow (-2^\rho, 2^\rho)$. The secret key is $sk = p$ and the public key is $pk = (x_0, x_1, \ldots, x_\tau)$.
2. *Encrypt*(pk, m): For $m \in \{0, 1\}$, select a random integer $r \in (-2^{\rho'}, 2^{\rho'})$ as the encryption noise and a subset $S \subseteq \{1, 2, \ldots, \tau\}$. Then calculate and output the ciphertext:

$$c = \left[m + 9r + 9 \sum_{i \in S} x_i \right]_{x_0}$$

3. *Evaluate*: The operation of this step is the same as the DGHV scheme, ciphertext mod x_0 after homomorphic calculation.
4. *Decrypt*(sk, c): The decryption process is as follows:

$$m' = (c \bmod p) \bmod 3$$

In the original scheme, $m + 2r + 2\sum_{i \in S} x_i$ has the same parity as m. Therefore, the modulus x_0 in the public key must be odd, otherwise the ciphertext of the two plaintext bits correspond to unique parity respectively. Because of the change of encryption coefficient and the uncertainty of the parity of random numbers, the parity of $m + 9r + 9\sum_{i \in S} x_i$ is independent of the parity of m and we do not need to restrict the parity of x_0 in our variant scheme. Further, in order to carry out the multiplication in the multi-key case, here we choose the coefficients to be 9 instead of 3.

Remark 1. In homomorphic operation, the noise change is that the noise after addition is equal to the sum of their respective noises and the noise after multiplication is equal to the product of their respective noises.

3.2. Parameter limitations

In order to make the subsequent proof clearer, the scheme described above is based on the original DGHV scheme and the restrictions of parameters can be consistent with [11].

1. $\rho = \omega(\log \lambda)$ in order to resist brute-force attacks on the noise.
2. $\eta \geq \rho \cdot \Theta(\lambda \log^2 \lambda)$ to perform homomorphic operations on "squashed decryption circuit".
3. $\gamma = \omega(\eta^2 \log \lambda)$ in order to resist various attacks on the approximate GCD.
4. $\tau > \gamma + \omega(\log \lambda)$ in order to apply the leftover hash lemma in the security proof.
5. $\rho' = 2\rho$ as a secondary noise parameter in encryption.

For convenience, we can take such a set of parameters, $\rho = \lambda$, $\eta = O(\lambda^2)$, $\gamma = O(\lambda^5)$, $\tau = \lambda + \gamma$ and $\rho' = 2\lambda$. In this way, the public key size of the scheme is $O(\lambda^{10})$, which is the same as that in [11]. In experiments, we will optimize and use a scheme with public key size reduced to $O(\lambda^3)$.

3.3. Correctness

Lemma 1. *For a fresh ciphertext $c \leftarrow Encrypt(pk, m)$, the noise will not be more than $\tau \cdot 2^{\rho'+2}$.*

Proof. Given the ciphertext c as follows:

$$c = \left[m + 9r + 9\sum_{i \in S} x_i \right]_{x_0}$$

Since x_0 is the largest one in pk, we have that

$$c = \left(m + 9r + 9\sum_{i \in S} x_i \right) + k \cdot x_0, \text{ where } |k| \le \tau.$$

For x_i, there exist integers q_i and r_i, such that $x_i = q_i \cdot p + r_i$, and $|r_i| < 2^{\rho}$, $|r| < 2^{\rho'}$, $\rho' \ge \rho + 4$. We get:

$$c \bmod p = m + 9r + 9\sum_{i \in S} r_i \tag{1}$$

Therefore, $(c \bmod p) \bmod 3$ and m are the same. Its absolute value is:

$$|c \bmod p| \le 2^3 \cdot 2^{\rho'+1} + 2^4\tau \cdot 2^{\rho} < 2^3 \cdot 2^{\rho'+1} + \tau \cdot 2^{\rho'+1} < \tau \cdot 2^{\rho'+2} \tag{2}$$

Definition 1 (*Permitted Circuit* [10]). For a circuit C, we extend it to integers, that is, operations are applied to integers rather than bits. For any $i \ge 1$ and any integer set which the absolute value of each number in is less than $\tau^i \cdot 2^{i(\rho'+2)}$, the permitted circuit outputs a result with an absolute value of up to $2^{\eta-3-n} < p/(4(\lambda+1))$, where $n = \lceil \log_2(\lambda+1) \rceil$. Let C_ε represent the set of permitted circuits. We conclude that:

Theorem 2. *Our improved scheme is correct for C_ε.*

Proof. Suppose that C is a permitted circuit in C_ε with t inputs. c is the output after circuit operation. C' is the generalized circuit of C, and its operations are over integers. For c_i, $i = 1, 2, \ldots, t$:

$$\begin{aligned} c \bmod p &= C'(c_1, \ldots, c_t) \bmod p \\ &= C'(c_1 \bmod p, \ldots, c_t \bmod p) \bmod p \end{aligned}$$

According to Lemma 1, we know that for each ciphertext, $|c_i \bmod p| < \tau \cdot 2^{\rho'+2}$. So we obtain the following formula according to the definition of C_ε :

$$|C'(c_1 \bmod p, \ldots, c_t \bmod p)| \le 2^{\eta-4} \le p/8$$

That is to say $c \bmod p = C'(c_1 \bmod p, \ldots, c_t \bmod p)$. Further for $(c \bmod p) \bmod 3$, we have:

$$[c \bmod p]_3 = C(m_1, \ldots, m_t) \tag{3}$$

Therefore, the result obtained by decryption is the same as that of the same operation on the plaintext and the correctness of our improved scheme is guaranteed.

Remark 2. From the definition of the permitted circuit, we cannot directly judge whether a given calculation can be allowed. Therefore, a sufficient condition for the permission of a multivariate polynomial f is given for the related arithmetic circuit C. If the degree of polynomial f is d and $\|f\|_1$ is the sum of absolute values of its coefficients, then whether $C \in C_\varepsilon$ can be judged by:

$$d \le \frac{\eta - 3 - n - \log \|f\|_1}{\rho' + 2 + \log \tau} \tag{4}$$

As in [11], polynomials satisfying (4) are permitted polynomials and the set of them is defined as $\mathcal{P}_\varepsilon$.

4. The multi-key homomorphic encryption scheme

In this section, we give a detailed description of our MKHE scheme relied on the improved DGHV scheme. Our goal is to build a framework that supports performing calculation on encrypted data under different keys in cloud environment. As shown in Fig. 2, our scheme consists of data owner, cloud server and data user which can also be data owner. A trusted third party (KGC) is responsible for the generation and distribution of keys for the system. Assuming that there are t users in the system, for the sake of clarity, we take two ciphertexts encrypted by Alice and Bob as examples in the figure. The detailed process of each part will be described below.

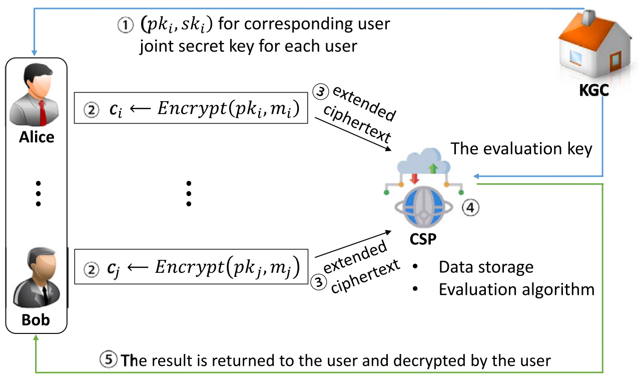

① (pk_i, sk_i) for corresponding user
joint secret key for each user
KGC
Alice
② $c_i \leftarrow Encrypt(pk_i, m_i)$
③ extended ciphertext
The evaluation key
④
CSP
Bob
② $c_j \leftarrow Encrypt(pk_j, m_j)$
③ extended ciphertext
• Data storage
• Evaluation algorithm
⑤ The result is returned to the user and decrypted by the user

Fig. 2. Multi-key homomorphic encryption algorithm framework.

4.1. Scheme construction

We assume there are t participants in the multi-key setting.

1. *MK.KeyGen*(λ): Each user obtains his own public and secret keys from the trusted third party, which is generated by running *KeyGen* algorithm in Section 3.1, and the secret key p_i shall be with the form of $9k+1$:

   $$pk_i = \left(x_{i,0}, x_{i,1}, \ldots, x_{i,\tau}\right), \; sk_i = p_i$$

   Hence, the sequence of secret keys in the trusted party is $(sk_1, sk_2, \ldots, sk_t) = (p_1, p_2, \ldots, p_t)$, where $p_i \in [2^{\eta-1}, 2^{\eta})$. Then, similarly, a key p_{t+1} is generated. The joint key is calculated by multiplying all the p_i generated above, namely:

   $$P = \prod_{i=1}^{t+1} p_i$$

   This is executed by the trusted party and sent to participating users.
2. *MK.Encrypt*(pk_i, m_i): For one user, the corresponding public key is pk_i and the message to be encrypted is $m_i \in \{0, 1\}$. Run *Encrypt* algorithm to calculate c_i:

   $$c_i = \left[m_i + 9r_i + 9\sum_{j \in S_i} x_{i,j} \right]_{x_{i,0}}$$

   where $r_i \in (-2^{\rho'}, 2^{\rho'}), S_i \subseteq \{1, 2, \ldots, \tau\}$.
3. *MK.Extend*(c_i, ek_i): After the user get his own secret key and the joint secret key, he can calculate the extended key required for ciphertext extension, namely:

   $$ek_i = \frac{P}{p_i} = d_i$$

   For the ciphertext c_i, it can be expanded to the joint secret key using ek_i and obtain the extended ciphertext $\hat{c}_i$ by running the extension procedure described below:

   $$\hat{c}_i = c_i \cdot \frac{ek_i}{3} = c_i \cdot \frac{d_i}{3}$$

4. *MK.Evaluate*$(pk, C, (\hat{c}_1, \ldots, \hat{c}_\ell, evk))$: Given the circuit C with ℓ inputs and ℓ extended ciphertexts $(\hat{c}_1, \ldots, \hat{c}_\ell)$, homomorphic operation is performed on the extended ciphertexts by the cloud. The specific operations are as follows:

   (a) *MK.Add*$(\hat{c}_1, \hat{c}_2)$: $c_{add} = \hat{c}_1 + \hat{c}_2$
   (b) *MK.Mult*$(\hat{c}_1, \hat{c}_2)$:

   $$c_{mult} = \frac{\hat{c}_1 \cdot \hat{c}_2}{evk}$$

   where the evaluation key evk is p_{t+1}, which is generated by the trusted party and sent to the cloud server.
5. *MK.Decrypt*(P, c): Here c is the extended ciphertext or the ciphertext after homomorphic operation on extended ciphertexts. During decryption, in order to get m', the participant first calculate the following formula with the joint secret key P:

   $(c \bmod P) \bmod 3$

   m' is 0 when the above result is an integer or its fractional part is greater than 0.5, otherwise m' is 1.

4.2. Correctness of ciphertext extension

The user extend the ciphertext c_i to the joint secret key by using the extended key ek_i. According to the scheme, the verification is as follows, where S_i is a subset of public key of user i and k_i is an integer.

$$\begin{aligned}
&\left(\hat{c}_i \bmod P\right) \bmod 3 \\
&= \left(\left(c_i \cdot \frac{ek_i}{3}\right) \bmod P\right) \bmod 3 \\
&= \left(\left(\left(m_i + 9r_i + 9\sum_{j\in S_i} x_{i,j} + k_i x_{i,0}\right) \cdot \frac{d_i}{3}\right) \bmod P\right) \bmod 3 \\
&= \left(m_i \frac{d_i}{3} + 3r_i d_i + 3\sum_{j\in S_i} r_{i,j} d_i\right) \bmod 3 \\
&= \left(m_i \frac{d_i}{3}\right) \bmod 3
\end{aligned} \tag{5}$$

First, when $m_i = 0$, we have $\left(m_i \frac{d_i}{3}\right) \bmod 3 = 0$. d_i is an odd integer with the form of $9k+1$ because the essence of d_i is the product of some primes with the form of $9k+1$. Then, when $m_i = 1$, $\left(m_i \frac{d_i}{3}\right) \bmod 3$ will not be an integer and its fractional part is less than 0.5. Therefore, the corresponding plaintext is 1 according to the decryption rule.

In addition, the conversion between decimals and integers can be done by encoding and decoding. If a decimal is rounded directly, some valid numbers may be affected. In order to guarantee precision within a certain range $1/\Delta$, where $\Delta > 0$, multiply by Δ when encoding and divide by Δ when decoding. For example, when we round $x = 1.666$, we do not want to get the closest integer 2, but in order to preserve the precision of 0.2, we can define $\Delta = 5$, retain the precision of $1/\Delta = 0.2$, $\Delta x = 8.33 \approx 8$, and $8/\Delta = 1.6$, which will not affect the correct decryption.

It can be seen from Eq. (1) that the noise of unextended ciphertext c_i is $\left(c_i \bmod p_i\right) = m_i + 9r_i + 9\sum_{j\in S_i} r_{i,j}$, and $\left(c_i \bmod p_i\right) < \frac{p_i}{2}$ by the decryption correctness of the improved scheme. From Eq. (5), we can conclude that the noise of the extended ciphertext is:

$$\hat{c}_i \bmod P = \left(m_i + 9r_i + 9\sum_{j\in S_i} r_{i,j}\right) \cdot \frac{d_i}{3}$$

Hence, $\hat{c}_i \bmod P < \frac{p_i}{2} \cdot \frac{d_i}{3} = \frac{p_i \cdot d_i}{6} < \frac{P}{2}$, which meets the condition for correct decryption.

4.3. Analysis of homomorphic operations

For the MKHE scheme, homomorphic addition and homomorphic multiplication are redefined. Suppose there are two ciphertexts c_1 and c_2, which are encrypted by their respective public keys pk_1 and pk_2. By the extended keys, $\hat{c}_1$ and $\hat{c}_2$ can be obtained, which are the extended ciphertexts of c_1 and c_2 respectively. The forms of $\hat{c}_1$ and $\hat{c}_2$ are:

$$\hat{c}_1 = \left(m_1 + 9r_1 + 9\sum_{n\in S_1} x_{1,n} + k_1 x_{1,0}\right) \cdot \frac{d_1}{3} \tag{6}$$

$$\hat{c}_2 = \left(m_2 + 9r_2 + 9\sum_{k\in S_2} x_{2,k} + k_2 x_{2,0}\right) \cdot \frac{d_2}{3} \tag{7}$$

4.3.1. Addition

For homomorphic addition of ciphertexts $\hat{c}_1$ and $\hat{c}_2$, add them directly and output: $c_{add} = \hat{c}_1 + \hat{c}_2$. We now show the result c_{add} can be decrypted correctly under the joint secret key P to get message $m_{add} = \left(m_1 + m_2\right) \bmod 2$, that is, the XOR operation of two plaintext bits. We first calculate the following formula:

$$\begin{aligned}
&\left(\hat{c}_1 + \hat{c}_2\right) \bmod P \\
&= \left(m_1 + 9r_1 + 9\sum_{n\in S_1} r_{1,n}\right) \cdot \frac{d_1}{3} + \left(m_2 + 9r_2 + 9\sum_{k\in S_2} r_{2,k}\right) \cdot \frac{d_2}{3} \\
&= m_1 \frac{d_1}{3} + 3r_1 d_1 + 3\sum_{n\in S_1} r_{1,n} d_1 + m_2 \frac{d_2}{3} + 3r_2 d_2 + 3\sum_{k\in S_2} r_{2,k} d_2
\end{aligned} \tag{8}$$

$$\left(\left(\hat{c}_1 + \hat{c}_2\right) \bmod P\right) \bmod 3 = \left(m_1 \frac{d_1}{3} + m_2 \frac{d_2}{3}\right) \bmod 3 \tag{9}$$

The following analysis shows that c_{add} can be obtained according to the decryption criterion. In the correctness verification of the extended ciphertext, we know that d_i is odd. So neither $\frac{d_1}{3}$ nor $\frac{d_2}{3}$ is an integer here. Then, according to the possible values of m_1 and m_2, it can be divided into the following four situations:

1. When $m_1 = 0$ and $m_2 = 0$, $\left(m_1 \frac{d_1}{3} + m_2 \frac{d_2}{3}\right) \bmod 3 = 0$ is an integer. From the criterion, we have:

   $m' = 0 = \left(m_1 + m_2\right) \bmod 2$
2. When $m_1 = 1$ and $m_2 = 0$, $\left(m_1 \frac{d_1}{3} + m_2 \frac{d_2}{3}\right) \bmod 3 = \left(m_1 \frac{d_1}{3}\right) \bmod 3$ is not an integer and its fractional part is less than 0.5. From the criterion, we have:

   $m' = 1 = \left(m_1 + m_2\right) \bmod 2$
3. When $m_1 = 0$ and $m_2 = 1$, $\left(m_1 \frac{d_1}{3} + m_2 \frac{d_2}{3}\right) \bmod 3 = \left(m_2 \frac{d_2}{3}\right) \bmod 3$ is similar to case 2.
4. When $m_1 = 1$ and $m_2 = 1$, $\left(m_1 \frac{d_1}{3} + m_2 \frac{d_2}{3}\right) \bmod 3$ is not an integer but its fractional part is greater than 0.5. From the criterion, we have:

   $m' = 0 = \left(m_1 + m_2\right) \bmod 2$

Thus, in any case, m_{add} can be obtained by *MK.Decrypt*. In addition, the noise of c_{add} is equal to the sum of their respective noises from Eq. (8).

4.3.2. Multiplication

In the homomorphic multiplication of extended ciphertexts $\hat{c}_1$ and $\hat{c}_2$, the evaluation key evk generated by the trusted party is used instead of simply multiplying the ciphertexts. We now describe how the result c_{mult} can be decrypted correctly under the joint secret key P to get the message $m_{mult} = m_1 \cdot m_2$. First, c_{mult} is:

$$c_{mult} = \frac{\hat{c}_1 \cdot \hat{c}_2}{evk}$$

where evk is a large prime number and the common divisor of ek_1 and ek_2. Further detailed description is:

By analyzing from the perspective of the trusted party responsible for distributing keys, $\frac{ek_1 \cdot ek_2}{evk}$ is a multiple of P. For the simplicity of the argument, we set it to P'. Hence, the calculation result c_{mult} is denoted by:

$$\left(m_1 m_2 + 9q' + k_2 m_1 x_{2,0} + 81q'' + k_1 m_2 x_{1,0} + k_1 k_2 x_{1,0} x_{2,0}\right) \frac{P'}{9}$$

where q' is the sum of all formulas in the following form:

$$m_i r_j,\ m_i \sum_{k \in S_j} x_{j,k},\ k_i x_{i,0} r_j,\ k_i x_{i,0} \sum_{k \in S_j} x_{j,k}$$

q'' is the sum of all formulas in the following form:

$$r_i r_j,\ r_i \sum_{k \in S_j} x_{j,k},\ \sum_{n \in S_i} x_{i,n} \cdot \sum_{k \in S_j} x_{j,k}$$

The above formula is obtained by substituting Eqs. (6), (7) into c_{mult}, where $i \neq j \in \{1, 2\}$. So $\left(c_{mult} \bmod P\right)$ is:

$$\frac{\hat{c_1} \cdot \hat{c_2}}{evk} \bmod P = \left(m_1 m_2 \frac{P'}{9} + k_2 m_1 x_{2,0} \frac{P'}{9} + k_1 m_2 x_{1,0} \frac{P'}{9}\right) \bmod P$$

Because 3 is the factor of $x_{i,0}$, we can make k_1 and k_2 be multiples of 3 by selecting the appropriate subset S_i, we have:

$$\frac{\hat{c_1} \cdot \hat{c_2}}{evk} \bmod P = m_1 m_2 \frac{P'}{9} \bmod P$$

As long as there is $m_i = 0$, $\left(c_{mult} \bmod P\right) \bmod 3$ is 0 which is an integer, so $m' = m_1 \cdot m_2 = 0$. When $m_1 = 1 = m_2 = 1$, $\left(c_{mult} \bmod P\right) \bmod 3$ is not an integer and its fractional part is less than 0.5 because P' has the same form of $9k + 1$ as d_i. From the criterion, we have $m' = m_1 \cdot m_2 = 1$.

From the mentioned above, it can be seen that by multiplying the extended key by $1/3$, $m_1 m_2$ is multiplied by $P'/9$ instead of P', so that $\left(c_{mult} \bmod P\right) = m_1 m_2 \frac{P'}{9} \bmod P$ meets the condition for correct decryption. Other noise components will also be multiplied by $P'/9$, such as $m_1 r_2 \frac{P'}{9}, m_2 r_1 \frac{P'}{9}$, in this case, even r_1, r_2 are small, the sum of noise components will also affect correct decryption. In order to avoid decryption failure, we modify the original scheme in Section 3 and change the corresponding coefficient to 9 to ensure $9 m_1 r_2 \cdot \frac{P'}{9}$ can be converted into multiple of P.

5. Security of our schemes

In this part, we prove the security of the improved scheme and the multi-key scheme. The improved scheme is semantically secure under the variant of approximate GCD assumption, i.e., error-free approximate GCD assumption. The multi-key scheme also depends on the difficulty of large integer factorization to ensure the security.

5.1. Security of the improved scheme

We prove that our scheme is semantically secure under the (stronger) error-free approximate GCD assumption, and the security proof follows the strategy in [11]: The adversary who destroys the security of this scheme can be transformed into an least significant bit (LSB) predictor of $(z \bmod p)$, where z is an integer. This in turn can recover p in the approximate GCD problem.

For specific integers p and q_0, we define the following modified distribution:

$$\mathcal{D}'_\rho\left(p, q_0\right) = \left\{ \text{Choose } q \leftarrow Z \cap \left[0, q_0\right), r \leftarrow Z \cap (-2^\rho, 2^\rho) \right.$$
$$\left. \text{Output}: x = p \cdot q + r \right\}$$

Definition 2 ((ρ, η, γ)-*Error-free Approximate GCD* [29]). Let p be an η-bit prime, q_0 be a random integer in $[0, 2^\gamma/p)$ and q_0 be three times that of an integer which is square free and without prime factors less than 2^λ. Given $x_0 = q_0 \cdot p$, polynomially many samples randomly chosen from $\mathcal{D}'_\rho\left(p, q_0\right)$, output p.

Theorem 3. *Suppose $\mathcal{A}$ is an adversary of our improved scheme with non-negligible advantage ε, then a simulator $\mathcal{B}$ can be constructed to solve the (ρ, η, γ)-error-free-approximate-GCD problem. The probability of success of simulator $\mathcal{B}$ is at least $\varepsilon/2$. The running time of $\mathcal{B}$ can be represented by a polynomial about λ, $1/\varepsilon$ and the running time of $\mathcal{A}$.*

Proof. Consider that an adversary $\mathcal{A}$ can break the semantic security of our improved scheme with advantage ε in polynomial time, that is, the probability of $\mathcal{A}$ outputting a correct plaintext for a set of public keys and a ciphertext (generated by *KeyGen* and *Encrypt*) is at least $1/2 + \varepsilon$. We construct a simulator $\mathcal{B}$ using $\mathcal{A}$. For $\mathcal{B}$, given x_0 with γ bits, $\mathcal{B}$ recovers p by sampling polynomial samples from the distribution $\mathcal{D}'_\rho\left(p, q_0\right)$. The process is as follows:

5.1.1. Generation of public key

The public key is created by $\mathcal{B}$. First, define $x_0 = q_0 \cdot p$, where q_0 is three times that of an integer which is square free and without prime factors less than 2^λ. Then, by sampling x_i from the distribution $\mathcal{D}'_\rho\left(p, q_0\right)$ for $i = 1, \ldots, \tau$, $\mathcal{B}$ obtains the public key $pk = \left(x_0, x_1, \ldots, x_\tau\right)$. In this way, the public key distribution created by $\mathcal{B}$ is the same as that generated by *KeyGen*.

5.1.2. A LSB predictor

In this step, $\mathcal{B}$ constructs a high-accuracy LSB predictor using $\mathcal{A}$ to achieve the goal recovering p. For an integer $z \in [0, 2^\gamma)$, q_z is the quotient of z with respect to p and the predictor will output a reliable prediction about the least significant bit of $q_p(z)$. $\mathcal{B}$ interacts with $\mathcal{A}$ as Algorithm 1.

Algorithm 1 Least Significant Bit Learning

```
Input: An integer $z \in [0, 2^\gamma)$, a set of public keys generated by $\mathcal{B}$,
    $pk = \left(x_0, x_1, \ldots, x_\tau\right)$.
Output: The least significant bit of $q_p(z)$.
1: for i=1 to $poly(\lambda)/\varepsilon$ do        // $\varepsilon$ is the advantage of $\mathcal{A}$
2:     Randomly choose $m_i \in \{0, 1\}$ and $r_i \in \left(-2^{\rho'}, 2^{\rho'}\right)$;
3:     Calculate $c_i = \left[z + m_i + 9r_i + 9\sum_{j \in S_i} x_j\right]_{x_0}$;
4:     Ask $\mathcal{A}$ and get a prediction $a_i \leftarrow \mathcal{A}\left(c_i, pk\right)$;
5:     Let $b'_i = a_i \oplus \text{parity}(z) \oplus m_i$;  // $b'_i$ is the parity of $q_p(z)$
6: end for
7: Select the majority among $b'_i$ as $b_i$;
8: Return $b_i$.
```

According to Leftover Hash Lemma [43], the distribution of ciphertext c_i of the above subroutine in line 3 is the same as that of direct encryption of $\left[r_p(z)\right]_2 \oplus m_i$. Because p is an odd number, we have $\left[q_p(z)\right]_2 = \left[r_p(z)\right]_2 \oplus \text{parity}(z)$. If $\mathcal{A}$ can predict the encrypted bit with a significant advantage under pk, that is, $a_i = \left[r_p(z)\right]_2 \oplus m_i$. So:

$$b_i = a_i \oplus \text{parity}(z) \oplus m_i = \left[r_p(z)\right]_2 \oplus \text{parity}(z) = \left[q_p(z)\right]_2$$

When $\mathcal{A}$ guesses successfully, b_i represents the LSB prediction of $q_p(z)$. Therefore, the subroutine can output $\left[q_p(z)\right]_2$ with high probability.

5.1.3. Binary GCD algorithm

Under the condition that the LSB oracle of $\left[q_p(z)\right]_2$ can obtain according to the above subroutine, a binary GCD algorithm can be constructed as in [11]. For arbitrary two integers $z_i = q_p(z_i) \cdot p + r_p(z_i), i = \{1, 2\}$, we solve the odd part of GCD$(q_p(z_1), q_p(z_2))$. The detailed description is as Algorithm2.

Since $r_p(z_i) \ll p$, deducting the parity bit does not alter the quotient of z relative to p, but only the remainder, namely, $q_p(z_i - \text{parity}(z_i)) = q_p(z_i)$. So if z'_i is the result after recalculation in line 7, we have:

$$q_p(z'_i) = \frac{q_p(z_i)}{2},\ r_p(z'_i) = \frac{r_p(z_i - \text{parity}(z_i))}{2}$$

$$\left|r_p(z'_i)\right| \leq \frac{\left|r_p(z_i) + 1\right|}{2} \leq \left|r_p(z_i)\right|$$

Algorithm 2 Binary GCD

```
Input: Two integers $z_1 = q_p(z_1) \cdot p + r_p(z_1), z_2 = q_p(z_2) \cdot p + r_p(z_2), r_p(z_i) \ll p$.
Output: The odd part of $\mathrm{GCD}(q_p(z_1), q_p(z_2))$.
1: while $z_2 \neq 0$ do
2:     if $z_2 > z_1$ then
3:         Exchange them, $z_1 \Leftrightarrow z_2$; // To ensure $z_1$ is large
4:     end if
5:     Get the LSB predictions of $q_p(z_i)$ using Algorithm 1 and denoted by $b_i$;
6:     if $z_i$ with $b_i = 0$ then
7:         Recalculate $z_i$, $z_i = (z_i - \mathrm{parity}(z_i))/2$; // The new $z_i$ is an integer
8:     else if the parity bits of $q_p(z_i)$ are both 1 then
9:         Let $z_1 = z_1 - z_2$ and $b_1 = 0$; // Redefine to make $z_1$ even
10:    end if
11: end while
12: Return the final value of $q_p(z_1')$ as the odd part of $\mathrm{GCD}(q_p(z_1), q_p(z_2))$ for initial integers.
```

For z_1 replaced by $z_1 - z_2$ in line 9, then will perform the calculation in line 7 in the next cycle, we record the results of the two steps as z_1', z_2'' temporarily. So, we get:

$$\left|r_p(z_1'')\right| = \frac{\left|r_p(z_1) - r_p(z_2) - \mathrm{parity}(z_1')\right|}{2} \leq \max\left\{r_p(z_1), r_p(z_2)\right\}$$

Therefore, $r_p(z_i)$ will not exceed the largest of the original two numbers. The above process is equivalent to continuously using binary GCD on $q_p(z_i)$. Finally, two integers $z_1', z_2' = 0$ will be obtained after $O(\gamma)$ iteration, and $q_p(z_1')$ is the odd part of the original two integers $\mathrm{GCD}(q_p(z_1), q_p(z_2))$.

5.1.4. Solution of p

To solve p, $\mathcal{B}$ selects a pair of elements z_1^*, z_2^* from $\mathcal{D}'_\rho(p, q_0)$ and runs the binary GCD algorithm. The probability that $q_p(z_1^*)$ and $q_p(z_2^*)$ are coprime is at least $\frac{6}{\pi^2} \approx 0.6$. So we will get an integer $\tilde{z} = 1 \cdot p + r$, $|r| < 2^\rho$ in Algorithm 2. Otherwise, $\mathcal{B}$ reselects two numbers from $\mathcal{D}'_\rho(p, q_0)$. Then, let $z_1 = z_1^*, z_2 = \tilde{z}$ and use the binary GCD algorithm on these two numbers. The binary representation of $q_p(z_1^*)$ can be obtained from the sequence of parity bits of $q_p(z_1^*)$ in the iterative process, that is, $\mathcal{B}$ can get $q_p(z_1^*)$. Finally, $\mathcal{B}$ finds p by calculating $p = \lfloor z_1^* / q_p(z_1^*) \rceil$.

So far, we have completed how simulator $\mathcal{B}$ uses $\mathcal{A}$ to solve p under the condition that $\mathcal{A}$ can guess the plaintext with certain advantage. Then, we will analyze the probability of success of $\mathcal{B}$. As demonstrated in [11]: We denote the set of $p \in [2^{\eta-1}, 2^\eta)$ by $\mathcal{P}$. If $\mathcal{A}$ has an advantage ε in guessing the encrypted bits under pk, for at least $\varepsilon/2$ of p in $\mathcal{P}$, the advantage of $\mathcal{A}$ is at least $\varepsilon/2$. Go a step further, for such a fixed p, the advantage of $\mathcal{A}$ is at least $\varepsilon/4$ for at least $\varepsilon/4$ of its corresponding public key. In line 4 of Algorithm 1, $\mathcal{A}$ has advantages at least $\varepsilon/4 - \mathrm{negl}$. Then the correct answer will be returned with an overwhelming probability from the majority, $\mathcal{B}$ will utilize this to find p.

Therefore, for such p, the success probability of simulator $\mathcal{B}$ is at least $\varepsilon/4 - \mathrm{negl}$ in a round. By repeating above process $(4/\varepsilon) \cdot \omega(\log \lambda)$ times with random public keys, p will be recovered with a significant probability. In addition, the overall success probability of $\mathcal{B}$ is the corresponding density of $\mathcal{P}$, namely at least $\varepsilon/2$. At this point, we have completed the proof of Theorem 3.

The security of the scheme is reduced to the error-free AGCD problem by Theorem 3. Since the error-free AGCD assumption holds, that is, the problem is difficult for any polynomial time solver, our improved encryption scheme is semantically secure.

Table 2
Summary of integer factorization algorithms.

Algorithm	Time complexity
Trial division	$O(\sqrt{n}/\log n)$
Pollard's rho algorithm [44]	$O(n^{\frac{1}{4}}/\log n)$
Continued fraction method	$O\left(\exp\left(c\sqrt{\log n \log\log n}\right)\right)$
Elliptic curve factorization [45]	$O\left(\exp\left(2+o(1)\right)\sqrt{\log p \log\log p}\right)$
Quadratic sieve [46]	$O\left(\exp\left(1+o(1)\right)\sqrt{\log n \log\log n}\right)$
General number field sieve [47]	$O\left(\exp\left(\left(\frac{64}{9}\right)^{\frac{1}{3}} + o(1)\right)(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right)$

5.2. Security of multi-key homomorphic encryption scheme

In the multi-key scheme, each user encrypts the message with his own public key, and the encryption algorithm adopts the improved DGHV scheme, so the security inherits the single-key scheme, which has been proved in the previous section. According to the proof in [29], an adversary $\mathcal{A}$ with advantage ε can be converted into a solver $\mathcal{B}$ to solve the error-free approximate GCD. Here we argue that using prime number of the form $9k+1$ will not enhance the advantage of the adversary. We mainly from the public key and ciphertext two aspects of analysis. In the public key, the selection of q_0 is random, and random number r is added to the composition of other public key elements. Similarly, random noise is added to each ciphertext during encryption, so that the randomness of the generated public key and ciphertext will not be changed because of the form of secret key. Therefore, no additional advantage is given to the $\mathcal{A}$, that is, this selection method does not affect the security of the scheme.

Compared with previous multi-key schemes, such as [36–38,40], we introduce the extended key in the process of ciphertext extension. Therefore, it is necessary to ensure that the generation of extended key will not affect the security of the scheme, that is, the user cannot obtain secret keys of other users through cracking the extended key. Then we respectively prove that users and the cloud server in the system are zero-knowledge.

For user i, his own public and secret keys and the joint secret key generated by a trusted third party are received. Then user i calculates the extended key ek_i according to the joint secret key and the personal secret key. The essence of extended keys ek_i is the product of some large prime numbers. Hence, the security depends on the intractability of the problem of factoring large integers. In addition, when the number of users is 2, the indecomposability of the keys is guaranteed due to the existence of the evaluation key. In Table 2, we list some effective methods to deal with the integer factoring. So far, for the large integer factorization problem, most of the research work is to improve existing algorithms.

None of these algorithms succeed in factoring the product of 1024-bit prime numbers. In the parameter selection of our scheme, the bit size of prime p_i is $\eta = \lambda^2$. The product of two or even more primes will far exceed the factorization range of these algorithms, so it can resist the known attack on integer factorization. Therefore, in addition to knowing relevant information of the user's own keys, the user cannot further decompose to obtain secret keys of other users. That is to say, although the user knows the joint secret key, the joint secret key will not reveal the information of others.

For the cloud server, the evaluation key is received and used in the homomorphic multiplication. Generating the evaluation key by the trusted third party is feasible because it is the same as generating the secret key for a user. However, it is impossible for the cloud server to obtain the participating users' secret keys and the joint secret key based on the evaluated key. Therefore, the evaluation key is zero-knowledge to the cloud server.

Table 3
Comparison of relevant schemes.

Scheme	ρ	η	γ	α	β	τ	Constraint condition	Public key	Multi-key	Difficult problem
DGHV [11]	$O(\lambda)$	$O(\lambda^2)$	$O(\lambda^5)$	/	/	$O(\lambda^5)$	$\tau \geq \gamma + \omega(\log \lambda)$	$O(\lambda^{10})$	×	Approximate GCD
CMNT [29]	$O(\lambda)$	$O(\lambda^2)$	$O(\lambda^5)$	$O(\lambda)$	$O(\lambda^2)$	/	$\alpha \cdot \beta^2 \geq \gamma + \omega(\log \lambda)$	$O(\lambda^7)$	×	Error-free approximate GCD
CNT [30]	$O(\lambda)$	$O(\lambda^2)$	$O(\lambda^5)$	$O(\lambda^2)$	/	$O(\lambda^3)$	$\alpha \cdot \tau \geq \gamma + \omega(\log \lambda)$	$O(\lambda^5)$	×	Error-free approximate GCD
Chen [31]	$O(\lambda)$	$O(\lambda^2)$	$O(\lambda^5)$	$O(\lambda^2)$	$O(\lambda^{1.5})$	/	$\alpha \cdot \beta^2 \geq \gamma + \omega(\log \lambda)$	$O(\lambda^{3.5})$	×	Error-free approximate GCD
Ours	$O(\lambda)$	$O(\lambda^2)$	$O(\lambda^5)$	$O(\lambda^2)$	$O(\lambda)$	/	$\alpha \cdot \beta^3 \geq \gamma + \omega(\log \lambda)$	$O(\lambda^3)$	√	Error-free approximate GCD Integer factorization

5.3. Some discussion

5.3.1. Decryption mode

In the previous multi-key schemes [36–38,40], decryption requires secret keys of all participants, which is mainly caused by the construction of the joint secret key being a concatenation. However, since it is impractical for one user to obtain secret keys of all users, only the trusted party with secret keys of all participants can decrypt. Therefore, it is necessary to construct a two-round MPC protocol to complete distributed decryption such as [36,40]. The distributed decryption protocol is mainly divided into two steps: (1) each user executes partial decryption of the corresponding entry; (2) the results of partial decryption are combined to complete final decryption.

It is difficult for users to decompose the joint secret key in our multi-key scenario, so we take user i as an example, where user i has his own keys (pk_i, sk_i) and the joint secret key. In the cloud environment, two-round MPC is not required to decrypt multi-key ciphertexts from the cloud, and other users can be offline. In addition, for the data that user i wants to share with other users, the cloud server stores the extended ciphertexts, and user i may still use and download from the cloud. Besides decrypting by the joint secret key, we provide another option, that is, user i can still decrypt by using his own secret key. The principle is as follows:

According to the algorithm, it can be known that the user's extended key and personal secret key are coprime. Therefore, there is an inverse of ek_i under modulus p_i, which can be obtained by the Extended Euclidean Algorithm and satisfy:

$$ek_i \cdot ek_i^{-1} \equiv 1 \bmod p_i$$

For the extended ciphertext $\hat{c}_i$ of user i, the modulus can be returned to p_i by using the inverse ek_i^{-1}, and then decrypted with his secret key p_i, that is:

$$m' \leftarrow \left(3\hat{c}_i \cdot ek_i^{-1}\right) \bmod p_i \bmod 3$$

5.3.2. How to manage noise growth

Multiplication leads to faster noise growth than addition in homomorphic encryption. Noise grows almost exponentially with the number of multiplications in the original framework [11]. The modulus is the upper bound for correct decryption. So noise reaches the limit after performing logarithmic multiplications. FHE is achieved by bootstrapping, that is, homomorphic decryption to refresh noise. However, the bootstrapping process of standard FHE under the circular security is very cumbersome.

It is first proposed FHE without bootstrapping through key-switching and modulus-switching in [17]. A DGHV version of modulus-switching algorithm was first presented in [30], running the algorithm after each multiplication to construct the DGHV scheme without bootstrapping. Similarly, this technique can be applied to our scheme to generate a set of decreasing modulus based on the joint secret key. Here we describe the conversion process between two modulus as Algorithm 3.

According to Lemma 1 and Lemma 2 [30], it can be known ciphertext c'' after switching is under sk_2, and the noise is P'/P of the original.

Algorithm 3 Modulus Switching

```
Input: Public parameters, two modulus $sk_1 = P, sk_2 = P'$ and the ciphertext $c$ under $sk_1$.
Output: The ciphertext $c''$ under $sk_2$.
1: Let $\kappa = 2\gamma + n_1$; // $n_1, n_2$ are the sizes of $sk_1, sk_2$, $\gamma$ is the size of public key elements
2: Randomly generate a vector $\mathbf{s}$ of $\Theta$ bits;
3: Generate a vector $\mathbf{y}$ containing $\Theta$ numbers with a precision of $\kappa$ bits after the binary point and satisfy:
   $$\frac{2^{n_2}}{P} = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon \bmod 2^{n_2+1}, |\varepsilon| \leq 2^{-\kappa}$$
4: Calculate $\mathbf{s}'$ =Powersof2$\left(\mathbf{s}, n_2\right)$;
5: Calculate the encryption of $\mathbf{s}'$ under $sk_2$ and get ciphertext vector $\boldsymbol{\sigma} = P' \cdot \mathbf{q} + \mathbf{r} + \left\lfloor \mathbf{s}' \cdot \frac{P'}{2^{n_2+1}} \right\rceil$; // where $\mathbf{q} \leftarrow \left(\mathbb{Z} \cap \left[0, 2^{\gamma}/P'\right)\right)^{(n_2+1)\cdot\Theta}$, $\mathbf{r} \leftarrow \left(\mathbb{Z} \cap \left(-2^{\rho'}, 2^{\rho'}\right)\right)^{(n_2+1)\cdot\Theta}$
6: for $i = 1$ to $\Theta$ do
7:     Calculate $c_i = \lfloor c \cdot y_i \rceil \bmod 2^{n_2+1}$;
8: end for
9: Compute $\mathbf{c}' = $ BitDecomp$\left(\mathbf{c}, n_2\right)$,where $\mathbf{c} = (c_1, \cdots, c_\Theta)$;
10: Let $c'' = 2\langle \boldsymbol{\sigma}, \mathbf{c}' \rangle + [c]_2$;
11: Return $c''$.
```

6. Experimental performance and analysis

In this section, we simulate our scheme and the contrastive schemes in Python programming language, mainly compares the storage capacity of public key, the time of key generation and the function of these schemes. The setting of the experimental environment is 64-bit Windows 7, the CPU is Intel Core i5-4590 clocked at 3.30 GHz and the memory is 4 GB. In addition, in the implementation of the algorithm, we use the gmpy2 database to optimize the operation.

6.1. Parameter size and function analysis

In Table 3, we compare the theoretical values of the parameters in [11,29–31] with our scheme, where λ is the security parameter. α is the bit-length of the noise used for encryption. β is related to the number of the subgroup public key elements, τ is the number of integers in public key and $\beta = \sqrt{\tau}$ in [29,31], $\beta = \sqrt[3]{\tau}$ in our scheme. Besides, α and β meet the constraints in Table 3 in order to use the leftover hash lemma in the reduction to approximate GCD assumption. In the experiment, the cubic form used in the public key element during encryption instead of linear or quadratic form. In other words, the public key elements can be obtained by multiplying $x_{i,0}$, $x_{j,1}$ and $x_{k,2}$, where $1 \leq i,j,k \leq \beta$. There are β elements in each part, so 3β elements need to be saved in the public key. The original public key can be obtained by multiplying these three groups of elements. Similar to the quadratic offset compression scheme proposed by Chen et al. [31], we reserve 3β offsets $\delta_{i,b}$, where $1 \leq i \leq \beta$, $b \in \{0,1,2\}$ in the public key. The sizes of $\delta_{i,b}$ and β are $O(\lambda^2)$ and $O(\lambda)$ respectively, so the public key has the size $O(\lambda^3)$, which is optimal. In addition, due to the large integer factoring problem, our proposed scheme can be extended to multi-key

Table 4
Value of security parameter.

Security parameter level	Actual value of λ
Toy	42
Small	52
Medium	62
Large	72

Table 5
The parameters of toy level.

Scheme	ρ	η	$\gamma \times 10^{-6}$	α	β	τ
DGHV [11]	16	1088	0.16	/	/	158
CMNT [29]	16	1086	0.16	/	12	/
CNT [30]	27	1026	0.15	936	/	158
Chen [31]	26	1040	0.15	936	12	/
Ours	26	1050	0.15	936	6	/

Table 6
The parameters of small level.

Scheme	ρ	η	$\gamma \times 10^{-6}$	α	β	τ
DGHV [11]	24	1626	0.86	/	/	527
CMNT [29]	24	1632	0.86	/	23	/
CNT [30]	41	1558	0.83	1476	/	572
Chen [31]	41	1640	0.85	1476	23	/
Ours	41	1646	0.85	1476	9	/

Table 7
The parameters of medium level.

Scheme	ρ	η	$\gamma \times 10^{-6}$	α	β	τ
DGHV [11]	32	2177	4.23	/	/	2110
CMNT [29]	32	2176	4.20	/	44	/
CNT [30]	56	2128	4.20	2016	/	2110
Chen [31]	56	2240	4.25	2016	44	/
Ours	56	2250	4.25	2016	11	/

Table 8
The parameters of large level.

Scheme	ρ	η	$\gamma \times 10^{-6}$	α	β	τ
DGHV [11]	39	2653	19.00	/	/	7654
CMNT [29]	39	2652	19.00	/	88	/
CNT [30]	71	2698	19.35	2556	/	7659
Chen [31]	71	2840	19.00	2556	88	/
Ours	71	2846	19.00	2556	20	/

homomorphic encryption, while the other comparison schemes cannot. Then we set the parameters, calculate the actual storage capacity and the time consumption of five schemes, and analyze the comparison results in detail.

6.2. Comparison of actual storage and time

Public key size and time consumption are the main factors affecting the performance of the scheme. In the experiment, the size of security parameters are set according to [10] and λ is divided into four different security levels in Table 4.

After the security parameter is fixed, the values of other parameters in the scheme according to theoretical calculation are too large to facilitate operation. During the experiment, according to these four levels of the security parameter, concrete values of other parameters are shown from Tables 5 to 8.

Because the public key generated by DGHV [11] is too large, it can reach tens of GB at the large level, and the public key generation time is up to more than ten hours. Therefore, in the subsequent description of this paper, we mainly compare with other three schemes.

Fig. 3(a) shows the change of key generation time under different security levels. With the increase of security parameter, the key generation time of CMNT scheme [29] increases significantly. When the security parameter is 72, the key generation time reaches 43 min, while the CNT scheme [30] takes less time than CMNT. When the security parameter is 72, the time is 13.9 min. Chen's [31] and our scheme have little difference in time in minutes. Therefore, we further compare our scheme with Chen's scheme in Fig. 3(b). It can be seen that the key generation of our scheme is faster than Chen's scheme. Although when the security level is low, the time of our scheme is less than one second different from that of Chen's scheme, the time consumption of our scheme is no more than 10 s at large level while Chen's scheme needs 20.12 s.

Fig. 3(c) shows the effect of security parameter on storage capacity of public key. The public key storage spaces increase with the increase of the security parameter. Distinctly, the public key storage capacity of CMNT [29] is the highest and increases most obviously. At large level, the public key size reaches 406.5MiB while the other three schemes are smaller. Therefore, we further compare our scheme with CNT [30] and Chen's scheme [31] in Fig. 3(d). It can be seen that the public key size of our scheme is the smallest. When the security level is low, our public key size is only half of Chen's scheme. When the security parameter reaches 72, the memory occupied by x_0 in the public key will be much larger than that of other elements, Therefore, the storage capacity of public key is not significantly reduced at large level, but it is also lower than other schemes.

In Fig. 4, we further compare the number of elements in the public key. CMNT scheme and Chen's scheme are equivalent to dividing the public key into two subsets and there are 2β public key elements. In our optimization, it is divided into three subsets. When the security parameters are increased, β will not grow too fast. So there are fewer elements stored in our scheme. For encryption, our public key compression scheme is more complex than Chen's scheme in form, but no additional operation is introduced. At the same time, the decryption steps of the above homomorphic encryption schemes are actually the same, so there is little difference in encryption and decryption time.

6.3. Performance analysis of multi-key scheme

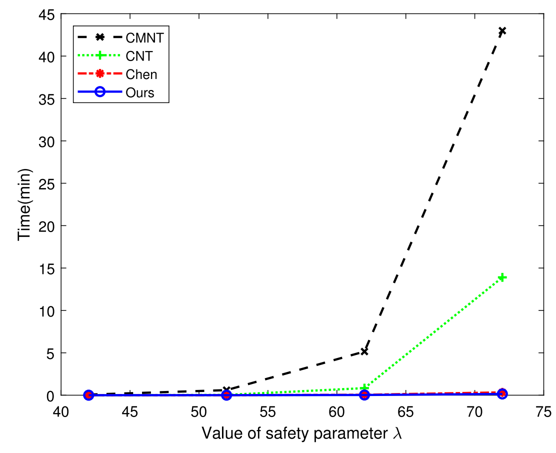
In our multi-key homomorphic encryption scheme, we introduce the extended key in the process of ciphertext extension. The calculation of the extended key is closely related to the number of participating users.

Table 9
Time to calculate the extended key.

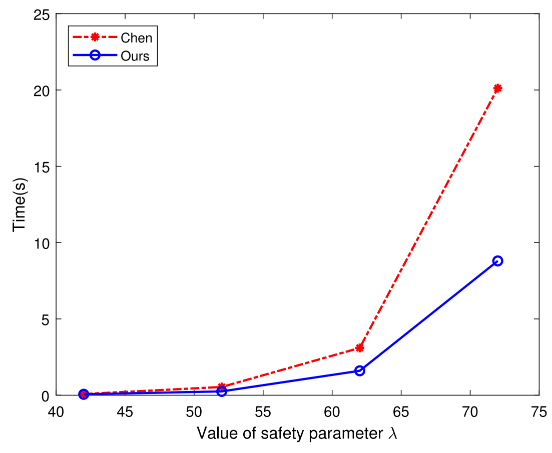
Number of users	10	20	30	40	50
Time(ms)	12	24	33	45	54

In Table 9, we show the time consumed to calculate the users' extended key under different number of users. It takes only about 1 millisecond on average to calculate the extended key of a user. Fig. 5 shows the impact of number of users on the generation time of the secret key and the public key respectively. It can be seen that the key generation time increases linearly with the increase of the number of users. When the number of users is 50, the sum of the two times is about 80 s. Hence, the proportion of calculated extended key in the whole key generation stage is very small, even negligible.

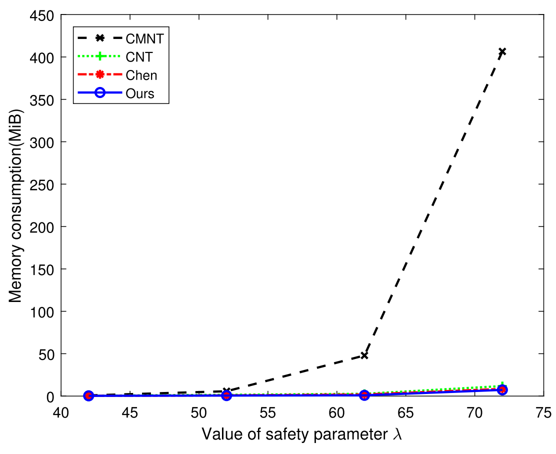
Table 10 shows the running times of ciphertext extension, homomorphic operation of extended ciphertexts and decryption using the joint secret key in our multi-key scheme at different security levels. In the experiment, we assume that the number of users is 20. From Table 10, we can see that there is little time difference between decrypting the extended ciphertext with the joint secret key and the original single key scheme. Because they are the same modular operation, so the difference between them can be ignored. In our scheme, the addition operation is the fastest for the operation of extended ciphertexts. Both ciphertext extension and multiplication are more complex, so they take more time. Furthermore, homomorphic multiplication is not simply multiplying, so it takes the longest time, but the time is still within the acceptable range.



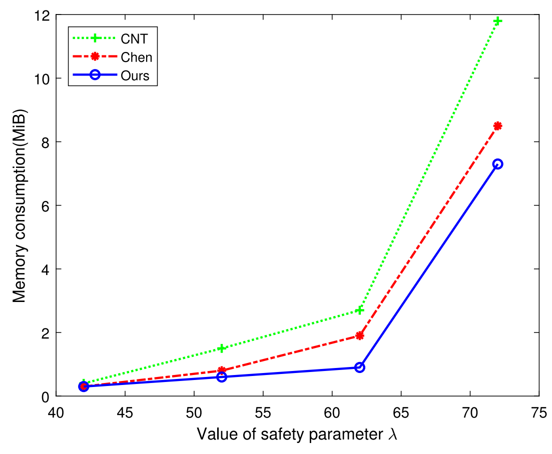
(a)



(b)



(c)



(d)

Fig. 3. Comparison of time and storage capacity.

Table 10
Running time of our multi-key scheme.

Security parameter level	Ciphertext expansion	Addition	Multiplication	Decryption
Toy	0.06 s	0.02 s	0.12 s	0.02 s
Small	0.23 s	0.08 s	0.65 s	0.03 s
Medium	0.98 s	0.51 s	3.34 s	0.05 s
Large	4.2 s	2.12 s	13.95 s	0.07 s

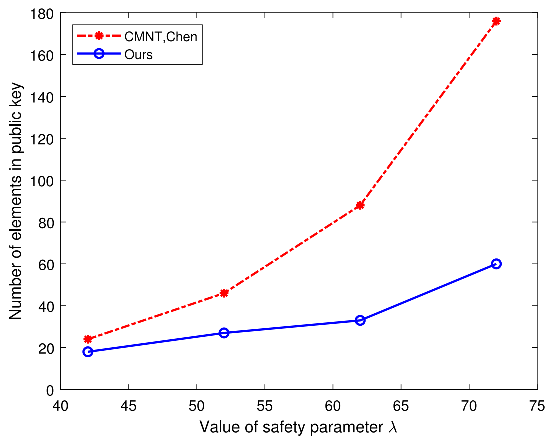


Fig. 4. Comparison of the number of elements in the public key.

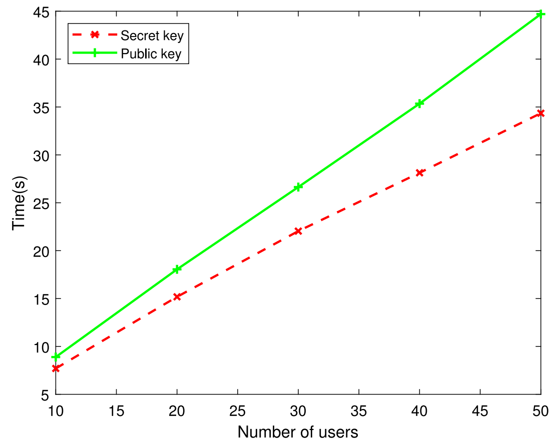


Fig. 5. Impact of number of users.

7. Conclusion

Considering the diversity of users in the cloud computing environment, MKHE supports information storage and sharing and secure homomorphic calculation from different users. It is more suitable to solve privacy and security issues in the cloud computing, which is of great practical significance. However, the existing MKHE schemes weaken the functionality of MKHE in the expansion of ciphertext dimension and computational complexity. In this paper, we first improve the DGHV scheme [11] and propose a DGHV-type MKHE scheme. Compared with the original DGHV scheme, on the premise of ensuring security, an extended key is introduced to expand the function of the scheme. Compared with other MKHE schemes, our scheme is easier to understand in ciphertext composition and calculation form. In addition, in the experiment, the cubic form is used to reduce the public key size of the scheme to $O(\lambda^3)$. Through experiments, we evaluate the generation time of the keys, storage capacity, expansion and homomorphic operation. However, introducing a trusted KGC when generating joint key may be difficult to meet in some practical applications. Whether to cancel the setting of KGC in joint key distribution is an important research content. We should also strive to make MKHE better applicable to the cloud environment.

CRediT authorship contribution statement

Xuelian Li: Conceptualization, Resources, Data curation. **Hui Li:** Methodology, Visualization, Writing-original draft. **Juntao Gao:** Formal analysis, Writing-review and editing, Funding acquisition. **Runsong Wang:** Software, Validation, Investigation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported in part by the Key Research and Development Program of Shaanxi (No. 2021ZDLGY06-04) and by the Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201802). All authors approved the version of the manuscript to be published.

References

[1] Chettri L, Bera R. A comprehensive survey on internet of things (IoT) toward 5G wireless systems. IEEE Internet Things J 2020;7(1):16–32.
[2] Gill SS, Tuli S, Xu M, Singh I, Singh KV, Lindsay D, et al. Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges. Internet Things 2019;8:100–18.
[3] China Academy of Information and Communications Technology. Cloud computing whitepaper 2021. 2021, [Online]. Available: http://www.caict.ac.cn/kxyj/qwfb/bps/202107/t20210727_381205.htm.
[4] Shen J, Zhou TQ, Chen XF, Li J, Susilo W. Anonymous and traceable group data sharing in cloud computing. IEEE Trans Inf Forensics Secur 2018;13(4):912–25.
[5] Pu YW, Hu CQ, Deng SJ, Alrawais A. R^{2}PEDS: a recoverable and revocable privacy-preserving edge data sharing scheme. IEEE Internet Things J 2020;7(9):8077–89.
[6] Bethencourt J, Song D, Waters B. New techniques for private stream searching. ACM Trans Inf Syst Secur 2009;12(3):1–32.
[7] Popa RA, Li FH, Zeldovich N. An ideal-security protocol for order-preserving encoding. In: 2013 IEEE Symposium on Security and Privacy. Berkeley, CA, USA; 2013, p. 463–77.
[8] Zhang C, Zhu L, Xu C, Sharif K, Zhang C, Liu X. PGAS: Privacy-preserving graph encryption for accurate constrained shortest distance queries. Inform Sci 2020;506:325–45.
[9] Furukawa J. Short comparable encryption. In: 13th International Conference Cryptology and Network Security, vol. 8813. Heraklion, Crete, Greece: Springer; 2014, p. 337–52.
[10] Gentry C. Fully homomorphic encryption using ideal lattices. In: 41st Annual ACM Symposium on Theory of Computing. Bethesda, MD, USA: ACM; 2009, p. 169–78.
[11] van Dijk M, Gentry C, Halevi S, Vaikuntanathan V. Fully homomorphic encryption over the integers. In: Advances in Cryptology – EUROCRYPT, vol. 6110. French Riviera: Springer; 2010, p. 24–43.
[12] Li J, Song DJ, Chen SC, Lu XF. A simple fully homomorphic encryption scheme available in cloud computing. In: 2nd IEEE International Conference on Cloud Computing and Intelligence Systems. Hangzhou, China: IEEE; 2012, p. 214–7.
[13] Kocabas Ö, Soyata T. Utilizing homomorphic encryption to implement secure and private medical cloud computing. In: 8th IEEE International Conference on Cloud Computing. New York, USA; p. 540–7.
[14] Ren SQ, Tan BHM, Sundaram S, Wang T, Ng Y, Chang V, et al. Secure searching on cloud storage enhanced by homomorphic indexing. Future Gener Comput Syst 2016;65:102–10.
[15] López-Alt A, Tromer E, Vaikuntanathan V. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: 44th Annual ACM Symposium on Theory of Computing. New York, USA: ACM; 2012, p. 1219–34.
[16] Chen H, Dai W, Kim M, Song Y. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: 2019 ACM SIGSAC Conference on Computer and Communications Security. London, UK: ACM; 2019, p. 395–412.
[17] Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping. ACM Trans Comput Theory 2014;6(3):1–36.
[18] Ma J, Naas S, Sigg S, Lyu X. Privacy-preserving federated learning based on multi-key homomorphic encryption. Int J Intell Syst 2022;37(9):5880–901.
[19] Asharov G, Jain A, López-Alt A, Tromer E, Vaikuntanathan V, Wichs D. Multiparty computation with low communication, computation and interaction via threshold FHE. In: Advances in Cryptology – EUROCRYPT, vol. 7237. Cambridge, UK: Springer; 2012, p. 483–501.
[20] Mouchet C, Troncoso-Pastoriza JR, Bossuat J, Hubaux J. Multiparty homomorphic encryption from ring-learning-with-errors. Proc Priv Enhanc Technol 2021;2021(4):291–311.
[21] Park J. Homomorphic encryption for multiple users with less communications. IEEE Access 2021;9:135915–26.
[22] Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology – CRYPTO, vol. 8042. Santa Barbara, CA, USA: Springer; 2013, p. 75–92.
[23] Dyer J, Dyer ME, Xu J. Practical homomorphic encryption over the integers for secure computation in the cloud. Int J Inf Sec 2019;18(5):549–79.
[24] Rivest RL, Adleman LM, Dertouzos ML. On data banks and privacy homomorphisms. Found Secure Comput 1978;4(11):169–78.
[25] Elgamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory 1985;31(4):469–72.
[26] Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology – EUROCRYPT. Prague Czech Republic: Springer; 1999, p. 223–38.
[27] Gentry C. Toward basing fully homomorphic encryption on worst-case hardness. In: Advances in Cryptology – CRYPTO, vol. 6223. Santa Barbara, CA, USA: Springer; 2010, p. 116–37.
[28] Gentry C, Halevi S. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In: IEEE 52nd Annual Symposium on Foundations of Computer Science. Palm Springs, CA, USA; 2011, p. 107–9.
[29] Coron J-S, Mandal A, Naccache D, Tibouchi M. Fully homomorphic encryption over the integers with shorter public keys. In: Advances in Cryptology – CRYPTO, vol. 6841. Santa Barbara, CA, USA: Springer; 2011, p. 487–504.
[30] Coron J-S, Naccache D, Tibouchi M. Public key compression and modulus switching for fully homomorphic encryption over the integers. In: Advances in Cryptology – EUROCRYPT, vol. 7237. Cambridge, UK: Springer; 2012, p. 446–64.
[31] Chen LQ, Lim M, Fan ZJ. A public key compression scheme for fully homomorphic encryption based on quadratic parameters with correction. IEEE Access 2017;5:17692–700.
[32] Chen Y, Nguyen PQ. Faster algorithms for approximate common divisors: breaking fully-homomorphic-encryption challenges over the integers. In: Advances in Cryptology – EUROCRYPT, vol. 7237. Cambridge, UK: Springer; 2012, p. 502–19.
[33] Nuida K, Kurosawa K. (Batch) fully homomorphic encryption over integers for non-binary message spaces. In: Advances in Cryptology – EUROCRYPT, vol. 9056. Sofia, Bulgaria: Springer; 2015, p. 537–55.
[34] Chongchitmate W, Ostrovsky R. Circuit-private multi-key FHE. In: 20th Public-Key Cryptography – PKC, vol. 10175. Amsterdam, Netherlands: Springer; 2017, p. 241–70.
[35] Zhou TP, Li NB, Lai QQ, Yang XY, Han YL, Liu WC. Efficient multi-key fully homomorphic encryption over prime cyclotomic rings with fewer relinearisations. IET Inf Secur 2021;15(6):472–86.

[36] Mukherjee P, Wichs D. Two round multiparty computation via multi-key FHE. In: Advances in Cryptology – EUROCRYPT, vol. 9666. Vienna, Austria: Springer; 2016, p. 735–63.

[37] Peikert C, Shiehian S. Multi-key FHE from LWE, Revisited. In: 14th Theory of Cryptography – TCC, vol. 9986. Beijing, China: Springer; 2016, p. 217–38.

[38] Brakerski Z, Perlman R. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: Advances in Cryptology – CRYPTO, vol. 9814. Santa Barbara, USA: Springer; 2016, p. 190–213.

[39] Regev O. On lattices, learning with errors, random linear codes, and cryptography. J ACM 2009;56(6):1–40.

[40] Chen L, Zhang Z, Wang X. Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension. In: 15th Theory of Cryptography – TCC, vol. 10678. Baltimore, MD, USA: Springer; 2017, p. 597–627.

[41] Zhang P, Huang T, Sun X, Zhao W, Liu H, Lai S, et al. Privacy-preserving and outsourced multi-party K-means clustering based on multi-key fully homomorphic encryption. IEEE Trans Dependable Secure Comput 2022;1–12.

[42] Kim T, Kwak H, Lee D, Seo J, Song Y. Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition. IACR Cryptol ePrint Arch 2022;347.

[43] HÅstad J, Impagliazzo R, Levin LA, Luby M. A pseudorandom generator from any one-way function. SIAM J Comput 1999;28(4):1364–96.

[44] Brent RP, Pollard JM. Factorization of the eighth Fermat number. Math Comput 1981;36(154):627–30.

[45] Lenstra H. Factoring integers with elliptic curves. Ann of Math 1987;126(3):649–73.

[46] Silverman RD. The multiple polynomial quadratic sieve. Math Comput 1987;48(177):329–39.

[47] Lenstra AK, Lenstra HW, Manasse MS, Pollard JM. The number field sieve. In: Lenstra AK, Lenstra HW, editors. The development of the number field sieve, vol. 1554. Berlin, Germany: Springer; 1993, p. 11–42.