Hindawi
Security and Communication Networks
Volume 2017, Article ID 7695751, 11 pages
https://doi.org/10.1155/2017/7695751

Research Article

An Efficient Secret Key Homomorphic Encryption Used in Image Processing Service

Pan Yang, Xiaolin Gui, Jian An, and Feng Tian

School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China

Correspondence should be addressed to Xiaolin Gui; xlgui@mail.xjtu.edu.cn

Received 30 April 2016; Revised 26 January 2017; Accepted 26 March 2017; Published 14 May 2017

Academic Editor: Zonghua Zhang

Homomorphic encryption can protect user's privacy when operating on user's data in cloud computing. But it is not practical for wide using as the data and services types in cloud computing are diverse. Among these data types, digital image is an important personal data for users. There are also many image processing services in cloud computing. To protect user's privacy in these services, this paper proposed a scheme using homomorphic encryption in image processing. Firstly, a secret key homomorphic encryption (IGHE) was constructed for encrypting image. IGHE can operate on encrypted floating numbers efficiently to adapt to the image processing service. Then, by translating the traditional image processing methods into the operations on encrypted pixels, the encrypted image can be processed homomorphically. That is, service can process the encrypted image directly, and the result after decryption is the same as processing the plain image. To illustrate our scheme, three common image processing instances were given in this paper. The experiments show that our scheme is secure, correct, and efficient enough to be used in practical image processing applications.

1. Introduction

Along with the arrival of the cloud computing fever, there emerge a lot of service outsourcing applications based on cloud computing platform (such as SaaS). Users who request the service just need to upload their data to the service and wait for the result. This brings users great benefits, but also the risk of privacy disclosure, because the service provider (SP) can access users' plain sensitive information arbitrarily. To balance the privacy with usability of data in cloud computing, many computable encryption technologies are proposed, such as homomorphic encryptions [1–5]. The idea of these encryptions can be represented as follows.

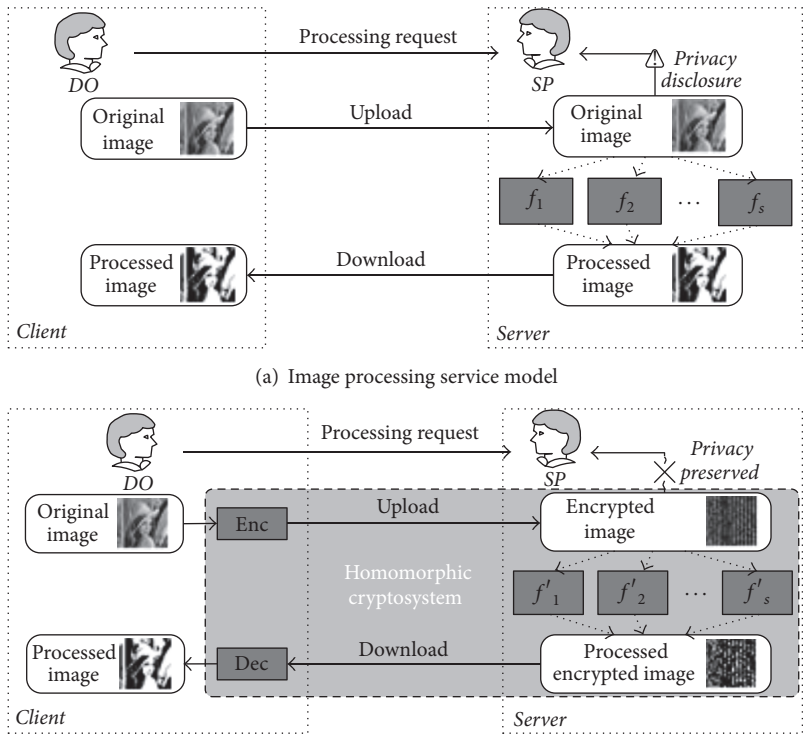
"Enc" represents the encryption process, and "Dec" represents the decryption process. For a function $f(\pi_0, \pi_1, \ldots, \pi_t)$ taking the plaintexts as input, there exists a function $f'(\psi_0, \psi_1, \ldots, \psi_t)$ (where $\psi_k = \mathrm{Enc}(\pi_k)$) taking the ciphertexts as input, such that

$$\mathrm{Dec}\left(f'\left(\psi_0, \psi_1, \ldots, \psi_t\right)\right) = f\left(\pi_0, \pi_1, \ldots, \pi_t\right). \quad (1)$$

According to (1), user uploads the encrypted data to service, and instead of using f to operate on user's plain data, the service can operate on the encrypted data by using f' and return to user the encrypted result. After decryption, user will get the expected result which is the same as f. Thus, the service can run correctly without knowing user's plain data. And then, the privacy is safe from disclosure.

Homomorphic encryption seems a good way to protect privacy in service outsourcing applications, especially when handling the integer data type. But as the data types in cloud computing are diverse, how to use homomorphic encryption in other types is still a challenging problem. For example, with the popularization of photograph equipment, a large amount of digital images are generated every day. It has become one of the most popular forms of personal data for users. Consequently, the online image processing services are widely used for users to edit their images. But the image may also contain privacy that the user does not want SP to see. To protect the privacy in this data type, we proposed a scheme using homomorphic encryption in image processing services.

In broad terms, image processing includes all kinds of operations on the image. But it is hard to handle the privacy issues in all kinds of image processing using one scheme.



(a) Image processing service model

(b) Encrypted image processing service model

Figure 1: The encrypted image processing model.

So in this paper, the image processing mainly refers to the processes based on pixels. That is, the new color of pixel is computed according to the old ones. The operations on plain pixels can be seen as a function f. Taking the old pixels and some other parameters (if necessary) as inputs, f outputs the new pixel. When using the traditional image encryptions [6–8], operating on the encrypted pixels is meaningless. Thus no one can process the encrypted image without decryption. However, by encrypting each pixel separately using homomorphic encryption, the service can operate on the encrypted pixels using f'. Then the service can process the encrypted image without decryption.

Unfortunately, there are no appropriate homomorphic encryptions which can be used in image encryption yet. Unpadded-RSA [1] and ElGamal [2] cryptosystems satisfy (1) when f consists of only multiplication operations. But it needs both addition and multiplication operations in image processing. Neither do Goldwasser and Micali [3] or Paillier [4] cryptosystems fit image encryption, as f consists of only addition operations. Gentry's fully homomorphic encryption [5] is too complex to apply in image processing, because it encrypts one bit each time, with the runtime more than 30 seconds even in "toy" security level [9]. The homomorphic cryptosystems proposed in [10, 11] are oriented to integers, but as the floating numbers are involved in operations in image processing, these cryptosystems cannot be used either. Besides, these are all public key encryptions which need a large key size to ensure the security. Thus, the sizes of ciphertexts are too large to store or transmit online.

Through the above analysis, we need an efficient homomorphic encryption which can support both addition and multiplication operations on floating numbers. Then we can use it to encrypt the image and process the encrypted image directly. It seems that no existing work has ever proposed any solution for this problem. Our contributions are as follows:

(i) We propose an encrypted image processing model based on homomorphic encryption.

(ii) We improve Gentry's homomorphic encryption to propose an efficient secret key homomorphic encryption which can support addition and multiplication operations on floating numbers (IGHE).

(iii) We use IGHE in image encryption and give the instances of processing encrypted image.

The rest of this paper is organized as follows. Section 2 outlines our encrypted image processing model and defines the notions that we need. In Section 3, we describe the improvement of Gentry's homomorphic encryption (IGHE). Section 4 gives the method of image encryption using IGHE, followed by the instances of processing encrypted image. The experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

2. Encrypted Image Processing Model

First of all, we construct an encrypted image processing model based on homomorphic encryption; see Figure 1.

Figure 1(a) is a common online image processing service model. The service has a set of image processing functions as $\{f_1, f_2, \ldots, f_s\}$. When image data owner (DO) uploads the image for a specific kind of process request, service provider (SP) will choose the corresponding function f_k to process the image. As SP can access the original image, the privacy may be leaked. In Figure 1(b), DO encrypts the image using a homomorphic encryption before uploading. Thanks to this encryption, SP can operate on the encrypted image with the function f'_k corresponding to f_k and return to DO the encrypted result image. After decryption, DO can get the correct processed image. The result is the same as Figure 1(a). But as the image in server is always in encrypted form, the privacy is preserved.

To realize the model, constructing an efficient homomorphic cryptosystem which supports both addition and multiplication operations on floating numbers is the key problem. We will discuss this in the next section. For clear description, the important notations used in this paper are summarized in Notations.

3. Improvement of Gentry's Homomorphic Encryption

3.1. The Initial Construction of Gentry's Scheme. Gentry denoted his initial scheme as ε [5]. It started by fixing a ring R and an ideal lattice $I \subset R$, with $\mathbf{B}_I$ as the basis. Then he used an algorithm IdealGen(R, $\mathbf{B}_I$) to output the public and secret keys $\mathbf{B}_J^{\text{pk}}$ and $\mathbf{B}_J^{\text{sk}}$ which are the bases of some ideal J, such that $I + J = R$. The plaintext space Π_ε was $[R]_{\mathbf{B}_I}$, and "$\text{Add}_{\mathbf{B}_I}$", "$\text{Mult}_{\mathbf{B}_I}$" were the addition and multiplication in Π_ε. For $\pi \in \Pi_\varepsilon$, ε set the ciphertext $\psi \xleftarrow{R} \pi + I + J$. In decryption, $\pi \leftarrow (\psi \bmod \mathbf{B}_J^{\text{sk}}) \bmod \mathbf{B}_I$. For any function $f(\pi_0, \pi_1, \ldots, \pi_t)$ consisting of "$\text{Add}_{\mathbf{B}_I}$" and "$\text{Mult}_{\mathbf{B}_I}$", $f'(\psi_0, \psi_1, \ldots, \psi_t)$ was generated by replacing "$\text{Add}_{\mathbf{B}_I}$" and "$\text{Mult}_{\mathbf{B}_I}$" with "$\text{Add}_{\mathbf{B}_J^{\text{pk}}}$" and "$\text{Mult}_{\mathbf{B}_J^{\text{pk}}}$", respectively.

Gentry proved that scheme ε satisfies (1), that is,

$$\text{Dec}_\varepsilon\left(\mathbf{B}_J^{\text{sk}}, f'\left(\psi_0, \psi_1, \ldots, \psi_t\right)\right) = f\left(\pi_0, \pi_1, \ldots, \pi_t\right). \quad (2)$$

But ε is too complex to encrypt image. For one thing, to prevent the attacker from working out the secret key, the dimension n of public key needs to be large enough ($n \geq 512$ [9]). For another, ε encrypts one bit each time ($\{0, 1\}$ need to be translated into Π_ε as $\{\mathbf{0}, \mathbf{1}\}$). Thus, supposing that each of the elements of vector has l bits, the ciphertext will be $n \cdot l$ times larger than plaintext. Therefore, the byte of the encrypted image file will be enlarged at least $512l$ times. This is unrealistic, not to mention the extra cost for operating on each encrypted bit in the image processing. So we need to simplify ε to make it more efficient.

3.2. Simplifying Gentry's Scheme. Our simplified scheme is denoted as ε_s. To reduce the size of key, we discard the public key encryption part. That is, the part of $\pi + I$ is reserved, but the public key encryption part (the ideal J) will no longer be used. The ciphertext of π is set as $\psi \xleftarrow{R} \pi + I$. In decryption, $\pi \leftarrow [\psi]_{\mathbf{B}_I}$. Then, the key problem is how to construct the ideal lattice I.

For any $\mathbf{b} \in R$, the ideal generated by $\mathbf{b}$ is $(\mathbf{b}) = \{\mathbf{b} * \mathbf{r} \mid \mathbf{r} \in R\} = \{\text{Rot}(\mathbf{b}) \cdot \mathbf{r} \mid r \in R\}$. According to Notations, $(\mathbf{b})$ is an ideal lattice with $\text{Rot}(\mathbf{b})$ as the basis. Thus, we set $\mathbf{B}_I \leftarrow \text{Rot}(\mathbf{b})$, and I is $(\mathbf{b})$. In ε_s, $\mathbf{b}$ can be seen as the secret key. The encryption and decryption algorithms are described as follows.

$Enc_{\varepsilon s}(\mathbf{b}, \pi)$. It takes as input the key $\mathbf{b}$ and a plaintext π. It sets $\mathbf{r} \xleftarrow{R} R$ and outputs $\psi \leftarrow \pi + \mathbf{b} * \mathbf{r}$.

$Des_{\varepsilon s}(\mathbf{b}, \psi)$. It takes as input the key $\mathbf{b}$ and a ciphertext ψ. It outputs $\pi \leftarrow [\psi]_{\mathbf{B}_I}$, where $\mathbf{B}_I = \text{Rot}(\mathbf{b})$.

Besides, there is an algorithm denoted as Eval which operates on ciphertext.

$Eval_{\varepsilon s}(f, \boldsymbol{\Psi})$. It takes as input a set of ciphertexts $\boldsymbol{\Psi} = \{\psi_0, \psi_1, \ldots, \psi_t\}$, where $\text{Dec}_{\varepsilon s}(\mathbf{b}, \psi_k) = \pi_k$, and a function $f(\pi_0, \pi_1, \ldots, \pi_t)$ consisting of "$\text{Add}_{\mathbf{B}_I}$" and "$\text{Mult}_{\mathbf{B}_I}$". It outputs $f'(\psi_0, \psi_1, \ldots, \psi_t)$ by replacing "π_k", "$\text{Add}_{\mathbf{B}_I}$", and "$\text{Mult}_{\mathbf{B}_I}$" with "$\psi_k$", "+", and "$*$", respectively.

Now let us consider the correctness of $\text{Eval}_{\varepsilon s}$; that is, ε_s satisfies (1).

Proof. For $\boldsymbol{\Psi} = \{\psi_0, \psi_1, \ldots, \psi_t\}$, $\psi_k = \pi_k + i_k$, where $\pi_k \in \Pi$, and $i_k \in I$, we have

$$\begin{aligned}
&\text{Eval}_{\varepsilon s}\left(f, \boldsymbol{\Psi}\right) = f'\left(\psi_0, \psi_1, \ldots, \psi_t\right) \\
&\quad = f'\left(\pi_0 + i_0, \pi_1 + i_1, \ldots, \pi_t + i_t\right), \\
&\text{Dec}_{\varepsilon s}\left(\mathbf{b}, \text{Eval}_{\varepsilon s}\left(f, \boldsymbol{\Psi}\right)\right) \\
&\quad = f'\left(\pi_0 + i_0, \pi_1 + i_1, \ldots, \pi_t + i_t\right) \bmod \mathbf{B}_I \\
&\quad = f'\left(\pi_0, \pi_1, \ldots, \pi_t\right) \bmod \mathbf{B}_I.
\end{aligned} \quad (3)$$

In $f(\pi_0, \pi_1, \ldots, \pi_t)$, instead of computing "$\text{Add}_{\mathbf{B}_I}$" and "$\text{Mult}_{\mathbf{B}_I}$", we can compute "+" and "$*$" to get a result $\pi \in R$ and then output $\pi \bmod \mathbf{B}_I$. Thus,

$$f'\left(\pi_0, \pi_1, \ldots, \pi_t\right) \bmod \mathbf{B}_I = f\left(\pi_0, \pi_1, \ldots, \pi_t\right). \quad (4)$$

That is, ε_s satisfy (1). □

The security of ε_s is the same as ε, that is, semantic security, as they are all based on ICP.

Definition 1 (ideal coset problem (ICP) in [5]). Fix R, $\mathbf{B}$, and an algorithm Samp_1 that efficiently samples R. The challenger sets $b \xleftarrow{R} \{0, 1\}$. If $b = 0$, it sets $\mathbf{r} \xleftarrow{R} \text{Samp}_1(R)$, $\mathbf{t} \leftarrow \mathbf{r} \bmod \mathbf{B}$. If $b = 1$, it samples $\mathbf{t}$ uniformly from $R \bmod \mathbf{B}$. The problem is guessing b by given $(\mathbf{t}, \mathbf{B})$.

Theorem 2. *Suppose that there is an algorithm A_1 that breaks the semantic security of ε_s with advantage ϵ. Then there is an algorithm A_2 that solves the ICP with advantage $\epsilon/2$.*

Proof. The challenger sends A_2 an ICP instance $(\mathbf{t}, \mathbf{B})$. A_2 sets $\beta \xleftarrow{R} \{0,1\}$, $\mathbf{t}_0 \leftarrow \mathbf{t}$ and samples $\mathbf{t}_1$ uniformly from $R \bmod \mathbf{B}$. Then, A_2 gives $\psi_\beta \leftarrow \mathrm{Enc}_{\varepsilon s}(\mathbf{B}, \mathbf{t}_\beta)$ to A_1; A_1 sends back a guess of β as β'. Finally, A_2 guesses b as $b' = \beta \odot \beta'$.

If $b = 0$, ψ_0 is a well-formed ciphertext. As $\mathbf{t}_1$ is uniformly random element of $[R]_{\mathbf{B}}$, and $\mathbf{r}$ is uniformly random element of R, then ψ_1 is uniformly random element of R. In this case, A_1 should have advantage ϵ to get $\beta' = \beta$, which translates into an advantage of ϵ for A_2 to get $b' = \beta \odot \beta' = 1$.

If $b = 1$, both $\mathbf{t}_0$ and $\mathbf{t}_1$ are uniformly random elements of $R \bmod \mathbf{B}$. In this case, whatever β is, the ciphertexts ψ_β are uniformly random element of R. A_1's advantage is 0, which translates into an advantage of 0 for A_2 to get $b' = \beta \odot \beta' = 1$.

Overall, A_2's advantage is $\epsilon/2$. □

The ICP asks one to decide whether $\mathbf{t}$ is uniform modulo $\mathbf{B}$ ($b = 1$) or whether it was chosen according to a known "clumpier" distribution induced by Samp_1 ($b = 0$). According to [5], there is no such algorithm that can solve the ICP with an unnegligible probability. Thus, according to Theorem 2, the advantage of algorithm A_1 breaking the semantic security of ε_s can be ignored; that is, ε_s is semantic security.

When $\mathbf{B}_I = \mathrm{Rot}(\mathbf{b})$, the modular operation $[\mathbf{v}]_{\mathbf{B}_I}$ can be calculated as $\mathbf{v} - \mathbf{b} * \lfloor \mathbf{b}^{-1} * \mathbf{v} \rceil$ or denoted as $[\mathbf{v}]_{\mathbf{b}}$. As "$*$" and $\mathbf{b}^{-1}$ can be operated with $O(n \cdot \log n)$ operations using the Fast Fourier Transform (FFT) [12], the time complexity of ε_s is $O(n \cdot \log n)$. Furthermore, as a secret key encryption, a small value of n will make ε_s security enough. Thus, ε_s is efficient and secure for being used in image encryption. But the plaintext space of ε_s is $[R]_{\mathbf{b}}$, while, in image processing, the data types include integer and float. So we need a mapping between the floats and R.

3.3. Expanding Plaintext Space into Floats

3.3.1. A Mapping between Integers and R. For $\forall \mathbf{v} \in R$, $\mathbf{v}$ can be written as the polynomial $\sum_{i=0}^{n-1} v_i \cdot x^i \bmod g(x)$. If we set x a fixed value (e.g., $x = 2$ or 10), each $\mathbf{v}$ is corresponding to a number; this numeralization of $\mathbf{v}$ is denoted as $\eta_x(\mathbf{v})$. Meanwhile, any integer v with $|v| < \eta_x(g(x))$ is equal to the polynomial $\sum_{i=0}^{n-1} v_i \cdot x^i \bmod g(x)$, where $v_i = \pm\lfloor |v|/x^i \rfloor \bmod x$; the sign of v_i is the same as v. Thus, each number v is corresponding to an element in R, denoted as $\rho_x(v)$. Thus, for an integer plaintext $|p| < \eta_x(g(x))$, in encryption (denoted as $\mathrm{Enc}_{\varepsilon 1}$), after getting $\pi \leftarrow \rho_x(p)$, it outputs $\psi \leftarrow \mathrm{Enc}_{\varepsilon s}(\mathbf{b}, \pi)$. In decryption (denoted as $\mathrm{Dec}_{\varepsilon 1}$), when getting $\pi \leftarrow \mathrm{Dec}_{\varepsilon s}(\mathbf{b}, \psi)$, it outputs $p \leftarrow \eta_x(\pi)$.

This scheme is denoted as ε_1. We discuss the conditions to make ε_1 satisfy (1), that is, $\mathrm{Dec}_{\varepsilon 1}(\mathbf{b}, h'(\psi_0, \ldots, \psi_t)) = h(p_0, \ldots, p_t)$, where $\psi_k = \mathrm{Enc}_{\varepsilon 1}(\mathbf{b}, p_k)$. The operations in h are "+" and "×" in real number field.

Lemma 3. *For an integer N, if $h(p_0, \ldots, p_t) < \min\{N, \eta_x(g(x))\}$ and $1/(2 \cdot \|\mathbf{b}^{-1}\|) \geq N$, then ε_1 satisfies (1).*

Proof. Obviously $\eta_x(\pi_k) = p_k$, where $\pi_k = \rho_x(p_k)$. If $h(p_0, \ldots, p_t) < \eta_x(g(x))$, then $\eta_x(h''(\pi_0, \ldots, \pi_t)) = h(p_0, \ldots, p_t)$, where h'' consists of "+" and "$*$" in R. Besides, we have proved that $\mathrm{Dec}_{\varepsilon s}(\mathbf{b}, f'(\psi_0, \ldots, \psi_t)) = f(\pi_0, \ldots, \pi_t)$, where f' is equivalent to h'.

As f consists of "$\mathrm{Add}_{\mathbf{B}_I}$" and "$\mathrm{Mult}_{\mathbf{B}_I}$", $[h'']_{\mathbf{B}_I} = f$. Denoting the output of h'' as $\mathbf{h}$, then $\|\mathbf{h}\| = \sqrt{\sum_i h_i^2} \leq \sum_i h_i \cdot x^i = \eta_x(\mathbf{h}) < N \leq 1/(2 \cdot \|\mathbf{b}^{-1}\|)$. According to Lemma 1 in [5], if $\|\mathbf{h}\| < 1/(2 \cdot \|\mathbf{b}^{-1}\|)$, then $\mathbf{h} \in [R]_{\mathbf{B}_I}$, that is, $h'' = [h'']_{\mathbf{B}_I}$.

Above all, we have $\mathrm{Dec}_{\varepsilon 1}(\mathbf{b}, h') = \eta_x(\mathrm{Dec}_{\varepsilon s}(\mathbf{b}, f')) = h$, that is, ε_1 satisfies (1). □

3.3.2. A Mapping between Floats and R. Now we need to expand the plaintext space into floats. To encrypt a floating number p which is accurate to x^{-m} (i.e., $p \cdot x^m - \lfloor p \cdot x^m \rceil = 0$), we can change it into integer $p \cdot x^m$ and encrypt $p \cdot x^m$ to output the ciphertext $\psi \leftarrow \mathrm{Enc}_{\varepsilon 1}(\mathbf{b}, p \cdot x^m)$. In this case, $\mathrm{Dec}_{\varepsilon 1}(\mathbf{b}, \psi) = p \cdot x^m$, and $\mathrm{Dec}_{\varepsilon 1}(\mathbf{b}, \psi_1 + \psi_2) = (p_1 + p_2) \cdot x^m$. So, the correct output of decryption should be $\mathrm{Dec}_{\varepsilon 1} \cdot x^{-m}$. But as $\mathrm{Dec}_{\varepsilon 1}(\mathbf{b}, \psi_1 * \psi_2) = (p_1 \cdot p_2) \cdot x^{2m}$, to keep the multiplication homomorphic, we redefine it ("$*^{(m)}$") as $\mathbf{u} *^{(m)} \mathbf{v} = \mathbf{u}^{(m)} * \mathbf{v}$, where $\mathbf{u}^{(m)} = \mathbf{u} \cdot x^{-m} \bmod g(x)$, that is, $u_i^{(m)} = u_{i+m \bmod n}$.

Lemma 4. *If $(u \cdot v) \cdot x^{2m} \leq \eta_x(g(x))$ and $(u \cdot v) \cdot x^m - \lfloor (u \cdot v) \cdot x^m \rceil = 0$, then $Dec_{\varepsilon 1}(\mathbf{b}, \psi_u *^{(m)} \psi_v) = (u \cdot v) \cdot x^m$, where $\psi_u = Enc_{\varepsilon 1}(\mathbf{b}, u)$, $\psi_v = Enc_{\varepsilon 1}(\mathbf{b}, v)$.*

See the proof in Appendix B.

Finally, our homomorphic encryption (IGHE) is described as follows.

KeyGen(n, N). It takes as input the security parameters n and N. It chooses a random vector $\mathbf{b}$ from $\mathbb{Z}^n$ repeatedly until $1/(2 \cdot \|\mathbf{b}^{-1}\|) > N$, and then it outputs $(\mathbf{b}, \mathbf{b}^{-1})$.

Enc$(\mathbf{b}, m, p)$. It takes as input the key $\mathbf{b}$ and a plaintext $p \in (-N, N)$. It sets $\pi \leftarrow \rho_x(\lfloor p \cdot x^m \rceil)$; then, after choosing a random vector $\mathbf{r}$ from $\mathbb{Z}^n$, it outputs $\psi \leftarrow \pi + \mathbf{b} * \mathbf{r}$.

Dec$(\mathbf{b}, m, \psi)$. It takes as input the key $\mathbf{b}$ and a ciphertext ψ. It sets $\pi \leftarrow [\psi]_{\mathbf{b}}$ and outputs $p \leftarrow \eta_x(\pi) \cdot x^{-m}$.

Eval(f, m, Ψ). It takes as input a function $f(p_0, p_1, \ldots, p_t)$ and a set of ciphertexts $\boldsymbol{\Psi} = \{\psi_0, \psi_1, \ldots, \psi_t\}$, where $\mathrm{Dec}(\mathbf{b}, m, \psi_k) = p_k$. It outputs $f' = \{\psi_0, \psi_1, \ldots, \psi_t\}$ by replacing "p_k", "+", and "×" in f with "ψ_k", "+", and "$*^{(m)}$" in R, respectively.

According to Lemmas 3 and 4, if f includes t multiplications, as long as $f \leq \eta_x(g(x)) \cdot x^{-t \cdot m}$ and $f \cdot x^{t \cdot m} - \lfloor f \cdot x^{t \cdot m} \rceil = 0$, IGHE satisfies (1). To gain a high accuracy with a small value of m, we set $x = 10$. Next, we will use IGHE to realize the encrypted image processing model in Figure 1(b).

We have proved that ε_s is semantic security. If algorithm A_3 can break the semantic security of IGHE, then, for ε_s, one can first map the ciphertext ψ_β into float and use A_2 to break the security of ε_s. Thus, IGHE is also semantic security.

So far, we have introduced our homomorphic encryption IGHE which is simplified and improved from Gentry's homomorphic encryption. The differences between IGHE and Gentry's scheme are as follows:

(i) IGHE is a symmetric encryption, while Gentry is a public key encryption. The purpose of public key is for service to use the public key to operate on the encrypted data. We omitted the public key encryption part, because we have another way to let service operate on the encrypted data without using key (see Section 4.2). The main benefit of symmetric encryption is that a smaller size of secret key can reach to the security level as high as public key. Then, in the same security level, as the size of key is smaller, our scheme is more efficient than Gentry's scheme.

(ii) Gentry's scheme is universal and fully homomorphic; it takes each bit as the plaintext. As the image processing can also be presented as a batch of operations of bit, theoretically the scheme can be used for encrypted image processing. But considering the complexity, it is hard to implement. While our scheme takes the byte as plaintext, and, after inducing the homomorphic addition and multiplication operations on floats, we can easily realize the encryption image processing.

4. Encrypted Image Processing Using IGHE

4.1. Image Encryption Using IGHE. The values of colors are less than 256, and it usually has only one multiplication in image processing. By choosing an appropriate value of m and n, after process, it is easy to keep the result less than $10^{n-m} - 10^{-m}$ ($\eta_x(g(x)) \cdot x^{-m}$ when $x = 10$) and accurate to 10^{-m}. Thus, IGHE can be applied in encrypted image processing.

An image with w width and h height consists of $w \times h$ pixels. As we know, a pixel can be seen as a combination of the three-primary colors, we denote the pixel in the ith row and jth column as $P_{i,j}(p_0, p_1, p_2)$, with p_0, p_1, p_2 representing the color of red, green, and blue, respectively. Firstly, we use KeyGen to output the key $\mathbf{b}$. Then, after reading the image file and getting the pixels array $\mathbf{P} = \{P_{i,j}\}$, we encrypt $\mathbf{P}$ by calling ImgEnc($\mathbf{b}, m, \mathbf{P}$).

ImgEnc($\mathbf{b}, m, \mathbf{P}$). For each pixel $P_{i,j}(p_0, p_1, p_2)$ in $\mathbf{P}$, it encrypts p_0, p_1, p_2 sequentially to get $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2$, where $\mathbf{c}_k \leftarrow \text{Enc}(\mathbf{b}, m, p_k)$. The ciphertext of $P_{i,j}$ is denoted as $\mathbf{C}_{i,j}(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2)$. After getting all $\mathbf{C}_{i,j}$, it outputs $\mathbf{C} \leftarrow \{\mathbf{C}_{i,j}\}$ as a $w \times h$ ciphertexts array of $\mathbf{P}$.

For a given image, there are $w \cdot h$ pixels with each having 3 integers. ImgEnc encrypts all these plaintexts separately. Thus, the time complexity of ImgEnc is $O(w \cdot h \cdot n \cdot \log n)$. $\mathbf{C}$ is the set of all the encrypted pixels, as each ciphertext is an n-dimensional column vector; $\mathbf{C}_{i,j}$ is an $n \times 3$ matrix. Thus, the size of $\mathbf{C}$ is $O(w \cdot h \cdot n)$.

In decryption algorithm, each $\mathbf{C}_{i,j}$ in $\mathbf{C}$ will be decrypted separately to get the pixel $P_{i,j}(p_0, p_1, p_2)$.

ImgDec($\mathbf{b}, m, \mathbf{C}$). For each cipher pixel $\mathbf{C}_{i,j}$ in $\mathbf{C}$, we decrypt $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2$ sequentially to get $P_{i,j}(p_0, p_1, p_2)$, where $p_k \leftarrow \lfloor \text{Dec}(\mathbf{b}, m, \mathbf{c}_k) \rceil$; if $p_k < 0$, set $p_k \leftarrow 0$; if $p_k > 255$, set $p_k \leftarrow 255$. Then, after getting all $P_{i,j}$, it outputs $\mathbf{P} \leftarrow \{P_{i,j}\}$ as the pixels array of plain image.

As the color must be an integer in $[0, 255]$, after getting $p_k \leftarrow \text{Dec}(\mathbf{b}, m, \mathbf{c}_k)$, the result will be rounded into the nearest integer in $[0, 255]$. The time complexity of ImgDec is $O(w \cdot h \cdot n \cdot \log n)$.

4.2. Processing Encrypted Image. We concentrate on the processing method which is based on addition and multiplication operations on the pixels. In these processes, the new color p' in a pixel is calculated by a function $f(p_0, p_1, \ldots, a_0, a_1, \ldots)$, where p_k represents the color of a pixel and a_k represents the data from SP. As IGHE is a secret key encryption, SP does not have the key to encrypt a_k. However, SP can translated a_k into $\rho_x(\lfloor a_k \cdot x^m \rceil)$, which is a special ciphertext with $\mathbf{r} = \mathbf{0}$ in Enc($\boldsymbol{b}, m, a_k$). We denote this translation as Enc(m, a_k), which can be run by SP without the key $\mathbf{b}$.

As shown in Figure 1(b), firstly, user uploads the encrypted image by calling ImgEnc and asks for a processing method request corresponding to the function f. Then, SP returns to user the encrypted result by calling Eval(f, m, Ψ). Finally, by calling ImgDec, user will get the correct plain result same as Figure 1(a). In Eval, the function $f'(\mathbf{c}_0, \mathbf{c}_1, \ldots, \psi_{(a)0}, \psi_{(a)1}, \ldots)$ was generated to process the encrypted image, where $\mathbf{c}_k \leftarrow \text{Enc}(\mathbf{b}, m, p_k)$, and $\psi_{(a)k} \leftarrow \text{Enc}(m, a_k)$. In these methods, as SP does not need any key, there is no need to share the key. User just needs to keep the secret key safely.

In the image processing, we may like to change the color of the image, add a watermark (or photo frame) on the image, or change the size of the image. To illustrate our scheme, we use these three common kinds of image processing as instances.

4.2.1. Color Transformation. The original color transformation is based on the color matrix [13] **CM** which is a 5×5 matrix. The ith row and jth column are denoted as cm_{ij}, where cm_{ij} is often a floating number. Given **CM** and the original pixel $P_{i,j}(p_0, p_1, p_2)$, the resulting pixel $P'_{i,j}({p'}_0, {p'}_1, {p'}_2)$ is computed as

$$\begin{aligned} {p'}_k &= f_1\left(P_{i,j}, cm_{k0}, \ldots, cm_{k4}\right) \\ &= p_0 \cdot cm_{k0} + p_1 \cdot cm_{k1} + p_2 \cdot cm_{k2} + \alpha_{i,j} \cdot cm_{k3} \\ &\quad + cm_{k4}, \end{aligned} \tag{5}$$

where k is 0, 1, 2, and $\alpha_{i,j}$ is the transparency of the pixel $P_{i,j}$. For the image without transparency, the default value of $\alpha_{i,j}$ is 255.

In the instance, DO wants to change the color of his image, such as grayness and binarization. But he does not know the value of **CM**, while there are many kinds of color transformation operations in SP, with each kind corresponding to a value of **CM**.

DO uploads the ciphertext $\mathbf{C}$, then, in encrypted image color transformation, SP uses the encrypted form of **CM**

$(\psi_{ij} \leftarrow \mathrm{Enc}(m, cm_{ij}))$ to compute each encrypted resulting pixel $\mathbf{C}'_{i,j}(\mathbf{c}'_0, \mathbf{c}'_1, \mathbf{c}'_2)$.

$$\begin{aligned}\mathbf{c}'_k &= f'_1\left(\mathbf{C}_{i,j}, \psi_{k0}, \ldots, \psi_{k4}\right)\\ &= \mathbf{c}_0 *^{(m)} \psi_{k0} + \mathbf{c}_1 *^{(m)} \psi_{k1} + \mathbf{c}_2 *^{(m)} \psi_{k2}\\ &\quad + \mathrm{Enc}\left(m, \left(\alpha_{i,j} \cdot cm_{k3} + cm_{k4}\right)\right).\end{aligned} \tag{6}$$

As SP knows $\alpha_{i,j}, cm_{k3}, cm_{k4}$, he can compute and then encrypt $\alpha_{i,j} \cdot cm_{k3} + cm_{k4}$ as one plaintext.

4.2.2. Image Addition. Give two images $\mathbf{P}_1$, $\mathbf{P}_2$, add $\mathbf{P}_1$ on $\mathbf{P}_2$ with a transparency tp; then the resulting pixel $P'_{i,j}$ is

$$P'_{i,j} = f_2\left(\mathrm{tp}, P_{1(i,j)}, P_{2(i,j)}\right)\mathrm{tp} \cdot P_{1(i,j)} + (1 - \mathrm{tp}) \cdot P_{2(i,j)}. \tag{7}$$

In one instance, IO gives SP two encrypted images $\mathbf{C}_1$ and $\mathbf{C}_2$ and asks SP to add them with the transparency tp. The encrypted resulting pixel is

$$\begin{aligned}\mathbf{C}'_{i,j} &= f'_2\left(\mathrm{tp}, \mathbf{C}_{1(i,j)}, \mathbf{C}_{2(i,j)}\right)\\ &= \mathrm{Enc}\left(m, \mathrm{tp}\right) *^{(m)} \mathbf{C}_{1(i,j)}\\ &\quad + \mathrm{Enc}\left(m, 1 - \mathrm{tp}\right) *^{(m)} \mathbf{C}_{2(i,j)}.\end{aligned} \tag{8}$$

In another instance, IO gives SP one encrypted image $\mathbf{C}$ and chooses a plain image $\mathbf{P}$ (e.g., a photo frame) from SP to add on the encrypted image with the transparency tp. In SP's image, the pixels may have different transparencies $\alpha_{i,j}$; when added on $\mathbf{C}_{i,j}$, the true transparency of $P_{i,j}$ is $\mathrm{tp} \cdot \alpha_{i,j}$. And the encrypted resulting pixel is

$$\begin{aligned}\mathbf{C}'_{i,j} &= f'_2\left(\mathrm{tp}, P_{i,j}, \alpha_{i,j}, \mathbf{C}_{i,j}\right)\\ &= \mathrm{Enc}\left(m, \mathrm{tp} \cdot \alpha_{i,j} \cdot P_{i,j}\right)\\ &\quad + \mathrm{Enc}\left(m, 1 - \mathrm{tp} \cdot \alpha_{i,j}\right) *^{(m)} \mathbf{C}_{i,j}.\end{aligned} \tag{9}$$

As SP knows $\alpha_{i,j}$, tp, p_k, he can compute and then encrypt $\mathrm{tp} \cdot \alpha_{i,j} \cdot p_k$ and $1 - \mathrm{tp} \cdot \alpha_{i,j}$ as one plaintext.

4.2.3. Image Scaling. Image scaling is also common image processing. Here we discuss the quadratic linear interpolation method. The size of original image $\mathbf{P}$ is $w \times h$. Suppose that, after scaling, the size of $\mathbf{P}'$ becomes $\lambda \cdot w \times \gamma \cdot h$. Then, $P'_{i,j}(p'_0, p'_1, p'_2)$ $(i = 0, 1, \ldots, \lambda \cdot w - 1,\ j = 0, 1, \ldots, \gamma \cdot h - 1)$ is computed according to four adjacent pixels in $\mathbf{P}$. They are $P_{s,t}, P_{s,t+1}, P_{s+1,t}, P_{s+1,t+1}$, where $s = \lfloor i/\lambda \rfloor$, $t = \lfloor j/\gamma \rfloor$, and

$$\begin{aligned}P'_{i,j} &= f_2\left(P_{s,t}, P_{s,t+1}, P_{s+1,t}, P_{s+1,t+1}, u, v\right)\\ &= P_{s,t} \cdot (1 - u) \cdot (1 - v) + P_{s,t+1} \cdot u \cdot (1 - v)\\ &\quad + P_{s+1,t} \cdot (1 - u) \cdot v + P_{s+1,t+1} \cdot u \cdot v,\end{aligned} \tag{10}$$

where $u = i/\lambda - s$, $v = j/\gamma - t$.

In the instance, IO asks SP to scale the encrypted image $\mathbf{C}$ λ times in width and γ times in height. So the encrypted resulting pixel is

$$\begin{aligned}\mathbf{C}'_{i,j} &= f'_3\left(\mathbf{C}_{s,t}, \mathbf{C}_{s,t+1}, \mathbf{C}_{s+1,t}, \mathbf{C}_{s+1,t+1}, u, v\right)\\ &= \mathbf{C}_{s,t} *^{(m)}_{m} \mathbf{t}_0 + \mathbf{C}_{s,t+1} *^{(m)} \mathbf{t}_1 + \mathbf{C}_{s+1,t} *^{(m)} \mathbf{t}_2\\ &\quad + \mathbf{C}_{s+1,t+1} *^{(m)} \mathbf{t}_3,\end{aligned} \tag{11}$$

where $\mathbf{t}_0 = \mathrm{Enc}(m, (1 - u) \cdot (1 - v))$, $\mathbf{t}_1 = \mathrm{Enc}(m, u \cdot (1 - v))$, $\mathbf{t}_2 = \mathrm{Enc}(m, (1 - h) \cdot v)$, and $\mathbf{t}_3 = \mathrm{Enc}(m, u \cdot v)$.

5. Experiments

We run some experiments to test the performance of our scheme. All the experiments are carried out on a computer equipped with two processors, each with 3.1 GHz clock speed, and the computer has 4 GB RAM.

5.1. Security. Now we analyze the statistical security of the encrypted image.

In our scheme, the ciphertext $\mathbf{c}_k \in R$ is an n-dimensional integer vectors. We use 4 bytes to represent the integer; then each element of $\mathbf{c}_k$ can represent a new pixel in the encrypted image. The top byte represents the transparency α, and the rest represent the color of red, green, and blue, respectively. So a plain pixel will be changed into $n \times 3$ new pixels in encrypted image. We store the encrypted image in .PNG format and calculate the histograms of original and encrypted image, respectively. Figure 2(a) is the original image, and its histogram is shown in (b). Figure 2(c) is a part of the encrypted image, and the histogram is shown in (d).

Figure 2 indicates that the histogram of the encrypted image is fairly uniform and is significantly different from that of the original image. Thus, our scheme can resist the statistical analysis attack.

5.2. Correctness. We do the same type of processing both on the original image $\mathbf{P}$ and on the encrypted image $\mathbf{C}$ to get the processed image $\mathbf{P}'$ and $\mathbf{C}'$. The error between $P'_{i,j}$ and $\mathbf{C}'_{i,j}$ is

$$\mathrm{err}_{i,j} = \frac{\sum_{k=0}^{2}\left|\mathrm{Dec}\left(b, m, \mathbf{c}'_k\right) - p'_k\right|}{3}. \tag{12}$$

If $\mathrm{err}_{i,j} \neq 0$, the encryption pixel of $\mathbf{C}'_{i,j}$ is error. If all the $\mathrm{err}_{i,j}$ are zero, our scheme is correct.

As the floating numbers are accurate to 10^{-m}, the value of m will affect the correctness of our scheme. To test this, we choose different value of m. For each m, we encrypt an image $\mathbf{P}$ to get $\mathbf{C}$ and do the same kind of image processing on $\mathbf{P}$ and $\mathbf{C}$ to get $\mathbf{P}'$ and $\mathbf{C}'$. Then, we compute the $\mathrm{err}_{i,j}$ for every pixel.

We fix $n = 8$. Table 1 shows the correctness of our scheme with different value of m. The "Error Rate" means the rate of error $\mathbf{C}'_{i,j}$ in $\mathbf{C}$. "None" means decrypted $\mathbf{C}$ with no process. In "color transformation (CT)," each time we set the value of $\mathbf{CM}$ randomly. The elements of $\mathbf{CM}$ are accurate to 0.001. In "image addition (Add.)," we add the same image

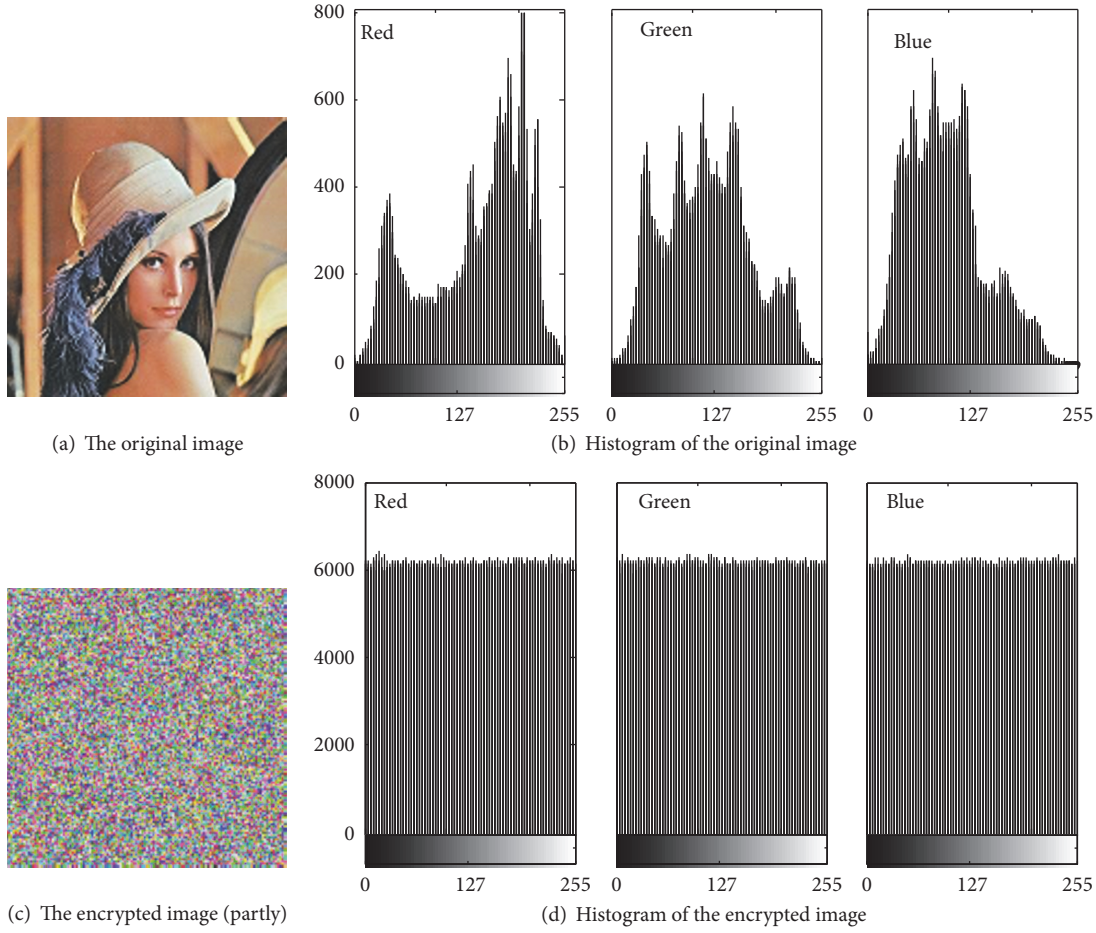


Figure 2: The histogram of the original and encrypted image.

Table 1: The correctness with different value of m.

m	Error rate (%)				Max. error			Average error		
	None	CT	Add.	Scal.	CT	Add.	Scal.	CT	Add.	Scal.
1	0	28.1	99.3	54.4	17	13	26	2.34	5.46	5.45
2	0	15.5	0	37.6	3	0	3	0.26	0	0.54
3	0	0	0	8.4	0	0	1	0	0	0.08
4	0	0	0	0	0	0	0	0	0	0

on **P** and **C** several times. Each time we set a different value of transparency ranging from 0 to 100%. In "image scaling (Scal.)," each time we choose λ from $(0, 4]$, and set $\gamma = \lambda$.

In Table 1, the error becomes less and smaller as m increases. When m increases to 4, there is no error; then our scheme is correct. The error rate of "None" is zero, so no matter what m is, if we decrypt an encrypted image with no process, we can get the original image with no error. That is because the value of color is integers, so the accuracy of 10^0 is enough.

In "Add.", the transparency is a number with two decimals, so when $m = 2$, there is no error any more. In "CT", as the random value of **CM** has three decimals, the error is zero when $m = 3$. In "Scal.", the data may be an infinite decimal. But since the value of color is smaller than 256, when $m \geq 4$, $256 \cdot 10^{-m} < 0.1$. So there will be no error after rounding the result to the nearest integer.

Figure 3 shows some result of image $\mathbf{P}'$ (left image) and image $\mathrm{Dec}(\mathbf{C}')$ (right image) after the image processing when $n = 8$ and $m = 4$. In (a), the left image is the original image. We encrypt the left image and decrypt the cipher image to get the right image. In (b), we remove the color (grayness) of the original image; this is realized by setting **CM** a specific value. In (c), we set the elements of **CM** random values. And in (d), we choose a photo frame to add on the original image; the transparency is 60%.

The two images in each figure look the same. According to Table 1, they do have the same pixel in the same position.

(a) No processing (b) Color Trans. (grayness)

(c) Color Trans. (randomly) (d) Image addition

Figure 3: The result of $\mathbf{P}'$ and Dec($\mathbf{C}'$).

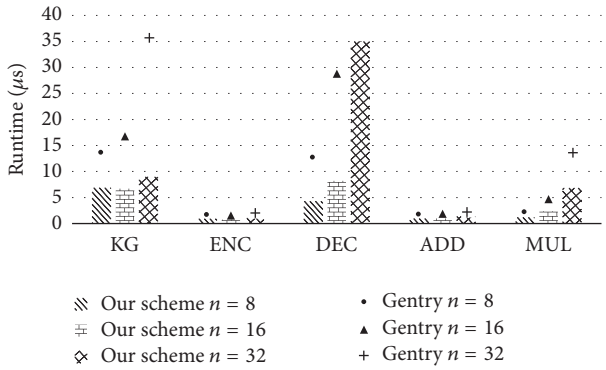


Figure 4: The runtime of IGHE and Gentry's scheme.

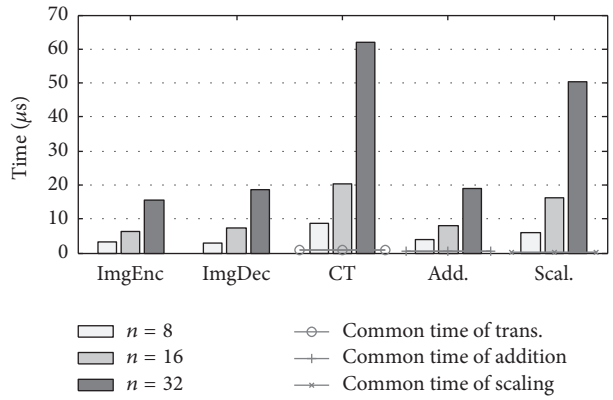


Figure 5: The running time of each algorithm.

Above all, the image can be processed in encrypted form. After decryption, user will get the same image as processing directly on plain image. Thus, our scheme is correct.

5.3. Efficiency. Figure 4 shows the runtime of the key generation, encryption, decryption, addition, and multiplication processes of our scheme IGHE and Gentry's scheme.

We can see, when, in the same value of n, our scheme is more efficient than Gentry's scheme. When n is 32, the runtime of "DEC" of Gentry is even more than 100 ms, which is not marked in Figure 4. What is more, note that as Gentry's scheme is a public encrypt system, to keep security, n is at least 512 [9]. Then, the running time of each process will be more than 5 s. Thus, IGHE is more efficient than Gentry's scheme.

Encryption and decryption will cost extra running time. Besides, processing encrypted image is slower than processing plain image. We test the efficiency of ImgEnc, ImgDec, and the three encrypted image processing algorithms (CT, Add., and Scal.). We choose different values of n and record the running time of each algorithm. Figure 5 shows the average running time of each algorithm on one pixel. The common time means the running time of plain image processing.

From Figure 5, we can see that all the running time increases as n increases. When $n = 8$, all algorithms can process each pixel in 10 μs. This is 10 times slower than processing the plain pixel. But for a 1-megapixel image, the process can be finished in 10 seconds. To protect the privacy, this compromise of efficiency is worthy. When n is 16 or 32, it may take a long time to process the large size image, but our scheme is still suitable for the small size image.

5.4. Size of Encrypted Image. After encryption, the size of encrypted image is enlarged. By setting different value of n, we record the size of secret key, the size of encrypted image,

Table 2: The size of key and image in different value of n.

n	Key (byte)	Width (pixel)	Height (pixel)	.PNG (Kb)
Plain	—	256	256	192
8	128	768	2048	6K
16	256	768	4096	12K
32	512	768	8192	24K

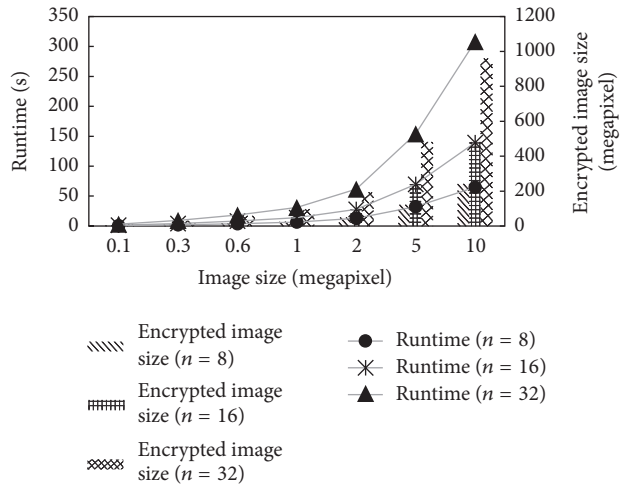


Figure 6: Runtime of CT and encrypted picture size with different original picture size.

and the space of encrypted image file in .PNG format. See Table 2. "Plain" means the original image.

As described in Section 5.1, the key and ciphertext of IGHE are n-dimensional vectors, with each element having a 4-byte integer. The size of secret key is $O(n)$. A plain color takes one byte. After encryption, the encrypted color takes $4n$ bytes, with n bytes representing the transparency. So, in the encrypted image, the width grows by 3 times, and the height grows n times. And the size of .PNG file is enlarged $4n$ times.

Figure 6 shows that, with the size of image increasing, both the size of encrypted image and the runtime of CT process are increasing.

For $n = 16$, the time for processing a 5-megapixel image is about 1 minute, but the size of the encrypted image is up to 200 megapixels. So considering both the efficiency and the communication overhead, our scheme is suitable for small size image encryption and processing, but not proper for larger size of image yet. According to Figure 6, for $n = 8$, the proper size range may be less than 2 megapixels, and, for $n = 16$, it should be less than 1 megapixel.

6. Conclusion

In this paper, we improve Gentry's homomorphic encryption to propose an efficient, simplified secret key homomorphic encryption (IGHE). We use it in the image encryption, so that the image can be processed in encrypted form to protect the privacy. Any image processing that consists of addition and multiplication operations on single pixel can be translated into the encrypted-formed process. We give the color transformation, image addition, and image scaling as the examples.

By using our scheme, the image can be processed in the server in encrypted form; after decryption in client, user can get the correct processed image that is the same as processing the plain image. So our scheme can protect privacy in online image processing. Our scheme is secure, but it enlarges the running time of process and the space for storing encrypted image. So our scheme is not proper for processing large size of image yet.

As the value of color is less than 256, when translating into vector, most elements of the vector are zero. To use these elements, we will research on translating batch colors in one vector to decrease the running time and the size of encrypted image.

Appendix

A. Proving $\mathbf{u} * \mathbf{v} = \mathrm{Rot}(\mathbf{u}) \cdot \mathbf{v}$

Proof. Suppose $\mathbf{u} * \mathbf{v} = \mathbf{w}$. The multiplication can be viewed as

$$\left(\sum_{i=0}^{n-1} v_i \cdot x^i \cdot \sum_{i=0}^{n-1} u_i \cdot x^i\right) \bmod g(x) = \left(\sum_{k=0}^{2n-2} c_k \cdot x^k\right) \bmod g(x), \quad \text{where } c_k = \sum_{i+j=k} u_i \cdot v_j. \tag{A.1}$$

Furthermore, as

$$\sum_{k=0}^{2n-2} c_k \cdot x^k = \left(c_n + c_{n+1} \cdot x + \cdots + c_{2n-2} \cdot x^{n-2}\right) \cdot \left(x^n - 1\right) + \left(c_0 + c_n\right) + \left(c_1 + c_{n+1}\right) \cdot x + \cdots + \left(c_{n-2} + c_{2n-2}\right) \cdot x^{n-2} + c_{n-1} \cdot x^{n-1}, \tag{A.2}$$

thus,

$$\left(\sum_{k=0}^{2n-2} c_k \cdot x^k\right) \bmod g(x) = \left(c_0 + c_n\right) + \left(c_1 + c_{n+1}\right) \cdot x + \cdots + \left(c_{n-2} + c_{2n-2}\right) \cdot x^{n-2} + c_{n-1} \cdot x^{n-1}. \tag{A.3}$$

The coefficient of x^s $(s = 0, 1, \ldots, n-1)$ is

$$w_s = \sum_{k \equiv s \bmod n} c_k = \sum_{k \equiv s \bmod n} \left(\sum_{i+j=k} u_i v_j\right) = \sum_{i+j \equiv s \bmod n} u_i v_j. \tag{A.4}$$

Besides, suppose

$$\text{Rot}(\mathbf{u}) \cdot \mathbf{v} = \begin{bmatrix} u_0 & u_{n-1} & \cdots & u_1 \\ u_1 & u_0 & \cdots & u_2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{n-1} & u_{n-2} & \cdots & u_0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{n-1} \end{bmatrix}, \tag{A.5}$$

then, $w_s = \sum_{i+j \equiv s \bmod n} u_i \cdot v_j$.

Thus, $\mathbf{u} * \mathbf{v} = \text{Rot}(\mathbf{u}) \cdot \mathbf{v}$. □

B. Proving Lemma 4

Proof. We fix x, and suppose the floating number u is accurate to x^{-m}, that is, $\sum_{i=0}^{n-1} u_i \cdot x^i = u \cdot x^m$.

Then, $(\sum_{i=0}^{n-1} u_i \cdot x^i \cdot \sum_{j=0}^{n-1} v_j \cdot x^j) = u \cdot v \cdot x^{2m}$.

Suppose $\sum_{i=0}^{n-1} u_i \cdot x^i \cdot \sum_{j=0}^{n-1} v_j \cdot x^j = \sum_{k=0}^{2n-2} c_k \cdot x^k$, where $c_k = \sum_{i+j=k} u_i \cdot v_j$.

We have $(u \cdot v) \cdot x^m \le \eta_x(g(x)) \Rightarrow c_k = 0$ if $k \ge n + m$.

And $(u \cdot v) \cdot x^m - \lfloor (u \cdot v) \cdot x^m \rceil = 0 \Rightarrow c_k = 0$ if $k < m$.

Then, $\sum_{k=m}^{n+m-1} c_k \cdot x^k = u \cdot v \cdot x^{2m}$ or $\sum_{k=m}^{n+m-1} c_k \cdot x^{k-m} = u \cdot v \cdot x^m$.

Besides, $\mathbf{u} = [u_0 \ \cdots \ u_{n-1}]$, and $\mathbf{u}^{(m)} = [u_m \ u_{m+1} \ \cdots \ u_{n-1} \ u_0 \ u_1 \ \cdots \ u_{m-1}]$.

Suppose $\mathbf{u} *^{(m)} \mathbf{v} = \mathbf{w}$; then $w_s = \sum_{i+j-m \equiv s \bmod n} u_i \cdot v_j = \sum_{k-m \equiv s \bmod n} c_k$.

As $c_k = 0$ if $k < m$ or $k \ge n + m$, then $\sum_{s=0}^{n-1} w_s \cdot x^s = \sum_{s=0}^{n-1} \sum_{k-m \equiv s \bmod n} c_k \cdot x^s = \sum_{k=m}^{n+m-1} c_k \cdot x^{k-m}$.

Thus, $\eta_x(\mathbf{u} *^{(m)} \mathbf{v}) = \sum_{k=m}^{n+m-1} c_k \cdot x^{k-m} = (u \cdot v) \cdot x^m$.

Furthermore, as $[(\mathbf{b} * \mathbf{r}_1 + \mathbf{u}) *^{(m)} (\mathbf{b} * \mathbf{r}_1 + \mathbf{v})]_{\mathbf{b}} = \mathbf{u} *^{(m)} \mathbf{v}$,

$$\text{Dec}_{\varepsilon 1}\left(\mathbf{b}, \psi_u *^{(m)} \psi_v\right) = (u \cdot v) \cdot x^m. \tag{B.1}$$

□

Notations

R: The commutative ring $R = \mathbb{Z}[x]/g(x)$, where $g(x) \in \mathbb{Z}[x]$ is monic and of degree n. In this paper, we set $g(x) = x^n - 1$, same as [14]

$\mathbf{v}$: The bold lowercase letter represents the element of R, that is, $\mathbf{v} \in R$. It can be written both as a polynomial, for example, $\sum_{i=0}^{n-1} v_i \cdot x^i \bmod g(x)$, and as a vector, for example, $\mathbf{v} = (v_0 \ \ v_1 \ \ \cdots \ \ v_{n-1})^T$. Besides, we denote $(0 \ \ 0 \ \ \cdots \ \ 0)^T$ as $\mathbf{0}$ and $(1 \ \ 0 \ \ \cdots \ \ 0)^T$ as $\mathbf{1}$

I: The ideal lattice in R. $\mathbf{B}_I \in \mathbb{Z}^{n \times n}$ is the basis of I, that is, $I = \{\mathbf{B}_I \cdot \mathbf{r} \mid \mathbf{r} \in R\}$ [15]

Rot($\mathbf{v}$): The circulant matrix generated by $\mathbf{v}$. The ith row and jth column in Rot($\mathbf{v}$) are $v_{(i-j) \bmod n}$

$*$: The multiplication in ring R, $\mathbf{u} * \mathbf{v} = (\mathbf{u} \cdot \mathbf{v}) \bmod g(x) = \text{Rot}(\mathbf{u}) \cdot \mathbf{v}$. See the proof in Appendix A. Besides, the addition in R is the same as the addition in $\mathbb{Z}^n$, still denoted as "+"

$\mathbf{r} \xleftarrow{R} R$: The value of $\mathbf{r}$ is randomly chosen from R

$[\mathbf{v}]_{\mathbf{B}_I}$: $\mathbf{v} \bmod \mathbf{B}_I$. It can be efficiently computed as $\mathbf{v} - \mathbf{B}_I \cdot \lfloor \mathbf{B}_I^{-1} \cdot \mathbf{v} \rceil$, where "$\lfloor\ \rceil$" rounds each element of the vector to the nearest integer

$\text{Add}_{\mathbf{B}_I}, \text{Mult}_{\mathbf{B}_I}$: The modular addition operations. $\text{Add}_{\mathbf{B}_I}(\mathbf{u}, \mathbf{v}) = [\mathbf{u} + \mathbf{v}]_{\mathbf{B}_I}$, $\text{Mult}_{\mathbf{B}_I}(\mathbf{u}, \mathbf{v}) = [\mathbf{u} * \mathbf{v}]_{\mathbf{B}_I}$.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by NSFC under Grant no. 61472316, Research Fund for the Doctoral Program of Higher Education of China under Grant no. 20120201110013, Scientific and Technological Project in Shaanxi Province under Grants no. 2016ZDJC-05 and no. 2014JQ8322, Basic Science Research Fund in Xi'an Jiaotong University (no. XJJ2014049 and no. XKJC2014008), and Shaanxi Science and Technology Innovation Project (2013SZS16-Z01/P01/K01).

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the Association for Computing Machinery*, vol. 21, no. 2, pp. 120–126, 1978.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *Institute of Electrical and Electronics Engineers. Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[3] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[4] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '99)*, vol. 1592 of *Lecture Notes in Computer Science*, pp. 223–238, Springer, Prague, Czech Republic.

[5] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC '09*, pp. 169–178, Bethesda, Md, USA, June 2009.

[6] J. Fridrich, "Image encryption based on chaotic maps," in *Proceedings of the IEEE International Conference on Systems*, vol. 2, pp. 1105–1110, Orlando, Fla, USA, 1997.

[7] N. K. Pareek, V. Patidar, and K. K. Sud, "Image encryption using chaotic logistic map," *Image and Vision Computing*, vol. 24, no. 9, pp. 926–934, 2006.

[8] J. B. Shen, X. G. Jin, and C. A. Zhou, "A color image encryption algorithm based on magic cube transformation and modular arithmetic operation," in *Proceedings of the Advances in Multimedia Information Processing*, vol. 3768, pp. 270–280, Jeju Island, Korea.

[9] C. Gentry and S. Halevi, "Implementing Gentry's fully-homomorphic encryption scheme," in *Proceedings of the 30th Annual International Conference on the Theory and Applications of Cryptographic (EUROCRYPT '11)*, vol. 6632 of *Lecture Notes in Computer Science*, pp. 129–148, Springer, Tallinn, Estonia.

[10] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Proceedings of the 31st Annual International Cryptology Conference (CRYPTO '11)*, vol. 6841 of *Lecture Notes in Computer Science*, pp. 487–504, Springer, Santa Barbara, Calif, USA.

[11] M. Hojsík and V. Půlpánová, "A fully homomorphic cryptosystem with approximate perfect secrecy," in *Proceedings of the 13th international conference on Topics in Cryptology (CT-RSA '13)*, pp. 375–388, San Francisco, Calif, USA, 2013.

[12] P. J. Davis, *Circulant Matrices*, John Wiley & Sons, New York, NY, USA, 1979.

[13] Using a Color Matrix to Transform a Single Color, 2014, http://msdn.microsoft.com/en-us/library/windows/desktop/ms533875(v=vs.85).aspx.

[14] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: a ring-based public key cryptosystem," in *Proceedings of the International Algorithmic Number Theory Symposium (ANTS '98)*, vol. 1423 of *Lecture Notes in Computer Science*, pp. 267–288, Springer, Portland, Ore, USA.

[15] D. Micciancio, "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions," *Computational Complexity*, vol. 16, no. 4, pp. 365–411, 2007.