Fully Homomorphic Encryption of Floating-Point Matrices for Privacy-Preserving Image Processing

Prabhavathi Krishnegowda, Anandaraju M Boregowda

***Abstract*— Existing Fully Homomorphic Encryption schemes mainly operate on integer data in the finite field. However, in many real-life applications, the data is in floating-point number format, and the homomorphic encryption of floating-point information is challenging. This paper presents a new homomorphic encryption method for matrices whose elements are floating-point numbers. The proposed homomorphic encryption of Floating-Point Data Matrices is applied for secure image processing where the loss of accuracy due to the finite length of the floating-point format representation is insignificant. Here, the encryption/decryption keys are also in floating-point format. New techniques of homomorphic weighted addition, multiplication, and DCT calculation are presented. Homomorphic proxy re-encryption is the novelty of the proposed method.**

***Keywords*— Double side encryption, Fully Homomorphic encryption, Homomorphic DCT, Homomorphic proxy re-encryption, Matrix keys, Weighted addition.**

I. Introduction

IMAGES are a visual depiction of relevant information for easy rcognition. Billions of images are produced and distributed worldwide in arts & entertainment, E-commerce, science & education, geographical & tourism maps, health care & diagnostic imaging services, etc. Invariably, these images are saved in public or dedicated Cloud Servers (CS), providing efficient storage and distribution at a low cost. When medical images carrying sensitive information are stored in Cloud Servers, security and confidentiality are the image owner's primary responsibility [1]. The conventional solution is to use image encryption or steganography [2]. Image encryption is tailor-made for homomorphic operations. Therefore, we use image encryption to achieve the desired privacy and security when the images are stored in the Cloud Server.

Homomorphic Encryption (HE) allows the algebraic and arithmetic computations to be carried out over the encrypted data in the CS without decryption and guarantees full accuracy on the decryption of the results from the cipher domain [3]. Thus, with HE, the processing unit at the CS can do computations without any access to the original data. Therefore, the data owner can delegate privacy-preserving Image Processing operations to the CS, which can handle the heavy computational load.

The HE of two scalar plaintext numbers x_1 and x_2 is governed by the defining rule as follows. Let $enc(x_1)$ and $enc(x_2)$ be the encrypted ciphertext numbers of x_1 and x_2, respectively. Then,

$$enc(x_1) \otimes enc(x_2) = enc(x_1 \otimes x_2)$$

Here, $\otimes$ stands for the algebraic operations: + (addition), – (subtraction), * (multiplication), and so on.

We can extend the homomorphic encryption and operations to matrices where the operands are matrices, and the encryption is formulated such that,

$$E(\boldsymbol{X}_1) \phi E(\boldsymbol{X}_2) = E(\boldsymbol{X}_1 \phi \boldsymbol{X}_2) \quad (1)$$

Here, ϕ stands for the matrix operation: + (addition), – (subtraction), * (multiplication), and **.*** (element-wise multiplication), *etc*. The encryption function $E(\ldots)$ that satisfies (1) is the HE for matrices. Since matrices represent digital images, HE of matrices leads to Homomorphic Image Processing (HIP).

The principal objective of the proposed scheme is to:

- Develop a novel Matrix Encryption scheme for the HIP using floating-point matrix keys.

The proposed method is denoted by **HE-FPMK** (Homomorphic Encryption using floating-point matrix keys)

The remaining part of the paper is organized as follows. Section II gives a brief review of the related work. Section III explains the basic Homomorphic Encryption and Decryption. In section IV, applications of HE-FPMK, in image processing are presented. In section V, the performance metrics of HE_FPMK and comparison with other methods are discussed. Section VI concludes the paper.

II. Related Work

We have several well-established techniques for encrypting digital images [4] to achieve the required level of security and adequate privacy. However, we consider those schemes implementing efficient homomorphic encryption suitable for *image processing* in the cipher domain. In [5-6], the authors have used SEAL (Simple Encrypted Arithmetic Library) developed by Microsoft Research for the

Manuscript received April 20, 2023; revised September 8, 2023 .

Prabhavathi Krishnegowda is a research scholar in the Department of ECE, BGS Institute of Technology, Adichunchanagiri University, B. G. Nagara, Karnataka, India. (Corresponding author. Phone +91 90194 56669. e-mail: pdevimtech@gmail.com).

Anandaraju M Boregowda is a Professor in the Department of ECE, BGS Institute of Technology, Adichunchanagiri University, B. G. Nagara, India. (e-mail: mb.anandaraju@gmail.com).

Homomorphic Encryption (HE) of images. Logistic regression is implemented in [5], and Homomorphic image resizing is carried out using bilinear and bicubic interpolations. Image compression/decompression operations are carried out using DCT and inverse DCT transformations in the cipher domain [6]. However, SEAL uses well-known BV and CKKS schemes where noise components are added, which limits multilevel HE operations. In [7], HE is deployed for the scale-invariant feature transform (SIFT) of images. The SIFT technique can search and match the images.

Here, image privacy is achieved by splitting it into two shares and storing them in two separate Cloud Servers for SIFT implementation. Afterward, the partial results are combined to reconstruct the final result. Therefore, the proposed method has higher communication overhead and increased computational complexity. In [8], the authors have created CryptoNets, which is trained using encrypted data. Queries can be sent, and the response from the CryptoNet can be received in encrypted form. Therefore, the CS that operates the CryptoNet cannot gain any information about the original data. However, the proposed method uses HE based on noise vectors and dual moduli, which results in reduced throughput due to the increased computational overhead when images are used as data inputs.

In [9], the authors have designed a Convolutional Neural Network (CNN) classifier that works in the cipher domain using Fully Homomorphic Encryption. The classification activity is outsourced to a high-power CS. The real numbers are represented in fixed point formats to implement homomorphic operations. Hence, the proposed method has higher computational complexity, which is the main disadvantage when working with large images. In [10], the Scale Invariant Feature Transform (SIFT) of images has been implemented using homomorphic encryption, which is carried out based on NTRU (N^{th}-degree Truncated polynomial Ring Units). NTRU encryption supports easy scale-up so that the large-sized images can be processed efficiently in the cipher domain. The proposed method uses SIFT coefficients for image matching. However, the inaccuracy of homomorphic division is reflected in the calculated SIFT coefficients, and the overall error is substantially higher compared to SIFT operation in the plaintext domain. In [11], morphological transformations like dilation, erosion, opening, etc., are carried out on the encrypted images using homomorphic multiplication implemented in the binary field. Here, the authors have used monoid algebra that is susceptible to CPA (chosen plaintext attack) and CCA (chosen ciphertext attack).

In [12], the authors have securely outsourced the BTC (Block Truncation Coding) of images to the CS. Homomorphic BTC is accomplished after encrypting the image using block permutation and diffusion, which do not modify the BTC parameters. Therefore, image compression via BTC is carried out successfully in the cipher domain. However, image compression using BTC suffers substantial information loss and is not acceptable for fine-grained images where sensitive features are embedded, like medical images. In [13], a symmetric encryption/decryption scheme using two-stage modular operations has been presented. Here, the image matrices are encrypted/decrypted element by element, which increases the time complexity, and the consequent computational cost quadratically with respect to the size of the image.

In [14], Elliptic Curve Cryptography (ECC) is used to provide homomorphic addition of images. The basic principle of the Elgamal method is adopted to exchange cryptographic data. The major drawback of this scheme is the lack of homomorphic multiplication, and hence its applicability is restricted. Another disadvantage of this method is increased computational complexity due to ECC, which is not favourable for handling large-sized image data.

In [15], the authors have used a fully homomorphic encryption scheme for remote authentication using digital signatures based on biometric images like fingerprints, iris, faces, and so on. In the proposed scheme, the encrypted biometric images are used for the verification of the authenticity of the source by comparing and matching processes in the cipher domain. Hence, this scheme is limited only to authentication and cannot be applied for any additional processing of images. In [16], the segmentation of encrypted images and edge detection schemes have been presented using Gaussian filtering and Sobel operators in the cipher domain. The authors have used Paillier homomorphic encryption for implementing Gaussian and Sobel filtering. However, this method uses block-wise computations to cover the full image, resulting in higher computational costs. In [17], the authors have used fully homomorphic encryption to implement image addition and brightness enhancement in the cipher domain. However, the brightness enhancement is achieved using the scalar multiplication of the image matrix, which increases the brightness uniformly throughout but cannot provide selective brightness control. Additionally, the HE scheme adopted in this work uses the addition of small random noise to provide security. This process prevents multi-depth multiplication. In [18], Craig Gentry uses Ideal Lattices to implement fully homomorphic encryption. Here, the basic plaintext space is binary, and to encrypt integers, they should be converted to their equivalent binary and then proceed for further processing. Similarly, after decryption, the process should be reversed. This process involves a higher computational cost. Also, in this Gentry method, the key sizes are relatively large.

In [19], the authors have improved the Gentry method to reduce the key sizes and also to include the real numbers in the message (plaintext) space. The proposed method is called IGHE (Improved Gentry Homomorphic Encryption), where the key sizes are reduced compared to the Gentry method. Here, the real numbers are represented in the fixed-point format. The authors have implemented homomorphic addition, colour transformation, and scaling. In IGHE, the encryption and decryption are carried out element-wise, and hence, the computational cost is higher compared to our HE-FPMK, where the operations are carried out matrix-wise. In [20], 2D-DCT and 2D-IDCT operations are securely outsourced to the cloud. However, additional random matrices are used for homomorphic encryption that results in higher computational overhead compared to HE-FPMK. In [21], the image matrix is converted block-wise into a frequency domain transform and then encrypted based on reversible mathematical operations based on the symmetric key. The decryption process is the inverse of the encryption process and is carried out using the symmetric key. This method involves a higher computational cost due to the block-wise conversion from the space domain to the frequency domain and vice-versa.

In [22], the Paillier Homomorphic Cryptosystem is used for image encryption pixel-by-pixel. Even though the homomorphic addition in the Paillier scheme is relatively less complex, the homomorphic multiplication is highly complex. Additionally, pixel-wise operations increase the computational overhead for large-sized images. In [23], the authors have developed 'Matrix Operation for Randomization and Encryption (MORE)' to provide privacy for training the Alex-net convolutional network. Randomization is provided by augmenting the plaintext matrix by a random diagonal element. However, the scheme is block-oriented and hence computationally expensive.

In [24], the author has used Concrete-Numpy Python APIs to encrypt the image matrix as well as to decrypt it. The built-in library uses extended TFHE (Torus Fully Homomorphic Encryption), which is faster compared to Gentry's FHE scheme. However, this method cannot be used directly on the float data types.

III. Homomorphic Encryption and Decryption

HE-FPMK uses special matrices, whose elements are floating-point numbers as keys for encryption and decryption of image matrices.

A. Matrix Key for Decryption:

Symbol $\boldsymbol{D}$ is used to represent the matrix decryption key. It is a floating-point matrix of size $m{\times}n$ in real field $\mathbb{R}$ with $m > n$. Thus, $\boldsymbol{D}\in \mathbb{R}^{(m\times n)}$. Matrix $\boldsymbol{D}$ is generated such that.

$$\boldsymbol{D}^{\mathrm{T}} * \boldsymbol{D} = \boldsymbol{I}_{n\times n} \tag{2}$$

In (2), $\boldsymbol{D}^{\mathbf{T}}$ is the transpose of $\boldsymbol{D}$. Appendix describes the generation of this semi-orthogonal matrix $\boldsymbol{D}$.

B. Matrix keys for encryption

Matrix keys for encryption are generated from the base matrix, denoted by $\boldsymbol{E}$, which is obtained from $\boldsymbol{D}^{\mathrm{T}}$ as,

$$\boldsymbol{E} = \boldsymbol{D}^{\mathrm{T}} \tag{3}$$

The size of $\boldsymbol{E}$ is $n{\times}m$. From (2) and (3),

$$\boldsymbol{E} * \boldsymbol{D} = \boldsymbol{I}_{n\times n} \tag{4}$$

1) Left Null Space of $\boldsymbol{D}$

The size of $\boldsymbol{D}$ is ($m{\times}n$) with ($m > n$). Hence, $\boldsymbol{D}$ has the left null space [16]. Let this null space of $\boldsymbol{D}$ be represented by $\boldsymbol{F}$ as described in the Appendix. Then,

$$\boldsymbol{F} * \boldsymbol{D} = \boldsymbol{0}_{(m-n)\times n} \tag{5}$$

In (5), the size of $\boldsymbol{F}$ is (m–n)×m. Pre-multiplication of both sides of (5) by a random matrix $\boldsymbol{R}\{1\}_{nx(m-n)}$ gives,

$$\boldsymbol{R}\{1\}_{nx(m-n)} * (\boldsymbol{F} * \boldsymbol{D})_{(m-n)xn}$$
$$= \boldsymbol{R}\{1\}_{nx(m-n)} * \boldsymbol{0}_{(m-n)xn} = \boldsymbol{0}_{nxn}$$

On removing the size indicating subscripts, we get $\boldsymbol{R}\{1\}*(\boldsymbol{F}*\boldsymbol{D}) = \boldsymbol{0}$. This can be rewritten as,

$$(\boldsymbol{R}\{1\}*\boldsymbol{F})*\boldsymbol{D} = \boldsymbol{0} \tag{6}$$

The size of ($\boldsymbol{R}\{1\}*\boldsymbol{F}$) is $n{\times}(m–n) \times (m–n){\times}m = n{\times}m$. Now, Let matrix $\boldsymbol{E}\{1\}$ be formed as,

$$\boldsymbol{E}\{1\} = \boldsymbol{E} + \boldsymbol{R}\{1\}*\boldsymbol{F} \tag{7}$$

On post multiplying both sides of (7) by $\boldsymbol{D}$,

$$\boldsymbol{E}\{1\}*\boldsymbol{D} = \mathrm{E*D} + \boldsymbol{R}\{1\}*\boldsymbol{F}*\boldsymbol{D} \tag{8}$$

From (4), (5), and (8),

$$\boldsymbol{E}\{1\}*\boldsymbol{D} = \boldsymbol{I}_{n\times n}$$

Here, the size of $\boldsymbol{E}\{1\}$ is ($n{\times}m$) which is same as that of $\boldsymbol{E}$. In (7), $\boldsymbol{E}\{1\}$ is obtained by perturbing $\boldsymbol{E}$ by the random matrix $\boldsymbol{R}\{1\}$. Therefore, we denote $\boldsymbol{E}\{1\}$ as the randomized version of $\boldsymbol{E}$. In our proposed scheme HE-FPMK, matrix $\boldsymbol{E}\{1\}$ is the encryption key.

C. Multiple Versions of E{1}

In (7), $\boldsymbol{R}\{1\}$ is an arbitrary random matrix and can have dissimilar values as $\boldsymbol{R}\{2\}$, $\boldsymbol{R}\{3\}$,…, $\boldsymbol{R}\{\mathrm{i}\}$,… and so on. Correspondingly, $\boldsymbol{E} + \boldsymbol{R}\{i\}$ can have dissimilar values as $\boldsymbol{E}\{2\} = \boldsymbol{E} + \boldsymbol{R}\{2\} * \boldsymbol{F}$, $\boldsymbol{E}\{\mathbf{3}\} = E + \boldsymbol{R}\{3\} * \boldsymbol{F}$, ... and so on as,

$$\boldsymbol{E}\{i\} = \boldsymbol{E} + \boldsymbol{R}\{i\} * \boldsymbol{F} \tag{9}$$

In (9), elements of R{i} 's are chosen in the range [0 to +1] with uniform distribution. This is found to give a good random spread for E{i} 's. Thus, $\boldsymbol{E}\{1\}$, $\boldsymbol{E}\{2\}$,…, $\boldsymbol{E}\{\mathrm{i}\}$,… and so on, are different versions of $\boldsymbol{E}$. Then, post multiplying both sides of (9) by $\boldsymbol{D}$ gives,

$$\boldsymbol{E}\{i\} * \boldsymbol{D} = (\boldsymbol{E} + \boldsymbol{R}\{i\} * \boldsymbol{F}) * \boldsymbol{D}$$
$$= \boldsymbol{E} * \boldsymbol{D} + \boldsymbol{R}\{i\} * \boldsymbol{F} * \boldsymbol{D} \tag{10}$$

Then, using (4) and (5) in (10) gives,

$$\boldsymbol{E}\{i\} * \boldsymbol{D} = \boldsymbol{I}_{n\times n} \tag{11}$$

From (3), we have $\boldsymbol{D} = \boldsymbol{E}^{\mathrm{T}}$. Using this in (11) gives,

$$\boldsymbol{E}\{i\} * \boldsymbol{E}^{\mathrm{T}} = \boldsymbol{I}_{n\times n} \tag{12}$$

Multiple versions, $\boldsymbol{E}\{1\}$, $\boldsymbol{E}\{2\}$,…, *etc*., are used for successive encryptions to prevent Chosen Plaintext Attack (CPA).

D. Image Matrix Encryption

In HE-FPMK, an image matrix is encrypted using $\boldsymbol{E}\{i\}$'s and the $\boldsymbol{D}$ matrix as the secret keys. The encryption mode can be a *single side* or *double side,* as will be explained in this section.

1) Single Side Encryption with post multiplication

In Single Side Encryption (SSE), the image matrix is multiplied by the key matrix on the right (post-multiplication) or on the left (pre-multiplication). Let matrix $\boldsymbol{A}$ represent the pixel intensities of the grayscale image to be encrypted. Let the size of $\boldsymbol{A}$ be $k{\times}n$. The data type of the elements of $\boldsymbol{A}$ is uint8, whose range is 0 to 255. The image matrix $\boldsymbol{A}$ is encrypted to get the cipher matrix $\boldsymbol{C}$ as,

$$\boldsymbol{C} = \boldsymbol{A}*\boldsymbol{E}\{i\} \tag{13}$$

In (13), $\boldsymbol{E}\{i\}$ is the i^{th} version of $\boldsymbol{E}$ for some $i \in \mathrm{Z}^{+}$, and the size of $\boldsymbol{E}\{\mathrm{i}\}$ is $n{\times}m$. Therefore, the size of $\boldsymbol{C}$ is $k{\times}m$.

The decryption of $\boldsymbol{C}$ is carried out to get matrix $\boldsymbol{B}$ as,

$$\boldsymbol{B} = \boldsymbol{C}*\boldsymbol{D} \tag{14}$$

Here, the size of $\boldsymbol{D}$ is $m{\times}n$.

2) Correctness of Decryption

Substituting for cipher matrix $\boldsymbol{C}$ from (13) in (14), gives $\boldsymbol{B} = (\boldsymbol{A}*\boldsymbol{E}\{\mathrm{i}\})*\boldsymbol{D} = \boldsymbol{A}*\boldsymbol{E}\{i\}*\boldsymbol{D}$. Since $\boldsymbol{E}\{i\}*\boldsymbol{D} = \boldsymbol{I}_{n\times n}$, matrix $\boldsymbol{B} = \boldsymbol{A}$, which is the original plain matrix.

3) SSE with pre-multiplication:

Here, the size of $\boldsymbol{A}$ is taken as $n\times k$, and matrix $\boldsymbol{C}$ is obtained as $\boldsymbol{C} = \boldsymbol{E}\{i\}^{\mathrm{T}} *\boldsymbol{A}$, and the decryption is carried out as $\boldsymbol{B} = \boldsymbol{D}^{\mathrm{T}}*\boldsymbol{C}$. This means $\boldsymbol{B} = \boldsymbol{D}^{\mathrm{T}}*\boldsymbol{E}\{i\}^{\mathrm{T}}*\boldsymbol{A}$, and since $\boldsymbol{D}^{\mathrm{T}}*\boldsymbol{E}\{\boldsymbol{i}\}^{\mathrm{T}} = \boldsymbol{I}_{n\times n}$ [from the transpose of Eq. (11)], $\boldsymbol{B} = \boldsymbol{A}$.

Single-side encryption can be used only for additive or subtractive homomorphic calculations. It cannot be used for homomorphic multiplication. Therefore, in HE-FPMK, we use the double side HE, which enables homomorphic addition/subtraction and homomorphic multiplication.

B. *Double side encryption*

Hereafter, the Double side encryption is simply called Homomorphic Encryption (HE). The encryption of an image matrix $\boldsymbol{A}$ of size $n\times n$ is carried out as,

$$\boldsymbol{C} = \boldsymbol{D}*\boldsymbol{A}*\boldsymbol{E}\{i\} \tag{15}$$

Here, the sizes of $\boldsymbol{D}$, $\boldsymbol{A}$, and $\boldsymbol{E}\{i\}$are $m\times n$, $n\times n$, and $n\times m$, respectively. Therefore, the size of $\boldsymbol{C}$ is $m\times m$. Matrix $\boldsymbol{C}$ is sent to the Cloud Server (CS) for secured storage and distribution. The computational complexity of (15) is two matrix multiplications, each having $m*n^2$ floating-point multiplications. Taking m nearly equal to n, (In this paper, m is taken equal to n+2), the overall complexity would be of the order of $(2*n^3)$.

1) Decryption of $\boldsymbol{C}$

The end user receives matrix $\boldsymbol{C}$ from the CS and decrypts the cipher matrix $\boldsymbol{C}$ as,

$$\boldsymbol{B} = \boldsymbol{E}*\boldsymbol{C}*\boldsymbol{D} \tag{16}$$

The decryption key $\boldsymbol{D}$ has been made available to the end user through a secured channel. Matrix $\boldsymbol{E}$ is the transpose of $\boldsymbol{D}$ (see Equation (3)), and thus, the decrypter has access to $\boldsymbol{E}$ and $\boldsymbol{D}$ (but no access to $\boldsymbol{E}\{\mathrm{i}\}$'s). The correctness of decryption can be verified by substituting for $\boldsymbol{C}$ from (15) in (16). Then,

$$\boldsymbol{B} = \boldsymbol{E}*(\boldsymbol{D}*\boldsymbol{A}*\boldsymbol{E}\{\mathrm{i}\})*\boldsymbol{D} = \boldsymbol{E}*\boldsymbol{D}*\boldsymbol{A}*\boldsymbol{E}\{\mathrm{i}\}*\boldsymbol{D} \tag{17}$$

From (4), $\boldsymbol{E}*\boldsymbol{D} = \boldsymbol{I}_{n\times n}$ and from (9) and $\boldsymbol{E}\{1\}*\boldsymbol{D} = \boldsymbol{I}_{n\times n}$. Using these relations in (17), we get $\boldsymbol{B} = \boldsymbol{A}$, which verifies the correctness of the decryption formula (16).

C. *Homomorphic Addition of Images*

Let the two plaintext images be represented by matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$, of sizes $n\times n$, which are to be added. They are encrypted as,

$$\left.\begin{aligned}\boldsymbol{C}_1 &= \boldsymbol{D} * \boldsymbol{A}_1 * \boldsymbol{E}\{1\}\\ \boldsymbol{C}_2 &= \boldsymbol{D} * \boldsymbol{A}_2 * \boldsymbol{E}\{2\}\end{aligned}\right\} \tag{18}$$

$\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are sent to the CS, which adds $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ as,

$$\boldsymbol{C}_3 = \boldsymbol{C}_1 + \boldsymbol{C}_2 \tag{19}$$

$\boldsymbol{C}_3$ is sent to the end user, who decrypts it as,

$$\boldsymbol{B}_3 = \boldsymbol{E} * \boldsymbol{C}_3 * \boldsymbol{D} \tag{20}$$

On substituting for $\boldsymbol{C}_3$ from (19) and further substituting for $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ from (18), Equation (20) gives,

$$\begin{aligned}\boldsymbol{B}_3 &= \boldsymbol{E} * (\boldsymbol{D} * \boldsymbol{A}_1 * \boldsymbol{E}\{1\} + \boldsymbol{D} * \boldsymbol{A}_2 * \boldsymbol{E}\{2\}) * \boldsymbol{D}\\ &= \boldsymbol{E} * \boldsymbol{D} * \boldsymbol{A}_1 * \boldsymbol{E}\{1\} * \boldsymbol{D} + \boldsymbol{E} * \boldsymbol{D} * \boldsymbol{A}_2 * \boldsymbol{E}\{2\} * \boldsymbol{D}\end{aligned}$$

Using (4) and (11) in the above Equation gives

$$\boldsymbol{B}_3 = \boldsymbol{A}_1 + \boldsymbol{A}_2.$$

The addition of images is generally used for:

- Insert the time stamp or a text or an icon, etc.
- Camouflage a part of an image to hide the identity.
- Add a background image to the target image.
- Overlay the detected edges of an image onto itself.

D. *Homomorphic multiplication*

The Homomorphic multiplication takes place in the CS to get $\boldsymbol{C}_3$ as,

$$\boldsymbol{C}_3 = \boldsymbol{C}_1*\boldsymbol{C}_2 \tag{21}$$

where $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are defined in Equation (18).

The size of $\boldsymbol{C}_3$ is $(m\times m)$. The decryption of $\boldsymbol{C}_3$ is carried out by the end user as,

$$\boldsymbol{B}_3 = \boldsymbol{E} * \boldsymbol{C}_3 * \boldsymbol{D} \tag{22}$$

On substituting for $\boldsymbol{C}_3$ from (21) and further substituting for $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ from (18), Equation (22) gives,

$$\boldsymbol{B}_3 = \boldsymbol{E} * \boldsymbol{D} * \boldsymbol{A}_1 * \boldsymbol{E}\{1\} * \boldsymbol{D} * \boldsymbol{A}_2 * \boldsymbol{E}\{2\} * \boldsymbol{D}$$

Substituting for $\boldsymbol{E} * \boldsymbol{D}$ from (4), $\boldsymbol{E}\{1\} * \boldsymbol{D}$ and $\boldsymbol{E}\{2\} * \boldsymbol{D}$ from (11), we have, $\boldsymbol{B}_3 = \boldsymbol{A}_1 * \boldsymbol{A}_2$. Thus, the multiplication in the cipher domain yields the correct product on decryption.

IV. Applications of Homomorphic Encryption in Image Processing

A few applications of HE-FPMK in image processing are presented in this section to demonstrate its novelty.

A. *Homomorphic Calculation of DCT of images*

The 2-dimensional DCT (Discrete Cosine Transform) of an image matrix $\boldsymbol{A}$ of size $n\times n$ can be calculated using the built-in function `dct2(`$\boldsymbol{A}$`)` or using the dct-matrix `dctmtx(`n`)` [25]. Let the dct-matrix of size $n\times n$ be denoted as

$$\boldsymbol{G} = \mathtt{dctmtx}(n) \tag{23}$$

Then, the 2D DCT of $\boldsymbol{A}$, represented by $\boldsymbol{H}$, is given by [25] as

$$\boldsymbol{H} = \boldsymbol{G}*\boldsymbol{A}*\boldsymbol{G}^{\mathrm{T}} \tag{24}$$

Thus, the calculation 2D DCT is translated into the process of matrix multiplication, which can be accomplished using the HE of $\boldsymbol{G}$ and $\boldsymbol{A}$ as follows. Matrix $\boldsymbol{A}$ is encrypted as,

$$\boldsymbol{C}_1 = \boldsymbol{D}*\boldsymbol{A}*\boldsymbol{E}\{i\} \tag{25}$$

Matrix $\boldsymbol{G}$ is encrypted as,

$$\boldsymbol{C}_2 = \boldsymbol{D}*\boldsymbol{G}*\boldsymbol{E} = \boldsymbol{D}*\boldsymbol{G}*\boldsymbol{D}^{\mathrm{T}} \tag{26}$$

In (26), $\boldsymbol{E}$ is equivalent to $\boldsymbol{D}^{\mathrm{T}}$ as given by (3). Now, in the cipher domain, $\boldsymbol{C}_3$ is obtained as,

$$\boldsymbol{C}_3 = \boldsymbol{C}_2*\boldsymbol{C}_1*\boldsymbol{C}_2^{\mathrm{T}} \tag{27}$$

Now, the end user decrypts $\boldsymbol{C}_3$, as usual, to get $\boldsymbol{B}_3$ as,

$$\boldsymbol{B}_3 = \boldsymbol{E}*\boldsymbol{C}_3*D \tag{28}$$

On substituting for $\boldsymbol{C}_3$ from (27) and further substitution for $\boldsymbol{C}_2$ and $\boldsymbol{C}_1$ from (26) and (25) yields,

$$\begin{aligned}\boldsymbol{B}_3 &= \boldsymbol{E}*\boldsymbol{C}_2*\boldsymbol{C}_1*\boldsymbol{C}_2^{\mathrm{T}}*\boldsymbol{D}\\ &= \boldsymbol{E}*\boldsymbol{D}*\boldsymbol{G}*\boldsymbol{E}*\boldsymbol{D}*\boldsymbol{A}*\boldsymbol{E}\{i\}*\boldsymbol{E}^{\mathrm{T}}*\boldsymbol{G}^{\mathrm{T}}*\boldsymbol{D}^{\mathrm{T}}*\boldsymbol{D}\end{aligned} \tag{29}$$

Using (4), (12), and (2) in (33) gives $\boldsymbol{B}_3 = \boldsymbol{G}*\boldsymbol{A}*\boldsymbol{G}^{\mathrm{T},}$ which is the same as $\boldsymbol{H}$. Similarly, the inverse DCT of $\boldsymbol{A}$ can be determined using $\boldsymbol{C}_3 = \boldsymbol{C}_2^{\mathrm{T}}*\boldsymbol{C}_1*\boldsymbol{C}_2$.

For large-size image matrices, homomorphic DCT calculation can be carried out using block-wise processing. Then, this method is faster as we calculate $\boldsymbol{C}_2$ only once. A major security advantage of this method is that the CS cannot access the secret key $\boldsymbol{D}$ even when it knows $\boldsymbol{C}_2$ due to the product format of $\boldsymbol{C}_2$ as given by (26). Here, no random matrices are used to hide the secret keys as in [20].

Similar homomorphic encryption can be carried out for calculating `fft2(`$\boldsymbol{A}$`)` using `dftmtx(`n`)` and their transposes.

B. *Weighted Addition of Images*

The addition of images with different weights is used in visible watermarking, a fusion of images, image morphing, etc. Using HE-FPMK, the weighted addition of images can be delegated to the cloud server while maintaining image privacy. Let $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ be the image matrices to be added with scalar weights w_1 and w_2 to get the weighted sum $\boldsymbol{A}_3$ as,

$$\boldsymbol{A}_3 = w_1 * \boldsymbol{A}_1 + w_2 * \boldsymbol{A}_2 \tag{30}$$

In some special applications, w_1 and w_2 are fractions in the range 0 to 1 with constraint $w_1 + w_2 = 1$, so that the increase in one component is balanced by the corresponding decrease in the other component. Otherwise, w_1 and w_2 can be any floating point weights according to the needs of the problem. Weighted addition in the cipher domain is carried out as follows.

Matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are encrypted according to (18), as

$$\left.\begin{aligned} \boldsymbol{C}_1 &= \boldsymbol{D} * \boldsymbol{A}_1 * \boldsymbol{E}\{1\} \\ \boldsymbol{C}_2 &= \boldsymbol{D} * \boldsymbol{A}_2 * \boldsymbol{E}\{2\} \end{aligned}\right\}$$

Weighted addition is performed in the CS as,

$$\boldsymbol{C}_3 = w_1 * \boldsymbol{C}_1 + w_2 * \boldsymbol{C}_2 \tag{31}$$

Matrix $\boldsymbol{C}_3$ is decrypted by the end user to get $\boldsymbol{B}_3$ as,

$$\boldsymbol{B}_3 = \boldsymbol{E} * \boldsymbol{C}_3 * \boldsymbol{D} \tag{32}$$

On substituting for $\boldsymbol{C}_3$ from (31) and further substituting for $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ from (18), Equation (32) gives,

$$\begin{aligned} \boldsymbol{B}_3 &= \boldsymbol{E} * (w_1 * \boldsymbol{D} * \boldsymbol{A}_1 * \boldsymbol{E}\{1\} + w_2 * \boldsymbol{D} * \boldsymbol{A}_2 * \boldsymbol{E}\{2\}) * \boldsymbol{D} \\ &= w_1 * \boldsymbol{E} * \boldsymbol{D} * \boldsymbol{A}_1 * \boldsymbol{E}\{1\} * \boldsymbol{D} \\ &\quad + w_2 * \boldsymbol{D} * \boldsymbol{A}_2 * \boldsymbol{E}\{2\} * \boldsymbol{D} \end{aligned} \tag{33}$$

On substituting (4) and (11) in (33), we get,

$$\boldsymbol{B}_3 = w_1 * \boldsymbol{A}_1 + w_2 * \boldsymbol{A}_2$$

This means the decrypted matrix. $\boldsymbol{B}_3 = \boldsymbol{A}_3$ and it is the weighted addition as required by (31).

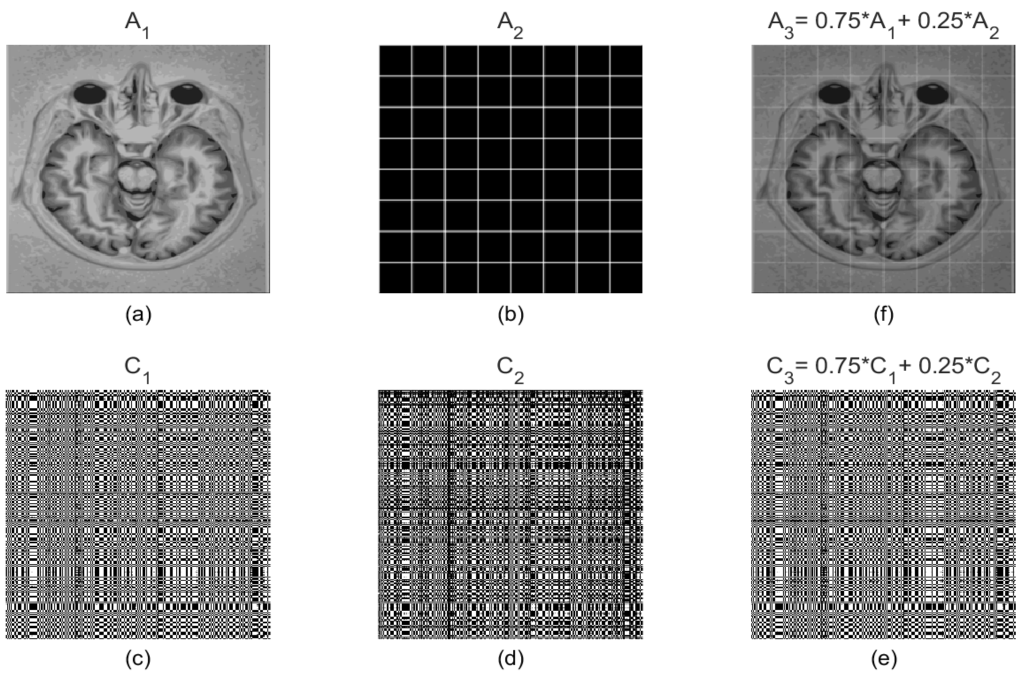

Fig. 1. Homomorphic weighted addition of images

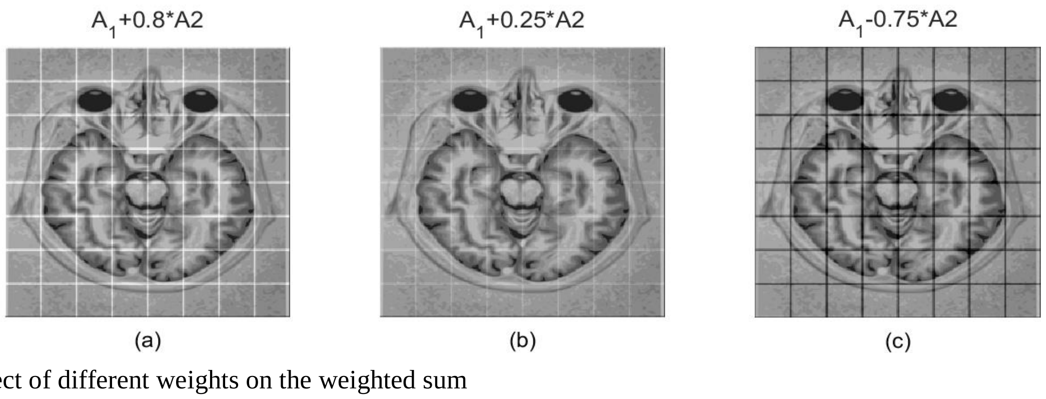

Fig. 2. Effect of different weights on the weighted sum

Example 1. Here, a square grid image $\boldsymbol{A}_2$ is superimposed on an MRI image $\boldsymbol{A}_1$ with weighted addition, as shown in Fig. 1. Matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are expressed in the double precision format in the range [0, 1] so that the range of weighted sum is relatively small. (Matlab function `im2double(…)` is used to convert the image matrix from uint8 to double format).

The effect of choosing weights $w_1 = 0.75$ and $w_2 = 0.25$ is shown in Fig. 1. Here, subplots (c) and (d) show the cipher images of plain images at subplots (a) and (b), respectively. The weighted cipher sum C_3 is shown in Fig. 1(e), and the final decrypted weighted sum image is shown in Fig. 1(f).

The effect of different weights on the resulting weighted sum is shown in Fig. 2. In Fig. 2(c), $w_1 = 1$ and $w_2 = -0.75$. Thus, it can be seen that the HE-FPMK can handle weighted sums with positive as well as negative weights comfortably.

C. Homomorphic Proxy Re-encryption of Images

Homomorphic Proxy Re-encryption (HPRE) is the procedure where a given ciphertext intended for a user (say user *U*) is re-encrypted to enable another user (say user *V*) to decrypt it correctly. The HPRE process is designed in such a way that the re-encrypter itself is incapable of discovering the plaintext. Also, it cannot access either the encryption keys or the decryption keys of the end users (user *U* and user *V*). Fig. 3 shows the basic block diagram of a PRE scheme.

1) Single Side Homomorphic Proxy Re-encryption

Single Side Homomorphic Proxy Re-encryption (SSHPRE) scheme uses two distinct sets of encryption and decryption keys for two distinct users. Let the primary user be denoted as user U whose

cryptographic keys and encryption are same as in section III. B. Thus, the SSE is carried out as in (13) except that $\boldsymbol{E}\{i\}$ is written as $\boldsymbol{E}_{\mathrm{U}}$, for easy writing as,

$$\boldsymbol{C}_{\mathrm{U}} = \boldsymbol{A} * \boldsymbol{E}_{\mathrm{U}} \tag{34}$$

In (34), $\boldsymbol{A}$ is the plain image matrix of size $k \times n$ to be encrypted, $\boldsymbol{E}_{\mathrm{U}}$ of size $n \times m$ is the encryption key for user U, and $\boldsymbol{C}_{\mathrm{U}}$ of size $k \times m$ is the encrypted matrix with respect to user U. The encrypted matrix $\boldsymbol{C}_{\mathrm{U}}$ is sent to the CS which also houses the proxy re-encrypter as shown in Figure 3.

The decryption of $\boldsymbol{C}_{\mathrm{U}}$ is carried out as usual, as,

$$\boldsymbol{B}_{\mathrm{U}} = \boldsymbol{C}_{\mathrm{U}} * \boldsymbol{D}_{\mathrm{U}} \tag{35}$$

where $\boldsymbol{D}_{\mathrm{U}}$ of size $m \times n$ is the decryption key matrix of user U with the property,

$$\boldsymbol{E}_{\mathrm{U}} * \boldsymbol{D}_{\mathrm{U}} = \boldsymbol{I}_{n \times n} \tag{36}$$

From (34), (35), and (36), it can be verified that $\boldsymbol{B}_{\mathrm{U}} = \boldsymbol{A}$.

Now, a second set of distinct cryptographic keys is generated for the secondary user V, similar to as explained in section III. B. Let $\boldsymbol{E}_{\mathrm{V}}$ generically represent the encryption key, and the decryption key of user V be $\boldsymbol{D}_{\mathrm{V}}$ with properties similar to (36),

$$\boldsymbol{E}_{\mathrm{V}} * \boldsymbol{D}_{\mathrm{V}} = \boldsymbol{I}_{n \times n} \tag{37}$$

The Proxy Re-encrypter unit within CS provides HPRE as follows. Initially, the Key Generation Center (KGC) of the data owner generates all the cryptographic keys: $\boldsymbol{E}_{\mathrm{U}}$, $\boldsymbol{D}_{\mathrm{U}}$, $\boldsymbol{E}_{\mathrm{V}}$, and $\boldsymbol{D}_{\mathrm{V}}$. Keys $\boldsymbol{D}_{\mathrm{U}}$ and D_{V} are sent through secured channels to user U and user V, respectively.

2) Re-encryption key

In SSHPRE, the proxy re-encryption key, denoted by $\boldsymbol{E}_{\mathrm{UV}}$ is generated by the KGC, and it is given by,

$$\boldsymbol{E}_{\mathrm{UV}} = \boldsymbol{D}_{\mathrm{U}} * \boldsymbol{E}_{\mathrm{V}} \tag{38}$$

Here, the sizes of $\boldsymbol{E}_{\mathrm{UV}}$, $\boldsymbol{D}_{\mathrm{U}}$, and $\boldsymbol{E}_{\mathrm{V}}$ are $m \times m$, $m \times n$, and $n \times m$, respectively. The KGC sends $\boldsymbol{E}_{\mathrm{UV}}$ to the proxy re-encrypter at the beginning of the session. At the proxy re-encrypter, the presence of

$\boldsymbol{E}_{\mathrm{UV}}$ does not reveal any information about $\boldsymbol{D}_{\mathrm{U}}$ and $\boldsymbol{E}_{\mathrm{V}}$. Thus, the secrecy of the keys is not breached at the proxy re-encrypter.

3) Re-encryption operation

During the HPRE operation, the proxy Re-encrypter receives the present cipher matrix $\boldsymbol{C}_{\mathrm{U}}$ (intended for user U) and translates $\boldsymbol{C}_{\mathrm{U}}$ to C_{V} (suitable for user V) using $\boldsymbol{E}_{\mathrm{UV}}$ as,

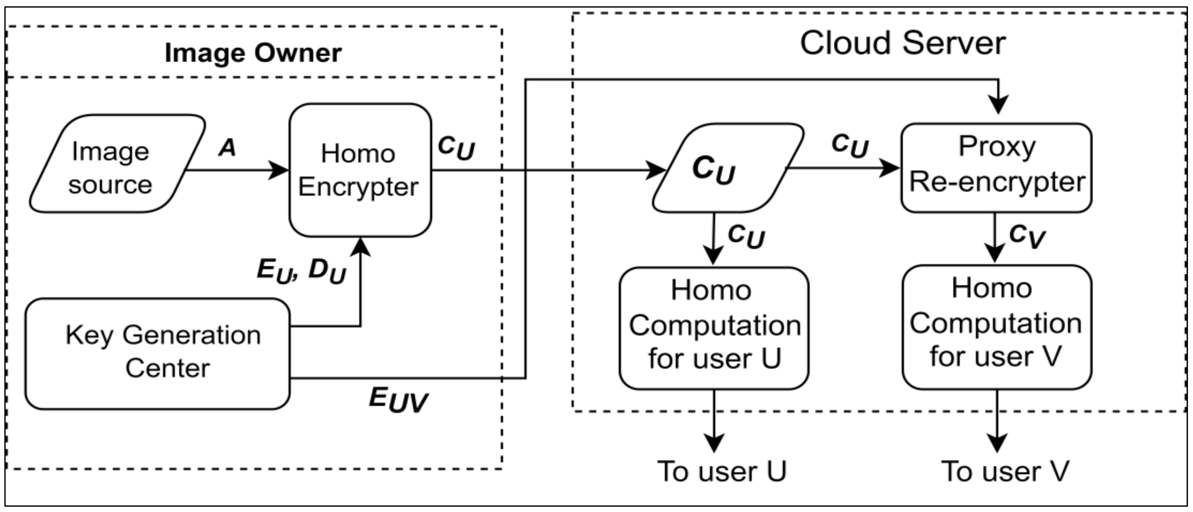

Fig. 3. Basic Proxy Re-encryption Scheme

$$\boldsymbol{C}_{\mathrm{V}} = \boldsymbol{C}_{\mathrm{U}}*\boldsymbol{E}_{\mathrm{UV}} \tag{39}$$

This C_V with size $k\times m$ is received by user V, and he/she decrypts $\boldsymbol{C}_V$ as,

$$\boldsymbol{B}_{\mathrm{V}} = \boldsymbol{C}_{\mathrm{V}}*\boldsymbol{D}_{\mathrm{V}} \tag{40}$$

4) Correctness of SSHPRE

On substituting for $\boldsymbol{C}_V$ from (39) in (40), we have,

$$\boldsymbol{B}_{\mathrm{V}} = \boldsymbol{C}_{\mathrm{U}}*\boldsymbol{E}_{\mathrm{UV}}*\boldsymbol{D}_{\mathrm{V}} \tag{41}$$

On further substituting for $\boldsymbol{C}_U$ from (34) and for $\boldsymbol{E}_{UV}$ from (38), we have,

$$\boldsymbol{B}_{\mathrm{V}} = \mathrm{A}*\boldsymbol{E}_{\mathrm{U}}*\boldsymbol{D}_{\mathrm{U}}*\boldsymbol{E}_{\mathrm{V}}*\boldsymbol{D}_{\mathrm{V}} \tag{42}$$

Applying the property $\boldsymbol{E}_U*\boldsymbol{D}_U = \boldsymbol{I}_{n\times n} = \boldsymbol{E}_V*\boldsymbol{D}_V$, we get $\boldsymbol{B}_V = \boldsymbol{A}$. This proves the correctness of SSHPRE. When the proxy re-encrypter serves multiple users, the CS can provide one-to-many secure data distribution. HPRE can also be designed using the double side HE with higher security.

V. Performance Analysis

In HE-FPMK, homomorphic encryption and decryption are carried out using floating point (double precision) matrix keys. The

performance metrics of these operations are discussed in this section.

A. Histogram analysis

The histogram of an image gives the distribution of pixels based on their intensity levels. In purely shuffle-based image encryption schemes, the original intensity levels are not changed while their locations are shuffled.

Therefore, the histogram of the encrypted image remains the same as that of the original one. In such a situation, the histogram

of the encrypted image can be used to identify the original image. Hence, apart from shuffling, image diffusion is used to completely alter the histogram for full privacy

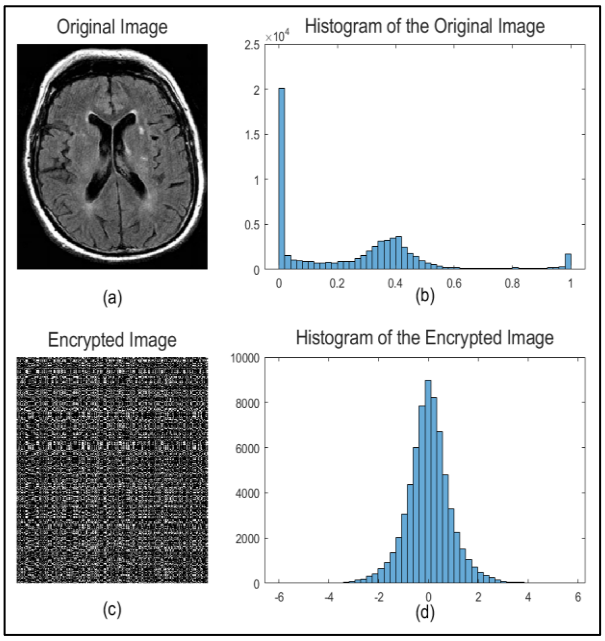

Fig. 4. Histograms of the original and the encrypted

preservation. Thus, additional processing is involved in shuffle-based methods. On the other hand,
in HE-FPMK, the basic encryption operation itself changes all the pixel intensities, and the histogram of the encrypted image is entirely different from the original one. Fig. 4 shows the histograms of the original image and that of the encrypted image.

In Fig. 4, the bell-shaped (Gaussian) distribution is due to the special characteristics of the encryption keys which are derived from the orthogonal matrix $\boldsymbol{Q}$ (see Appendix) obtained through the QR decomposition. From Figure 4(b) and 4(d), we see that the histograms of the original and the encrypted image are entirely dissimilar.

B. Security of HE-FPMK

Some of the security aspects of HE-FPMK are discussed in this section. Brute force guessing of secret keys is almost impossible as each element of a secret key is a 64-bit floating point number, and the probability of correctly guessing the secret key is 2^{-64}.
Additionally, the size of each key is $m\times n$. Hence, the overall probability of correct guessing is $2^{-64*m*n,}$ which is extremely low.

1) Chosen Plaintext Attack

In HE-FPMK, successive encryptions use randomized encryption keys, namely E{i} 's. Therefore, the knowledge of cipher matrices cannot reveal the encryption keys.

Chosen Ciphertext Attack

In HE-FPMK, the Chosen Ciphertext Attack (CCA) can be prevented using the digital signature scheme that authenticates the cipher matrix as well as its source. Prevention of CCA will be implemented in the next version of HE-FPMK.

C. Computational cost of HE-FPMK

The computational cost of encryption and decryption in HE-FPMK are measured in terms of 'bit multiplications.' The cost of the addition is ignored.

1) Computational cost of encryption

In HE-FPMK, the image matrix is encrypted as,

$$\boldsymbol{C} = \boldsymbol{D}*\boldsymbol{A}*\boldsymbol{E}\{\mathrm{i}\}$$

Here, the sizes of $\boldsymbol{D}$, $\boldsymbol{A}$, and $\boldsymbol{E}\{\mathrm{i}\}$are $m\times n$, $n\times n$, and $n\times m$, respectively, and the multiplication $\boldsymbol{D}*\boldsymbol{A}$ involves $m*n^2$ Floating Point (FP) multiplications. Then, $(\boldsymbol{D}*\boldsymbol{A})*\boldsymbol{E}\{i\}$ requires m^2*n multiplications. The total number of multiplications is $m*n^2+ m^2*n$. Since $m = n+2 \cong n$, we have $2*n^3$ FP multiplications. Taking 64 bits for a double precision FP number, the total number of 'bit multiplications.' incurred for encryption, represented by TBME is,

$$\mathrm{TBME} = 2*n^3*64 = 128*n^3 \tag{43}$$

2) Computational cost of decryption: The decryption process is given by (Eq. (16)) as,

$$\boldsymbol{B} = \boldsymbol{E}*\boldsymbol{C}*\boldsymbol{D}$$

Here, the sizes of $\boldsymbol{E}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are $n\times m$, $m\times m$, and $m\times n$, respectively. Therefore, the total number of floating point multiplications in Zp, is $n*m^2+ m^2*n \cong 2*n^3$. Similar to as in encryption, the total number of 'bit multiplications.'

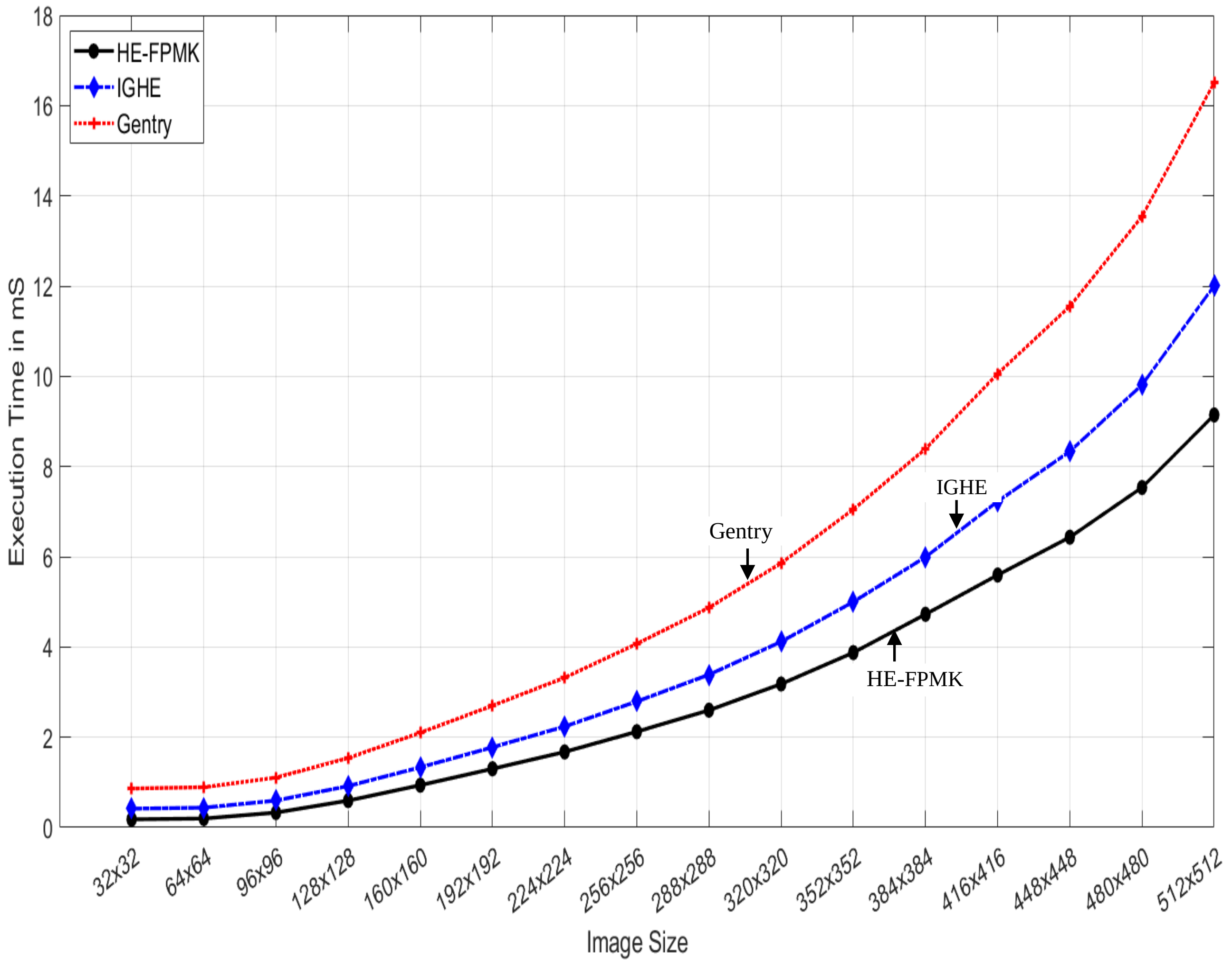

Fig. 5. Comparison of execution times for homomorphic encryption of image matrices

incurred for decryption is same as that for encryption and is represented by TBMD as,

$$\text{TBMD} = 2*n^3*64 = 128*n^3$$

Thus, TMBD is proportional to the third power of *n*.

D. Comparison of the runtime of HE-FPMK with other methods

Here, the runtime of HE-FPMK encryption is compared with those of the Gentry (GHV) scheme [18] and Yang's IGHE [19]. In this simulation, the grayscale image size is varied from 32x32 to 512x512, where both the height and width are incremented by 32 in each step. The runtimes of HE-FPMK, IGHE, and Gentry methods are experimentally determined, and the results are plotted as shown in Fig. 5.

In Fig. 5, the execution times obtained are machine-dependent, and hence, the plots represent only the relative values. From the plots in Fig. 5, it can be seen that the HE-FPMK method has the lowest execution time compared to the other two methods.

E. Ciphertext Expansion Ratio

Ciphertext Expansion Ratio (*CER*) is the ratio of the ciphertext matrix size to that of its plaintext matrix. A higher *CER* incurs a higher computational and communication cost. A smaller *CER* ensures better performance. *CER* is defined as,

$$CER = \frac{Size\ of\ Ciphermatrix\ in\ bits}{Size\ of\ Plain\ matrix\ in\ bits} \qquad (44)$$

In HE-FPMK, as per Equation (15), the size of the cipher matrix **C** is (m×m), and that of plaintext matrix **A** is (n×n). Here, the elements of **C** are of size 64 bits (type double), and those of **A** are 8 bits. Therefore, for the double side encryption, from (46), it can be seen that

$$CER(bytes) = \frac{m*m*64}{n*n*8}$$

Since *m* = (*n*+2), as an approximation, *m* can be taken equal to *n*, and then the *CER*(bytes) = 64/8 = 8. In terms of the number of matrix elements, *CER*(no. of elements) is,

$$CER = \frac{m*m}{n*n} = \frac{(n+2)*(n+2)}{n*n} \approx 1$$

The plain matrix size (in bytes), the sizes of the Encryption keys, and the cipher matrices for HE-FPMK, IGHE, and the Gentry method, along with the corresponding ratios, are shown in Table 1. In HE-FPMK, the Encryption key and the cipher matrix are in float64, whereas in IGHE these values are in bytes. Hence, the sizes and ratios are higher compared to IGHE. In the Gentry method, the basic unit of plaintext is a bit. Therefore, the elements of the plain matrix are converted from bytes to bits and vice-versa during decryption. Hence, in the Gentry method, an 8-fold increase occurs in the sizes of the key and cipher matrix.

TABLE I
Sizes of keys, cipher matrices, and the ratios

	HE-FPMK	IGHE	Gentry
Plain Matrix Size	n*n	n*n	n*n
Encryption Key Size	n*m*64	n*n	64*n*n
Cipher Matrix Size	m*m*64	n*n	64*n*n
Keyspace Expansion Ratio	$64*\left(\frac{m}{n}\right)$ ≈ 64	1	64
Cipher matrix Expansion Ratio	$64*\left(\frac{m*m}{n*n}\right)$ ≈ 64	1	64

VI. Conclusion

A new method of homomorphic encryption for floating point data is presented using matrix keys. These matrix keys can encrypt image matrices directly without the need for element-by-element encryption. The proposed method uses randomized encryption to prevent Chosen Plaintext Attack. The floating point homomorphic encryption is useful for weighted addition and the calculation of DCT and FFT of images, including their inverses. Homomorphic proxy re-encryption is another new contribution of this work. Our proposed method can be extended to HDR (High Dynamic Range) images.

Appendix

In HE-FPMK, the decryption matrix key $\boldsymbol{D}$ is generated using the QR decomposition of a random square matrix $\boldsymbol{S}$ of size $m \times m$. In HE-FPKM, matrix S is obtained using the Matlab function `randi(…)` as,

$$\boldsymbol{S} = \texttt{randi([-10,10],}m\texttt{)} \tag{A1}$$

Here, the range (-10 to +10) is found to give good results during homomorphic encryption/decryption. Then, the QR decomposition of $\boldsymbol{S}$ is obtained using the Matlab function `qr(…)` as,

$$(\boldsymbol{Q},\ \boldsymbol{UT}) = \texttt{qr(}\boldsymbol{S}\texttt{)} \tag{A2}$$

In (A2), $\boldsymbol{Q}$ is the $m \times m$ orthogonal matrix, and UT is the $m \times m$ upper triangular matrix. Since $\boldsymbol{Q}$ is orthogonal,

$$\boldsymbol{Q}^T * \boldsymbol{Q} = \boldsymbol{I}_{\boldsymbol{m}\times\boldsymbol{m}} \tag{A3}$$

Here, the elements of $\boldsymbol{Q}$ are floating point numbers. The matrices $\boldsymbol{Q}^{\mathrm{T}}$ and $\boldsymbol{Q}$ are partitioned into sub-matrices as,

$$\boldsymbol{Q}^T = \begin{bmatrix} \boldsymbol{E}_{\boldsymbol{n}\times\boldsymbol{m}} \\ -- \\ \boldsymbol{F}_{(\boldsymbol{m}-\boldsymbol{n})\times\boldsymbol{m}} \end{bmatrix}, \qquad \boldsymbol{Q} = \left[\boldsymbol{D}_{\boldsymbol{m}\times\boldsymbol{n}} \mid \boldsymbol{G}_{\boldsymbol{m}\times(\boldsymbol{m}-\boldsymbol{n})}\right] \tag{A4}$$

In (A4) n is taken as $n = (m–2)$ or $m = n+2$. From (A4), it can be seen that,

$$\boldsymbol{E}_{\boldsymbol{n}\times\boldsymbol{m}} = \boldsymbol{D}^{\boldsymbol{T}}_{\boldsymbol{m}\times\boldsymbol{n}} \tag{A5}$$

To match the partitions of $\boldsymbol{Q}^{\mathrm{T}}$ and $\boldsymbol{Q}$, the matrix $\boldsymbol{I}_{\boldsymbol{m}\times\boldsymbol{m}}$ is also partitioned as,

$$\boldsymbol{I}_{\boldsymbol{m}\times\boldsymbol{m}} = \begin{bmatrix} \boldsymbol{I}_{\boldsymbol{n}\times\boldsymbol{n}} & | & \boldsymbol{0}_{\boldsymbol{n}\times(\boldsymbol{m}-\boldsymbol{n})} \\ -- & | & -- \\ \boldsymbol{0}_{(\boldsymbol{m}-\boldsymbol{n})\times\boldsymbol{n}} & | & \boldsymbol{I}_{(\boldsymbol{m}-\boldsymbol{n})\times(\boldsymbol{m}-\boldsymbol{n}} \end{bmatrix} \tag{A6}$$

Substituting (A4) and (A6) in (A3) leads to,

$$\begin{bmatrix} \boldsymbol{E}_{\boldsymbol{n}\times\boldsymbol{m}} \\ — \\ \boldsymbol{F}_{(\boldsymbol{m}-\boldsymbol{n})\times\boldsymbol{m}} \end{bmatrix} * \left[\boldsymbol{D}_{\boldsymbol{m}\times\boldsymbol{n}} \mid \boldsymbol{G}_{\boldsymbol{m}\times(\boldsymbol{m}-\boldsymbol{n})}\right] =$$

$$\begin{bmatrix} \boldsymbol{I}_{\boldsymbol{n}\times\boldsymbol{n}} & | & \boldsymbol{0}_{\boldsymbol{n}\times(\boldsymbol{m}-\boldsymbol{n})} \\ -- & | & -- \\ \boldsymbol{0}_{(\boldsymbol{m}-\boldsymbol{n})\times\boldsymbol{n}} & | & \boldsymbol{I}_{(\boldsymbol{m}-\boldsymbol{n})\times(\boldsymbol{m}-\boldsymbol{n}} \end{bmatrix} \tag{A7}$$

From (A7), we have,

$$\boldsymbol{E}_{\boldsymbol{n}\times\boldsymbol{m}} * \boldsymbol{D}_{\boldsymbol{m}\times\boldsymbol{n}} = \boldsymbol{I}_{\boldsymbol{n}\times\boldsymbol{n}} \tag{A8}$$

$$\boldsymbol{F}_{(\boldsymbol{m}-\boldsymbol{n})\times\boldsymbol{m}} * \boldsymbol{D}_{\boldsymbol{m}\times\boldsymbol{n}} = \boldsymbol{0}_{(\boldsymbol{m}-\boldsymbol{n})\times\boldsymbol{n}} \tag{A9}$$

From (A8) and (A5), it can be seen that,

$$\boldsymbol{D}^{\boldsymbol{T}}_{\boldsymbol{m}\times\boldsymbol{n}} * \boldsymbol{D}_{\boldsymbol{m}\times\boldsymbol{n}} = \boldsymbol{I}_{\boldsymbol{n}\times\boldsymbol{n}} \tag{A10}$$

On removing the dimension subscripts, $\boldsymbol{D}^{\mathrm{T}} * \boldsymbol{D} = \boldsymbol{I}$. Now, the property (A9) means, $\boldsymbol{F}_{(\boldsymbol{m}-\boldsymbol{n})\times\boldsymbol{m}}$ is the left-null space of $\boldsymbol{D}_{\boldsymbol{m}\times\boldsymbol{n}}$. On removing the dimension subscripts, $\boldsymbol{F} * \boldsymbol{D} = 0$.

References

[1] R. Kui, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, 16(1), 2012, pp. 69-73.

[2] H. Ghanbari-Ghalehjoughi, M. Eslami, S. Ahmadi-Kandjani, M. Ghanbari-Ghalehjoughi, Z. Yu, "Multiple layer encryption and steganography via multi-channel ghost imaging," *Optics and Lasers in Engineering*, vol. 134, 2020, 106227, ISSN 0143-8166, pp. 1-12.

[3] G. K. Mahato and S. K. Chakraborty, "A comparative review on homomorphic encryption for cloud security," *IETE Journal of Research*, 2021, pp. 1-10.

[4] S. Kumar, B. K. Singh, Akshita, S. Pundir, S. Batra and R. Joshi, "A survey on symmetric and asymmetric key based image encryption," *2nd International Conference on Data, Engineering and Applications* (IDEA), 2020, pp. 1-5.

[5] N. Dowlin, *et al*., "Manual for using homomorphic encryption for bioinformatics," *in Proceedings of the IEEE*, vol. 105, no. 3, pp. 552-567, March 2017.

[6] W. Fu, R. Lin, D. Inge, "Fully Homomorphic Image Processing," *CoRR abs/1810.03249*, 2018, pp. 1-12.

[7] Q. Wang, *et al*., "Catch me in the dark: Effective privacy-preserving outsourcing of feature extractions over image data," *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1-9.

[8] N. Dowlin, *et al*., "CryptoNets: applying neural networks to encrypted data with high throughput and accuracy," *In Proceedings of the 33rd International Conference on International Conference on Machine Learning* - vol. 48 (*ICML'16*). *JMLR.org*, pp. 201–210. 2016.

[9] T. Shortell and S. Ali. "Secure Convolutional Neural Network using FHE." *ArXiv abs/1808.03819*, 2018, pp. 1-14.

[10] L. Jiang, C. Xu, X. Wang, B. Luo, and H. Wang, "Secure outsourcing SIFT: Efficient and Privacy-Preserving Image Feature Extraction in the Encrypted Domain," *in IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 179-193, 2020.

[11] C. Lupaşcu, C. Pleşca, and M. Togan, "Privacy-Preserving Morphological Operations for Digital Images," *2020 13th International Conference on Communications (COMM)*, pp. 183-188. 2020.

[12] M. Jiang and H. Yang, "Secure Outsourcing Algorithm of BTC Feature Extraction in Cloud Computing," *in IEEE Access*, vol. 8, pp. 106958-106967, 2020.

[13] A.M. Vengadapurvaja, G. Nisha, R. Aarthy, N. Sasikaladevi, "An Efficient Homomorphic Medical Image Encryption Algorithm For Cloud Storage Security," *Procedia Computer Science*, vol. 115, pp. 643-650, 2017.

[14] L. Li, A. AbdEl-Latif, and X. Niu, "Elliptic curve EIGamal based homomorphic image encryption scheme for sharing secret images," *Signal Processing*. Vol. 92, pp. 1069-1078. 2012.

[15] G. Pradel and C. Mitchell, "Privacy-Preserving Biometric Matching Using Homomorphic Encryption," *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 494-505, 2021.

[16] D. Chen, W. Chen, J. Chen, P. Zheng, and J. Huang, "Edge Detection and Image Segmentation on Encrypted Image with Homomorphic Encryption and Garbled Circuit," *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1-6, 2018.

[17] R. Challa, G. VijayaKumari and B. Sunny, "Secure Image processing using LWE based Homomorphic encryption," *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1-6, 2015.

[18] C. Gentry, S. Halevi, and V. Vaikuntanathan, "A Simple BGN-Type Cryptosystem from LWE," In: Gilbert, H. (eds) *Advances in Cryptology – EUROCRYPT 2010. Lecture Notes in Computer Science, vol 6110.* Springer, Berlin, Heidelberg. pp. 1-15, 2010.

[19] P. Yang, X. Gui, J. An, and F. Tian, "An Efficient Secret Key Homomorphic Encryption Used in Image Processing Service," *Security and Communication Networks.* pp. 1-11, 2017.

[20] D. An, S. Zhang, J. Lu, and Y. Li, "Efficient and Privacy-Preserving Outsourcing of 2D-DCT and 2D-IDCT," *Wireless Communications and Mobile Computing*. pp. 1-9, 2020.

[21] A. Vishnoi, A. Aggarwal, A. Prasad, M. Prateek, and S. Aggarwal, "Image Encryption Using Homomorphic Transform," *Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT)*, Kannur, India, pp.1455-1459, 2022.

[22] Z. Muneef, H. bahjat, A. Abdulhoseen, "Image Encryption Paillier Homomorphic Cryptosystem," *Iraqi Journal Of Computers, Communications, Control And Systems Engineering*, vol. 21, No. 4, pp. 29-36. 2021.

[23] R. Hari Kishore, A. Chandra Sekhar, P. Patro, P. Chaganti, "A novel homomorphic and matrix operation for randomization encryption schemes for privacy in cloud computing architecture," *Journal of Theoretical and Applied Information Technology*, vol.101. No. 3. pp. 1038-1053, 2023.

[24] R. Bredehoft, "Encrypted Image Filtering Using Homomorphic Encryption, " *Blog /Tutorial*, Concrete ML. February 23, 2023.https: //www.zama.ai/post/encrypted-image-filtering-using-homomorphic-encryption (Accessed on 21-Jun-2023)

[25] Discrete cosine transform matrix (dctmtx). https://in.mathworks. com/help/images/ref/dctmtx.html# bvighg3. (Accessed on 21-Jun-2023).

Prabhavathi Krishnegowda (Corresponding author) is a research scholar.
Anandaraju M Boregowda is a Professor.
In the Department of ECE, BGS Institute of Technology, Adichunchanagiri University, B. G. Nagara, Karnataka, India.