Privacy-Preserving Image Filtering and Thresholding Using Numerical Methods for Homomorphically Encrypted Numbers

Sharmila Devi Kannivelu

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Electrical Engineering

University of Washington

2021

Committee:

Sunwoong S Kim, Chair

Kaibao Nie

Geetha Thamilarasu

Program Authorized to Offer Degree:

Electrical Engineering

University of Washington

Abstract

Privacy-Preserving Image Filtering and Thresholding Using Numerical Methods for Homomorphically Encrypted Numbers.

Sharmila Devi Kannivelu

Chair of the Supervisory Committee:
Sunwoong S Kim

Electrical Engineering

Homomorphic encryption (HE) is an important cryptographic technique that allows one to directly perform computation on encrypted data without decryption. In HE-based applications using digital images, a user often encrypts a private image captured on a local device. This image can contain noise that negatively affects the results of HE-based applications. To solve this problem, this thesis proposes a HE-based locally adaptive Wiener filter (HELAWF). For small-sized encrypted input data, pixels that have no dependency when sliding a window are encoded into the same ciphertext. For division in the adaptive filter, which is not supported by conventional HE schemes, a numerical approach is adopted. Image thresholding is a method of segmenting a region of interest and is used in many real-world applications. Typically, image thresholding contains a comparison operation, but this operation is not supported in conventional

HE schemes. To solve this problem, a numerical approach for comparison operation is used in the proposed HE-based image thresholding (HETH). The proposed HELAWF and HETH designs are integrated and implemented as a proof-of-concept client-server model. In practical HE schemes, the number of consecutive multiplications on encrypted data is limited. Therefore, the number of iterations of the numerical methods used in the integrated design is carefully chosen. To the best of the author's knowledge, this thesis is the first work that applies numerical division and comparison operation over encrypted data to image processing (IP) algorithms. The proposed solutions can address important privacy issues in IP applications in internet-of-things and cyber-physical systems, where many devices are connected through a vulnerable network.

Abbreviations

Advanced Encryption Standard ..…….……..……..…. (AES)

Brakerski/Fan-Vercauteren ……...…….. (BFV)

Cheon, Kim, Kim, and Song …….…….. (CKKS)

Computed Tomography ……...…… (CT)

Convolutional Neural Network …….…….. (CNN)

Cyber-Physical Systems ……..…… (CPS)

Discrete Fourier Transform ……..…… (DFT)

Fully Homomorphic Encryption …….. (FHE)

Homomorphic Encryption …… (HE)

HE-based Locally Adaptive Wiener Filter …… (HELAWF)

HE-based Thresholding …… (HETH)

Identity Theft Resource Center…….…….…… (ITRC)

Image Processing ……..…..……..……. (IP)

Internet of Things ……..…… (IoT)

Leveled Homomorphic Encryption …….. (LHE)

Locally Adaptive Wiener Filter …… (LAWF)

Magnetic Resonance Imaging ……...…… (MRI)

Mean Square Error ……..…….. (MSE)

Partially Homomorphic Encryption …….. (PHE)

Peak Signal-to-Noise Ratio ……..……. (PSNR)

Red-Green-Blue color space …….. (RGB)

Residue Number System …….. (RNS)

Simple Encryption Arithmetic Library…….…… (SEAL)

Somewhat Homomorphic Encryption ……. (SHE)

Y-Cb-Cr color space …….. (YCbCr)

Table of Contents

List of Table

List of Figures

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. Sunwoong S Kim, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. I would also like to thank Dr. Kaibao Nie and Dr. Geetha Thamilarasu for your great help in providing suggestions during my research and feedback for the improvement of this thesis.

Chapter 1: Introduction:

1.1 Motivation & Problem Statement:

In today's data driven world, security and privacy has become an important feature in any consumer device or application. With the rapid increase of IoT and smart devices, large amounts of data are being transferred over multiple connected systems in a network. Even simple day-to-day applications that are performed using smartphones involve substantial amounts of data transfer without the conscious knowledge of the user. For example, the Google photos app can create a collage from the photos taken or sharpen the blurry photos within seconds. Our simple handheld phones do not have the power or resources to handle such data processing. All this processing does not happen inside the smart phones but on the external cloud server where unlimited resources are available on demand. As shown in Figure 1, many stand-alone devices have limited power and resources and hence outsource large computations to external cloud servers. Cloud computing has become a cost-effective model to solve the problems of resource shortage in embedded devices.

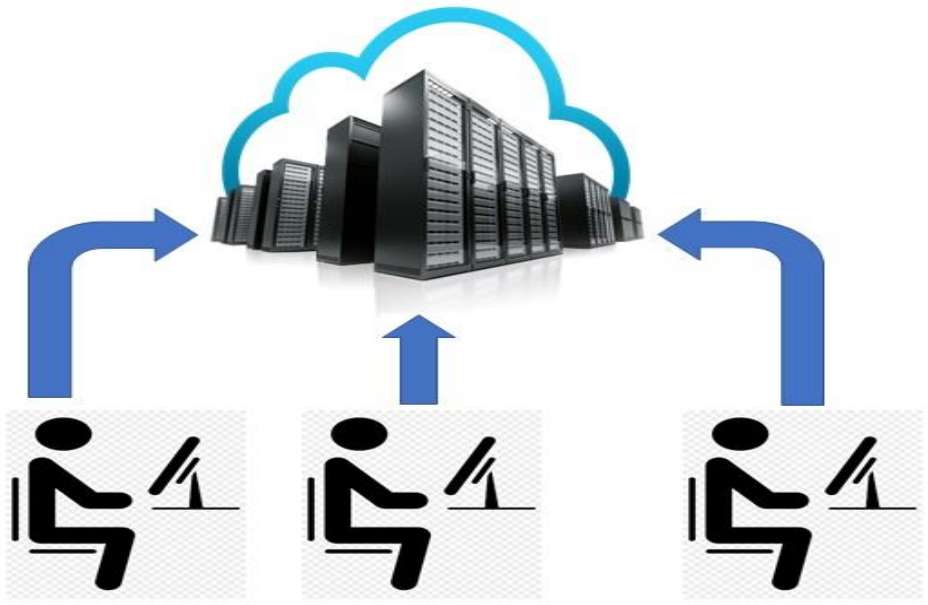


Figure 1: Data sharing in cloud computing.

Cloud computing is becoming a popular tool these days with its applications widely ranging from healthcare, education, genomics, entertainment, storage services, financial services, big data analysis, e-commerce, and e-governance. In healthcare, medical professionals and patients from all over the world can access and share information via cloud sharing. Genome sequencing, which helps to identify complex disease patterns, makes use of cloud resources. Cloud computing also provides services to government agencies such as smart grid power distribution and management. Big data analysis, involving large datasets of high velocity, uses the cloud resources to analyze and model the patterns from the data. All these examples show that cloud computing is becoming indispensable.

Despite the proliferating advantages of cloud computing there are several security concerns owing to large amount of data. In recent days several data breaches are happening across companies that handle cloud data. In particular, the year 2021 is set to break a record with 1291 breaches recorded till October by the ITRC [1]. Specifically in fields where sensitive data sharing and computations are involved, such as medical and financial fields, privacy concerns should be addressed. Tampering of such sensitive and confidential data can lead to serious problems for the parties involved. For instance, hacking the smart grid can cause disruptions to the power distribution and lead to failure of the grid. Similarly hacking financial data can lead to heavy financial losses for companies and individuals. In cases of a medical data breach, the patient's name, social security number and other critical details like insurance information and even images involved in the patient's treatment can be compromised.

To mitigate the privacy problems, secure cloud computing techniques are to be followed. Privacy-preserving data analysis and privacy-preserving data computations have become a pivotal solution to tackle these problems. With the advent of new cryptography techniques and

developments in secure hardware, privacy-preserving computations are becoming feasible. HE, Secure multi-party computation, Blinding, Differential privacy, Pseudonymization are different privacy-preserving computational methods that are currently used. Limited Disclosure technology and Anonymous credentials are some of other privacy enhancing methods that are currently being researched [2].

HE is a cryptographic technique that performs computations on encrypted data without decrypting the data first. The results are also in encrypted form which when decrypted will be equal (or similar) to results obtained from computations over unencrypted data. Since the data is processed in an encrypted domain, the cloud servers do not have access to the raw data. Thus, a secure data transfer and computation is possible in a connected network domain.

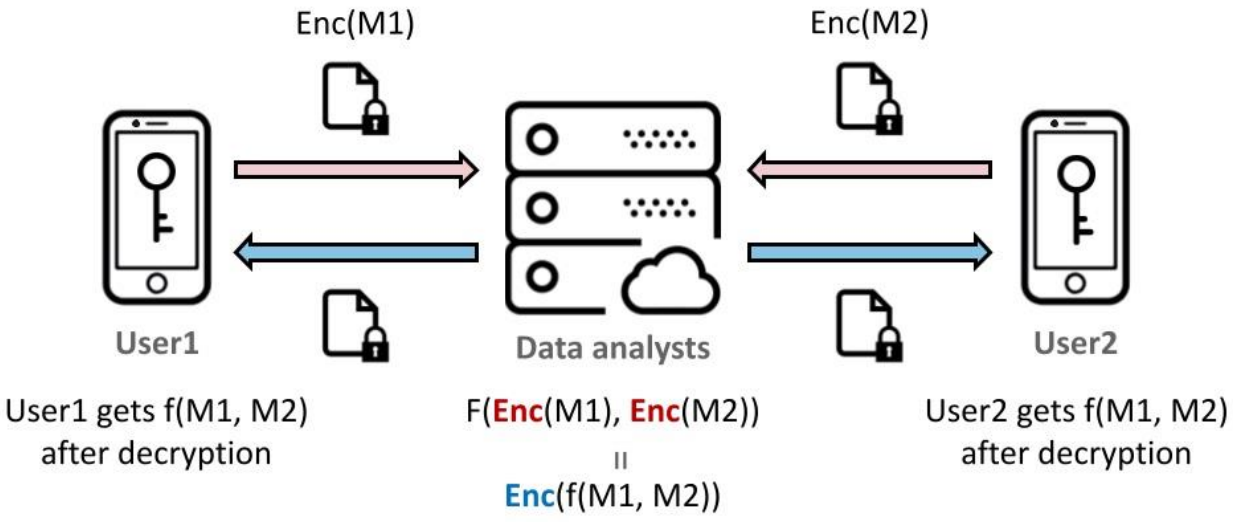


Figure 2: Data sharing in HE.

In this thesis, a system based on HE technique is proposed as shown in Figure 2. For example, two users need to analyze their messages M1 and M2 together but not separately over a cloud server. Both the users encrypt the messages and send them to the data analysts over cloud. The analysts perform some operations using both the encrypted data. The result of the operations will also be in encrypted form which is then sent back to the users. The users can then decrypt the result and get the actual result, which will be the same as (or similar to) performing the operation on the unencrypted message M1 and M2. In this method user 1 does not have access to message M2 and user 2 does not have access to message M1. But the combined computation results are available to both the users which can be used for further processing. For sensitive information analysis such as medical IP, HE can help to remove the privacy barriers and provide a secure way to perform computations.

1.2 Proposed System:

The medical and healthcare sector have numerous images that are being constantly analyzed and used in diagnosis. There are cases where the hospital may need to refer the images to specialized doctors or perform advanced processing for further prognosis. Also, there might be situations where machine learning models may need to be used, to predict the probability of diagnosis based on the IP results. The machine learning models cannot be applied on the local hospital machines since the models are trained using datasets that are collected from multiple hospital sources. The patient confidentiality clause prevents the hospital from using such services from third-party cloud servers. This thesis proposes a HE-based IP system that can be applied to situations, where cloud computing services can be used even in a privacy limited data sharing network.

Figure 3: Overall process flow of the proposed system.

The overall process flow of the proposed system is shown in Figure 3. A doctor or a client can encrypt an image, which makes the image pixels unreadable. This encrypted image is then sent to the cloud server via a public network. The server performs the IP algorithms on the encrypted image using HE methodologies. The resulting image, which is also in encrypted form, is then sent back to the client through the public network. The client then decrypts the image and analyses the results.

To encrypt and decrypt the image, the client uses special key codes called public keys and secret keys. These keys are not shared with the server and hence the server has no access to the unencrypted image data. Since the image data is encrypted during both the network transfers, tampering of data during transmission does not divulge any information on the image or patient details.

1.3 Thesis Structure:

- ★ This chapter, Chapter 1 provides the motivation for this research and briefly discusses the problem statement. A top-level overview of the proposed solution is also discussed.
- ★ Chapter 2 provides some background knowledge that is required to understand this thesis. The background information covers topics related to IP techniques such as image noise filtering and image segmentation, history of HE and the related terminologies and numerical methods that are used in HE.
- ★ Chapter 3 covers the proposed privacy-preserving image filtering. It discusses the required encoding methodologies for implementing the HE-based filtering algorithm.
- ★ Chapter 4 covers privacy-preserving image thresholding and the integration of HE-based filtering and HE-based thresholding.
- ★ Chapter 5 lists out the evaluation results of the proposed HE-based image filtering and thresholding designs and the final integrated design implemented as a client-server model.
- ★ Chapter 6 concludes this thesis and suggests future studies that can be carried out as an extension of this work.

Chapter 2: Background:

This chapter introduces background information required to understand the proposed work. A brief overview of IP techniques and HE is presented. The numerical methods which are used for the proposed HE-based IP designs are also introduced.

2.1 Image Processing:

One of the common data types for IoT and CPS systems is an image. In real world applications, images are considered to be two dimensional data denoted by notation *f(x, y)*. Here *(x, y)* denotes the coordinate position, and the magnitude of *f* at position *(x, y)* denotes the intensity at that point. A single image consists of many such intensity elements. These elements are called picture elements or *pixels* in short and form the basic building block of a digital image.

There are three basic ways in which an image can be manipulated - IP, Image Analysis and Computer Vision. IP refers to processes where the input and output are both images. Image analysis is different from IP in the sense that the input to a process will be an image and the output will be measurements from the image. Computer vision is an advanced technique, where the computers are able to make inferences based on the image analysis [3].

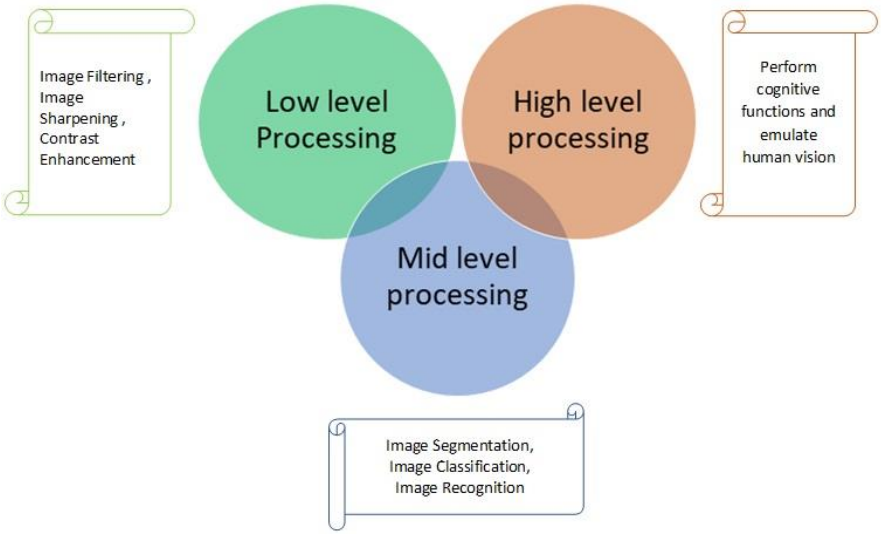


Figure 4: Image manipulation methods.

As shown in Figure 4, it is observed that there are no clear-cut boundaries and there is always an overlap between the processing methods. For example, when considering the case of medical images, capturing and preprocessing the image for any noise removal or contrast adjustments falls under the low-level IP. Segmenting the image for further diagnosis such as tumor detection or modeling the predictability of disease will fall under image analysis or computer vision depending on the complexity of the problem. In this research work scenarios of low- and mid-level IP techniques over encrypted domain are considered.

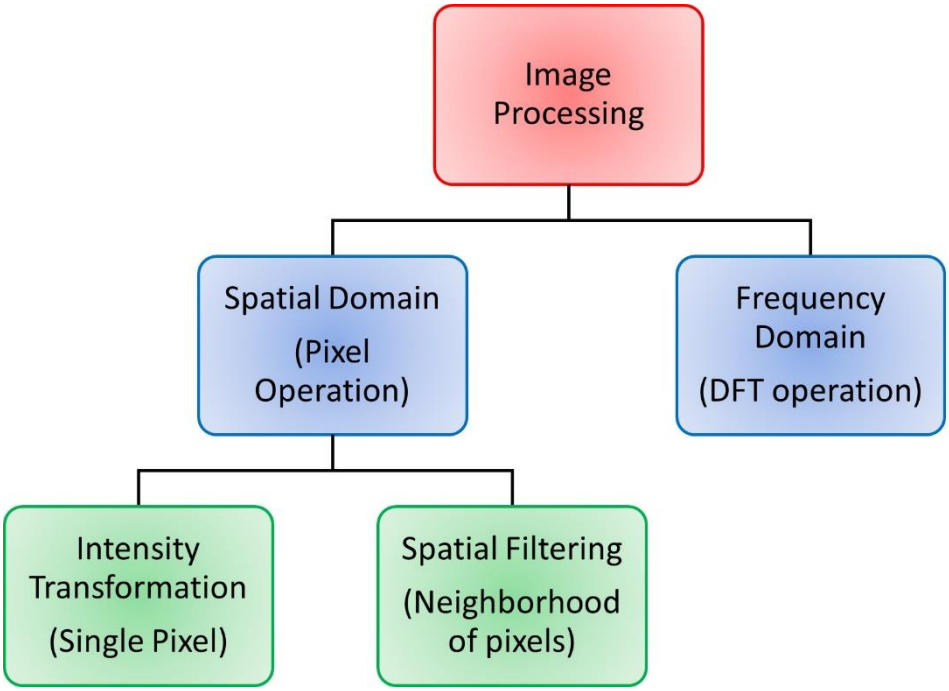


Figure 5: IP methods.

The two major IP methods are shown in Figure 5. In spatial domain processing, the operations are carried out on pixels directly whereas in the frequency domain analysis transforms such as DFT are used. IPs on spatial domain are further classified into two categories – Intensity Transformation and Spatial Filtering. Intensity transformation operations are usually single pixel operations without any dependency on the neighborhood pixels. Spatial filtering operates on a

window of neighborhood pixels to perform the pixel operations. Image thresholding is an example of intensity transformation whereas image filtering is an example of spatial filtering method. The proposed privacy-preserving system in this thesis provides a solution to implement both image filtering and image thresholding algorithms on the encrypted image data.

2.2 Image Filtering:

In HE-based IP applications, a client often encrypts a private image on a local device and sends it to a server where HE-based applications, also called homomorphic evaluations, are performed. If the image is taken with a camera sensor of the local device, inclusion of noise can lower the image quality and negatively affect the results of homomorphic evaluations. This problem can be solved by using HE-based image filtering methods.

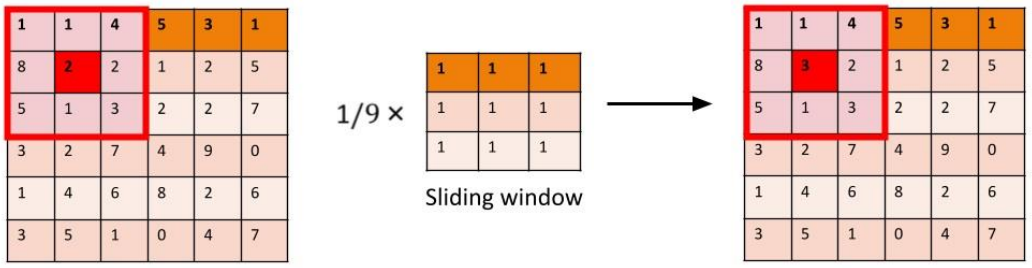
In practice, performing the filtering part on the client side is the best scenario because operations performed on unencrypted data are much faster than those on encrypted data. However, users generally do not want to perform tasks beyond en/decrypting and sending/receiving data. Also, they may lack the knowledge of what is important for the application or how to address the problem. Therefore, when providing services, it is better to perform image filtering on the server-side. The following is an introduction to popular image filtering algorithms.

2.2.1 Mean Filter & Median Filter:

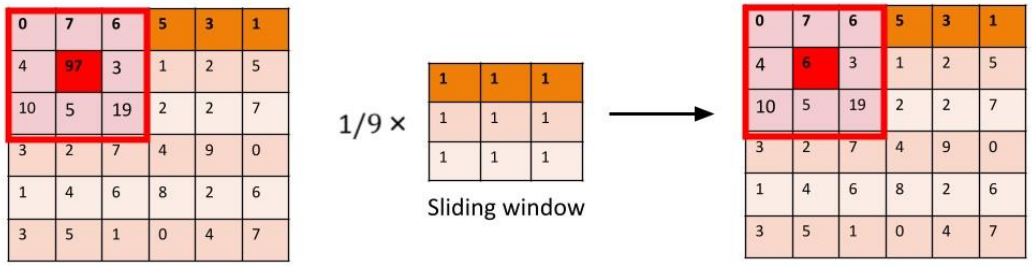
As mentioned in Chapter 2.1, spatial filters are widely used in image filtering [3] [4]. Mean filter is a linear spatial filter that replaces the pixels in a window with their average value. This effect reduces the irregular intensity variations in an image and produces a smoothing effect. This is very effective in reducing Gaussian noise and Uniform noise in an image. The blurring effect produced due to smoothing is determined by the window size.

Similarly, the median filter is a non-linear spatial filter that sorts pixels in a window according to their magnitude and uses a representative value, such as a median value, for the pixels. Median filters are better than mean filters since they produce less blurring effect for the same filter order. This filter is effective in cancelling extremely noticeable noise, such as salt-and-pepper noise. Figure 6-A and 6-B, shows the working of a mean filter and a median filter respectively.

Since the effectiveness of a filter depends on the noise type, it is advantageous to change the filter type or parameters based on the situation. Therefore, there have been many studies on adaptive filters that cope with various types of noise.



A. MEAN FILTER with 3x3 window



B. MEDIAN FILTER with 3x3 window

Figure 6: Working of mean and median filters.

2.2.2 Locally Adaptive Wiener Filter:

Adaptive filters consider the statistical characteristics of the image inside the window and determine the resulting pixel intensity [3] [4]. Typically, adaptive filters perform better than mean and median filters since they rely on the statistical parameters rather than a fixed value. But the computation complexity of the adaptive filter is higher compared to mean and median filters since statistical values need to be calculated at each step of the process.

Mean and variance are the two important statistical parameters. Any adaptive filters designed have their prime principle based on these statistical computations. Unlike the mean filter that uses constant filter weights, the weights of the adaptive filter are computed based on the local mean and local variance. For a 3 × 3 window, the local mean is computed as the mean of the pixels within the window. Similarly local variance is computed as the variance of the pixels within the selected window.

The Wiener filter is an adaptive filter that uses this statistical approach to estimate the original image by reducing the minimum mean square error. The formula for the well-known LAWF proposed by Lee [4] is shown in (1):

$$f(x,y) = m_l + \frac{\sigma_l^2}{\sigma_l^2+\sigma_o^2}\,(g(x,y) - m_l) \qquad (1)$$

where σ_l^2 , m_l, and σ_o^2 represent the local variance, local mean, and variance of overall noise, respectively. The local statistical values are calculated in a window, and the variance of overall noise is assumed to be a known constant. When the σ_l^2 value is close to 0, the adaptive Wiener filter works like the mean filter. On the other hand, if the σ_l^2 value increases, the resulting pixel intensity $f(x, y)$ has a value close to the original pixel intensity value, $g(x, y)$. Therefore, the LAWF

overcomes the blurring effect of the mean filter and gives a sharp resulting image. This research work implements the LAWF algorithm to remove noise in an image in the encrypted domain.

2.3 Image Thresholding:

Unlike image filtering which uses spatial filtering methods, image thresholding is an intensity transformation operation. The intensity transformation operations work on each individual pixel as opposed to windowing operations. Image thresholding plays a pivotal role in image segmentation analysis. The goal of image thresholding is to separate the image into groups of foreground and background. The foreground represents an area of interest in the image. All other regions excluding the area of interest form the background.

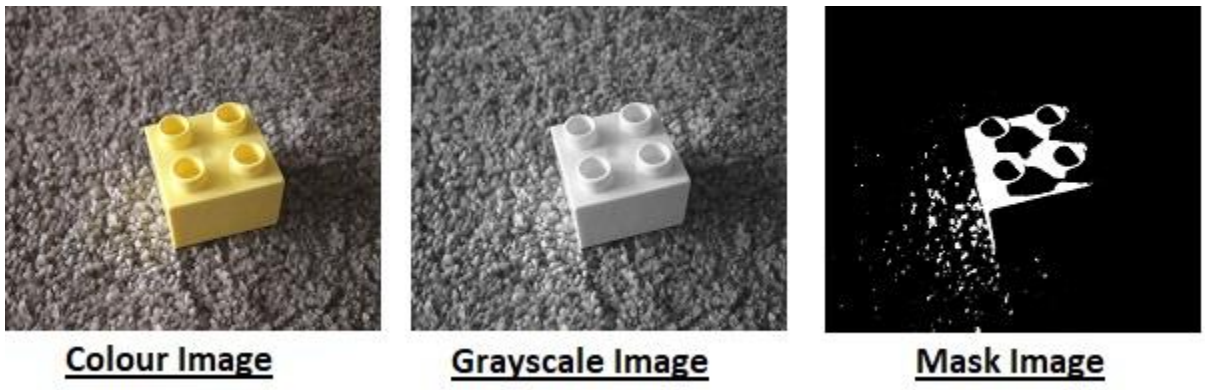


Figure 7: Example of image thresholding.

In Figure 7, which shows an example of thresholding, the yellow block is a region of interest, and the rest of the tiles are background. By performing thresholding, the background portions can be masked, and the region of interest is focused. Because of this property, the thresholded image is usually referred to as a mask image and by multiplying this mask image with the original image, the important features of the image can be highlighted. The level of masking is determined by the threshold value.

For color images, conversion into grayscale images are often performed in advance. Typically, a color image consists of 3 channels for red, green, and blue with each channel having pixel values ranging from 0 and 255. A grayscale image is a monochrome image and consists of a single channel with pixel values between 0 and 255. The result of thresholding will always be a binary image with a pixel value of '0' representing the background and a pixel value of '1' representing the foreground. This thresholding operation is usually denoted by the mathematical function shown in (2),

$$f(x,y) = \begin{cases} 1, & if\ g(x,y) > T \\ 0, & if\ g(x,y) \leq T \end{cases} \quad (2)$$

where *f(x, y)* represents the resulting pixel in the thresholded image, *g(x, y)* represents the pixel intensity in the grayscale image, and *T* represents the threshold value.

The selection of a threshold value is an important step in image thresholding. Typically, image thresholding methods are divided into two categories based on how a threshold value is selected, which is described in the following subchapters.

2.3.1 Global Thresholding:

When the selected thresholding value is constantly applied to the entire image, then the thresholding method is called global thresholding. Global thresholding is applicable to images where there is sufficient distinction in pixel intensities between foreground and background regions. Histogram method is commonly used to determine the global threshold value in such cases. Another popularly used method is the Otsu's method which identifies the threshold value using the statistical parameters [3]. This method has lesser computation time and is faster than the local thresholding method. In this research work global thresholding technique is used to create a mask image in an encrypted domain.

2.3.2 Local Thresholding:

A local thresholding method or an adaptive thresholding method, as the name implies, computes a different threshold for each pixel in an image [3]. This method is commonly used in images where there is non-uniform illumination or in images where foreground and background regions are not sufficiently distinct. This method is similar to spatial filtering where a window of neighborhood pixels is used to determine the threshold value for each pixel. The resulting binary image will have better mask characteristics than the global thresholding method, but the complexity is higher and hence computation time is longer than the global thresholding method.

In this thesis, a HE-based local thresholding method is not considered because a multiplicative depth, discussed later, is not sufficient.

2.4 Homomorphic Encryption:

HE, as specified earlier, is an encryption technique that allows computations on encrypted data without decryption. The results are also in encrypted form which when decrypted are the same as (or similar to) computations on unencrypted data. Homomorphism in Greek represents the same structure. In mathematics, homomorphic transformation is the property where one dataset can be transformed to another dataset, while still preserving the relationship between the elements in the dataset. Some of HE schemes are based on RNS [5]. This system is based on a set of *k* integers called the moduli (M), which are pairwise coprime. For example, an integer x, is represented in RNS [5] as a set of its reminders x_i as shown in (3),

$$\begin{gathered} x = \{x_1, x_2, \dots, x_{k-1}, x_k\}, \\ where \quad x_i = x \bmod m_i \quad \& \quad 0 \leq x_i \leq m_i \\ M = \{m_1, m_2, \dots, m_{k-1}, m_k\} \end{gathered} \tag{3}$$

In the RNS domain, the mathematical operations computed on the integer reminders will produce results similar to the computations performed on the original reminders. This property is made use of in HE schemes. But the limitation is that this property is not applicable for division and comparison operations. This limitation extends to the HE schemes, where division and comparison of encrypted data is not supported. Another major constraint of HE is that the number of operations that can be performed on encrypted data is limited. Once this limit is reached, further operations on the encrypted data do not retrieve correct results. This limited number of operations (especially multiplications) are denoted by the terminology called (multiplicative) circuit depth.

2.4.1 HE Process Flow:

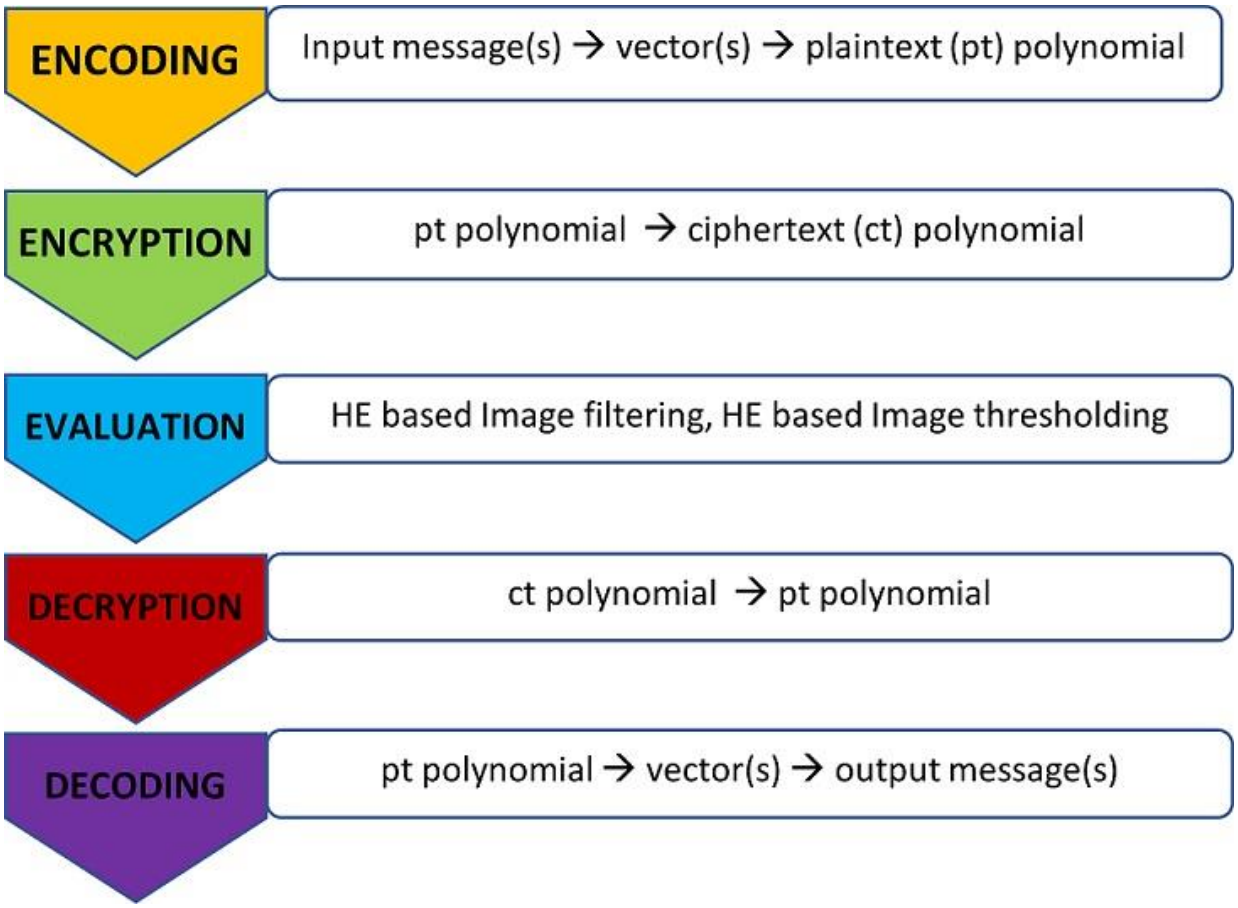


Figure 8: Overall process flow of a HE application.

The overall process flow of a HE-based application is shown in Figure 8. The initial step is encoding, where input messages are converted to a vector and further to a plaintext polynomial (pt). Encryption is the next step which converts a plaintext polynomial to a ciphertext polynomial (ct). Evaluation is the main step where HE-based processing is performed. In this thesis, image filtering and image thresholding are HE-based evaluations. The next step is decryption where the evaluated ciphertext is converted back to a plaintext polynomial. Decoding is the final step where the resulting plaintext polynomial is converted to an output vector and further to an output message.

The term plaintext refers to any message or information that is readable by a human or a machine. The term ciphertext refers to the encrypted message and it is not decipherable without the secret key or cipher.

2.4.2 Types of HE:

Different types of HE [6] are available based on the type and number of operations that can be performed on the encrypted data. Partially HE (PHE) allows only one type of mathematical operation to be performed on the encrypted data, i.e., either addition or multiplication. Somewhat HE (SHE) allows both operations addition and multiplication for a limited set of circuits. Leveled HE (LHE) is an extension of SHE, since it allows multiple operations for a predetermined circuit depth. Fully HE (FHE) is the most powerful encryption that allows both operations multiple times without any limitations on circuit depth. However, it includes additional step called bootstrapping that requires extremely high computational complexity to reset the circuit depth. Therefore, in this thesis an LHE scheme with a predetermined circuit depth is used.

Several generations of HE are currently available. Among these the BFV and CKKS schemes are the most popular. The BFV scheme does computations only on integer data whereas the CKKS scheme operates on floating-point data and complex numbers but with limited precision [6] [7]. This implies that the CKKS scheme yields approximate but close results instead of exact results. Since the intermediate results in image filtering and thresholding are of floating-point data type, the CKKS scheme is used in this thesis.

2.4.3 HE Libraries & Parameters of the CKKS Scheme:

Many open-source HE libraries are currently available. Table 1 lists some of them along with the HE schemes supported by them [6]. The SEAL library developed by Microsoft is used in this thesis. The reason for using it over other HE libraries is that the Microsoft research team is a leading research team in HE researches and standardization. The source code of SEAL libraries is well maintained and updated with the newest techniques periodically, which helps use it with ease. HE functions in this library are stable (and conservative), which is good for beginners.

Table 1: List of HE libraries

HE Library	**Developer**	**BFV**	**CKKS**
Helib	IBM	Yes	No
Microsoft SEAL	Microsoft	Yes	Yes
PALISADE	Consortium of DARPA-funded defense contractors and academics: NJIT, MIT, UCT and others	Yes	Yes
HEAAN	Seoul National University	No	Yes

As mentioned in Chapter 2.4.2, the major constraint in an LHE scheme is the circuit depth. In the SEAL library, the number of consecutive multiplications that can be performed on each ciphertext is limited [7]. The addition and subtraction operations do not consume circuit depth, but

still, they contribute to the noise factor in the encrypted result. HE parameters must be selected based on the required circuit depth. Table 2 lists the two important HE parameters that are considered in the CKKS scheme, which has a direct impact on determining the circuit depth.

Table 2: Parameters of the CKKS scheme that determine the circuit depth [7]

Polynomial degree (N)	N – should be a power of 2	**N should be large for a large circuit depth**
Coefficient modulus bit count (q) (or just bit-length of q) $q = q_1 \times q_2 \times \ldots \times q_k$	Constructed by a product of distinct prime numbers	$\boldsymbol{k-2}$ **= circuit depth**

The parameter N denotes a polynomial degree that is a degree of the polynomial ring. The N value should always be a power of 2 and it should be increased as the required circuit depth increases. Hence in case computations on encrypted data require more consecutive multiplications, N should be chosen to be a higher value. The maximum possible N value permitted by the SEAL library is 2^{15} [7].

The second parameter q, called coefficient modulus bit count, is constructed by the product of k distinct prime numbers. Here each number is allotted a specific number of bits – and the sum of all the bits is the overall coefficient modulus bit count. Each ciphertext multiplication consumes a prime modulus. In SEAL, the total bit counts are predetermined for a given N value and level of security. For instance, for the maximum N value of 2^{15} and a 128-bit security level, the predetermined q value is 885. This implies that 'k' numbers should be selected such that the sum of the total bits is less than or equal to 885. Once the 'k' value is fixed, the exact circuit depth is determined as k-2 in SEAL (k-1 in the original CKKS scheme). When the security level decreases, the coefficient bit count decreases which in turn decreases the circuit depth.

2.4.4 HE Encoding/Decoding Example:

In the SEAL library, both *plaintext* and *ciphertext* are in the form of polynomials, with the message being the coefficients of the polynomial equation. Specifically, in the CKKS scheme, which is the target HE scheme of this thesis, each ciphertext is considered as an array of the size equal to N/2 [7], where N represents the polynomial degree as mentioned in Chapter 2.4.3. If the N value is 2^{15}, the ciphertext has 16,384 slots, which means that a maximum of 16K values can be stored in this ciphertext.

The encoding process packs messages in an input vector into slots of the same plaintext polynomial. If the vector consists of a single element such as a constant value, then all the slots of the plaintext can be filled with the same value. These slots in the plaintext will correspond to the respective slots in the ciphertext. Any operation performed on this ciphertext is the same as performing the operation on 16K slot elements of this ciphertext in parallel. This helps us to process multiple elements together at the same time.

Figure 9 shows a simple example that adds two numbers that are encrypted homomorphically. Let us consider two encrypted numbers of M1 = 5 and M2 = 3 to be added. The first step is to convert both M1 and M2 to plaintext polynomials. These plaintext polynomials can be considered as arrays or vectors of size N/2, where N represents the polynomial degree. Plaintext P1 holds the value of 5 in all the slots. Similarly, plaintext P2 holds the value of 3 in all the slots. The next step is encryption, where the plaintext polynomials get converted to the ciphertext polynomials. Each ciphertext is similar to the plaintext with the same number of slots. The ciphertext C1 holds the encrypted value of 5 in all the slots and ciphertext C2 holds the encrypted value of 3 in all the slots. Now to compute any function *f* on C1 and C2, the relationship between the numbers 5 and 3 are maintained. Specifically, when encrypted 5 and encrypted 3 are added to

each other, using homomorphic library functions, the result obtained is an encrypted value of 8. This encrypted result when decrypted will give us a plaintext with values of 8 being filled in all slots. Decoding on the plaintext will give the result 8, which is the same as adding unencrypted numbers of 5 and 3.

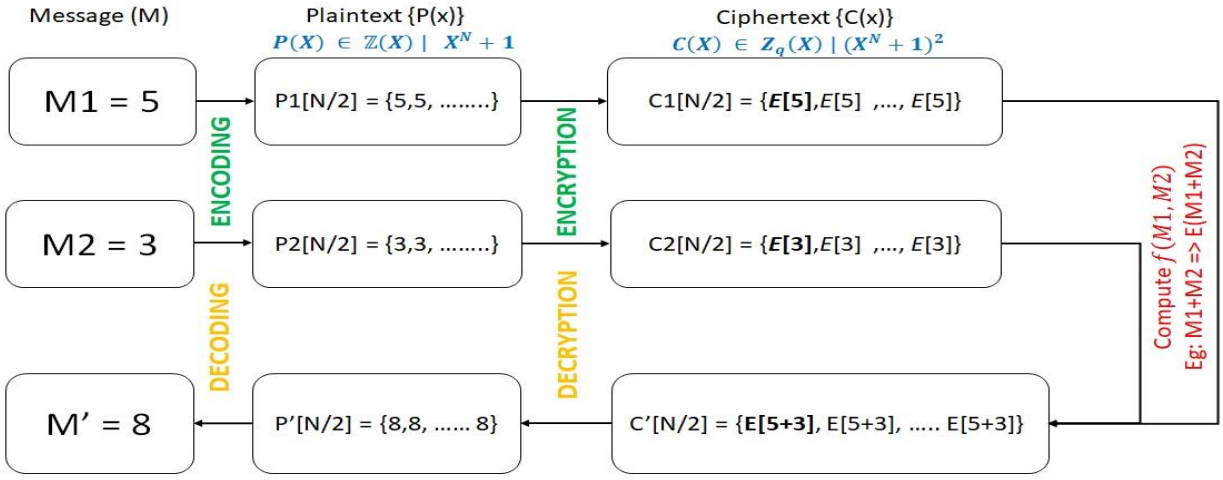


Figure 9: A simple example of addition of two numbers in the HE domain.

2.4.5 Key Generation:

HE libraries provide special functions to generate keys that are used in encryption and decryption. Other than the encryption and decryption keys, special keys such as Relin keys and Galois keys are also generated by the key generator function. Table 3 shows the generated keys.

Table 3 : List of keys generated by the key generation function

Function: *KeyGen (Param)* - generates keys for a given set of parameters.		
Keys used by en/decryption functions	**Public Key (pk)**	Converts a plaintext polynomial to a ciphertext polynomial (encryption)
	Secret Key (sk)	Converts a ciphertext polynomial back to a plaintext polynomial (decryption)
Keys used by mathematical functions	**Relin Key (rk)**	Reduces the ciphertext size after each ciphertext multiplication
	Galois Key (gk)	Rotates slots inside a ciphertext

2.4.6 HE Functions & Operations:

In addition to the key generation function, HE libraries also provide functions to encrypt and decrypt message data. For better clarity, these functions are referred to as processing functions and shown in Table 4.

Table 4: Basic processing functions provided by HE libraries

Encode (input vector)	packs multiple input messages into a plaintext polynomial message *'m'*
Encrypt $_{pk}$ (m)	encrypts a plaintext message *'m'* using the public key *'pk'*
Decrypt $_{sk}$ (ct)	decrypts a ciphertext *'ct'* using the secret key *'sk'*
Decode (plaintext)	unpacks the resulting plaintext polynomial into an output message vector

HE libraries also provide mathematical functions shown in Table 5, to perform computations on both ciphertexts and plaintext polynomials. Certain mathematical operations such as multiplication increase the size of the ciphertext polynomial. In such cases, it is necessary to reduce the ciphertext size back to the original form. This method of reduction in ciphertext size is called relinearization and it requires a special key called Relin keys. It should be noted that only ciphertext-to-ciphertext multiplication increases the size. Plaintext-to-ciphertext multiplication does not increase the size and hence relinearization is not required after this operation. Since the ciphertext consists of slots and can store different values in each slot, slot rotation can be useful for some operation such as convolution. This slot rotation is performed by the special keys called Galois keys.

Table 5: Mathematical functions provided by HE libraries

HomAdd (ct0, ct1)	Adds two ciphertexts. Subtraction between ciphertexts is a variant of this function.
HomMult $_{rk}$ (ct0, ct1)	Multiplies two ciphertexts. After each operation, relinearization is performed using the Relin keys to match the size of the resulting ciphertext to the size of the original ciphertext.
HomMultPlain (ct, m)	Multiplies a ciphertext *ct* with a plaintext message *m*. The result is also a ciphertext, but its size is maintained without relinearization.
HomRot $_{gk}$ (ct)	Rotates slots in a ciphertext 'ct' by a step size '*s* ' specified by the Galois keys. This function does not increase the ciphertext size.

2.5 Numerical Methods for HE:

In general, conventional HE schemes do not support every general arithmetic operations. For instance, there is no function to support division of ciphertexts. If a divisor is constant, division can be performed by taking a multiplicative inverse and then using the HomMultPlain function. But in cases where a divisor is constantly changing as in window-based filtering, there is no existing method to perform division over ciphertexts. Similarly, there are no available HE functions to compare two ciphertexts.

2.5.1 Inverse Algorithm:

To solve the division problem, Cheon *et al.* applied a popular numerical technique for multiplicative inverse operation called the Goldschmidt's algorithm [8] to the CKKS scheme [9]. This algorithm calculates the inverse of a number by an iterative process, where both dividend and divisor are multiplied by a common factor. The inverse of an input real number $x \in (0, 2)$ is calculated as by (4)

$$1/x = 1/(1-(1-x)) \approx \prod_{i=1}^{d}\left(1+(1-x)^{2^{i-1}}\right), \qquad (4)$$

where d refers to the number of iterations. In general, as the number of iterations increases, the result will be closer to the value of ideal *1/x*.

The algorithm to compute an inverse using the Goldschmidt algorithm is shown in Table 6 [8]. This algorithm is used to perform division over ciphertexts in this thesis. Specifically, an inverse of the changing divisor of LAWF in equation (1), described in Chapter 2.2.2, is calculated by using this algorithm.

Table 6: Algorithm to find an inverse of a number using the Goldschmidt's algorithm

Input	x ϵ (0,2); d ϵ N;
Output	An approximate value of *1/x*
1.	a[0] ← 2 – x
2.	b[0] ← 1 - x

3.	for (n=0; n<d; n++) do
	b[n+1] ← b[n] × b[n]
	a[n+1] ← a[n] × (1 + b[n+1])
	end for

4	return a[d]

2.5.2 Comparison Algorithm:

To solve the comparison problem, Cheon *et al.* initially applied an iterative approach to evaluate a rational function which is approximately equal to the comparison function [9]. But this method uses the inverse algorithm introduced in chapter 2.5.1, and hence the computational

complexity becomes high. This method does not achieve optimum efficiency since at least ‘d’ number of iterations must be performed for the inverse function to achieve a closely approximated result. This inverse is further used in an outer loop that increases the number of iterations significantly. With the limited circuit depth that is available in the SEAL library, this method could not be implemented with the acceptable results.

To reduce the computational complexity, a modified algorithm proposed by Cheon *et al.* [10] is used in this thesis. The modified comparison algorithm is based on the approximation of the “sign” function. This new algorithm approximates the compare function between two numbers ‘a’ and ‘b’ to a sign function as shown in (5):

$$comp\ (a, b) = (sgn(a - b) + 1)/2 \qquad \textbf{(5)}$$

The values of the inputs (a, b) should be within the range of 0 and 1. The algorithm returns an approximate value of ‘1’ when the value of ‘a’ is greater than ‘b’ and an approximate value of ‘0’ when the value of ‘a’ is less than the value of ‘b’. The sign function is represented with a complex polynomial equation shown in (6).

$$sgn(x) \approx f_n^d(x), \qquad where\ x = a - b$$

$$f_n(x) = \sum_{i=0}^{n} \frac{1}{4^i} \cdot \binom{2i}{i} \cdot x(1 - x^2)^i \qquad (6)$$

The algorithm to numerically perform comparison operation of two numbers is shown in Table 7 [10]. There are two parameters for iterative loops. One parameter is the ‘d’ in the outer loop and the other is the ‘n’ used for sgn(x) calculation. Both parameters need to be optimized to achieve a closely approximated value of ‘1’ and ‘0’ during comparison operation.

Table 7: Algorithm to compare two numbers using the numerical method

Input	(a,b) ϵ [0,1]; n, d ϵ N;
Output	An approximate value of 1, if a>b An approximate value of 0, if a<b An approximate value of ½, if a=b
1.	x ← a - b

2.	for (i=1, i<=d, i++) do
	x ← $f_n(x)$, where $f_n(x)$ = sgn (x)
	end for

3.	return ((x+1)/2)

2.6 Related Works:

In recent days, various studies using HE for privacy-preserving IPs have been carried out. For example, Ziad *et al*. developed CryptoImg, a modular library that provides IP operations for encrypted images [11]. The focus of this work was to implement low-level IP tasks using HE functions with minimum overhead. In this work, spatial filtering is performed by using a mean filter, where plaintext-ciphertext multiplication is performed for division. There is no option to implement an advanced filtering algorithm that uses a locally changing divisor.

Fu, Lin, and Inge proposed a HE-based method for image resizing and image compression/decompression [12]. This work also uses the Microsoft SEAL to implement the HE evaluations, but the BFV scheme for integers is used. Their main focus is to perform image compression, by transferring images from RGB color space to YCbCr color space. Since this conversion mostly uses additions and subtractions, they do not use any multiplicative circuit depth.

Another work uses HE to implement image addition and brightness adjustment of encrypted images by using learning with errors methodology [13]. Here the addition of images is performed on binary images, and for brightness adjustment grayscale images are used. The

addition of images uses additions on ciphertexts, which is a straightforward implementation. For brightness manipulation, a constant value is multiplied to all the pixels. This implementation uses a single multiplication operation and does not involve a large circuit depth.

Although many works have tried to implement IP in an encrypted domain, no work performs division or comparison operation on encrypted data. Most of the works that use HE to perform privacy-preserving IP target simple IP algorithms including only addition/subtraction or multiplication with a constant, which consumes a small circuit depth. In real-world applications, many IP algorithms require complex processing steps involving division and comparison operation of pixels. This thesis presents a potential solution to this issue.

Chapter 3: Privacy-Preserving Image Filter:

In this chapter, a novel encoding method for a small number of images is proposed. This encoding method is optimized for window-based filtering and increases the slot utilization significantly. The proposed HE-based LAWF (HELAWF) method is then presented. Note that this chapter is part of my earlier work [14].

3.1 Encoding Images for HE Filtering:

As mentioned in Chapter 2.2, image filtering algorithms require some mathematical operations to be performed on multiple pixels. For example, the 3×3 mean filter calculates the average value of nine pixels in a 3×3 window. When implementing it in the HE domain, these nine pixels should be added individually. If these nine pixels are encoded and encrypted into the slots of the same ciphertext, each pixel intensity cannot be extracted from this ciphertext without decryption and decoding, and therefore averaging on the encrypted nine pixels is impossible.

The most intuitive solution is to encode and encrypt each pixel in a separate ciphertext. For example, a grayscale image of the size 28×28 has 784 pixels and thus requires 784 ciphertexts. This number may be acceptable, but if the image resolution increases to HD or UHD, the number of ciphertexts increases significantly. This becomes very inefficient and impractical because an operation on each ciphertext takes a long time. In addition, huge transmission bandwidth between devices and/or memory size for ciphertexts are required. Specifically, it is fatal to embedded system platforms that have limited resources/power and real-time constraints.

In fact, many studies encode thousands to tens of thousands of images together to overcome this inefficiency and show good, amortized running time [9], [15], [16]. For example, pixels at (0, 0) in tens of thousands of images are encoded into the slots of the same ciphertext. However, in

real-world applications, processing only one or a few images may be more frequent than processing thousands to tens of thousands of images, which leaves most of the slots unused.

Figure 10 shows an encoding example that uses a 3×3 filter window for a 128×128 grayscale image. It should be noted that with the most intuitive method that encodes each pixel into a separate ciphertext this image requires 16,384 ciphertexts. When the center of the 3×3 window is placed at (1, 1), the nine pixels in this window should be placed in the same slot but with different ciphertexts, such as Slot 0s of Ciphertexts 0-8, for slot-wise operations.

Image matrix:

	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	…	Col 127
Row 0	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	…	(0, 127)
Row 1	(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	…	(1, 127)
Row 2	(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	…	(2, 127)
Row 3	(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	…	(3, 127)
Row 4	(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	…	(4, 127)
…	…	…	…	…	…	…	…	…
Row 127	(127, 0)	(127, 1)	(127, 2)	(127, 3)	(127, 4)	(127, 5)	…	(127, 127)

Ciphertext with slots:

	Slot 0	Slot 1	…	Slot 41	Slot 42	Slot 43	Slot 44	…	Slot 84	Slot 85	…
Ciphertext 0	(0, 0)	(0, 3)	…	(0, 123)	(0, 126)	(3, 0)	(3, 3)	…	(3, 123)	(3, 126)	…
Ciphertext 1	(0, 1)	(0, 4)	…	(0, 124)	(0, 127)	(3, 1)	(3, 4)	…	(3, 124)	(3, 127)	…
Ciphertext 2	(0, 2)	(0, 5)	…	(0, 125)	NOT USED	(3, 2)	(3, 5)	…	(3, 125)	NOT USED	…
Ciphertext 3	(1, 0)	(1, 3)	…	(1, 123)	(1, 126)	(4, 0)	(4, 3)	…	(4, 123)	(4, 126)	…
Ciphertext 4	(1, 1)	(1, 4)	…	(1, 124)	(1, 127)	(4, 1)	(4, 4)	…	(4, 124)	(4, 127)	…
Ciphertext 5	(1, 2)	(1, 5)	…	(1, 125)	NOT USED	(4, 2)	(4, 5)	…	(4, 125)	NOT USED	…
Ciphertext 6	(2, 0)	(2, 3)	…	(2, 123)	(2, 126)	(5, 0)	(5, 3)	…	(5, 123)	(5, 126)	…
Ciphertext 7	(2, 1)	(2, 4)	…	(2, 124)	(2, 127)	(5, 1)	(5, 4)	…	(5, 124)	(5, 127)	…
Ciphertext 8	(2, 2)	(2, 5)	…	(2, 125)	NOT USED	(5, 2)	(5, 5)	…	(5, 125)	NOT USED	…
	for Rows 0, 1, 2					for Rows 3, 4, 5					…

Figure 10: An example of the proposed encoding method using a 3×3 filter window for a 128×128 image.

When the 3×3 window moves right by 1, the pixels at (0,1), (0, 2), (0, 3), (1, 1), (1, 2), (1, 3), (2, 1), (2, 2), and (2, 3) are included in this window. These pixels cannot be mapped to the same ciphertext. However, the previous pixels at (0, 0), (1, 0), (2, 0) are never included in the 3×3

window with the pixels at (0, 3), (1, 3), and (2, 3). In other words, Column 0 and Column 3 of the image are independent. Therefore, the pixels at (0, 3), (1, 3), and (2, 3) are mapped to Slot 1s of Ciphertexts 0, 3, and 6. Similarly, the pixels at (0, 4), (1, 4), and (2, 4), which are independent from the pixels at (0, 1), (1, 1), and (2, 1), are mapped to Slot 1s of Ciphertexts 1, 4, and 7.

In the same way, the pixels at (0, 5), (1, 5), and (2,5) are mapped to Slot 1s of Ciphertexts 2, 5, and 8. With this method, nine ciphertexts are used for each three rows, and 43 slots of each ciphertext are used. Note that Slot 42s of Ciphertexts 2, 5, and 8 are unused. Considering 128 rows, 384 ciphertexts are required, which is a 98% reduction compared to the number of ciphertexts by the most intuitive method.

Further optimization is achieved by repositioning pixels on other rows into the same ciphertext. For instance, when the center of the 3×3 window is located at (2, 1), the pixels at (3, 0), (3, 1), and (3, 2) are included in this window with the pixels at (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), and (2, 2) but independent from the pixels at (0, 0), (0, 1), and (0, 2). Hence, the pixels on Row 3 are mapped to Ciphertexts 0-2 that are originally for Row 0 only. Similarly, the pixels on Row 4 and Row 5 are mapped to Ciphertexts 3-5 and Ciphertexts 6-8, respectively. By extending this method to all rows, a total of nine ciphertexts using 1,849 slots are required for a 128×128 grayscale image.

When the proposed encoding method is generalized for an image size C × R and a window size W × W, the number of ciphertexts c is calculated by using (7).

$$c = \left\lceil \frac{\lceil R/W \rceil \times \lceil C/W \rceil}{N/2} \right\rceil \times W^2, \qquad (7)$$

where N is the polynomial degree, which is one of the HE parameters and explained in Chapter 2.4.3. Table 8 shows the number of ciphertexts required for different image sizes using the proposed encoding method. For the sake of simplicity, the W and N values are fixed at 3 and 2^{15}, respectively. Due to the large N value, 128×128 and 256×256 images have many unused slots. To increase the utilization of slots, seven 128×128 images and a 256×256 image is additionally included in the existing ciphertexts. The last row shows the maximized slot utilization. About 90% of the slots are used for a small number of images.

Table 8: The number of ciphertexts per image and utilization depending on the image size when W=3 and N=2^{15}

Image Size	128 x 128	256 x 256	512 x 512	1024 x 1024
Ciphertexts/Image	9/8	9/2	18/1	72/1
Utilization (%)	90.28	90.28	89.24	89.24

3.2 HE Image Filtering Algorithms:

Before presenting the proposed HELAWF method, how to move the mean filter to the HE domain is introduced for comparison. It is straightforward to perform the mean filter on ciphertexts. First, pixels in the image are encoded as explained in Chapter 3.1. The first step is to calculate the sum of encrypted pixel intensity values in a window using HomAdds. The second step is to divide the sum that is still encrypted by the total number of pixels in the window. Since the divisor is an integer constant and the CKKS scheme supports arithmetic on real numbers, the inverse of the divisor is encoded in advance and multiplied to the encrypted sum using HomMultPlain. The algorithm for the mean filter window in the HE domain is shown in Table 9.

Table 9: Algorithm of the HE-based mean filter (for each window)

Input		g(x, y) - source pixel intensity in a n×n window ($0 \le x < n$, $0 \le y < n$)
Output		f(x, y) - filtered pixel intensity
1.	Find a sum of pixels in the window	$M_L \leftarrow$ HomAdds (Enc(g(0, 0)), …, Enc(g(n-1, n-1)))
2.	Find the local mean	$M_L \leftarrow$ HomMultPlain (M_L, $1/n^2$)
3.	Apply the mean to the resulting pixel intensity	$f(x, y) \leftarrow M_L$ (for $0 \le x < n$, $0 \le y < n$)

Unlike the mean filter, division in LAWF uses a locally computed variance as a divisor that changes as a window slides over an image. Therefore, HomMultPlain with a constant divisor cannot be used. For division over ciphertexts, the previous numerical approach introduced in chapter 2.5.1 is applied to the proposed HELAWF. The algorithm for the proposed HELAWF is shown in Table 10.

Table 10: Algorithm of the proposed HELAWF method (for each window)

Input		g(x, y) - source pixel intensity in a n×n window ($0 \le x < n$, $0 \le y < n$)
Output		f(x, y) - filtered pixel intensity
1.	Find a sum of pixels in the window	$M_L \leftarrow$ HomAdds (Enc(g(0, 0)), …, Enc(g(n-1, n-1)))
2.	Find the local mean	$M_L \leftarrow$ HomMultPlain (M_L, $1/n^2$)
3.	Find the mean square	$M_L^2 \leftarrow$ HomMult (M_L, M_L)
4.	Find a sum of squares of pixels in the window	for $0 \le x < n$ and $0 \le y < n$, a0 ← HomMult (Enc(g(x, y)), Enc(g(x, y))) Sum ← HomAdd (Sum, a0)

5.	Find the mean of the square sum	b0 ← HomMultPlain (Sum, $1/n^2$)
6.	Find the local variance	V_L ← HomSub (b0, M_L)
7.	Find the denominator by adding the variance of overall noise to the local variance	d0 ← HomAdd (V_L, V_0)
8.	Find the inverse of the denominator by using the Goldschmidt's algorithm and perform multiplication	T1 ← HomMult (V_L, Inv (d0))
9.	Apply the rest of the formula	T2 ← HomSub (g(x,y) - M_L)
		T3 ← HomMult (T1, T2)
		f(x, y) ← HomAdd (M_L, T3)

Besides HomAdd and HomMults, the proposed HELAWF requires HomRots for the proposed encoding method presented in Chapter 3.1. For example, as shown in Figure 11-A, to calculate the filtered pixel intensity value at (2, 1), the 3×3 window includes the pixels at (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 0), (3, 1), and (3, 2). However, the pixels at (3, 0), (3, 1), and (3, 2) are stored in the 43rd slots of Ciphertexts 0-2, respectively. To perform a slot-wise operation, the slots in Ciphertexts 0-2 are left-rotated by the step size of 43. Similarly, as shown in Figure 11-B, to compute the filtered pixel intensity value at (1, 2), the 3×3 window includes pixels at (0, 1), (0, 2), (0, 3), (1, 1), (1, 2), (1, 3), (2, 1), (2, 2), and (2, 3). This time, the pixels at (0, 3), (1, 3), and (2, 3) are stored in the first slots of Ciphertext 0, Ciphertext 3, and Ciphertext 6, respectively. Therefore, the slots in these three ciphertexts are left-rotated by the step size of 1. For 128 × 128 images, the step sizes of 1 and 43 are only required to compute all the filtered pixel intensity values.

The same principle can be used for a larger image size. For example, for 256 × 256 images, the step sizes of 1 and 86 are only required.

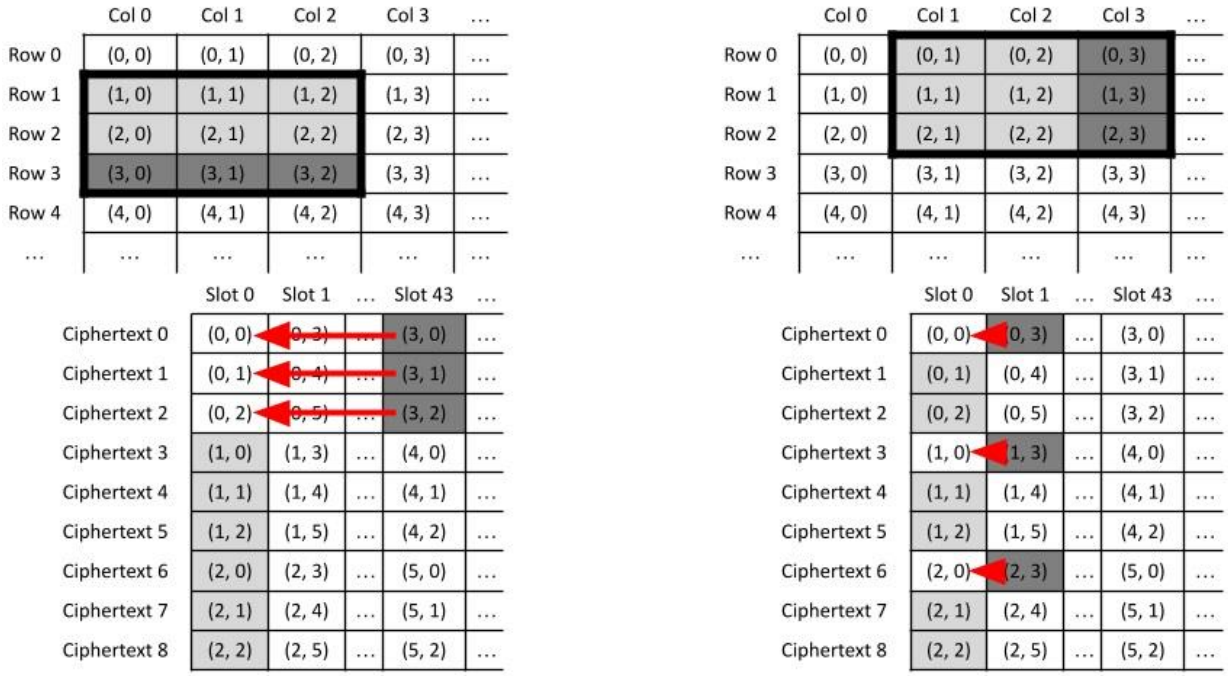


Figure 11: Slot rotation in ciphertexts for a 3×3 window and 128×128 images. (a) Step size of 43 (b) Step size of 1

The default KeyGen in SEAL creates Galois keys for $2 \times log_2 N - 1$ rotations to support arbitrary rotation amounts [8]. With the above-mentioned optimization that requires only two rotation amounts, the memory footprint for Galois keys is reduced from 3.5 GB to hundreds of MB.

Chapter 4: Privacy-Preserving Image Thresholding and Integration with Image Filtering:

The first part of this chapter proposes the proposed privacy-preserving image thresholding method. The second part covers the integrated design of the proposed privacy-preserving filtering and thresholding methods.

4.1 Privacy-Preserving Thresholding:

The proposed HE-based image thresholding (HETH) method is presented in this subchapter. It mainly targets medical images for which image thresholding is widely used. Since medical images are usually taken with uniform illumination, a global thresholding method gives sufficient results. In addition, a local thresholding method has higher computational complexity, even if the depth is excluded, a local thresholding method is not considered in this thesis.

For the proposed HETH method, the proposed encoding method shown in Chapter 3.1 is still used. For example, only nine ciphertexts are required for eight 128×128 grayscale images. Figure 12 shows an example of how HETH with the proposed encoding method works. Since a numerical comparison operation is used, the resulting mask values are not exact 1 and exact 0. To increase the accuracy (to get results closer to 0 and 1), the numbers of iterations in the proposed HETH method should be increased, which in turn consumes a large depth.

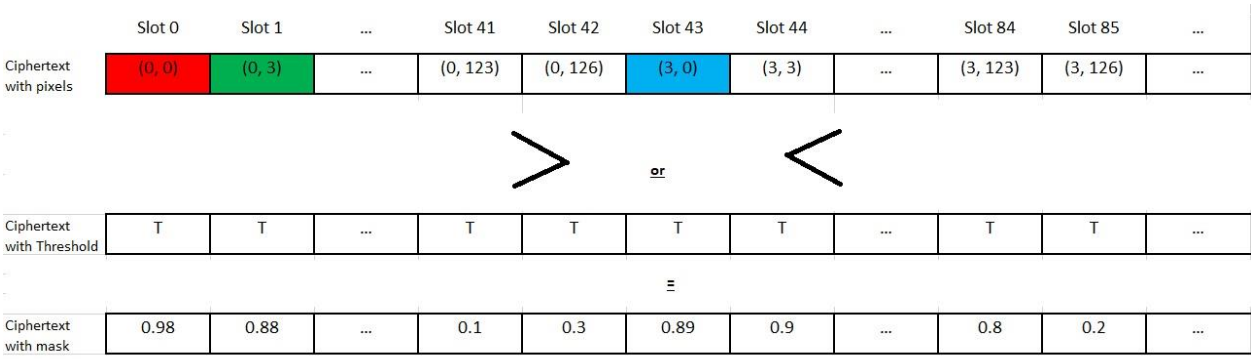


Figure 12: An example showing the comparison operation and the resulting mask values.

Table 11 shows the algorithm of the proposed HETH method. There are two sub functions in this algorithm: comp() and eval(). The comp function runs for "d" iterations. The input arguments to this function are the pixel value {A(i, j)} and the threshold value {T}. The comp function returns an approximate value of "1" or "0" based on the comparison. As the number of iterations "d" increases, the result closely approaches the exact value. Similarly, the eval function evaluates the sign function value fn(x). Here the iteration depends on the parameter "n", and as "n" value increases the value of the sign function converges closely to accurate results.

To optimize the use of circuit depth and achieve the maximum number of iterations, constants are reused during each iteration of the comp function. Similarly, the terms $x(1-x^2)$ and $(1-x^2)\cdot(1-x^2)$ are reused in the eval function. With this optimization, the eval function takes a circuit depth of 4. Thus, for each iteration in the comp function, the circuit depth increases by 4. A more detailed description about the depth consumption in the proposed HETH method is presented in Chapter 5.3.

Table 11: Algorithm of the proposed HETH method

Input	Ciphertext A – Image Pixels, Ciphertext B – Threshold,
Output	Ciphertext S – Binary Pixels

Functions:	***// comp (A, B)***
	{
	X[1] ← HomSub (A, B)
	for (i=1; i<=d; i++) do
	X[i+1] ← ***eval (X[i])***
	end for
	S ← HomMultPlain ([HomAdd (X[d+1] , 1)] , ½)
	return S
	}

	// eval (X)
	{
	T1 ← X
	F ← 0
	X_sq ← HomMult (X, X)
	T2 ← HomSub (1, X_sq)
	T3 ← HomMult (X, T2)
	for (i=0; i<=n; i++) do
	V1 ← 0.25^i
	V2 ← *combination (2×i, i)*
	V12 ← *V1×V2*
	for (t=1; t<=i/2; t++) do
	temp ← HomMult (T2, T2)
	end for
	V3 ← (i == even)? HomMult (T1, temp) : HomMult (T3, temp)
	f(i) ← HomMultPlain (V3, V12)
	F ← HomAdd (F, f(i)))
	end for
	return F
	}

4.2 Integration with HELAWF:

In this chapter, the proposed HETH is integrated with the proposed HELAWF. The overall scenario for this integrated design is shown in Figure 13.

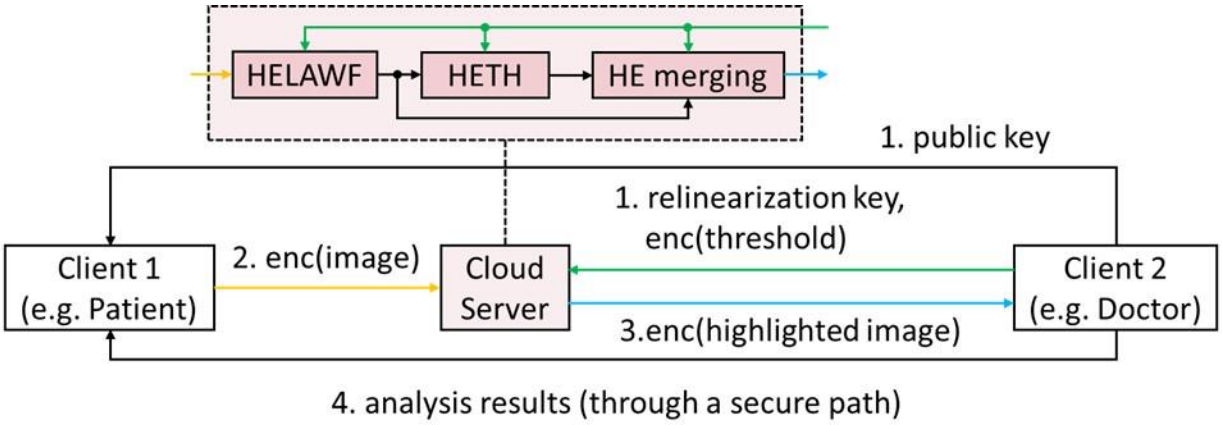


Figure 13: Overall scenario of the integrated design.

It is assumed that the first client (e.g., patient) has an image that is analyzed by the second client (e.g., doctor). However, the second client cannot analyze the image locally on his/her resource- and/or data-hungry machine and needs some processing to be done on a cloud server. In this case, the second client generates public key, secret key, and Relin key and then send the public key to the first client for encrypting the image. After that, the first client sends the encrypted image to the cloud server where HE-based IP or analysis is done. Meanwhile, the second client sends the generated Relin key directly to the cloud server for homomorphic evaluation.

As applications, the cloud server first performs the proposed HELAWF to remove noise included in the encrypted image. After noise removal, it performs the proposed HETH to get the encrypted mask image. The final step is to multiply the filtered image by the mask image in the HE domain to get a highlighted region of interest. Finally, the cloud server sends the resulting ciphertext back to the second client. This client uses his/her secret key to decrypt the resulting

image and analyzes the highlighted image. Once he/she makes a decision (e.g., diagnosis), the result is delivered to the first client through a secure channel.

The Integrated design follows the same encoding method as explained in chapter 3.1. Figure 14 shows an example of how the integrated design with the proposed encoding method works. If the filtered pixel value is lower than the threshold value, the resulting pixel value is approximately set to 0. Otherwise, the resulting pixel value is approximately set to 1.

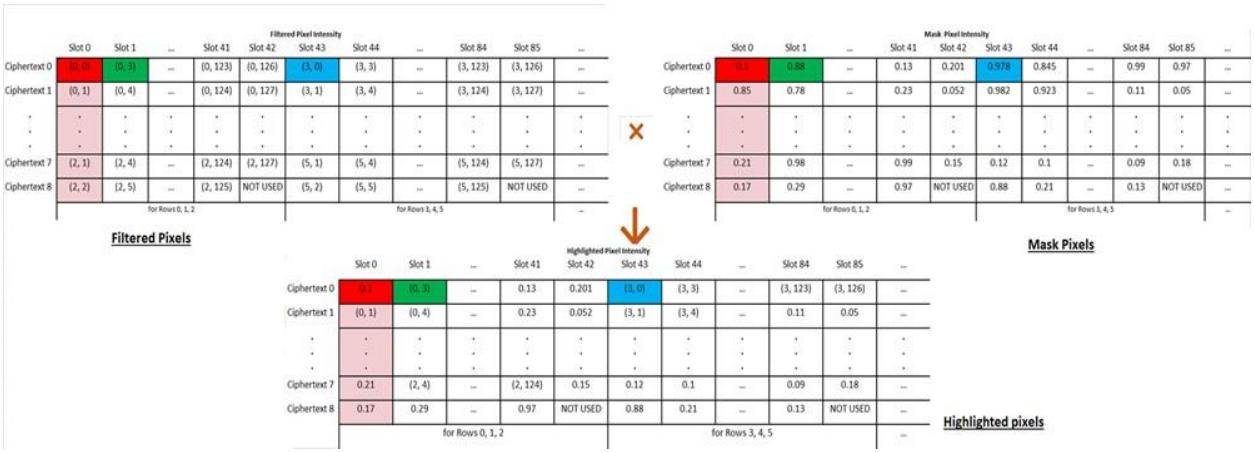


Figure 14: An example of the integrated design with the proposed encoding method.

Chapter 5: Evaluation:

This chapter evaluates the proposed HELAWF, HETH, and integrated designs. First of all, the experimental environment including HE parameters, test images, image quality metrics, and hardware platforms is introduced. Each of the proposed designs is then evaluated in terms of image quality and execution time.

5.1 Experimental Environment:

5.1.1 SEAL Parameters:

The proposed designs are implemented using CKKS functions of the Microsoft SEAL (version 3.6.2) [7]. As explained in chapter 2.4.3, there are mainly two HE parameter types in SEAL: polynomial degree N and coefficient modulus bit-length q. If a HE scheme is based on RNS, q is approximately set to a product of distinct prime numbers, and each prime is consumed whenever multiplication on ciphertexts is performed. A product of primes can be less than q, which helps reduce execution time in the SEAL, but not greater than q. In general, as the N value increases, the q value increases. Therefore, if a security level and arithmetic precision do not change in a HE scheme, the maximum depth increases as the N value increases. In the current SEAL version, the maximum available N value is 2^{15}. For the 128-bit security, which is widely used for recent real-world applications, the SEAL recommends 441 and 885 bits for N of 2^{14} and 2^{15}, respectively [8]. It is shown in Table 12. If 40 bits are allocated to each prime except for the first and last primes that typically use more bits, the maximum depths for N of 2^{14} and 2^{15} are around 8 and 19, respectively. These two depths are used for the proposed designs.

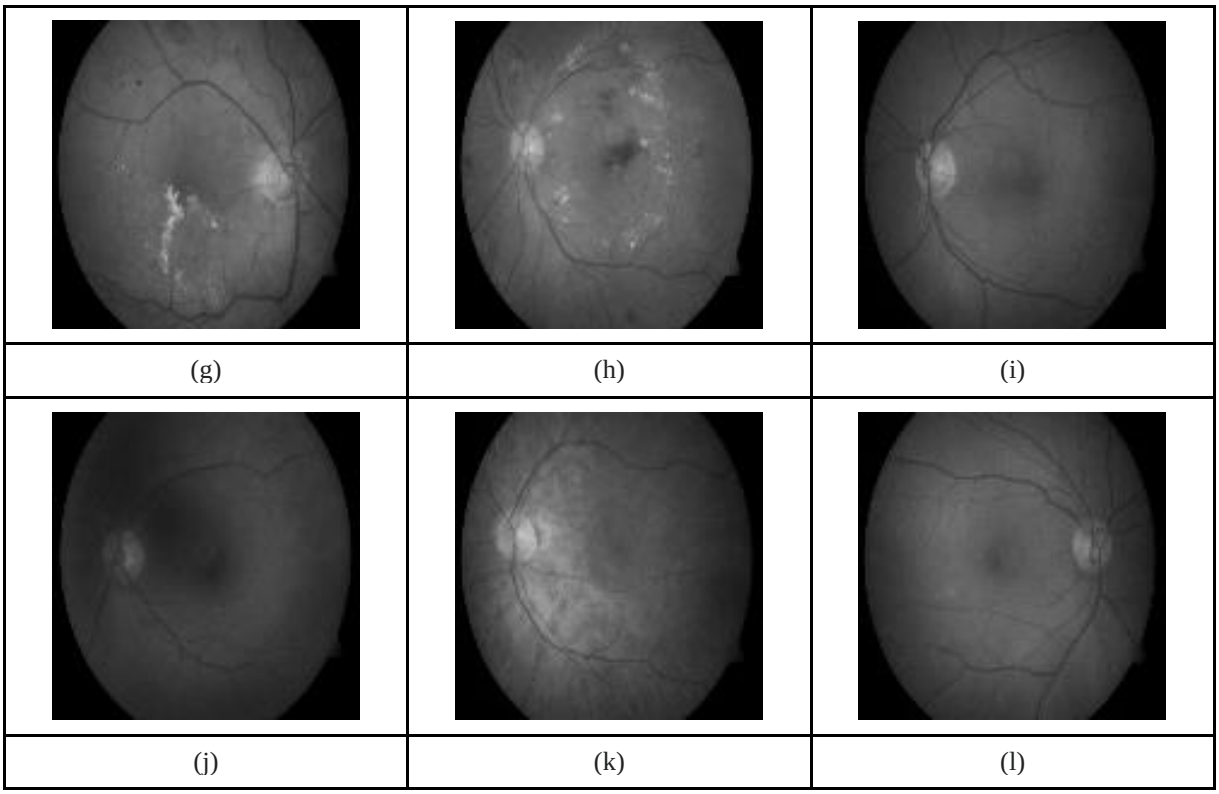
Table 12. Parameter sets of the CKKS scheme and corresponding circuit depths

	Polynomial degree N	Bit-length of q [7]	Maximum depth
Set 1	2^{14}	441	8
Set 2	2^{15}	885	19

5.1.2 Test Images:

As test images, twelve images shown in Figure 15 are used. In particular, Figures 15-A to 15-F are standard test images that are widely used in IP studies [17], and Figures 15-G to 15-L are popular diabetic retinopathy images (DIARETDB1) [18]. In evaluation, these test images are converted into 128 x 128 and 256 x 256 sized grayscale images in advance. These two image sizes generate the fewest number of ciphertexts when the proposed encoding method with the W value of 3 is used, as shown in Table 8.

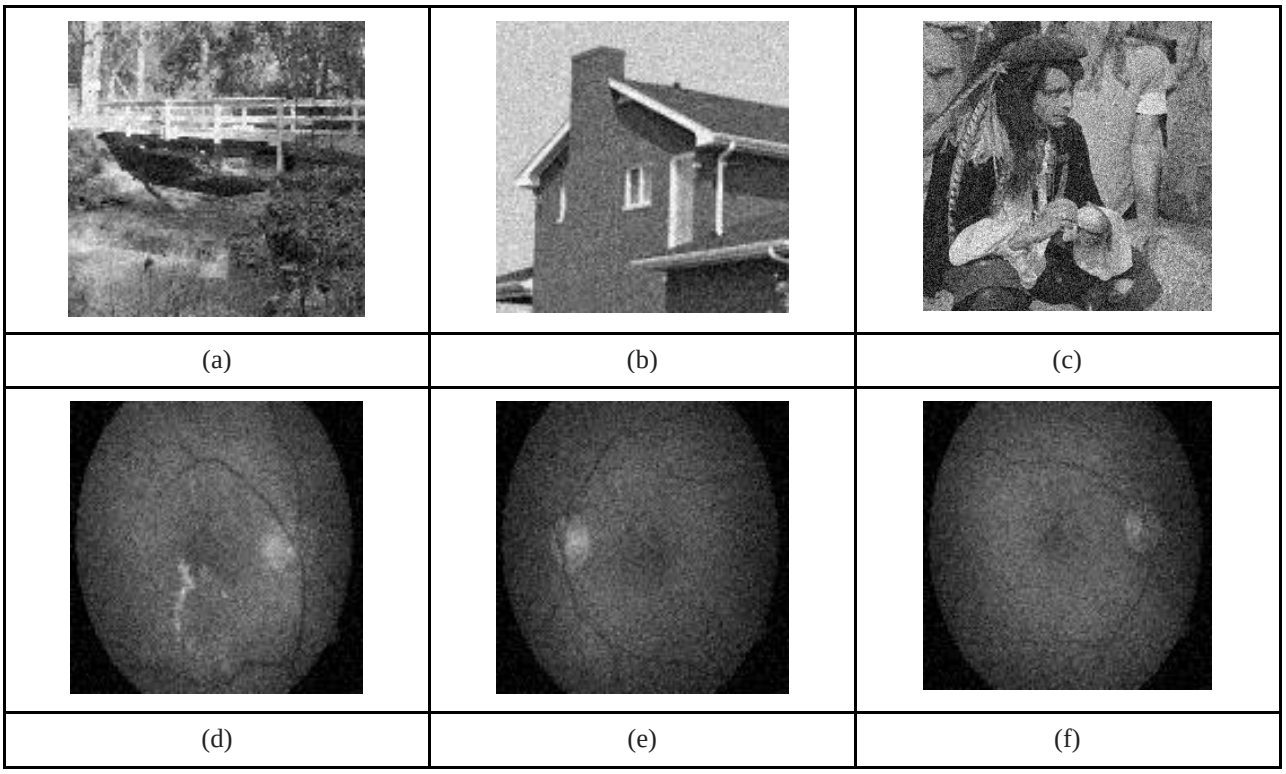
		
(a)	(b)	I)
		
I	I)	(f)



(g)	(h)	(i)
(j)	(k)	(l)

Figure 15: Test images [17] [18]. (a) boat (b) bridge (c) cameraman (d)II (e) Lena (f) pirate (g) DIARET1 (h) DIARET2 (i) DIARET3 (j) DIARET4 (k) DIARET5 (l) DIARET6.

To generate noisy images, Gaussian noise with a mean of 0.00 (m_{gn} = 0.00) is added to the twelve test images. Two values are used as standard deviation (σ_{gn}) to vary the noise level: 0.01 and 0.05. Figure 16 shows the generated noisy images with the Gaussian noise with σ_{gn} of 0.05. To save space, the noisy images generated from the six test images (bridge, house, pirate, DIARET1, DIARET3, and DIARET 6) are only shown. The noisy images are encoded and encrypted, and then used as input data for the proposed designs.



(a) (b) (c)

(d) (e) (f)

Figure 16: Test images with Gaussian noise (m_{gn} = 0.0 and σ_{gn} = 0.05). (a) bridge (b) house (c) pirate (dIIARET1 (e) DIARET3 (f) DIARET6.

To evaluate image quality of the proposed designs, PSNR values are calculated. If 8-bit grayscale input image *I* and output image *O*, of which resolution is $R \times C$, are compared to each other, the PSNR value is calculated by equations (8) and (9):

$$PSNR\ (dB) = 10 \times log_{10}(255^2\ /MSE), \tag{8}$$

$$MSE = \frac{\sum_{i=0}^{R-1}[O(i-,j) - I(i,j)]^2}{R \times C}. \tag{9}$$

5.1.3 Hardware Platforms and Implementation:

As explained in chapter 4.2, the proposed HELAWF, HETH, and merging process are performed on the server side, while other functions including encoding, encryption, decryption, and decoding are performed on the client side. The server functions are implemented on a Linux PC with the Intel Xeon W-2295 processor with the base frequency of 3.0GHz and the 128GB memory. For the client functions, Raspberry Pi (RPi) 4 Model B with the ARM Cortex-A72 processor with the frequency of 1.5GHz and the 2GB memory [19] is used. The execution times introduced in the following subchapters are the measured values on these hardware platforms. Figure 17 shows the implementation as a client server model.

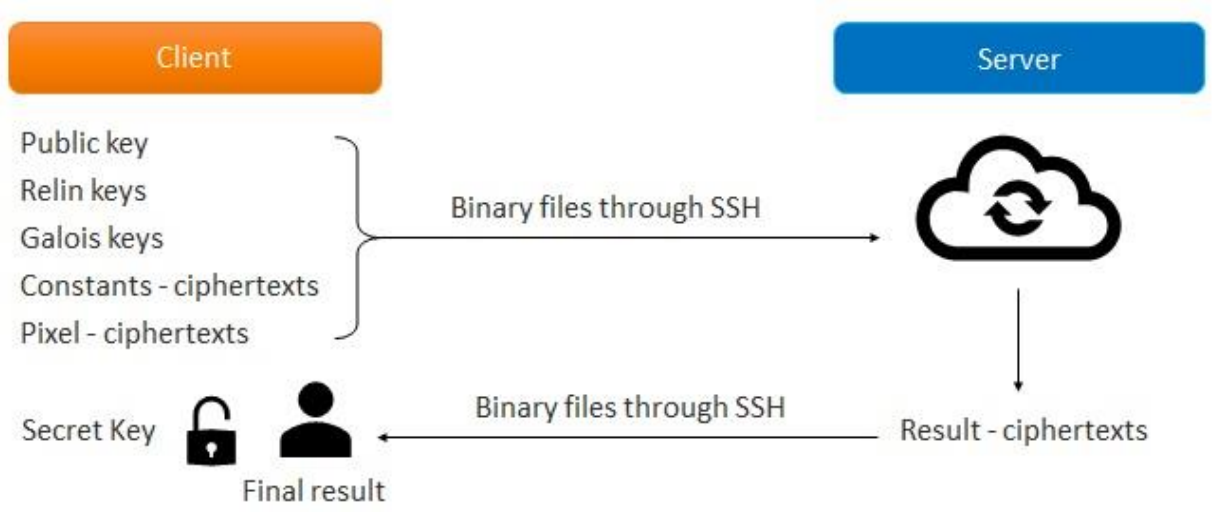


Figure 17: Client-server model implementation.

In this model, it is assumed that the client is a resource/power hungry device, such as an embedded board, and the server is a resource/power-rich device, such as a PC and a cloud. In practical scenarios, noisy images are obtained on the client side. In our original design, images are captured by a camera sensor attached to the client device, but for evaluation, it is replaced with two stages: reading test images and adding Gaussian noise to these images. The parameters used in noise generation are presented in Chapter 5.1.2.

Using the parameters of the CKKS scheme explained in Chapter 5.1.1, keys are generated. To be specific, the public key and secret key are generated and stored on the client side. Besides, the Relin keys and Galois keys are generated for HomMult and HomRot performed on the server, respectively. When all the keys are ready, pixels in noisy images are encoded into plaintext polynomials and then encrypted using the public key. The implementation in this thesis creates binary files for the input ciphertexts, Relin keys, and Galois keys and then transmits these files to the server through a network.

Once all the binary files are transmitted from the client, the server starts running the proposed HE-based implementations on the encrypted images, as described in Chapter 4.2. After that, the server creates another binary file for the resulting ciphertext which when decrypted will give the highlighted image. This ciphertext is transmitted back to the client through the network. In the final stage, the client decrypts the resulting ciphertext using the secret key and then decodes the resulting plaintext into a pixel vector. For evaluation, the resulting images constructed from this pixel vector are compared to the images obtained by the conventional integrated design. The PSNR values are calculated to measure the quality of the processed images.

5.2 Evaluation of Proposed HELAWF:

In this subchapter, the image quality and execution time of the proposed HELAWF design are evaluated. The filter window size is set to 3 x 3, and the value of $\sigma_o{}^2$ in (1) is set to 0.1. As reference designs, non-HE-based LAWF designs [4] without and with the approximate inverse operation are used. In the evaluation chapter, they are denoted as LAWF and approximate LAWF, respectively. For the previous and proposed designs using the numerical inverse operation, the d_{inv} values are set from 1 to 5.

5.2.1 Image Quality:

Figure 18 compares the PSNR values of the previous and proposed designs when the image size is 128x128 and the σ_{gn} value is set to 0.05. The average PSNR values for the twelve noisy images are shown as the leftmost group of bar graphs in Figure 19. When the previous LAWF and approximate LAWF designs are compared to each other, the approximate LAWF design shows lower PSNR values than the LAWF design. However, the PSNR difference is reduced as the value of d_{inv} of the approximate LAWF design increases. In particular, the average PSNR difference is less than 0.1% when the d_{inv} value becomes 4. The d_{inv} value at which the saturation starts is the same in the proposed HELAWF design. Specifically, when the d_{inv} value increases from 3 to 4, the average PSNR value increases by 0.04 dB, but when the d_{inv} value increases from 4 to 5, there is no change in the average PSNR value.

Despite this similarity in the d_{inv} value, the average PSNR value of the proposed design for all test images and all d_{inv} values are 0.86 dB lower than that of the previous approximate LAWF design. There are two specific features on this. First, the average PSNR difference of the previous approximate LAWF design and the proposed design increases as the d_{inv} value increases. For example, when the d_{inv} value is 1, the average PSNR difference is 0.69 dB, but when the d_{inv} value is 5, it increases to 0.97 dB. This is because the CKKS scheme performs rescaling that generates an error after every multiplication to match the given precision [20]. In the numerical inverse operation, a result with an error is used as an input to the next iteration. Therefore, as the d_{inv} value increases, the average PSNR difference increases. Second, the average PSNR difference in the six standard images is larger than that in the six DIARET images. This is because a filter window contains more edges in the six standard images. With fewer edges, σ_l in (1) has

smaller values (and errors), and consequently the numerical inverse operation using $\sigma_l+\sigma_o$ as an input produces smaller errors.

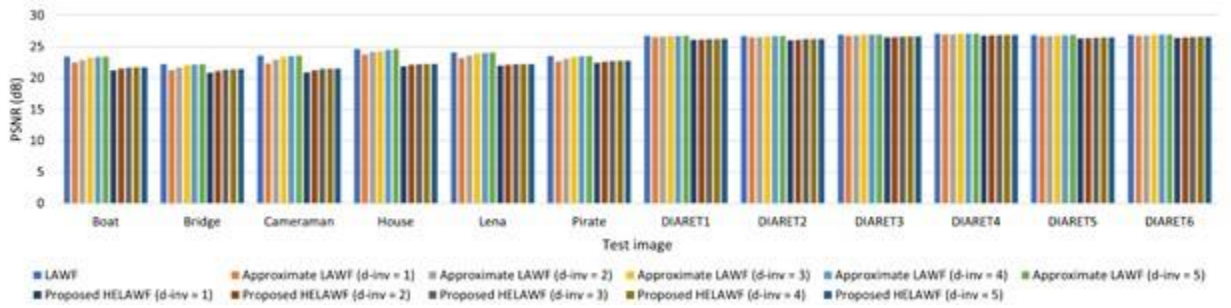


Figure 18: PSNR comparison of the twelve filtered images (image size = 128x128; σ_{gn} = 0.05).

Similarly, as the image size increases, the probability that edges are included in a filter window decreases, and therefore the average PSNR value increases. The second group of bar graphs in Figure 19 shows the average PSNR values when the image size is 256x256. Compared to the PSNR results for the 128x128-sized images, the average PSNR values of the previous approximate LAWF design and the proposed design for all the d_{inv} values increase by 0.97 dB and 1.89 dB, respectively. As described earlier, as fewer edges are included in a filter window, the error of the numerical inverse operation used in the proposed design is reduced. So, the PSNR increase in the proposed design is larger than that in the previous approximate LAWF design.

The last two groups of bar graphs show the PSNR values for the test images with Gaussian noise with σ_{gn} of 0.01. In this case, since noise is concentrated in a narrower area than the σ_{gn} of 0.05 case, filtering has a greater effect. Specifically, the average PSNR values of the previous approximate LAWF design and the proposed design for the two image sizes increase to 6 dB and 4 dB, respectively. The main reason for the 2 dB difference is that even if the amount of MSE change is similar, the PSNR change becomes larger as it changes at a smaller MSE value.

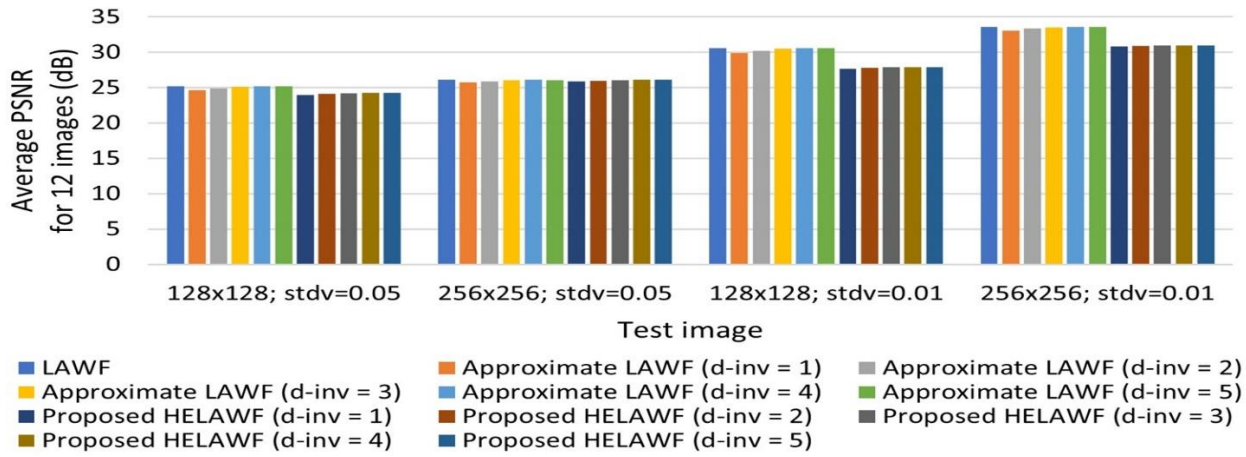


Figure 19: Average PSNR comparison for the 128x128- and 256x256-sized test images with σ_{gn} of 0.05 and 0.01.

5.2.2 Execution Time:

Table 13 shows the average execution time of the previous non-HE-based designs and proposed HELAWF design working on the PC. The image size for this experiment is set to 128x128. The previous non-HE-based LAWF designs show execution times of less than 0.1 seconds for an image. Although the approximate LAWF uses iterations, the execution time change according to the number of iterations is less than 0.01 seconds. Therefore, its results are not divided according to the d_{inv} value. As in the earlier image quality experiments, the d_{inv} values from 1 to 5 are used for the proposed HELAWF design. These values consume depths from 5 to 9 in the entire filtering. As shown in the third column, depths from 5 to 8 are supported by the HE parameter set 1. However, a depth of 9 is not supported by that parameter set, and therefore the HE parameter set 2 is used.

For nine ciphertexts that are the minimum in the proposed encoding method, the execution times of the proposed HELAWF designs with d_{inv} values from 1 to 5 are around 4, 5, 7, 8, and 22 seconds. When the HE parameter set 1 is used, the increase in execution time is approximately linear. However, when the HE parameter set 2 is used, the execution time increases more than twice compared to that with the HE parameter set 1 because the *N* value is doubled.

Table 13: Execution time in image filtering

Algorithm	Depth	Param.	Time (sec.) for	
			9 ciphertexts	1 image
LAWF	-	-	-	0.02
Approximate LAWF	-	-	-	0.03
Proposed HELAWF ($d_{inv} = 1$)	5	Set 1	3.84	0.48
Proposed HELAWF ($d_{inv} = 2$)	6	Set 1	5.16	0.65
Proposed HELAWF ($d_{inv} = 3$)	7	Set 1	6.67	0.83
Proposed HELAWF ($d_{inv} = 4$)	8	Set 1	8.33	1.04
Proposed HELAWF ($d_{inv} = 5$)	9	Set 2	21.92	2.74

As shown in Table 8, nine ciphertexts contain eight 128x128-sized images. Therefore, the execution time per image has a smaller number than the aforementioned number, as shown in the last column of Table 13. Specifically, when the HE parameter set 1 is used, each image is filtered in a second or less by the proposed design.

Table 14 shows the execution times in the numerical inverse operation out of the total execution times of the proposed HELAWF design. When the d_{inv} value is 1, only 7% of the total execution time is spent in the numerical inverse operation even though 2 HomMults out of 5 HomMults are performed in that operation. This is because HomRots that also require long execution times are used. Except for the numerical inverse operation, the same operations are performed regardless of the d_{inv} value in filtering. However, since the number of allocated primes increases by 1 as the d_{inv} increases by 1, the execution time of the non-inverse operations in filtering increases almost in proportion. As the d_{inv} value increases, both the execution time in the

numerical inverse operation and the total execution time increase almost linearly, and therefore the percentage increases until saturation.

Table 14. Execution time in calculating a multiplicative inverse

d_{inv}	1	2	3	4	5
Time (sec.)	0.28	0.66	1.15	1.77	5.40
Percentage (%)	7.29	12.79	17.24	21.25	24.64

5.3 Evaluation of Proposed HETH:

This subchapter evaluates the proposed HETH design in terms of image quality and execution time. A threshold for each image is pre-computed as the average pixel value of the image.

5.3.1 Image Quality:

Figure 20 compares the PSNR results of the proposed HETH design. To save space, only the results for the 6 DIARETDB1 images are shown. The reference resulting images used to calculate the PSNR values are generated by a conventional non-HE-based image thresholding method without the numerical comparison operation. Since saturation occurs near n_{comp} of 7, the results for n_{comp} of 5, 6, and 7 are only shown. Note that the PSNR values of a non-HE-based conventional method with the numerical operations are not shown in this figure because the average difference rate compared to those of the proposed HETH is only 1%.

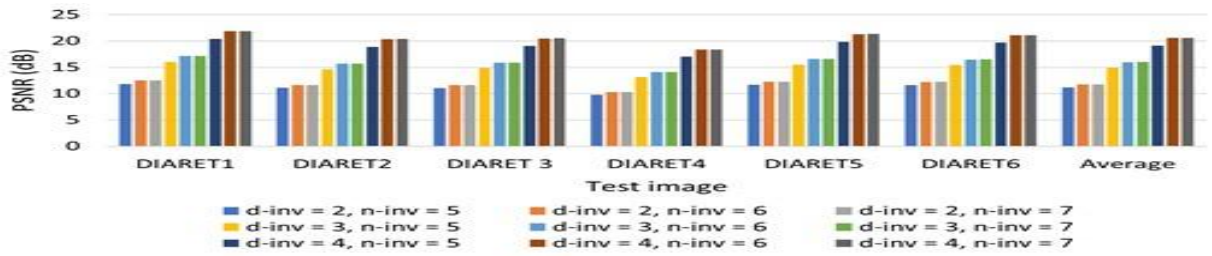


Figure 20: PSNR results of the proposed HETH design (image size = 128x128).

Several features are found from Figure 20. First, the PSNR values are more affected by d_{comp} than n_{comp} values. In particular, when the n_{comp} value increases from 5 to 7, the average PSNR value of the proposed design increases by 1 dB, but when the d_{comp} value increases from 2 to 4, the average PSNR value increases by 9 dB. This is because the computational complexity of the numerical comparison operation is more determined by the d_{comp} value than the n_{comp} value [10].

Second, the PSNR values shown in Figure 20 themselves are relatively low. The reason is that the numerical comparison operation outputs a real number between 0 and 1, not exactly 0 and 1. Nevertheless, as the d_{comp} value increases (especially when exceeding 3), resulting images become similar to the thresholded images of a conventional method without the numerical comparison operation perceptually. Figure 21 shows the thresholded DIARET images by a conventional method without the numerical comparison operation (Figures 21-A to 21-C) and those by the proposed HETH (Figures 21-D to 21-L). To save space, the thresholded images for the DIARET1, DIARET3, and DIARET6 images are only shown.

As shown in Figures 21-D to 21-F, when the d_{comp} value is 2 (and the n_{comp} value is 7), each original pixel value and the corresponding thresholded pixel value have a relationship close to a linear function rather than a step function. However, when the d_{comp} value increases to 4, the majority pixels have binary values except for a few pixels with values close to the threshold, as shown in Figures 21-J to 21-L.

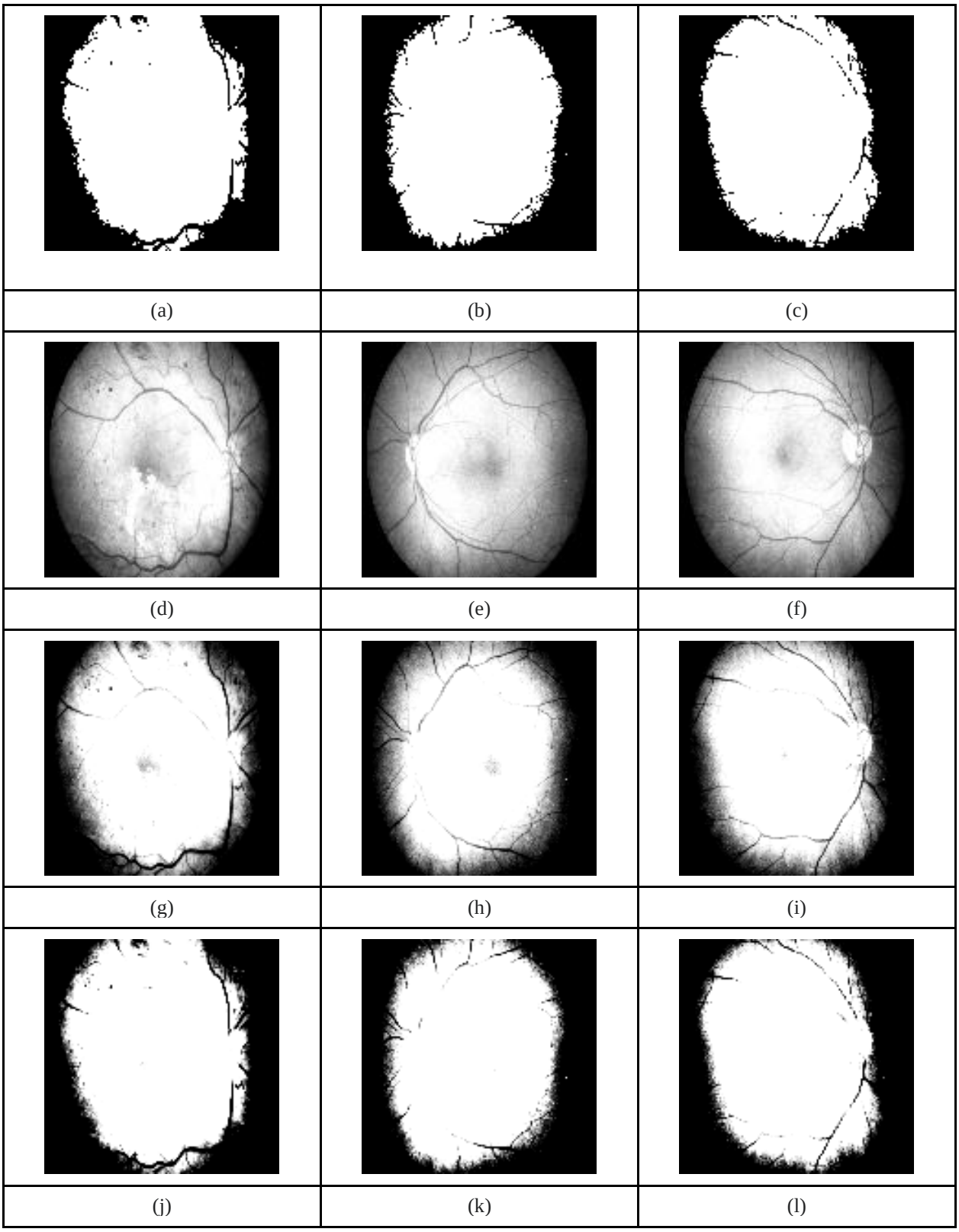


Figure 21: Thresholded DIARET1, DIARET3, and DIARET6 images. (a)-(c) by a conventional method without the numerical comparison operation (d)-(f) by the proposed HETH with d_{comp} of 2 and n_{comp} of 7 (g)-(i) by the proposed HETH with d_{comp} of 3 and n_{comp} of 7 (j)-(l) by the proposed HETH with d_{comp} of 4 and n_{comp} of 7.

5.3.2 Execution Time:

Table 15 shows the execution times of the proposed HETH design on the PC. The first two columns show the d_{comp} and n_{comp} values used in this experiment, and the third column shows the corresponding depths. As explained in chapter 4, the depth of the proposed HETH design depends on d_{comp} only. Since all the depths in Table 15 are larger than 8 that is the maximum depth of the HE parameter set 1, the HE parameter set 2 is used in this experiment. When the n_{comp} value increases while the d_{comp} value is fixed, the execution time increases almost linearly. However, the ratio varies depending on the fixed d_{comp} value, as shown in the last column of Table 15.

Table 15. Execution time of the proposed HETH design

d_{comp}	n_{comp}	Depth	Time (sec.)	ΔTime/Δn_{comp}
2	5	9	1.43	≈ 0.52
2	6	9	1.96	
2	7	9	2.46	
3	5	13	3.21	≈ 1.20
3	6	13	44.42	
3	7	13	5.60	
4	5	17	6.01	≈ 2.25
4	6	17	8.24	
4	7	17	10.50	

5.4 Evaluation of the Entire Integrated Design:

In this subchapter, the integrated design of the proposed HELAWF and HETH is evaluated. As input images, the twelve 128x128-sized test images with Gaussian noise with $m_{gn}$of 0.00 and σ_{gn} of 0.05 are used. To perform both proposed HELAWF and HETH with a given maximum depth of 19, which is provided by the HE parameter set 2, two (d_{inv}, d_{comp}, n_{comp}) sets are used: (4, 2, 7) and (1, 3, 7). The former maximizes the performance of HELAWF, and the latter maximizes the performance of HETH under the given maximum depth.

5.4.1 Image Quality:

Figure 22 shows the PSNR results of our integrated design. When calculating PSNR values, the results of a software implementation without the numerical operations are used. In terms of average PSNR value, (d_{inv}, d_{comp}, n_{comp}) of (1, 3, 7) for the numerical operations shows better results, of which PSNR value is 26 dB, than (d_{inv}, d_{comp}, n_{comp}) of (4, 2, 7), of which PSNR value is 22 dB. As shown in the previous experimental results, the results of the numerical comparison operation are more affected by the number of iterations than the results of the numerical inverse operation. Therefore, although the numerical comparison operation consumes a relatively large depth in the HE domain, it is advantageous to set the d_{comp} value to the maximum under the depth constraint in our integrated design.

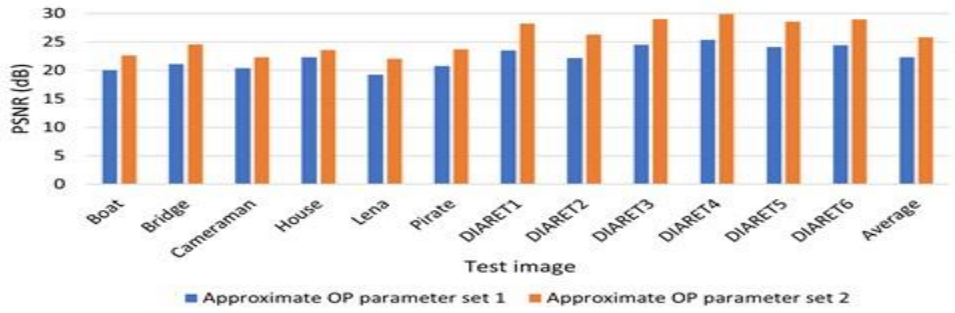


Figure 22: PSNR values of the integrated design of the proposed HELAWF and HETH.

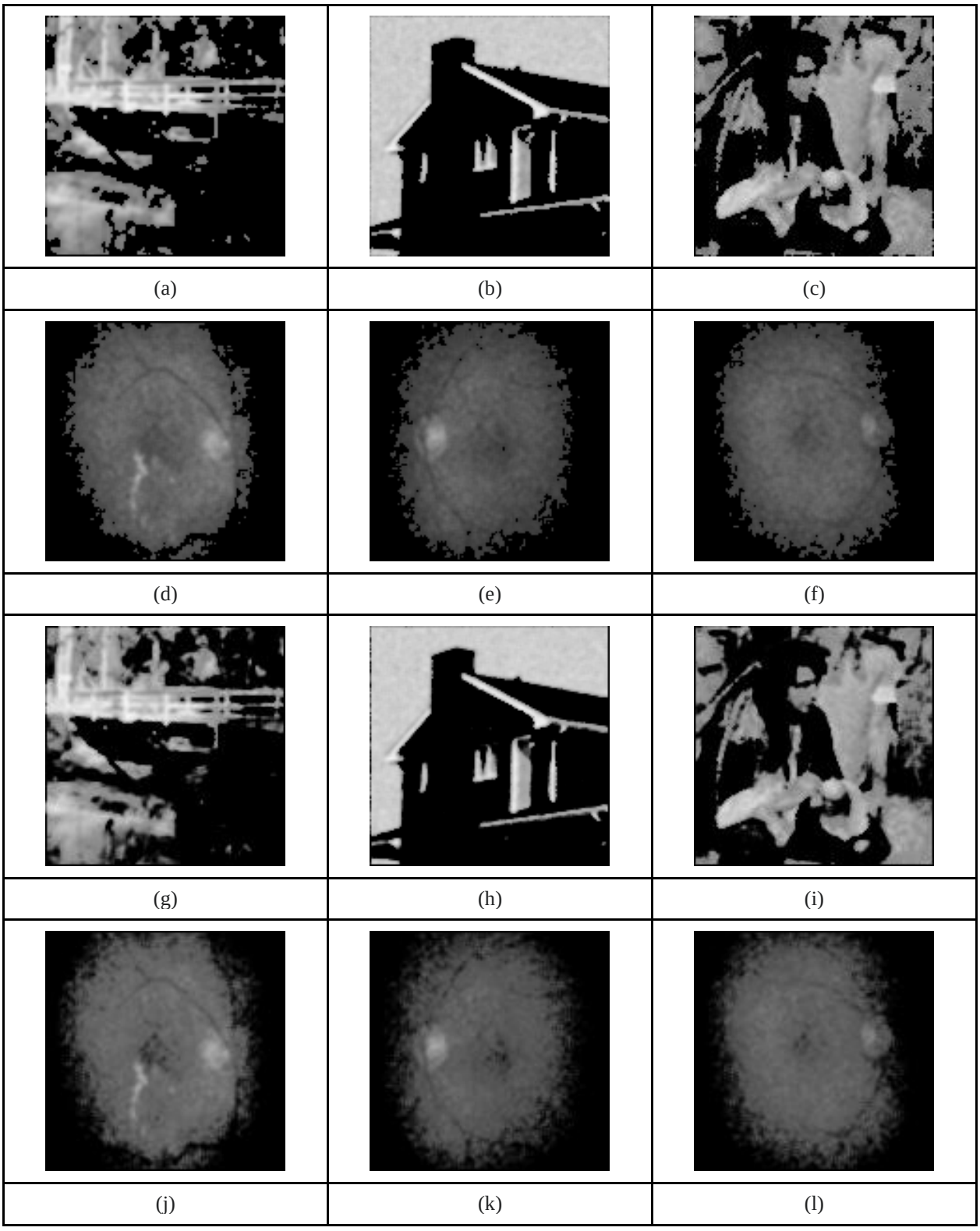


Figure 23: Filtered, thresholded, and merged images (a) reference-bridge (b) reference-house (c) reference-pirate (d) refereI-DIARET1 (e) reference-DIARET3 (f) reference-DIARET6 (g) proposed-bridge (h) proposed-house (i) proposed-pirate (j) proposed-DIARET1 (k) proposed-DIARET3 (l) proposed-DIARET6.

Figure 23 shows the resulting images that are filtered, thresholded, and then merged. To save space, the resulting images for the six input images (bridge, house, pirate, DIARET1, DIARET3, and DIARET6) are only shown. Figures 23-A to 23-F show the resulting images by the reference software without the numerical operations, and Figures 23-G to 23-L show the resulting images by the proposed integrated design with the parameter set 2 for the numerical operations. As in the reference resulting images, pixel values below thresholds are represented in black and the others are set to filtered values in the resulting images of the proposed design. Although pixels around thresholds have vague values, it can be alleviated if the given maximum depth increases.

5.4.2 Execution Time:

Table 16 shows the execution times of the integrated design of the proposed HELAWF and HETH working on the PC. The second row and third row show the two parameter sets for the numerical operations. The former set has HELAWF, HETH, and the merging part use 8 depths, 9 depths, and 1 depth, respectively. Therefore, the total depth is 18. On the other hand, the latter set has the processes use 5 depths, 13 depths, and 1 depth, respectively, and thus the total depth is 19. The seventh column shows the total execution time. Since all or almost all primes are used for the depths, the total execution time is longer than the sum of the execution times of the proposed HELAWF and HETH. The last column shows the execution time per image. The two parameter sets show 8-9 seconds for each image. Note that the execution time may significantly increase as the number of ciphertexts increases. A shown in Table 8, a larger image size generates more ciphertexts. This is one of the limitations of the proposed HE-based designs. However, for small-sized images, this thesis proves that the proposed designs work in real-time.

Table 16. Execution time of the proposed integrated design

d_{inv}	d_{comp}	n_{comp}	Depth			Time (sec.)	
			HELAWF	HETH	Entire	9 ciphers	1 image
4	2	7	8	9	18	76.05	9.51
1	3	7	5	13	19	67.09	8.39

Table 17 shows the execution times of the functions performed on the client side when eight 128x128-sized grayscale images are encoded and encrypted into nine input ciphertexts. Note that this table does not include the execution time in KeyGen because this function is initially executed once, and the generated keys are reused. As shown in the second row, the two HE parameter sets suggested in this thesis are used. The number of primes is set to the maximum number in each set.

Table 17. Execution time in the client functions (in milliseconds)

Device	RPi 4		PC	
HE parameter	Set 1	Set 2	Set 1	Set 2
Encoding	880	3,599	102	413
Encryption	2,667	11,947	169	766
Decryption	31	734	3	71
Decoding	495	4,243	744	625
Total in the client functions	4,073	20,523	348	1,875

For the HE parameter set 1, the total execution time in the client functions is 4 seconds. In that time, the execution times in the en/decoding functions related to the proposed encoding method account for 34%. This rate is similar even when the HE parameter set 2 is used (38%). However, the parameter set 2 increases the total execution time in the client functions to 21

seconds, which may be a burden in practical use. However, this execution time can be reduced if more hardware resources are available for the client side. For example, when the same functions are performed on the PC where the evaluation functions of the proposed designs are performed, the execution time is reduced by 91%, which is shown in the last two columns.

Note that the current SEAL version only supports single-thread processing. Therefore, the execution times of the client- and server- functions can be further reduced by using an open-source library supporting multi-thread processing. In addition, using custom hardware accelerators [21] [22] can help reduce the execution time.

5.4.3 Memory Usage:

In the scenario in this thesis, the client is assumed to be a device with less memory resources, which means that memory usage on the client side is important. Table 18 shows the memory usage on the client side when the integrated design is running. To measure the memory usage, binary files in Figure 17 were generated.

Table 18: Memory footprint on the client side

Binary files generated on the client side		**Memory footprint (for 9 ciphertexts)**
1.	Parameters (params.bin)	1 KB
2.	Galois Keys (gk.bin)	320.6 MB
3.	Relin Keys (rk.bin)	80.1 MB
4.	Ciphertexts (cipher.bin)	68.3 MB
5.	Constants (constants.bin)	53.1 MB
6.	Secret Key (sk.bin)	4 MB

Among the six binary files, the Galois key occupies the largest memory, followed by the Relin keys. Overall, this memory usage is greatly reduced by the proposed approach. Specifically, the size of Galois key is reduced by optimization as described in chapter 3.2, and the size of input ciphertexts is reduced by the proposed encoding method. Note that the keys used in homomorphic evaluation can be reused on the server side once they are generated on the client side. Therefore, this memory burden only occurs initially.

Chapter 6: Conclusion & Future Studies:

6.1 Conclusions:

In this thesis, HE-based IP designs using numerical methods are proposed. Specifically, the proposed HELAWF design includes a numerical inverse operation for division, and the proposed HETH uses a numerical comparison operation. This thesis proves that HE-based IP applications using the numerical methods can be implemented with recently popular HE parameters while avoiding time-consuming bootstrapping. In addition, unlike most earlier studies that encode tens of thousands of images to improve the amortized time, the proposed encoding method packs a small number of images into a small number of ciphertexts to reduce the execution time in homomorphic evaluation on the server, network bandwidth, and memory footprint. This provides general users with a more realistic solution. The proposed HE-based IP designs are based on an end-to-end HE design principle. Therefore, no further actions are required from the users after transmitting encrypted images.

6.2 Future Studies:

Despite these advantages, the execution time in the evaluation functions of the proposed designs remains a challenge. To mitigate this problem, modifying the current design to reduce the number of HomMults and HomRots can be performed. Although the proposed HETH uses a fixed threshold value, there are earlier studies using a locally computed threshold value. However, they contain multiple comparison operations, so it is difficult to implement with the current HE parameter sets. Since a HE-based image thresholding using a locally computed threshold value is useful in scenarios where non-uniform illumination exists in captured images, it could be an interesting future research topic. Implementing custom hardware accelerators to reduce the execution time could be another interesting research topic.

References

[1] ITRC Statistics, [Online] ITRC Statistics

[2] Privacy Enhancing Technologies, [Online]. Available: Privacy-enhancing technologies

[3] R. C. Gonzalez and R. E. Woods, Digital Image Processing. Englewood Cliffs, NJ, USA: Prentice-Hall, 2006.

[4] J. -S. Lee, "Digital Image Enhancement and Noise Filtering by Use of Local Statistics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, no. 2, pp. 165-168, Mar. 1980.

[5] Residual Number System, [Online] Available: Residue number system

[6] Homomorphic libraries, [Online]. Available: Homomorphic encryption

[7] H. Chen, K. Han, Z. Huang, A. Jalali, and K. Laine, "Simple Encrypted Arithmetic Library v2.3.0," [Online]. Available: Simple Encrypted Arithmetic Library v2.3.0

[8] R. E. Goldschmidt, "Applications of Division by Convergence," Ph.D. dissertation, Massachusetts Institute of Technology, 1964.

[9] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, "Numerical Methods for Comparison on Homomorphically Encrypted Numbers," in *Proc. ASIACRYPT*, 2019.

[10] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, "Efficient Homomorphic Comparison Methods with optimal complexity," in *Proc. ASIACRYPT*, 2020.

[11] M. Tarek Ibn Ziad, Amr Alanwar, Moustafa Alzantot, and Mani Srivastava, "CryptoImg: Privacy Preserving Processing Over Encrypted Images," in *Proc. SPC*, 2016

[12] William Fu, Raymond Lin, Daniel Inge, "Fully Homomorphic Image Processing," *arXiv preprint arXiv:1810.03249*, 2018.

[13] R. Challa, G. VijayaKumari and Sunny B, "Secure Image processing using LWE based Homomorphic encryption," in *Proc. ICECCT*, 2015.

[14] S. D. Kannivelu and S. Kim, "A Homomorphic Encryption-based Adaptive Image Filter Using Division Over Encrypted Data," in *Proc. RTCSA*, 2021.

[15] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A Full RNS Variant of Approximate Homomorphic Encryption," in *Proc. SAC*, 2018.

[16] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrrig, and J. Wernsing, "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy," in *Proc. ICML*, 2016.

[17] Standard Test Images, [Online]. Available: http://imageprocessingplace.com/root_files_V3/image_databases.htm

[18] Medical Test Images, [Online]. Available: DIARETDB1 - STANDARD DIABETIC RETINOPATHY DATABASE

[19] Raspberry Pi Trading Ltd., "Raspberry Pi 4 Computer Model B," [Online]. Available: https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf

[20] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Proc. ASIACRYPT,* 2017.

[21] S. Kim, K. Lee, W. Cho, J. H. Cheon, and R. A. Rutenbar, "FPGA-based Accelerators of Fully Pipelined Modular Multipliers for Homomorphic Encryption," in *Proc. ReConFig,* 2019.

[22] S. Kim, K. Lee, W. Cho, Y. Nam, J. H. Cheon, and R. A. Rutenbar, "Hardware Architecture of a Number Theoretic Transform for a Bootstrappable RNS-based Homomorphic Encryption Scheme," in *Proc. FCCM,* 2020.