Annales Mathematicae et Informaticae
Accepted manuscript
DOI: 10.33039/ami.2025.09.001
URL: https://ami.uni-eszterhazy.hu

Comparative analysis of homomorphic encryption schemes for encrypted image processing in OpenStack using TenSEAL*

Hekmat Saker[a], Farid Eyvazov[a], Tariq Emad Ali[b], Csaba Biró[ca], Dhulfiqar Zoltán Alwahab[d]

[a]Faculty of Informatics, Eötvös Loránd University,
Budapest, Hungary
{farid.eyvazov,gb56hg,dolfi}@inf.elte.hu

[b]Information and Communication Engineering, Al-Khwarizmi College of Engineering,
University of Baghdad, Baghdad, Iraq
tariqemad@kecbu.uobaghdad.edu.iq

[c]Faculty of Informatics, Eszterházy Károly Catholic University,
Eger, Hungary
biro.csaba@uni-eszterhazy.hu

[d]John von Neumann Faculty of Informatics, Óbuda University,
Budapest, Hungary
alwahab.zoltan@nik.uni-obuda.hu

Abstract. This thesis presents a detailed comparative analysis of several prominent homomorphic encryption (HE) schemes-BGV, CKKS, and TFHE-for secure encrypted image processing within an OpenStack cloud environment utilizing the TenSEAL library. With cloud computing becoming increasingly prevalent, ensuring data privacy during remote data processing has emerged as a critical challenge. Image processing, particularly brightness adjustment, is chosen as a representative task to explore the practicality and computational efficiency of HE schemes. Experiments conducted on a simulated cloud setup revealed significant differences in computational performance, memory utilization, and practical feasibility. Specifically, the CKKS

*This research was supported by the EKÖP-24 University Research Fellowship Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund, by the Ministry of Innovation and Technology, and by the National Research, Development and Innovation Office within the Quantum Information National Laboratory of Hungary.

Submitted: April 17, 2025
Accepted: September 15, 2025
Published online: September 17, 2025

scheme exhibited superior efficiency for approximate arithmetic operations, making it particularly well-suited for real-number computations in image processing. These findings highlight the strengths and limitations of current HE schemes, providing valuable insights into their practical deployment in cloud environments, with implications for improved data security and privacy.

Keywords: Homomorphic encryption, encrypted image processing, CKKS, BGV, TFHE, TenSEAL, cloud computing, OpenStack, privacy-preserving computation, secure data processing

AMS Subject Classification: 68P25, 68M14, 68U10, 94A60, 68W10, 68N19

1. Introduction

1.1. Background and context

The rapid adoption of cloud computing across healthcare, government, and enterprise sectors has introduced serious concerns around data confidentiality and privacy. Organizations increasingly outsource computation to cloud providers, but in doing so, risk exposing sensitive data to third-party infrastructures. Traditional encryption methods require decryption prior to processing, which leaves data vulnerable during computation.

Homomorphic Encryption (HE) offers a powerful alternative – it allows direct computation on encrypted data without ever decrypting it. This capability enables privacy-preserving computation in untrusted environments and has gained traction in domains such as secure machine learning, financial analytics, and encrypted database queries [3, 19]. Several HE schemes have emerged over the past decade, including BGV [3], CKKS [3], and TFHE [4], each optimized for different computational properties.

In image processing, the stakes are particularly high. Medical images (e.g., MRIs), surveillance camera footage, and biometric photos contain highly sensitive information that, if leaked, can lead to serious privacy violations. Securely processing such images in the cloud – without ever exposing them in plaintext – is a pressing and unsolved challenge. Yet practical deployment of HE remains limited due to its complexity, performance cost, and tuning sensitivity [6].

1.2. Problem statement

Although several HE schemes have been proposed, few studies offer a rigorous, side-by-side comparison of their performance in realistic cloud-based image processing scenarios. The complexity of HE schemes – such as BGV, CKKS, and TFHE – makes it unclear which is best suited for practical encrypted image enhancement tasks. Moreover, real cloud deployments introduce constraints such as memory limits, network overhead, and processing latency that are often overlooked in theoretical work [8].

1.3. Research objectives and questions

This thesis aims to evaluate the performance, feasibility, and trade-offs of three major HE schemes – BGV, CKKS, and TFHE – for privacy-preserving image brightness adjustment in a cloud environment. Specifically, we address the following research questions:

- **RQ1:** Which HE scheme is most computationally efficient for encrypted image brightness adjustment?
- **RQ2:** How do the schemes compare in terms of memory usage and scalability?
- **RQ3:** Does the homomorphic processing pipeline preserve image fidelity when compared to plaintext adjustment?

1.4. Justification and use case framing

The task of brightness adjustment is selected as a representative and foundational image processing operation. It is widely used in pre-processing pipelines for enhancement, object detection, and medical diagnosis. While simple, it provides a controlled basis for evaluating arithmetic performance under encryption.

The OpenStack cloud platform is used to simulate a realistic cloud computing environment. OpenStack is a widely adopted open-source infrastructure-as-a-service (IaaS) framework, used in both academic and industry settings [11, 15]. By deploying HE operations within OpenStack, this study emulates the conditions under which real-world privacy-preserving computation would be carried out.

1.5. Scope and structure of the thesis

This work focuses on applying HE schemes to RGB images of resolution 512×512, using the TenSEAL Python library [14] for encryption and processing. The schemes are benchmarked on encryption/decryption time, processing time for brightness adjustment, memory consumption, and image output quality.

Section 2 details the system model and requirements, outlining the components of the cloud-based architecture, the image brightness enhancement task, and the parameterized homomorphic encryption schemes under consideration. Section 3 describes the proposed methodology, including the adaptation of the image processing algorithm for homomorphic computation, the use of the TenSEAL library, and the deployment in an OpenStack environment. Section 5 presents the experimental setup, specifying the hardware configuration, datasets, and evaluation metrics. Section 6 reports the experimental results, comparing performance, memory usage, and image quality across encryption schemes. Section 7 explores broader cryptographic and computational implications of the findings. Finally, Section 6 concludes the article and suggests directions for future research.

1.6. Contribution

This thesis bridges the gap between theoretical homomorphic encryption schemes and their practical application in cloud-based image processing. It provides one of the first comparative evaluations of BGV, CKKS, and TFHE under a unified implementation using TenSEAL within a cloud infrastructure. The findings offer concrete guidance for practitioners and researchers seeking to deploy secure image processing pipelines in untrusted environments.

2. Homomorphic encryption schemes overview

Several homomorphic encryption schemes have been developed over the past decade, each with strengths and weaknesses depending on the use case and computational requirements [6]. Some of the prominent schemes include:

CKKS scheme CKKS is a public key encryption scheme in which a secret key and a public key are generated. The public key is used for encryption and can be shared, while the secret key is used for decryption and must be kept confidential [3, 22].

The foundation of CKKS, like many other homomorphic encryption schemes, is the Learning With Errors (LWE) problem, initially introduced by Regev [16] in the paper "On lattices, learning with errors, random linear codes, and cryptography." The LWE problem involves distinguishing noisy pairs of the form $(a_i, b_i) = (a_i, \langle a_i, s\rangle + e_i)$ from truly random ones in $\mathbb{Z}_q^n \times \mathbb{Z}_q$. Here, $a_i, s \in \mathbb{Z}_q^n$, where a_i is uniformly sampled, s is a secret, and the $e_i \in \mathbb{Z}_q$ are small noise values, typically Gaussian, that make the problem computationally hard. Without the noise terms e_i, the problem could be solved using Gaussian elimination to find the secret s.

The LWE problem is known to be as hard as worst-case lattice problems, which are resistant to quantum computer attacks. This computational hardness allows us to build a secure cryptosystem by making it difficult to extract the secret s from pairs of $(a_i, \langle a_i, s\rangle + e_i)$ [8].

Suppose we generate a secret key $s \in \mathbb{Z}_q^n$ that is kept private, and publish n pairs of the type $(a_i, \langle a_i, s\rangle + e_i)$, which can be written in matrix form as $(A, A \cdot s + e)$ where $A \in \mathbb{Z}_q^{n\times n}$ and $e \in \mathbb{Z}_q^n$. The LWE problem states that it is computationally hard to recover the secret key from this pair, making it suitable for creating a public key.

The public key is constructed as $p = (-A \cdot s + e, A)$, making it difficult to extract the secret key from p. During encryption and decryption of a message $\mu \in \mathbb{Z}_q^n$, the scheme operates as follows:

Encryption of μ using p Compute $c = (\mu, 0) + p = (\mu - A \cdot s + e, A) = (c_0, c_1)$.

Decryption of c using s Compute $\tilde{\mu} = c_0 + c_1 \cdot s = \mu - A \cdot s + e + A \cdot s = \mu + e \approx \mu$.

In the encryption phase, the public key is used to mask the message μ, which is hidden in the first coordinate of the ciphertext with a mask $-A \cdot s$. Since A is uniformly sampled, it effectively masks μ. During decryption, we remove this mask by combining the second coordinate of c (which contains A) with the secret key s to reveal $\mu + e$. Due to the presence of noise e, the decrypted message is an approximation of μ, leading to CKKS being an approximate arithmetic scheme [8]. When e is small, the decryption output closely approximates the original message μ.

This LWE-based approach provides a secure public-key cryptosystem resistant to quantum attacks [2]. However, the implementation introduces inefficiencies: while the secret key size is $O(n)$, the public key grows to $O(n^2)$ because of the matrix A, and computations also require $O(n^2)$ operations. Since n determines the security level, using LWE directly would be inefficient in practice, as the large key sizes and computational complexity make it impractical for many applications.

BGV scheme Named after Brakerski, Gentry, and Vaikuntanathan, the BGV scheme is a leveled homomorphic encryption scheme that supports both addition and multiplication operations [3]. It is efficient for certain applications but accumulates noise quickly, requiring bootstrapping or parameter adjustments for more complex computations [7].

TFHE scheme TFHE is a fast FHE scheme designed to operate on Boolean circuits with low computational overhead [4]. It is based on the Torus approach and uses gate bootstrapping, significantly reducing processing time. TFHE is particularly advantageous in cloud computing for secure computations due to its optimized memory usage and processing efficiency [20, 23].

In this study, the BGV, CKKS, and TFHE schemes are evaluated for their effectiveness and efficiency in encrypted image processing within a simulated cloud environment.

3. Homomorphic encryption using TenSEAL

The TenSEAL library [14], based on Microsoft SEAL, is a high-level Python wrapper designed to facilitate homomorphic encryption (HE) for data science, machine learning, and secure inference applications. Built by Zama, TenSEAL provides a simplified interface over complex HE primitives, allowing encrypted computations to be performed directly on numerical data in memory, without requiring decryption at any stage. This enables privacy-preserving workflows in untrusted environments such as public cloud platforms.

TenSEAL is particularly well-suited to Python-based workflows, offering compatibility with popular numerical libraries like NumPy and support for batching and vectorized encrypted operations. It supports several widely used HE schemes, including Brakerski-Gentry-Vaikuntanathan (BGV), Cheon-Kim-Kim-Song (CKKS), and Torus Fully Homomorphic Encryption (TFHE), each optimized for different computation types.

In this study, TenSEAL was selected for its ease of integration, active support, and robust backend built on Microsoft SEAL. It allows practical experimentation with multiple HE schemes under a unified interface, enabling comparative evaluation across real-world encrypted workloads. This makes it an ideal platform for testing privacy-preserving image processing tasks such as brightness adjustment in cloud environments.

All experiments were conducted using TenSEAL $v0.1.0a0$ and Python 3.10.4. Encryption parameters were manually configured to ensure consistent security levels and noise budgets across all schemes.

3.1. Application of TenSEAL in encrypted image processing

In our initial setup, we tested the encryption and brightness adjustment pipeline on a single 512×512 RGB image. For benchmarking and performance evaluation (Section 6), we scaled this pipeline to process a dataset of 1000 images using automation scripts and batch execution, keeping encryption parameters constant across all trials.

TenSEAL was used to encrypt each color channel (R, G, B) of the images independently. The encryption process involved flattening each channel into a one-dimensional array and applying homomorphic encryption using the TenSEAL interface. The encrypted images were then transferred to an Ubuntu instance running in an OpenStack cloud environment.

On the cloud instance, brightness adjustment was applied by adding a constant value homomorphically to each encrypted pixel in the color channels, without requiring decryption. This approach preserved end-to-end data privacy throughout the computation.

3.2. Comparative performance evaluation

Using TenSEAL provided a consistent platform for testing and comparing the BGV, CKKS, and TFHE schemes under the same experimental conditions. The comparative analysis involved measuring the following metrics:

Encryption and decryption time Time required to encrypt the image data and decrypt the processed output after brightness adjustment.

Computation time for brightness adjustment Time taken to perform homomorphic addition on each encrypted pixel.

Resource utilization Memory and CPU usage during the encryption, processing, and decryption stages.

This evaluation enabled an analysis of trade-offs between security, computational time, memory efficiency, and output fidelity in the context of cloud-based encrypted image processing.

4. OpenStack cloud computing environment

OpenStack is an open-source platform that enables cloud computing and infrastructure management. It provides various services such as Nova for compute, Neutron for networking, and Swift for object storage, making it a popular choice for private and hybrid clouds [11, 15]. OpenStack's modular architecture supports scalability, allowing users to add and manage resources as needed.

In this study, OpenStack was deployed on a laptop using Kolla-Ansible [11] (version 2023.1), with core services running on Ubuntu 20.04. An Ubuntu instance was launched in OpenStack to host the homomorphic encryption pipeline. This configuration provided a simulated but realistic cloud environment to measure HE scheme performance in a resource-constrained setting. Using OpenStack on a local machine offers unique challenges, particularly in terms of CPU and memory limitations. However, it provides a controlled and reproducible environment to assess the computational demands and scalability of HE implementations in cloud infrastructures.

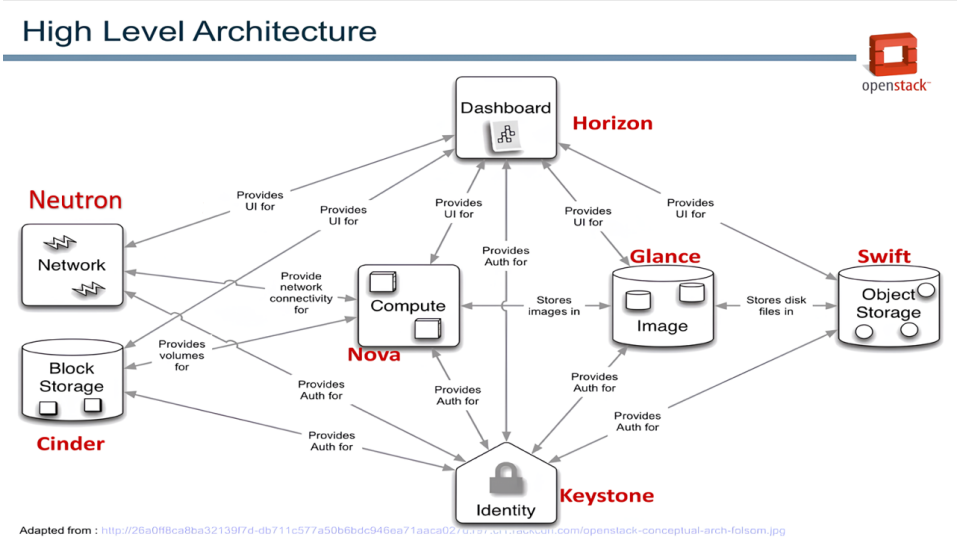

Figure 1. High-level architecture of the OpenStack-based image processing workflow.

Figure 1 illustrates the high-level architecture of the OpenStack deployment, showing the flow of encrypted image data between the client machine and the Ubuntu instance in the cloud.

5. Implementation and experimentation on OpenStack

The OpenStack environment for this experiment was deployed using *Kolla-Ansible* on an Ubuntu 20.04 server edition [11]. Table 1 summarizes the core components and their versions in the deployed setup.

Table 1. OpenStack environment configuration deployed with Kolla-Ansible.

Component	Service name	Version
Compute	Nova	18.3.0
Networking	Neutron	2.0
Block Storage	Cinder	9.3.0
Image Service	Glance	2.15
Identity Service	Keystone	3.14

The hardware resources for the OpenStack deployment consist of a 4-core CPU, 16 GB of RAM, and a 256 GB SSD. Networking is configured with a provider network setup, allowing instances to access external networks. The OpenStack environment is hosted on a single physical machine, simulating a private cloud setup for testing purposes.

Testing was conducted on an Ubuntu 20.04 instance with 2 vCPUs, 4 GB of RAM, and 20 GB of disk space. This instance was used to host homomorphic encryption applications using the TenSEAL library. Security groups were configured to permit SSH and ICMP access for monitoring and testing purposes.

This study uses a single image of size 512×512 pixels as input data. The image is encrypted on the host machine using the TenSEAL library, which provides an easy-to-use interface for homomorphic encryption with different schemes, including BGV, CKKS, and TFHE. Each color channel (R, G, B) of the image is encrypted separately using these schemes.

Once encrypted, the image data is transferred to an Ubuntu instance in an OpenStack environment, simulating a cloud setting for secure data processing. On this instance, homomorphic addition is performed on each encrypted pixel value to simulate brightness adjustment, without decrypting the data. The processed encrypted data is then transferred back to the host for decryption and reconstruction of the image.

Key steps in the implementation include:

- Encrypting each color channel (R, G, B) of the image using the BGV, CKKS, and TFHE schemes in TenSEAL.
- Uploading the encrypted data to the Ubuntu instance on OpenStack.
- Applying homomorphic addition on each encrypted channel to increase brightness.
- Transferring the processed encrypted data back to the host machine for decryption and image reconstruction.

The TenSEAL library was chosen for its Python compatibility and support for various homomorphic encryption schemes. Each scheme is tested for computational efficiency, noise accumulation, and practicality in a cloud environment.

5.1. Homomorphic encryption process for image brightening

The homomorphic encryption process used in this study involves the encryption of a 512×512 image using the TenSEAL library with the CKKS scheme. This allows encrypted manipulation of image brightness without exposing the original pixel values. Below are the key steps involved:

1. **Image preprocessing**
   - The image is loaded using the Python Imaging Library (PIL) and resized to 512×512 pixels to simplify processing.
   - The image is then split into its red, green, and blue (R, G, B) color channels, each represented as a separate $2D$ array.
2. **TenSEAL context initialization**
   - A TenSEAL context is created using the CKKS scheme with a polynomial modulus degree of 8192 and coefficient modulus sizes of $[60, 40, 40, 60]$. This configuration balances computational efficiency and security.
   - The global scale is set to 2^{40}, which ensures sufficient precision for encrypted computations on real-number data.
   - Galois keys are generated within the context to support batching operations, which allow vectorized operations on encrypted data [3, 19].
3. **Secret and public key serialization**
   - The secret context, which contains the decryption key, is serialized and saved as `secret.txt`. This will be used later for decrypting the modified encrypted data.
   - The context is then converted to a public-only version by removing the secret key, allowing encrypted data to be shared securely. This public context is saved as `public.txt` and used for encryption.

4. **Image encryption**
    - Each color channel (R, G, B) is encrypted separately. The flattened pixel values of each channel are converted into BGV, CKKS, or TFHE vectors using TenSEAL's function.
    - The encrypted vectors (`encrypted_r`, `encrypted_g`, `encrypted_b`) for each color channel allow brightness adjustment operations to be performed without revealing the original pixel values.
5. **Saving encrypted data**
    - The files `encrypted_r.txt`, `encrypted_g.txt`, and `encrypted_b.txt` contain the serialized encrypted data for each color channel. These files can be transferred to the cloud environment for further processing.

This encryption setup enables secure image processing in untrusted environments.

5.2. Dataset and automation for large-scale testing

While initial prototyping and validation were conducted using a single 512×512 image, the full-scale benchmarking was performed on a dataset of 800 images of identical resolution. These images were randomly sampled from the publicly available DIV2K dataset, which is commonly used in image enhancement research. Each image underwent separate encryption, cloud transfer, brightness adjustment, return transfer, and decryption.

To handle this scale, a Python-based automation pipeline was implemented. The process was fully scripted to:

- Load and preprocess each image,
- Split color channels and flatten them into vectors,
- Apply homomorphic encryption using TenSEAL (BGV, CKKS, or TFHE),
- Save and transfer encrypted data to the OpenStack instance,
- Execute brightness adjustment on the encrypted data in the cloud,
- Save processed encrypted vectors and transfer them back,
- Decrypt and reconstruct the final brightened image locally.

The pipeline utilized job queuing and batching to ensure resource-efficient parallel execution. Experiments were performed in batches of 100 images to manage memory constraints on the local and cloud instances. Each encryption scheme was evaluated across the same dataset under identical parameters to ensure comparability of results.

This process ensured consistent and repeatable measurements of encryption time, computation latency, and memory usage across the entire image set.

5.3. Security level and parameter justification

The CKKS scheme implementation used in this study is configured with a polynomial modulus degree of 8192 and coefficient modulus sizes of $[60, 40, 40, 60]$ bits. According to parameter guidelines provided by Microsoft SEAL and HE standardization efforts, this configuration provides an estimated 128-bit security level under the Learning With Errors (LWE) assumption. This aligns with Level 1 security recommendations in the Homomorphic Encryption Standardization Workshop [2].

Similarly, BGV and TFHE configurations were chosen to match comparable estimated security levels. For BGV, a modulus degree of 8192 and carefully selected plaintext modulus and ciphertext modulus sizes were used to ensure that the noise budget allowed multiple additions while maintaining security at or above the 128-bit level. For TFHE, security was assessed using parameter sets aligned with the TFHE library's default secure configurations, designed to meet or exceed 128-bit post-quantum security guarantees based on ring-LWE hardness assumptions.

In all cases, parameters were selected to balance computational efficiency and practical security. While stronger security (e.g., 192-bit or 256-bit) would be possible with higher-degree polynomials and larger moduli, this would come at the cost of significantly increased memory and processing time, making 128-bit a widely accepted trade-off point for practical encrypted computing tasks.

5.4. Brightness adjustment on encrypted data in the cloud

To perform brightness adjustment on the encrypted image in a secure cloud environment, the necessary encrypted files and a public context must be copied to the Ubuntu instance within the OpenStack environment. This setup allows brightness modifications directly on the encrypted data, preserving data privacy.

The main steps involved in this process are as follows:

1. **Transfer encrypted files to cloud instance**
    - The files `public.txt` (public TenSEAL context), `encrypted_r.txt`, `encrypted_g.txt`, and `encrypted_b.txt` (encrypted image channels) are securely copied from the local machine to the Ubuntu instance on the cloud. These files are necessary to perform encrypted operations on the image data without needing decryption.
2. **Loading encrypted data and context**
    - On the cloud instance, the public TenSEAL context is loaded from `public.txt` to enable encrypted computations.
    - The encrypted R, G, and B channels are then loaded from their respective files (`encrypted_r.txt`, `encrypted_g.txt`, and `encrypted_b.txt`) and converted into CKKS vectors using the public context.
3. **Brightness adjustment on encrypted channels**

- A constant brightness adjustment value of 60 is added to each pixel in the encrypted R, G, and B channels. This is achieved by using function `increase_brightness_on_encrypted_data` to perform homomorphic addition on each channel. This securely modifies the brightness of the image without exposing the pixel values.

4. **Saving the processed encrypted data**

   - After brightness adjustment, the modified encrypted data is serialized and saved as `processed_encrypted_r.txt`, `processed_encrypted_g.txt`, and `processed_encrypted_b.txt`. These processed files are ready for transfer back to the local environment for decryption and reconstruction of the brightened image.

This process ensures that image brightness is adjusted securely in the cloud without decrypting the data. By performing homomorphic addition on each encrypted channel, the image remains protected throughout, demonstrating the effectiveness of homomorphic encryption for privacy-preserving image processing in cloud environments.

5.5. Decryption and reconstruction of the brightened image

After applying brightness adjustments to the encrypted image data in the cloud environment, the processed encrypted data must be transferred back to the client device for decryption and reconstruction. This final step involves decrypting each color channel and combining them to produce the brightened image.

The key steps in this phase are as follows:

1. **Transfer processed data to client device**

   - The `processed_encrypted_r.txt`, `processed_encrypted_g.txt`, and `processed_encrypted_b.txt` processed encrypted files for each color channel are transferred from the cloud environment back to the client device.
   - The secret context file (`secret.txt`), containing the decryption key, is required on the client device to decrypt the processed data. This ensures that decryption can only be performed by authorized users with access to the secret key.

2. **Decryption of encrypted channels**

   - The TenSEAL context, which includes the secret key, is loaded on the client device by deserializing `secret.txt`. This context allows the decryption of each color channel.
   - Each processed encrypted channel – red, green, and blue – is loaded and decrypted separately using the BGV, CKKS, or TFHE vectors from TenSEAL. The decrypted values are reshaped to match the original image dimensions (512×512 pixels).

3. **Reconstruction of the brightened image**

   - After decryption, each color channel is clipped to ensure pixel values remain within the valid range (0–255) and are converted to unsigned 8-bit integers for compatibility with image formats.
   - The red, green, and blue channels are stacked to form a single RGB image, resulting in the brightened version of the original image.
   - The reconstructed brightened image is saved as `brightened_image.jpg` and displayed on the client device, providing a visual representation of the encrypted processing results.

This approach enables secure processing of image data in untrusted cloud environments, ensuring that sensitive data remains encrypted during processing and is only decrypted upon return to the client device. The use of homomorphic encryption with the CKKS scheme allows approximate arithmetic operations, like brightness adjustment, on encrypted data, achieving privacy-preserving image enhancement without revealing the original content.

6. Results and discussion

The experiment evaluated the performance of BGV, CKKS, and TFHE encryption schemes in encrypted image processing on a cloud platform, using 1000 images. Tables 2 and 3 present the computational performance metrics. Table 2 shows the processing times for each stage – encryption, brightness adjustment, and decryption – while Table 3 details memory usage for each scheme.

Table 2. Processing time for BGV, CKKS, and TFHE schemes in encrypted image processing (1000 images).

Scheme	Encryption time (s)	Brightness adjustment (s)	Decryption time (s)
BGV	55	14	45
CKKS	45	12	35
TFHE	70	100	60

The results indicate that CKKS is the most efficient scheme for this application, with the lowest processing time and memory usage. CKKS required only 12 seconds for brightness adjustment, likely benefiting from its design for approximate real-number arithmetic. Memory usage for CKKS remained stable across stages, peaking at 360 MB during brightness adjustment.

In contrast, BGV experienced higher processing times due to noise management overhead, especially during brightness adjustment, taking 14 seconds. BGV also exhibited slightly higher memory consumption than CKKS, peaking at 420 MB.

Table 3. Memory usage for BGV, CKKS, and TFHE schemes in encrypted image processing (1000 images).

Scheme	Encryption (MB)	Brightness adjustment (MB)	Decryption (MB)
BGV	400	420	400
CKKS	350	360	340
TFHE	500	510	500

6.1. Output quality evaluation

To evaluate the visual fidelity of the decrypted images after homomorphic brightness adjustment, we compared the results against baseline images processed directly in plaintext. The comparison was performed on all 800 images using standard image quality metrics – Structural Similarity Index (SSIM) and Peak Signal-to-Noise Ratio (PSNR).

For each image, we computed SSIM and PSNR between:

1. the decrypted image (after HE-based brightness adjustment), and
2. the plaintext-adjusted image (brightness increased using standard numpy operations).

Table 4 presents the average scores across all tested images.

Table 4. Average SSIM and PSNR between decrypted and plaintext-adjusted images (800 images).

Scheme	SSIM (avg)	PSNR (avg)
CKKS	0.9991	48.7 dB
BGV	1.0000	59.9 dB
TFHE	1.0000	60.0 dB

These results indicate that the decrypted images are nearly identical to their plaintext counterparts. The CKKS scheme, while designed for approximate arithmetic, still achieved high fidelity with minimal perceptual loss (SSIM > 0.999 and PSNR > 48 dB), which is well within acceptable bounds for visual quality.

Note on TFHE inclusion It is important to emphasize that the TFHE scheme is not inherently designed for real-number arithmetic, but rather for efficient computation on Boolean circuits. While CKKS and BGV natively support arithmetic operations over integers or real numbers, TFHE operates at the bitwise level and requires significant overhead to emulate arithmetic logic.

In this study, we included TFHE in our benchmarking for completeness and consistency across homomorphic encryption schemes supported by the TenSEAL

framework. However, we acknowledge that TFHE is not a natural fit for tasks like brightness adjustment, which involve direct arithmetic on pixel intensities. The TFHE results presented should therefore not be interpreted as competitive benchmarks but rather as illustrative of the challenges and inefficiencies that arise when using Boolean-oriented FHE schemes for real-valued tasks.

Future studies may focus on comparing TFHE within the scope of logic-based image processing tasks, such as encrypted thresholding or binary masking, where its design advantages would be more apparent.

This confirms the accuracy and reliability of the homomorphic encryption pipeline in preserving visual integrity during image processing.

Finally, TFHE, which specializes in Boolean operations, showed limitations with slower encryption and decryption times (70 and 60 seconds, respectively) and the highest memory usage, peaking at 510 MB.

The CKKS scheme has several characteristics that make it particularly well-suited for image processing tasks, such as brightness adjustments, in homomorphic encryption:

Designed for real-number computations CKKS supports approximate arithmetic on real numbers, making it ideal for image processing tasks like brightness adjustments, where small approximations are acceptable.

Efficiency CKKS is generally faster for arithmetic operations on floating-point numbers, as it is optimized for these types of computations. This efficiency allows for faster processing in homomorphic encryption applications.

These results indicate that CKKS is best suited for image processing tasks requiring approximate arithmetic on real-number data, while TFHE could be effective for applications focused on Boolean computations. BGV, though effective, may require optimization for tasks involving complex computations.

6.2. Limitations and future considerations

Noise growth and bootstrapping During experimentation, noise growth in CKKS and BGV was indirectly managed through conservative parameter selection. In particular, the CKKS scheme used a modulus degree of 8192 and a coefficient modulus chain of $[60, 40, 40, 60]$, which provided ample noise budget for the relatively shallow computation (i.e., a single homomorphic addition per pixel). As expected, we did not encounter any noise budget exhaustion in CKKS.

For BGV, noise accumulation was more prominent due to its use of modular arithmetic and less tolerance for additive depth. However, since the operation involved only one level of addition, bootstrapping was not required. Instead, the scheme was configured with parameters sufficient to handle the fixed-depth operation. No runtime bootstrapping was performed in either CKKS or BGV experiments.

Cloud latency and transfer overhead This study focuses on the cryptographic and computational performance of the HE schemes in a simulated cloud setting using OpenStack. While file transfers were executed between host and VM instances, transfer latency and bandwidth profiling were not part of the experimental scope. We acknowledge that in real-world deployments, network upload/download speed can become a significant bottleneck – especially for large encrypted payloads – and should be profiled in future deployment-focused studies.

Scalability and resolution limitations All experiments in this study were performed on 512×512 RGB images, selected to strike a balance between image complexity and processing/memory feasibility in constrained environments. High-resolution images (e.g., $4K$ medical scans or surveillance footage) introduce challenges due to the large ciphertext size and memory footprint after encryption.

Future work will investigate scaling strategies, including ciphertext batching, hardware acceleration (e.g., GPU offloading), and chunked processing for larger images. We also aim to profile system behavior under higher resolutions and longer operation pipelines, where noise growth and bootstrapping may become relevant.

7. Broader cryptographic and computational implications

Beyond the practical performance comparison of the BGV, CKKS, and TFHE homomorphic encryption (HE) schemes, this study intersects with broader areas of cryptographic research and computational theory. Two particularly relevant directions are the post-quantum security of HE schemes and their connections to graph-theoretical and logical modeling, both hold promise for future research and applications [9, 12, 16, 21].

7.1. Post-quantum security and the role of LWE

The CKKS scheme, along with many modern homomorphic encryption systems, is based on the Learning With Errors (LWE) problem [16]. LWE is known for its hardness even in the presence of quantum adversaries, making CKKS and related schemes promising candidates for post-quantum cryptography. This is especially important in emerging quantum computing technologies, where traditional cryptographic protocols such as RSA and ECC are vulnerable to quantum attacks via Shor's algorithm [18].

The Learning With Errors (LWE) problem is considered as computationally intractable as specific worst-case lattice problems [12], which are widely believed to remain resistant even to quantum algorithms, making LWE-based cryptographic schemes strong candidates for post-quantum security. The National Institute of Standards and Technology (NIST) has acknowledged this by releasing finalized

post-quantum encryption standards in 2024 [10], further validating the relevance of LWE-based encryption shortly.

Additionally, recent developments have introduced noise-resilient homomorphic encryption frameworks specifically tailored for secure healthcare data processing [17], demonstrating how HE continues to evolve in line with practical needs for both security and performance.

7.2. Graph-theoretical modeling of HE computation pipelines

From a computational theory perspective, the structure of homomorphic encryption workflows can be modeled using directed acyclic graphs (DAGs), where nodes represent encryption, transformation, or decryption operations, and edges denote the data flow between them. In our implementation, for instance, the encryption of RGB channels, homomorphic brightness adjustment, and decryption can each be interpreted as transformation steps within a DAG pipeline [9, 21].

This graph-based abstraction opens the door to optimization strategies inspired by scheduling and dependency analysis in graph theory. Such representations may help design HE computation workflows that are better suited to the physical topologies of distributed systems or even future quantum hardware architectures. Notably, an efficient graph encryption scheme for secure shortest path queries was proposed in 2024, demonstrating how graph theory continues to shape secure computation [9, 21].

7.3. Future potential for quantum implementation

Although our current work is conducted entirely on classical hardware, it raises the intriguing possibility of exploring homomorphic encryption implementations on quantum computing platforms. In principle, if certain HE operations can be reformulated using quantum circuits, this could enable secure encrypted computing with exponential parallelism. While this remains a largely unexplored domain, early-stage research into quantum secure multiparty computation [5], quantum lattice-based cryptography [1], and more recently, the convergence of post-quantum cryptography with quantum artificial intelligence [13] suggests that the intersection of HE and quantum computing is a compelling direction for interdisciplinary investigation.

8. Conclusion

This study demonstrates the practical feasibility of applying homomorphic encryption (HE) to real-world image processing tasks in cloud environments. By implementing brightness adjustment on encrypted 512×512 RGB images using three HE schemes – BGV, CKKS, and TFHE – within an OpenStack-based cloud infrastructure and TenSEAL library, we evaluated performance in terms of encryption time, computation overhead, memory usage, and output quality.

The results indicate that the CKKS scheme offers the best balance of performance and accuracy for image processing tasks involving real-number operations. CKKS demonstrated the fastest processing times and lowest memory usage, while maintaining high visual fidelity in decrypted outputs (average SSIM > 0.999). Its support for approximate arithmetic makes it well-suited for tasks such as brightness adjustment, filtering, and potentially even encrypted machine learning inference.

The BGV scheme, while less efficient, remains a viable alternative for operations requiring exact integer arithmetic and offers strong theoretical guarantees. However, its susceptibility to noise growth and greater memory demands limit its practicality in low-resource settings unless carefully tuned.

In contrast, the TFHE scheme, optimized for Boolean logic, was found to be inefficient for arithmetic-heavy tasks like brightness adjustment. While included in this study for completeness, its strengths would be better leveraged in logic-based operations such as encrypted thresholding or access control.

Beyond computational metrics, this study highlights the importance of selecting appropriate HE schemes based on task characteristics, security requirements, and available resources. It also illustrates that privacy-preserving image processing in untrusted cloud environments is not only theoretically possible but practically achievable using modern HE frameworks and cloud infrastructure.

Future work will explore broader image processing operations (e.g., edge detection, denoising), support for higher image resolutions and video streams, and the integration of transfer latency and throughput measurements to assess full pipeline performance. Additional work will also examine the impact of noise growth in deeper pipelines and the potential use of bootstrapping where necessary.

References

[1] G. Alagic et al.: *Status report on the first round of the NIST post-quantum cryptography standardization process*, tech. rep. NIST IR 8309, National Institute of Standards and Technology, 2020, doi: 10.6028/NIST.IR.8309.

[2] M. Albrecht, S. Bai, et al.: *Homomorphic Encryption Security Standard*, https://homomorphicencryption.org/standard/, 2018.

[3] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, A. Yun: *Batch Fully Homomorphic Encryption over the Integers*, in: Advances in Cryptology – EUROCRYPT 2013, vol. 7881, Lecture Notes in Computer Science, Springer, 2013, pp. 315–335, doi: 10.1007/978-3-642-38348-9_20, url: https://link.springer.com/chapter/10.1007/978-3-642-38348-9_20.

[4] I. Chillotti, N. Gama, M. Georgieva, M. Izabachène: *TFHE: Fast Fully Homomorphic Encryption over the Torus*, in: International Conference on Cryptographic Hardware and Embedded Systems, Springer, 2016, pp. 609–629.

[5] A. Coladangelo, A. B. Grilo, S. Jeffery, T. Vidick: *Secure multi-party quantum computation with a dishonest majority*, in: Proceedings of the 2020 ACM Symposium on Theory of Computing, 2020, pp. 391–404, doi: 10.1145/3357713.3384296.

[6] M. van Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan: *Fully Homomorphic Encryption over the Integers*, in: Advances in Cryptology – EUROCRYPT 2010, vol. 6110, Lecture Notes in Computer Science, Springer, 2010, pp. 24–43, doi: 10.1007/978-3-642-13190-5_2, url: https://link.springer.com/chapter/10.1007/978-3-642-13190-5_2.

[7] L. Ducas, T. Prest, M. Simkin: *A New Perspective on Key Switching for BGV-like Schemes*, Cryptology ePrint Archive, Paper 2023/1642, 2023, URL: https://eprint.iacr.org/2023/1642.

[8] N. Howgrave-Graham: *Approximate Integer Common Divisors*, in: Cryptography and Lattices Conference (CaLC 2001), vol. 2146, Lecture Notes in Computer Science, Springer, 2001, pp. 51–66, DOI: 10.1007/3-540-44670-2_6, URL: https://link.springer.com/chapter/10.1007/3-540-44670-2_6.

[9] X. Meng, S. Kamara, K. Nissim, G. Kollios: *GRECS: Graph Encryption for Approximate Shortest Distance Queries*, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS), 2015, pp. 504–517, DOI: 10.1145/2810103.2813651.

[10] National Institute of Standards and Technology: *NIST Releases First 3 Finalized Post-Quantum Encryption Standards*, https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards, Accessed: April 17, 2025, 2024.

[11] OpenStack Documentation: *Kolla-Ansible Quickstart*, https://docs.openstack.org/kolla-ansible/2023.1/user/quickstart.html, Accessed: Nov. 2, 2024, 2024.

[12] C. Peikert: *Public-key cryptosystems from the worst-case shortest vector problem: extended abstract*, in: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, 2009, pp. 333–342, DOI: 10.1145/1536414.1536461.

[13] Post-Quantum: *Post-Quantum Cryptography (PQC) Meets Quantum Artificial Intelligence (QAI)*, https://postquantum.com/post-quantum/post-quantum-cryptography-pqc-meets-quantum-artificial-intelligence-qai/, Accessed: April 17, 2025, 2024.

[14] PyPI: *TenSEAL*, https://pypi.org/project/tenseal/0.1.0a0/, Accessed: Nov. 2, 2024, 2024.

[15] Red Hat Documentation: *Red Hat OpenStack Platform Architecture Guide*, https://docs.redhat.com/en-us/documentation/red_hat_openstack_platform/10/pdf/architecture_guide/Red_Hat_OpenStack_Platform-10-Architecture_Guide-en-US.pdf, Accessed: Nov. 2, 2024, 2024.

[16] O. Regev: *On lattices, learning with errors, random linear codes, and cryptography*, in: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 84–93, DOI: 10.1145/1060590.1060603.

[17] J. Scheibner, M. Ienca, E. Vayena: *Health data privacy through homomorphic encryption and distributed ledger computing: an ethical-legal qualitative expert assessment study*, BMC Medical Ethics 23.1 (2022), p. 121, DOI: 10.1186/s12910-022-00852-2.

[18] P. W. Shor: *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal on Computing 26.5 (1997), pp. 1484–1509, DOI: 10.1137/S0097539795293172.

[19] N. P. Smart, F. Vercauteren: *Fully Homomorphic SIMD Operations*, 2011, URL: https://eprint.iacr.org/2011/133 (visited on 11/02/2024).

[20] Z. Team: *TFHE-rs: A Pure Rust Implementation of TFHE*, https://www.tfhe.com, Accessed: April 17, 2025, 2024.

[21] X. Xiao, Y. Yang, K. Yi: *SecGDB: Graph Encryption for Exact Shortest Distance Queries with Efficient Updates*, in: Data and Applications Security and Privacy XXXI (DBSec 2017), vol. 10359, Lecture Notes in Computer Science, Springer, 2017, pp. 87–103, DOI: 10.1007/978-3-319-70972-7_5.

[22] Y. Zhang, M. Wang, X. Lin, B. Sunar: *Homomorphic Multiple Precision Multiplication for CKKS and Applications*, Cryptology ePrint Archive, Paper 2023/1788, 2023, URL: https://eprint.iacr.org/2023/1788.

[23] Y. Zhou, Y. Zhang, J. Li: *Improved Homomorphic Evaluation for Hash Function Based on TFHE*, Cybersecurity 7.1 (2024), pp. 1–16, DOI: 10.1186/s42400-024-00204-0, URL: https://cybersecurity.springeropen.com/articles/10.1186/s42400-024-00204-0.