CONVOLUTION-FRIENDLY IMAGE COMPRESSION IN FHE

Axel Mertens[⋆] *Georgio Nicolas*[⋆] *Sergi Rovira*[†]

[⋆] KU Leuven - Cosic
[†] Pompeu Fabra University - WiSeCom

1. ABSTRACT

Fully Homomorphic Encryption (FHE) is a powerful tool that brings privacy and security to all sorts of applications by allowing us to perform additions and multiplications directly on ciphertexts without the need of the secret key. Some applications of FHE that were previously overlooked but have recently been gaining traction are data compression and image processing. Practically, FHE enables applications such as private satellite searching, private object recognition, or even encrypted video editing.

We propose a practical FHE-friendly image compression and processing pipeline where an image can be compressed and encrypted on the client-side, sent to a server which decompresses it homomorphically and then performs image processing in the encrypted domain before returning the encrypted result to the client.

Inspired by JPEG, our pipeline also relies on discrete cosine transforms and quantization to simplify the representation of an image in the frequency domain, making it possible to effectively use a compression algorithm. This pipeline is designed to be compatible with existing image-processing techniques in FHE, such as pixel-wise processing and convolutional filters. Using this technique, a high-definition (1024×1024) image can be homomorphically decompressed, processed with a convolutional filter and re-compressed in under 24.7s, while using 8GB memory.

2. INTRODUCTION

Data compression is a crucial subfield within the domain of data science, playing a pivotal role in efficiently managing the flood of data generated and processed daily. Its significance is clear in diverse applications such as internet browsing, telecommunications, and data storage. Recent developments in Fully Homomorphic Encryption (FHE) have opened the door to new privacy-preserving applications which use data compression such as private satellite searching and object recognition. FHE allows us to perform additions and multiplications directly on ciphertexts without the need of the secret key. Unfortunately, using FHE has the inherent drawback of ciphertext expansion - a blowup in the size of a plaintext when encrypted. Therefore, adding privacy to data compression and processing applications requires a robust data compression algorithm tailored for FHE.

This paper takes a big step in this direction, specifically for the homomorphic compression and processing of images. Possible applications of our work include satellite image-search, private object detection, facial recognition, etc. In this paper, we propose a compression scheme for the encryption of natural images in CKKS, that uses a modification of the method used in the JPEG-1 standard [1]. Our pipeline is as in figure 1, where an image is compressed and encrypted before being sent to a server for processing. The server performs decompression, processing and compression all in the encrypted domain, before sending it back to the client for decryption. The suggested scheme is designed to be compatible with server-side image processing with convolutional filters and pixel-wise image processing. The compression (decompression, resp.) algorithm is associative with encryption (decryption, resp.) algorithm, improving client-side performance.

2.1. Related Work

Very little research can be found on the topic of compression and decompression of encrypted images. There is more work on the related topic of ciphertext compression, where the ciphertext is compressed without taking into account the underlying plaintext data. This technique is independent of the scope of this paper, so both techniques can be used in the same applications.

In 2017, Canard et al. [2] analyzed the execution of a run-length coding scheme in FHE. The authors came to the conclusion that homomorphic run-length coding is impractical, due to the large number of comparisons it requires. They further argue that lossless compression in FHE cannot be guaranteed, stating that any lossless compression algorithm has a lousy worst-case behavior. In FHE, every algorithm achieves its worst-case behavior. In a followup work [3], the authors suggest an FHE-friendly homomorphic compression scheme. Unfortunately, they do not describe how to encode a compressed image nor how to perform processing on it. Moreover, their method results into images with less quality than those produced

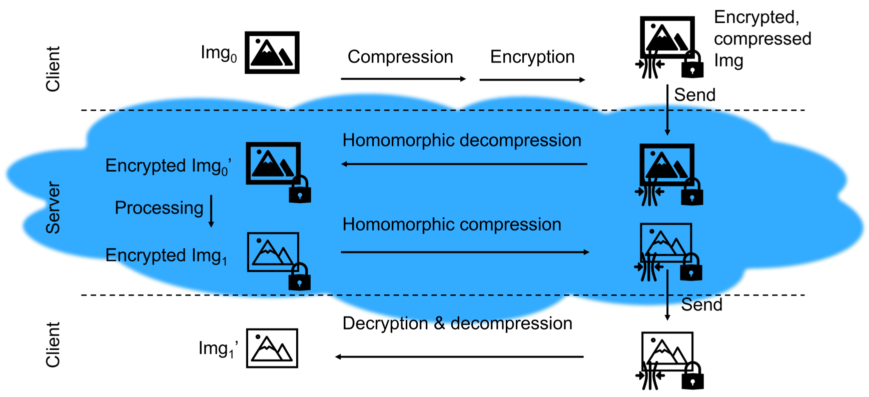

Fig. 1. Sketch of our pipeline

by our compression method.

In [4], the authors suggest a technique using invertible Bloom filters to compress an encrypted sparse array, with a joint decompression - decryption function. However, they suggest no solution for an encrypted decompression. While their method is more general than the one suggested in this paper, ours is more effective for images. We require fewer operations and achieve a higher compression rate. In [5], the author studies the leakage in systems that have both compression and encryption. He describes a number of attacks that use only the compression ratio as a side-channel. This study serves as a warning that for perfect privacy, the compression ratio should be secret.

3. FULLY HOMOMORPHIC ENCRYPTION

Since the first FHE scheme [6], research in FHE has seen rapid growth, leading to a wide range of schemes built from different techniques and security assumptions [7–10]. In this section we only provide the definition of a public-key HE scheme and give a bit more details regarding CKKS [9], the scheme used in this work. We refer the interested reader to the numerous surveys on the topic, for example [11].

Informally, a public-key *Fully Homomorphic Encryption* (FHE) scheme is an encryption scheme which allows us to perform arbitrary computations over encrypted inputs without decrypting them first. That is, given a ciphertext ct encrypting a plaintext m and an arbitrary function f, we can obtain a new ciphertext ct' encrypting $f(m)$ without decrypting ct. It is important to remark that HE schemes can only directly compute additions (XOR) and multiplications (AND) between ciphertexts. More complex functions are represented as arithmetic (boolean) circuits built from these two basic operations. The encryption function of most FHE schemes adds a random element to the message, referred to as *error* or *noise* in the literature. When a multiplication or an addition is performed, this error increases. After a given number of operations, the error is too big and the message cannot be recovered by the decryption algorithm. To solve this, an additional algorithm called *bootstrapping* is added to the scheme. This procedure reduces the noise of a ciphertext, allowing further operations. Unfortunately, bootstrapping is a costly operation. Therefore, it is common practice to set parameters of a HE scheme in such a way that the we can evaluate a function of our choice without the need of bootstrapping. This is called *leveled* FHE and is the approach that we take in this work.

Definition 3.1 (Homomorphic Encryption scheme) *A public-key homomorphic encryption scheme $\mathcal{E}$ consists of a set of probabilistic polynomial-time algorithms* (KeyGen, Enc, Dec, Eval) *such that:*

- $\mathsf{KeyGen}(1^\lambda)$*: outputs the secret key sk, the public key pk and the evaluation key evk given the security parameter λ. The evaluation key is also public, and it is used to perform the homomorphic operations over ciphertexts.*
- $\mathsf{Enc}_{pk}(m)$*: Outputs a ciphertext ct encrypting m under the public key pk.*
- $\mathsf{Dec}_{sk}(ct)$*: Outputs a message m. If the algorithm cannot recover m from ct, the output is $\perp$.*
- $\mathsf{Eval}_{evk}(f; ct_1, \dots, ct_n)$*: Given a function f and a list of ciphertexts $ct_1, \dots, ct_n$, outputs a ciphertext ct_f such that $\mathsf{Dec}_{sk}(ct_f) = f(m_1, \dots, m_n)$, where $ct_i \leftarrow \mathsf{Enc}_{pk}(m_i)$ for all $i \in \{1, \dots, n\}$.*

Among the most popular FHE schemes available today [7,9,12–14], we have chosen CKKS [9] since it allows us to compute directly on floating-point numbers, which is crucial for the fast evaluation of our compression and decompression algorithms. Furthermore, this allows us to work with non-integer kernels. The other key feature of CKKS is ciphertext packing. We expand on our use of ciphertext packing in Section 4.2.

3.1. Background on CKKS

The CKKS scheme [9] is an approximate homomorphic encryption scheme. The key difference between CKKS and other schemes, such as BGV [12] or TFHE [14], is that the decryption function of CKKS does not remove the encryption noise. This makes the decryption structure to be $m + e \mod q$ where m is the encrypted message, e is the encryption error and q a suitable modulus. If e is small enough compared to m, it would only alter the least significant bits of m. Another characteristic of CKKS is the native support for ciphertext packing, which allows packing multiple encrypted messages into a single ciphertext. More importantly, it provides the capability of performing computations between ciphertexts in a SIMD manner, reducing both effective memory and computational complexities.

4. OUR TECHNIQUE

When using lossy compression, data is represented more compactly, with the aim of retaining as much fidelity as possible to the original before compression. In the case of images, there are different methods to measure the level of fidelity or closeness. The measure closest to how the human eye perceives images is the Structural Similarity Index (SSI) [15]:

Definition 4.1 (Structural Similarity Index (SSI)) *SSI is defined as:* $SSI(x, y) = \frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)}$ *where μ_i is the pixel sample mean of image i, σ_i^2 is the variance of image i, σ_{xy}^2 is the covariance of images x and y, $c_1 = (0.01L)^2$, $c_2 = (0.03L)^2$, with L the dynamic range of the pixel values.*

It compares an imperfect image to its perfect version, and is a value between -1 and 1, where 1 is a perfect match, -1 is a perfectly negative correlation, and 0 means the two images are very different. The SSI is typically good at detecting structural deformations (stretching, rotation, etc) and other degradations such as blurriness or blockiness. Images with values of 0.95 are considered to be of good quality [16].

Another important metric is the compression ratio:

Definition 4.2 (Compression ratio) *Let B_0 be the number of bytes needed to represent some data X, and let B_1 be the number of bytes needed for the same, compressed data X'. The compression ratio is then $\frac{B_0}{B_1}$, also written as B_0:B_1.*

The ideal compression scheme has a high compression ratio, while maintaining an SSI that is close to 1.

Most traditional compression algorithms used in the clear are unfit for FHE. They often contain comparisons and do not have a constant run-time. FHE inflicts its own special set of requirements upon an algorithm. The most important property of a homomorphic compression scheme is that no additional data or metadata about the image leaks, even through side-channels such as the compression ratio or running-time. Secondly, the compression and decompression should be efficient. Thirdly, the ciphertext packing should be done in a way that a ciphertext structurally reflects the compression of a plaintext. With some inefficient packing methods, the ciphertext length would not be influenced by a (slightly) shorter plaintext. And last but not least, it should be possible to both compress and decompress the message without needing to decrypt it.
As an extra requirement, we also want the encrypted, decompressed form to support efficient image processing. The type of image processing we choose to focus on is using convolutional filters, but we also look at pixel-wise processing methods. Furthermore, our technique can also handle coloured images, both in the RGB format as in the YCbCr format, depending on the application.

4.1. Compression Algorithm.

In this section we introduce our algorithms for compression and decompression. We want to emphasize that we present them in unencrypted format, as the client will run them. The reader should keep in mind that the server will execute them in the encrypted domain. Specifically, in the server side, the image data ($\mathbf{a}$, $\mathbf{A}$, $\mathbf{B}$) is encrypted, while the sizes of vectors and matrices (m, n, c) remain public. Moreover, the computations are done in the encrypted domain.

The algorithm we suggest is based on the JPEG-standard.

As shown in Algorithm 2, we split the image into $m \times m$-sized blocks. We first center every pixel value in the $[-128, 128[$ interval. We call this operation $Level()$, while the inverse operation is called $Unlevel()$. Then, we apply the discrete cosine transform (DCT), which is done via two fixed-point matrix multiplications with DCT matrix T_m. This gives us representations of the frequency spectrum of each block of the image. The next step, still following the JPEG standard, is to apply a quantization, which scales each element of this frequency representation (lines 4-6 of alg. 2). The quantization matrix Q_m is defined through exhaustive testing and standardization. Hence, we are bound to the quantization matrices existing in prior research. We have only found a reliable matrix for $m = 8$ and a slightly less mature matrix for $m = 16$ [17].

Quantization results in a large number of zeros, especially towards the bottom right corner of each matrix. To exploit this redundancy, a *Zigzag* traversal is applied, turning the matrix into a one-dimensional vector, with these zeroes at the end. The function $Inverse\_zigzag()$ turns a one-dimensional vector back into a matrix, following the same zigzag pattern.

The next step is where we really diverge from the JPEG standard. JPEG uses Huffman encoding to represent this vector. Huffman coding,

like all other encoding schemes, works on the principle that more frequent symbols are represented with shorter codes compared to rarer symbols, resulting in an overall compression. This principle is unsuited for FHE, where consistency and constant-runtime are desired above all.

The encoding we suggest (see Algorithm 1) is more simple: we only keep a constant number c of non-zero elements of every block, discarding all other elements. Ideally, this number should be determined independent of the image, such as not to reveal any information about the level of texture present in the data. However, in practice, c leaks very little information about the image. It should be noted that predetermining this c can result in non-zero values being discarded. This causes additional artifacts (blockiness) in the image, but the effect is usually minimal.

Using this encoding, we make sure that every block of the image undergoes the same compression ratio. Furthermore, both compression and decompression are very simple actions, and can be done efficiently in FHE. The encoding itself is lossless, but can be lossy for some highly textured or discrete-tone images. Granted, the data will be compressed less than it would using a more evolved coding scheme (Huffman, Golomb, ...). As discussed earlier, this approach is impractical with homomorphic encryption.

Decompression works very similarly to compression, knowing that in the decoding, we pad with encryptions of 0 until we reach the required size.

Algorithm 1: Encode

```
    Input: blocks $\mathbf{a} = [a_0, a_1, \ldots, a_{n-1}]$ with
           $|a_i| = l_a$
    Input: Cutoff point $c < m^2$
    Output: blocks $\mathbf{b} = [b_0, b_1, \ldots, b_{n-1}]$ with
           $|b_i| = l_b$ and $l_a \geq l_b$
1   for $i \in [0, n-1]$ do
2       $b_i \leftarrow a_i[0:c]$ ;
3   end
4   return $\mathbf{b}$
```

4.2. Ciphertext packing.

In order to make our algorithms more efficient we exploit the packing capabilities of CKKS [9]. Recall that ciphertext packing refers to the ability to encrypt multiple messages in a single ciphertext.

In our work, we use ciphertext packing as follows. First, the image is divided into blocks of size $m \times m$, after which each block separately undergoes compression. Then, each position in a block is encrypted in a separate ciphertext. We now have c ciphertexts, one for the first position in a block, one for the second, etc. Therefore, if $m = 8$ for example, a decompressed image is represented using 64 ciphertexts, regardless of the dimension of the image. Figure 2 illustrates this principle without compression (for clarity). Furthermore, with this design every block-wise operation is automatically parallelized.

Algorithm 2: Compress

```
    Input: Encrypted image $\mathbf{A}$: split into $n$ $m \times m$
           blocks: $\mathbf{A} = [A_0, A_1, \ldots, A_{n-1}]$ with
           $A_i \in \mathbb{Z}^{m \times m}$
    Input: Cutoff point $c < m^2$
    Output: Encrypted, compressed image
           $\mathbf{B} = [b_0, b_1, \ldots, b_{n-1}]$ with $b_i \in \mathbb{Z}^c$
1   for $l \in [0, n-1]$ do
2       $C_l \leftarrow Level(A_l)$ ;
3       $D_l \leftarrow T_m C_l T_m^\top$ ;
4       for $(i, j) \in \mathbb{N}_m^2$ do
5           $E_{l,i,j} \leftarrow \lfloor(\frac{D_{l,i,j}}{Q_{m,i,j}})\rceil$ ;
6       end
7       $\mathbf{b_l} \leftarrow Zigzag(E_l)[0..c]$ ;
8   end
9   return $\{b_0, b_1, \ldots, b_{n-1}\}$
```

Observe that it is also possible to compress and process multiple images at a time. Compressing k images at once, there will be the same number of ciphertexts, but they will be k times longer.

Our packing technique is not unlike others in the literature. However, it is worth noting some differences. In CaReNets [18], images are combined to completely fill the ciphertexts, and leave no slots empty. While this makes the ciphertext more compact, it is not suitable for our situation where the compression and the processing are highly parallelized per block. The technique used in [19] is closer to our method, the only difference being that they create one ciphertext per location/pixel in an image, needing as many ciphertexts as one image has pixels. Contrarily, we create a ciphertext per frequency in a block, which is greatly fewer ciphertexts albeit longer ones.

4.3. Image processing.

The server needs to be able to perform convolutions on the encrypted (but decompressed) blocks of the image. This is hard to parallelize, as applying the convolution requires neighbouring pixel values, regardless of block borders. We suggest the following solution: as the very first step, before the client even encrypts/compresses the image, the client splits the image into blocks of size $(m-2) \times (m-2)$. Then, it pads these smaller blocks on the edges with the neighbouring cells, such that they end up with overlapping blocks of size $m \times m$. This system works for 3×3 kernels, which covers almost all convolutional filters used in practice. Recall that CKKS also provides supports for non-integer kernels.

This obviously reduces the compression ratio, but it does allow efficient image processing on the server, and avoids costly server-side packing operations, where the ciphertexts would have to be reconfigured. Also, it is worth noting that while we strive for a good compression ratio, the goal of this paper is not to achieve the optimal compression ratio. It rather is to provide a trade-off between

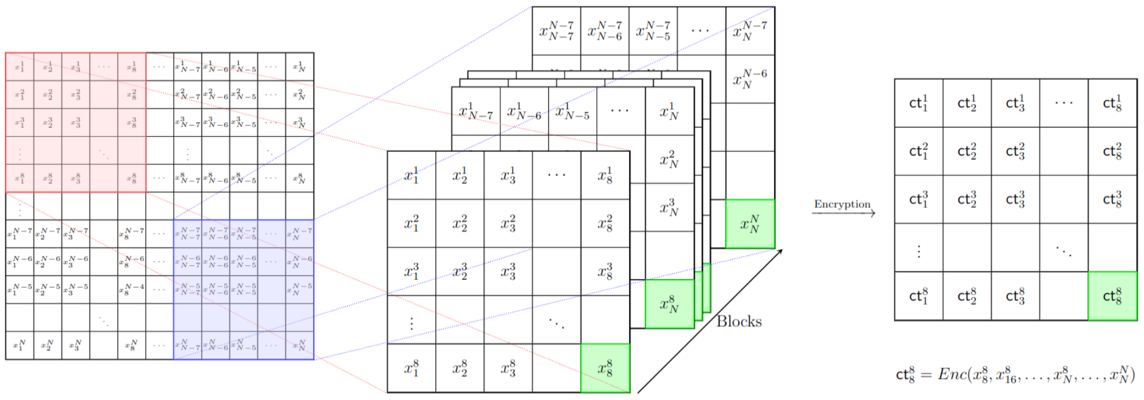
Encryption
Blocks
$\mathsf{ct}_8^8 = Enc(x_8^8, x_{16}^8, \ldots, x_N^8, \ldots, x_N^N)$

Fig. 2. Illustration of our ciphertext packing

```
Algorithm 3: Decompress
    Input: Block size $m$, which can be equal to
           either 8 or 16
    Input: Encrypted, compressed image
           $\mathbf{B} = [b_0, b_1, \ldots, b_{n-1}]$ with $b_i \in \mathbb{Z}^c$
    Output: Encrypted image
           $\mathbf{A} = [A_0, A_1, \ldots, A_{n-1}]$ with
           $A_i \in \mathbb{Z}^{m \times m}$
1  for $l \in [0, n-1]$ do
2      for $i \in [c, m^2[$ do
3          $b_{l,i} \leftarrow Enc(0)$
4      end
5      $C_l \leftarrow Inverse\_Zigzag(b_l)$ ;
6      for $(i, j) \in \mathbb{N}_m^2$ do
7          $D_{l,i,j} \leftarrow C_{l,i,j} \cdot Q_{l,i,j}$ ;
8      end
9      $E_l \leftarrow T_m^\top D_l T_m$ ;
10     $A_l \leftarrow Unlevel(E_l)$ ;
11 end
12 return $[A_0, A_1, \ldots, A_{n-1}]$
```

the compression ratio and efficient functionality. We have found that for this setting, keeping the SSI close to 0.95, most images require $c_8 = 30$ for $m = 8$, leading to a compression ratio of $100 : 83.3$. For $m = 16$, most images require $c_{16} = 70$, leading to a compression ratio of $100 : 35.7$. The simplest image processing actions, such as inversion and brightening, are pixel-based. They do not require any knowledge of neighbouring pixels, and thus do not require this whole padding trick. In these cases, the client can simply split the image in blocks of size $m \times m$. Here, we find that most images require $c_8 = 22$ (respectively $c_{16} = 63$) to keep the SSI ≥ 0.95, leading to a compression ratio of $100 : 34.4$ (respectively $100 : 24.6$).

In reality, the value of c can be determined for a specific image or a set of images by running the compression and decompression code in the clear, and verifying the image quality. The cutoff point could potentially be quite different after processing compared to before, depending on the operation that was performed.

5. RESULTS.

Both client and server were simulated a desktop with an Intel Core i7-13700 and 32gb of RAM. Our implementation[1] has been very moderately optimized, but could still be improved. Nonetheless, we think our numbers are promising as it would be safe to assume that the hardware setup we used is less powerful than a standard "server" and more powerful than a standard client.

We ran tests for four different image dimensions, and four algorithms each. The four algorithms are

1. Pixel-wise processing on 8×8 blocks.
2. Convolutional processing on 8×8 blocks.
3. Pixel-wise processing on 16×16 blocks.
4. Convolutional processing on 16×16 blocks.

Each timing in table 1 is for one 8-bit greyscale image of mentioned dimensions. The values are averaged over a

[1] `https://github.com/icip-24/img-processing-fhe`

few randomly selected images: 8 images of 512×512, 4 images for other dimensions. We used tailor-fitter parameters for each setting. The benchmarks are server-side timings, meaning that the compression and decompression are both done homomorphically. Server-side RAM Consumption varied between 1GB and 13GB based on the size of the image being processed.

The decompression is more costly than the compression, because the server needs to pad the compressed ciphertext with encryptions of zero and process them as well.

It is clear that the pixel-wise processing is highly efficient, but the 3×3 convolution as well is relatively fast. The largest image size we experimented on is considerably less fast, while the other image sizes are not as different. This has everything to do with the choices of CKKS-parameters. The parameter choices for 1024×1024 images do not allow for secure ciphertexts long enough to fit 2048×2048 images. Remember that the length of each ciphertext is equal to the number of blocks in the image.

We notice that the 8×8 techniques (1 and 2) are consistently faster, and keep the image quality the highest. This is expected, because these quantisation matrices are more mature, and because smaller blocks have more efficient matrix multiplication. We do not include example images that have been put through our pipeline, because with SSI-values above 0.9, it is very hard to tell the difference.

6. SUMMARY AND FUTURE DIRECTIONS

In this work, we described a technique to homomorphically compress and process natural images. This is a great first step towards improved and more advanced privacy-preserving image manipulation. We discussed the two types of packing we use, depending on the processing that should be applied (3×3 convolution or pixel-wise processing). Our implementation shows that the techniques using 8×8 blocks are more time-efficient, while the 16×16 blocks achieve more compression. The code we provide is not production ready and is only for research purposes.

The next step would be to move towards more complex image processing, such as face recognition and object detection.

Finally, an interesting question is whether it is possible to prove that entropy encoding is impractical in combination with FHE. If not, it should be possible to further improve the compression ratio we achieve.

Acknowledgement

This work was supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. FA8750-19-C-0502 and by the FWO under an Odysseus project GOH9718N. This work was supported by the Spanish Ministry of Economy and Competitiveness with reference number RTI2018-102112-B-I00. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the DARPA, the US Government, Cyber Security Research Flanders or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

7. REFERENCES

[1] International Organisation for Standardisation, “Information technology – Digital compression and coding of continous-tone still images - Requirements and guidelines,” Standard, CCITT, Geneva, CH, Sept. 1992.

[2] Sébastien Canard, Sergiu Carpov, Donald Nokam Kuate, and Renaud Sirdey, “Running compression algorithms in the encrypted domain: a case-study on the homomorphic execution of RLE,” Cryptology ePrint Archive, Report 2017/392, 2017, `https://eprint.iacr.org/2017/392`.

[3] Donald Nokam Kuate, Sébastien Canard, and Renaud Sirdey, “Towards video compression in the encrypted domain: A case-study on the H264 and HEVC macroblock processing pipeline,” in *Cryptology and Network Security - 17th International Conference, CANS 2018, Naples, Italy, September 30 - October 3, 2018, Proceedings*, Jan Camenisch and Panos Papadimitratos, Eds. 2018, vol. 11124 of *Lecture Notes in Computer Science*, pp. 109–129, Springer.

[4] Nils Fleischhacker, Kasper Green Larsen, and Mark Simkin, “Compressing encrypted data over small fields,” Cryptology ePrint Archive, Paper 2023/946, 2023, `https://eprint.iacr.org/2023/946`.

[5] John Kelsey, “Compression and information leakage of plaintext,” in *FSE 2002*, Joan Daemen and Vincent Rijmen, Eds. Feb. 2002, vol. 2365 of *LNCS*, pp. 263–276, Springer, Heidelberg.

[6] Craig Gentry, “Fully homomorphic encryption using ideal lattices,” in *41st ACM STOC*, Michael Mitzenmacher, Ed. May / June 2009, pp. 169–178, ACM Press.

[7] Junfeng Fan and Frederik Vercauteren, “Somewhat practical fully homomorphic encryption,” Cryptology ePrint Archive, Report 2012/144, 2012, `https://eprint.iacr.org/2012/144`.

image size	setting	decompression	processing	compression	total	SSI	compression ratio
256×256	1	8s	< 1s	5s	13s	0.95	$100 : 34.4$
256×256	2	11.3s	4s	9s	24.5s	0.935	$100 : 83.3$
252×252	3	39.5	1s	29s	69.5s	0.91	$100 : 24.6$
252×252	4	60s	19s	62s	131s	0.905	$100 : 35.7$
512×512	1	7.5s	< 1s	5s	13.5s	0.95	$100 : 34.4$
512×512	2	11s	4s	9s	24ss	0.935	$100 : 83.3$
504×504	3	39s	< 1s	29s	68s	0.92	$100 : 24.6$
504×504	4	50s	18.5s	62s	130.5s	0.92	$100 : 35.7$
1024×1024	1	8s	< 1s	5s	13s	0.98	$100 : 34.4$
1024×1024	2	11s	4s	9.7s	24.7s	0.975	$100 : 83.3$
1022×1022	3	41s	< 1s	29s	70s	0.96	$100 : 24.6$
1022×1022	4	61.5s	19s	62s	142.5s	0.97	$100 : 35.7$
2048×2048	1	107s	1s	70s	178s	0.98	$100 : 34.4$
2048×2048	2	107s	36.5s	79s	222.5s	0.98	$100 : 83.3$
2044×2044	3	137s	< 1s	99.5s	236.5s	0.975	$100 : 24.6$
2044×2044	4	136s	41.5s	136s	313.5s	0.975	$100 : 35.7$

Table 1. Experimental results (server-side)

[8] Zvika Brakerski and Vinod Vaikuntanathan, "Lattice-based FHE as secure as PKE," in *ITCS 2014*, Moni Naor, Ed. Jan. 2014, pp. 1–12, ACM.

[9] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT 2017, Part I*, Tsuyoshi Takagi and Thomas Peyrin, Eds. Dec. 2017, vol. 10624 of *LNCS*, pp. 409–437, Springer, Heidelberg.

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, Jan. 2020.

[11] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H. P. Fitzek, and Najwa Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012*, Shafi Goldwasser, Ed. Jan. 2012, pp. 309–325, ACM.

[13] Zvika Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti, Eds. Aug. 2012, vol. 7417 of *LNCS*, pp. 868–886, Springer, Heidelberg.

[14] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *ASIACRYPT 2016, Part I*, Jung Hee Cheon and Tsuyoshi Takagi, Eds. Dec. 2016, vol. 10031 of *LNCS*, pp. 3–33, Springer, Heidelberg.

[15] Z. Wang, E.P. Simoncelli, and A.C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, 2003, vol. 2, pp. 1398–1402 Vol.2.

[16] Thomas Zinner, Oliver Hohlfeld, Osama Abboud, and Tobias Hossfeld, "Impact of frame rate and resolution on objective qoe metrics," in *2010 Second International Workshop on Quality of Multimedia Experience (QoMEX)*, 2010, pp. 29–34.

[17] Adel Almohammad, Gheorghita Ghinea, and Robert M. Hierons, "Jpeg steganography: A performance evaluation of quantization tables," in *2009 International Conference on Advanced Information Networking and Applications*, 2009, pp. 471–478.

[18] Jin Chao, Ahmad Al Badawi, Balagopal Unnikrishnan, Jie Lin, Chan Fook Mun, James M. Brown, J. Peter Campbell, Michael F. Chiang, Jayashree Kalpathy-Cramer, Vijay Ramaseshan Chandrasekhar, Pavitra Krishnaswamy, and Khin Mi Mi Aung, "Carenets: Compact and resource-efficient CNN for homomorphic inference on encrypted medical images," *CoRR*, vol. abs/1901.10074, 2019.

[19] Nayna Jain, Karthik Nandakumar, Nalini K. Ratha, Sharath Pankanti, and Uttam Kumar, "Optimizing homomorphic encryption based secure image analytics," in *23rd International Workshop on Multimedia Signal Processing, MMSP 2021, Tampere, Finland, October 6-8, 2021*. 2021, pp. 1–6, IEEE.