Convolution-friendly Image Compression in FHE

Axel Mertens [1] Georgio Nicolas [1] Sergi Rovira [2]

[1] imec-COSIC, KU-Leuven

[2] WiSeCom, Universitat Pompeu Fabra (UPF)

April 25, 2024

Motivation

- Image processing on a server
  - Server doesn't know the image
  - Client doesn't necessarily know the process
- Compression for efficient data transfer
- Processing with convolutional filters
- Applications:
  - Cloud image processing
  - Secret satelite image search

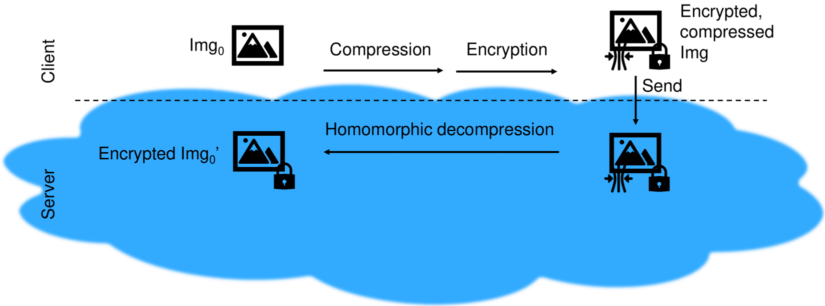
Existing techniques

- Compression:
  - Ciphertext compression: Can be combined with our technique. [MDK23]
  - [FLS23]: More general than for images, but no decompression without decryption. Only works for **very** sparse data.
- Image Processing:
  - Zama's Encrypted Image Filtering: Convolutional filters with Concrete

Pipeline

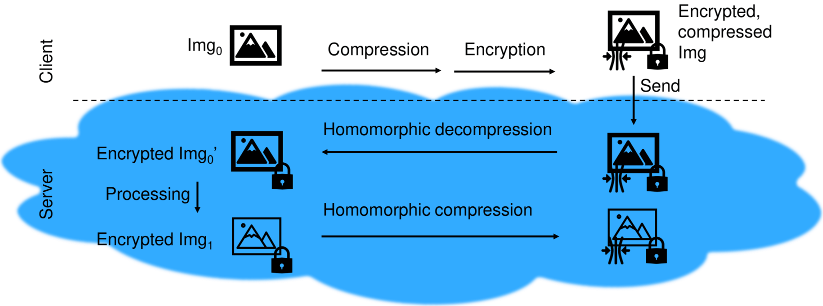

Client
Img_0
Compression
Encryption
Encrypted,
compressed
Img

Pipeline



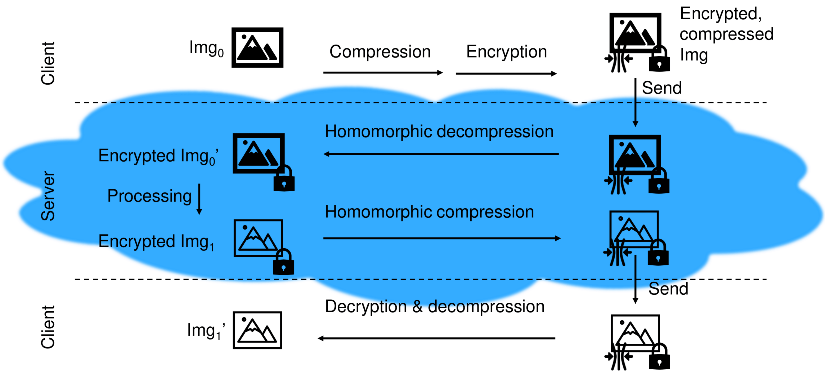
Client
Server
Img_0
Compression
Encryption
Encrypted,
compressed
Img
Send
Homomorphic decompression
Encrypted Img_0'

Pipeline



Client
Img_0
Compression
Encryption
Encrypted, compressed Img
Send
Server
Encrypted Img_0'
Homomorphic decompression
Processing
Homomorphic compression
Encrypted Img_1

Pipeline



Client
Img$_0$
Compression
Encryption
Encrypted, compressed Img
Send
Server
Homomorphic decompression
Encrypted Img$_0$'
Processing
Encrypted Img$_1$
Homomorphic compression
Send
Client
Decryption & decompression
Img$_1$'

Metrics

Metrics:

- Structural Similarity Index: metric for image quality, based on human perception
  $\rightarrow SSI \geq 0.9$ is a good quality.
  $\rightarrow SSI \geq 0.95$ is a very high quality.
- Compression ratio: how many bits are needed to represent 100 (uncompressed) bits

Compression in the clear: JPEG

- Lossy compression
- Natural images
- Can choose degree of compression
- Discrete Cosine Transform (DCT)
- Huffman Coding

1. **Split into blocks (8x8)**
2. Level
3. Discrete Cosine Transform
4. Quantization
5. Lossless encoding

154	123	123	123	123	123	123	136
192	180	136	154	154	154	136	123
254	198	154	154	180	154	123	123
239	180	136	180	180	166	123	123
180	154	136	167	166	149	136	136
128	136	123	136	154	180	198	154
123	105	110	149	136	136	180	166
110	136	123	123	123	136	154	136

Figure: The original image (block)

1. Split into blocks (8x8)
2. **Level**
3. Discrete Cosine Transform
4. Quantization
5. Lossless encoding

$M_{i,j} \in [0, 255]$ *to* $M_{i,j} \in [-128, 127]$

26	-5	-5	-5	-5	-5	-5	8
64	52	8	26	26	26	8	-18
126	70	26	26	52	26	-5	-5
111	52	8	52	52	38	-5	-5
52	26	8	39	38	21	8	8
0	8	-5	8	26	52	70	26
-5	-23	-18	21	8	8	52	38
-18	8	-5	-5	-5	8	26	8

Figure: Leveled image block

JPEG Compression (3/5)

1. Split into blocks (8x8)
2. Level
3. **Discrete Cosine Transform**
4. Quantization
5. Lossless encoding

$M^{DCT} = DCT \cdot M \cdot DCT^T$, where DCT is the DCT matrix.

162.3	40.6	20.0	72.3	30.3	12.5	-19.7	-11.5
30.5	108.4	10.5	32.3	27.7	-15.5	18.4	-2.0
-94.1	-60.1	12.3	-43.4	-31.3	6.1	-3.3	7.1
-38.6	-83.4	-5.4	-22.2	-13.5	15.5	-1.3	3.5
-31.3	17.9	-5.5	-12.4	14.3	-6.0	11.5	-6.0
-0.9	-11.8	12.8	0.2	28.1	12.6	8.4	2.9
4.6	-2.4	12.2	6.6	-18.7	-12.8	7.7	12.0
-10.0	11.2	7.8	-16.3	21.5	0.0	5.9	10.7

Figure: DCT of image block

JPEG Compression (4/5)

1. Split into blocks (8x8)
2. Level
3. Discrete Cosine Transform
4. **Quantization**
5. Lossless encoding

$M^{Q}_{i,j} = round(\frac{M_{DCT,i,j}}{Q_{i,j}})$, where Q is a standardized quantization matrix.

10	4	2	5	1	0	0	0
3	9	1	2	1	0	0	0
-7	-5	1	-2	-1	0	0	0
-3	-5	0	-1	0	0	0	0
-2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure: Quantized image block

JPEG Compression (5/5)

1. Split into blocks (8x8)
2. Level
3. Discrete Cosine Transform
4. Quantization
5. **Lossless encoding**

$Zigzag(M^Q)$ transforms the matrix into a vector, now ready for encoding.

10	4	2	5	1	0	0	0
3	9	1	2	1	0	0	0
-7	-5	1	-2	-1	0	0	0
-3	-5	0	-1	0	0	0	0
-2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure: Zigzag of quantized block

1. Split into blocks (8x8)
2. Level
3. Discrete Cosine Transform
4. Quantization
5. **Lossless encoding**

Zigzag(M^Q) transforms the matrix into a vector, now ready for encoding.

10	4	2	5	1	0	0	0
3	0	1	2	1	0	0	0
-7	-5	1	-2	-1	0	0	0
-3	-5	0	-1	0	0	0	0
-2	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure: Zigzag of quantized block

1. Split into blocks (8x8)
2. Level
3. Discrete Cosine Transform
4. Quantization
5. **Lossless encoding**

Zigzag(M^Q) transforms the matrix into a vector, now ready for encoding.

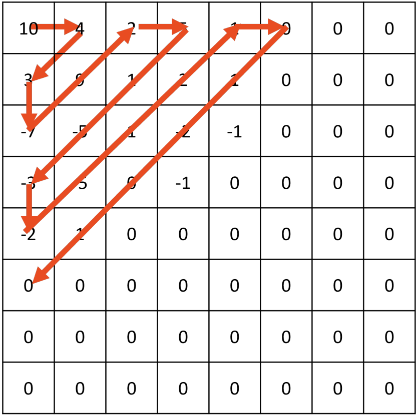


Figure: Zigzag of quantized block

JPEG Compression (5/5)

1. Split into blocks (8x8)
2. Level
3. Discrete Cosine Transform
4. Quantization
5. **Lossless encoding**

Zigzag(M^Q) transforms the matrix into a vector, now ready for encoding.

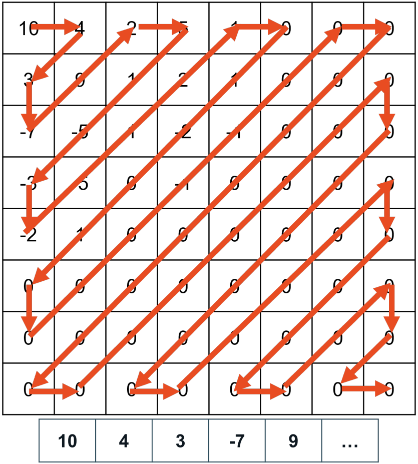


Figure: Zigzag of quantized block

Lossless Encoding

- Data compression uses predictability in the data, such that the entropy of the data increases with compression.
- Encoding example: Huffman
  - Construct a code specific for the data
  - Frequent symbols will get shorter binary representation
  - Rare symbols might get represented longer than they are

  - On average, this results in compression
  - Without data-specific code, compression is weaker
- Issues: Training is expensive + variable length code

JPEG Compression: Summary

1. Split into blocks (8x8)
2. Level
3. Discrete Cosine Transform
4. Quantization
5. Lossless encoding

- Lossy due to quantization
- Encoding uses predictability after quantization

- DCT consists of two matrix multiplications
- Quantization consists of an elementwise matrix multiplication and rounding

Compression and FHE: a difficult marriage

- Compression schemes have bad worst-case behaviour
- Homomorphic encryption intrinsically creates worst-case behaviour
  $\rightarrow$ No variable-length coding without conditional branching
- For privacy, we require constant compression factor and constant algorithm time

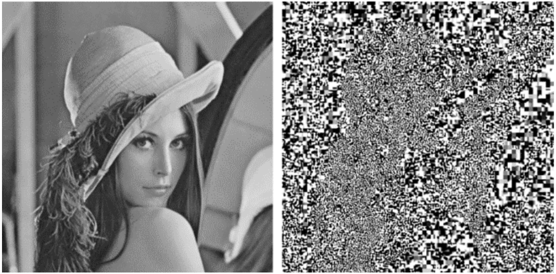


Figure: Information leakage from knowing only the compression factor of each block

Our solution

- Cut off a constant number of ending zeros
- On average: cut off between 30 and 40 values
- Very efficient, constant time, and constant compression

10	4	3	-7	9	2	5	1	-5	-3	-2	-5	1	2	1	0	1	-2	0	1	0	0	0	0	0	0	0	0	0	...

10	4	3	-7	9	2	5	1	-5	-3	-2	-5	1	2	1	0	1	-2	0	1	0	0	0	0	0	0	0	0	0	...

10	4	3	-7	9	2	5	1	-5	-3	-2	-5	1	2	1	0	1	-2	0	**1**	0	0	0	0	0	0	0	0	0	...

Image Processing

Pixel-wise processing

- for the most basic applications
  → inversion, brightening ...
- clear/secret multiplications

Convolutional filters

- different (3x3) kernels for different tasks
  → edge detection, blurring, sharpening ...
- clear/secret multiplications and additions

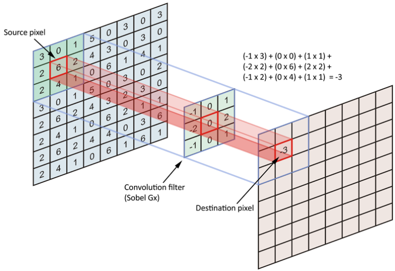


Figure: Illustration of a convolutional filter

Convolution-friendly compression

- Instead of 8x8 blocks, split the image into 6x6 blocks
- Pad each block with the neighbouring pixels
- The 8x8 blocks now have only a 6x6 part that is unique

154	123	123	123	123	123	123	136
192	180	136	154	154	154	136	123
254	198	154	154	180	154	123	123
239	180	136	180	180	166	123	123
180	154	136	167	166	149	136	136
128	136	123	136	154	180	198	154
123	105	110	149	136	136	180	166
110	136	123	123	123	136	154	136

Figure: Convolution on a 6x6 block needs an 8x8 block

Block size

Quantization matrices

- 8x8: well studied, great image quality
- 16x16: used in some video compressions. Slightly lesser quality.

- More compression for 16x16

Quick recap of FHE

- An FHE scheme allows us to perform arbitrary computations over encrypted data.
  - $\mathsf{Enc}(m_1 \triangle m_2) = \mathsf{Enc}(m_1) \triangle \mathsf{Enc}(m_2)$
- We can think of it as a commutative diagram:

$$\begin{tikzcd} \mathcal{C} \arrow[r, "\text{Eval(pk,f,}\cdot\text{)}"] \arrow[d, "\text{Dec(sk,}\cdot\text{)}"'] & \mathcal{C} \arrow[d, "\text{Dec(sk,}\cdot\text{)}"] \\ \mathcal{M} \arrow[r, "\text{f}(\cdot)"'] & \mathcal{M} \end{tikzcd}$$

- State-of-the-art schemes: BGV, B/FV, CKKS and TFHE.

Quick recap of FHE

- An FHE scheme allows us to perform arbitrary computations over encrypted data.
  - $\mathsf{Enc}(m_1 \triangle m_2) = \mathsf{Enc}(m_1) \triangle \mathsf{Enc}(m_2)$
- We can think of it as a commutative diagram:

$$\begin{tikzcd}
\mathcal{C} \arrow[r, "\text{Eval(pk,f,}\cdot\text{)}"] \arrow[d, "\text{Dec(sk,}\cdot\text{)}"'] & \mathcal{C} \arrow[d, "\text{Dec(sk,}\cdot\text{)}"] \\
\mathcal{M} \arrow[r, "\text{f(}\cdot\text{)}"'] & \mathcal{M}
\end{tikzcd}$$

- State-of-the-art schemes: BGV, B/FV, **CKKS** and TFHE.

Properties of CKKS

- Message space: $\mathbb{C}^{N/2}$
- Plaintext space: $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$
- Ciphertext space: $\mathcal{R}_q^2$, where $\mathcal{R}_q := \mathbb{Z}_q[X]/\langle X^N + 1\rangle$
- Native support for floating-point numbers.
- Great packing capabilities

$$\mathbb{C}^{N/2} \xrightarrow{\text{Encode}} \mathcal{R} \xrightarrow{\text{Encrypt}} \mathcal{R}_q^2$$

- It offers parallel computation in a SIMD manner.

Properties of CKKS

- Message space: $\mathbb{C}^{N/2}$
- Plaintext space: $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$
- Ciphertext space: $\mathcal{R}_q := \mathbb{Z}_q[X]/\langle X^N + 1\rangle$
- Native support for floating-point numbers. $\longrightarrow$ easy to implement JPEG
- Great packing capabilities

$$\mathbb{C}^{N/2} \xrightarrow{\text{Encode}} \mathcal{R} \xrightarrow{\text{Encrypt}} \mathcal{R}_q^2$$

- It offers parallel computation in a SIMD manner.

Properties of CKKS

- Message space: $\mathbb{C}^{N/2}$
- Plaintext space: $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$
- Ciphertext space: $\mathcal{R}_q := \mathbb{Z}_q[X]/\langle X^N + 1\rangle$
- Native support for floating-point numbers. $\longrightarrow$ easy to implement JPEG
- Great packing capabilities $\longrightarrow$ compact representation of encrypted images

$$\mathbb{C}^{N/2} \xrightarrow{\text{Encode}} \mathcal{R} \xrightarrow{\text{Encrypt}} \mathcal{R}_q^2$$

- It offers parallel computation in a SIMD manner.

Properties of CKKS

- Message space: $\mathbb{C}^{N/2}$
- Plaintext space: $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$
- Ciphertext space: $\mathcal{R}_q := \mathbb{Z}_q[X]/\langle X^N + 1\rangle$
- Native support for floating-point numbers. $\longrightarrow$ easy to implement JPEG
- Great packing capabilities $\longrightarrow$ compact representation of encrypted images

$$\mathbb{C}^{N/2} \xrightarrow{\text{Encode}} \mathcal{R} \xrightarrow{\text{Encrypt}} \mathcal{R}_q^2$$

- It offers parallel computation in a SIMD manner. $\longrightarrow$ efficient image processing

Properties of CKKS

- Message space: $\mathbb{C}^{N/2}$
- Plaintext space: $\mathcal{R} := \mathbb{Z}[X]/\langle X^N + 1\rangle$
- Ciphertext space: $\mathcal{R}_q := \mathbb{Z}_q[X]/\langle X^N + 1\rangle$
- Native support for floating-point numbers. $\longrightarrow$ easy to implement JPEG
- **Great packing capabilities** $\longrightarrow$ compact representation of encrypted images

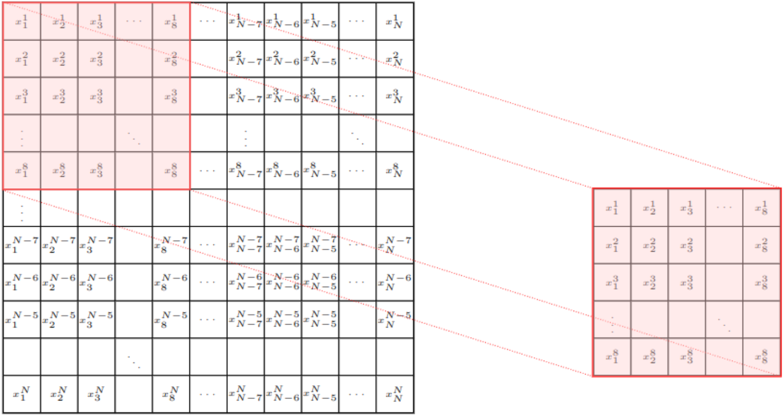
$$\mathbb{C}^{N/2} \xrightarrow{\text{Encode}} \mathcal{R} \xrightarrow{\text{Encrypt}} \mathcal{R}_q^2$$

- It offers parallel computation in a SIMD manner. $\longrightarrow$ efficient image processing

Image encryption

x_1^1	x_2^1	x_3^1	$\cdots$	x_8^1	$\cdots$	x_{N-7}^1	x_{N-6}^1	x_{N-5}^1	$\cdots$	x_N^1
x_1^2	x_2^2	x_3^2		x_8^2		x_{N-7}^2	x_{N-6}^2	x_{N-5}^2	$\cdots$	x_N^2
x_1^3	x_2^3	x_3^3		x_8^3		x_{N-7}^3	x_{N-6}^3	x_{N-5}^3	$\cdots$	x_N^3
$\vdots$			$\ddots$			$\vdots$			$\ddots$	
x_1^8	x_2^8	x_3^8		x_8^8	$\cdots$	x_{N-7}^8	x_{N-6}^8	x_{N-5}^8	$\cdots$	x_N^8
$\vdots$										
x_1^{N-7}	x_2^{N-7}	x_3^{N-7}		x_8^{N-7}	$\cdots$	x_{N-7}^{N-7}	x_{N-6}^{N-7}	x_{N-5}^{N-7}	$\cdots$	x_N^{N-7}
x_1^{N-6}	x_2^{N-6}	x_3^{N-6}		x_8^{N-6}	$\cdots$	x_{N-7}^{N-6}	x_{N-6}^{N-6}	x_{N-5}^{N-6}	$\cdots$	x_N^{N-6}
x_1^{N-5}	x_2^{N-5}	x_3^{N-5}		x_8^{N-5}	$\cdots$	x_{N-7}^{N-5}	x_{N-6}^{N-5}	x_{N-5}^{N-5}		x_N^{N-5}
			$\ddots$							
x_1^N	x_2^N	x_3^N		x_8^N	$\cdots$	x_{N-7}^N	x_{N-6}^N	x_{N-5}^N	$\cdots$	x_N^N

Image encryption



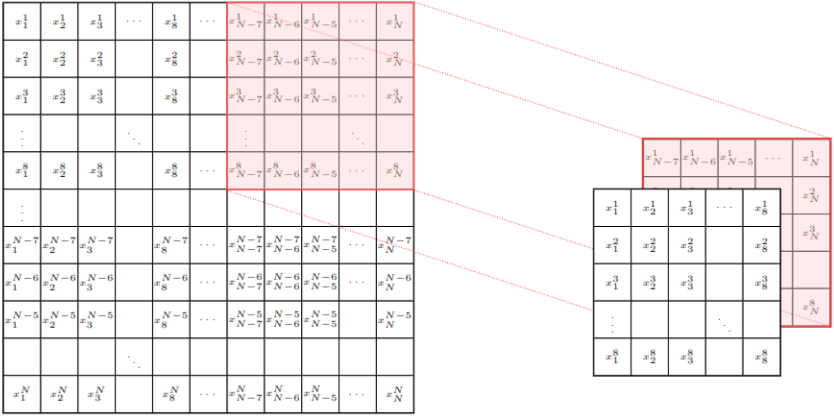

x_1^1 x_2^1 x_3^1 $\cdots$ x_8^1 $\cdots$ x_{N-7}^1 x_{N-6}^1 x_{N-5}^1 $\cdots$ x_N^1
x_1^2 x_2^2 x_3^2 x_8^2 x_{N-7}^2 x_{N-6}^2 x_{N-5}^2 $\cdots$ x_N^2
x_1^3 x_2^3 x_3^3 x_8^3 x_{N-7}^3 x_{N-6}^3 x_{N-5}^3 $\cdots$ x_N^3
$\vdots$ $\ddots$ $\vdots$ $\ddots$
x_1^8 x_2^8 x_3^8 x_8^8 $\cdots$ x_{N-7}^8 x_{N-6}^8 x_{N-5}^8 $\cdots$ x_N^8
$\vdots$
x_1^{N-7} x_2^{N-7} x_3^{N-7} x_8^{N-7} $\cdots$ x_{N-7}^{N-7} x_{N-6}^{N-7} x_{N-5}^{N-7} $\cdots$ x_N^{N-7}
x_1^{N-6} x_2^{N-6} x_3^{N-6} x_8^{N-6} $\cdots$ x_{N-7}^{N-6} x_{N-6}^{N-6} x_{N-5}^{N-6} $\cdots$ x_N^{N-6}
x_1^{N-5} x_2^{N-5} x_3^{N-5} x_8^{N-5} $\cdots$ x_{N-7}^{N-5} x_{N-6}^{N-5} x_{N-5}^{N-5} x_N^{N-5}
$\ddots$
x_1^N x_2^N x_3^N x_8^N $\cdots$ x_{N-7}^N x_{N-6}^N x_{N-5}^N $\cdots$ x_N^N
x_{N-7}^1 x_{N-6}^1 x_{N-5}^1 $\cdots$ x_N^1
x_N^2
x_N^3
x_N^8
x_1^1 x_2^1 x_3^1 $\cdots$ x_8^1
x_1^2 x_2^2 x_3^2 x_8^2
x_1^3 x_2^3 x_3^3 x_8^3
$\vdots$ $\ddots$
x_1^8 x_2^8 x_3^8 x_8^8

Image encryption

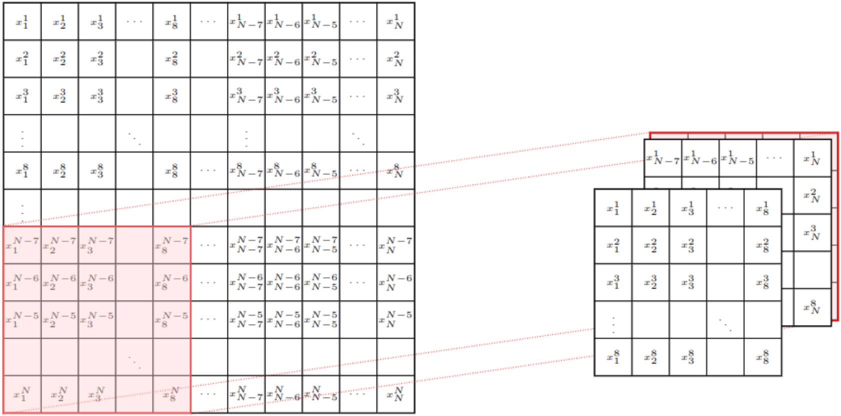


Image encryption

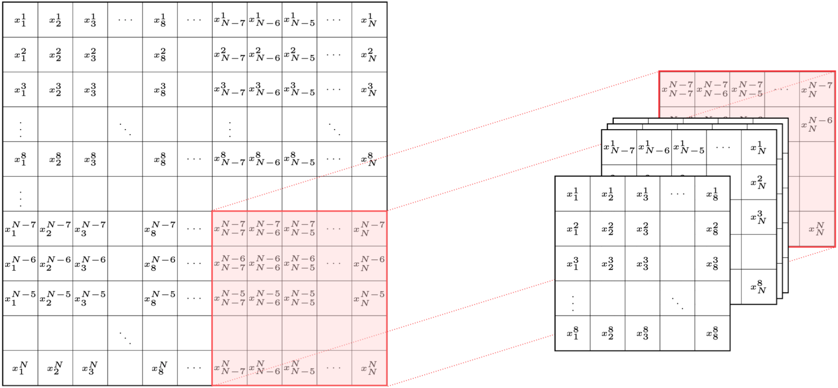


Image encryption

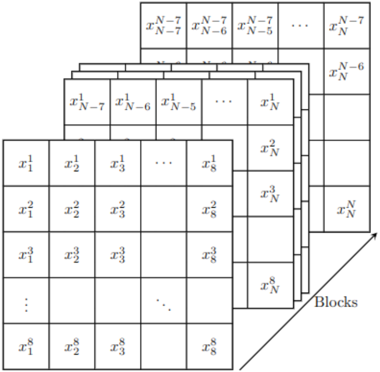


Image encryption

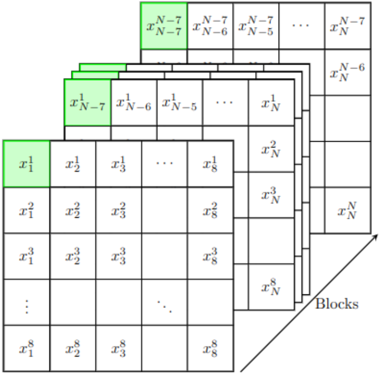
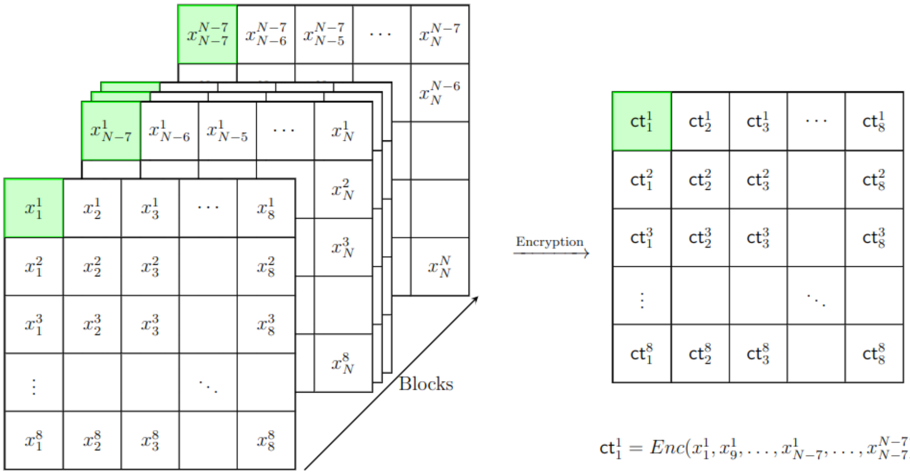
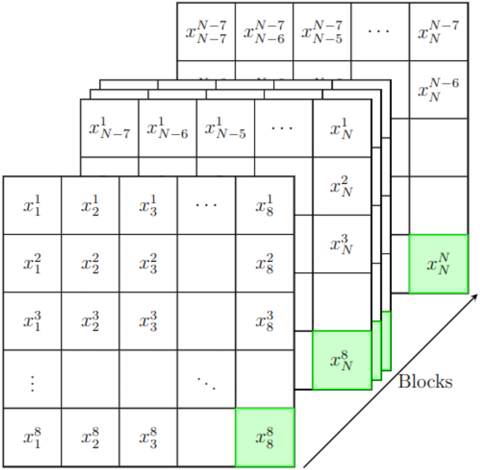
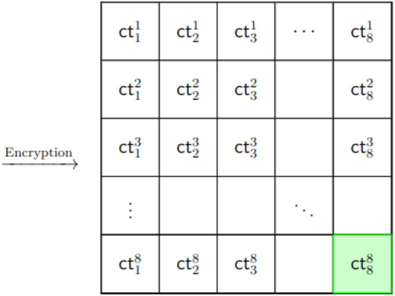


Image encryption



$$\mathrm{ct}^1_1 = Enc(x^1_1, x^1_9, \ldots, x^1_{N-7}, \ldots, x^{N-7}_{N-7})$$





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

- The **number of ciphertexts** needed to encrypt an image **is equal to 64 - cutting point**.
- We can **pack multiple images** (and images with multiple colour channels) with the **same number of ciphertexts**.
- We can apply **pixel-wise operations and convolutions** to all the blocks of an image (or to multiple images) **in parallel**.

- We use the CKKS implementation of OpenFHE
- Leveled approach, with depth $= 9$
- Standard parameters for 128-bits of security
- Simulation of client and server done in a desktop with an Intel Core i7-13700 and 32gb of RAM

4 settings:

1. Pixel-wise processing on 8x8 blocks
2. Convolutional processing on 8x8 blocks
3. Pixel-wise processing on 16x16 blocks
4. Convolutional processing on 16x16 blocks

Implementation and Performance

image size	setting	decompression	processing	compression	total	SSI	compression ratio
256×256	1	8s	< 1s	5s	13s	0.95	$100 : 34.4$
252×252	2	11.3s	4s	9s	24.5s	0.935	$100 : 83.3$
256×256	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	$100 : 34.4$
510×510	2	11s	4s	9s	24s	0.935	$100 : 83.3$
512×512	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	$100 : 34.4$
1020×1020	2	11s	4s	9.7s	24.7s	0.975	$100 : 83.3$
1024×1024	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	$100 : 34.4$
2046×2046	2	107s	36.5s	79s	222.5s	0.98	$100 : 83.3$
2048×2048	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: Experimental results (server-side). Each timing is for one 8-bit greyscale image of mentioned dimensions. The values are averaged over a few randomly selected images: 8 images of 512×512, 4 images for other dimensions.

Future work

- Formally prove that entropy encoding is impractical in FHE.
- More complex image processing eg. object detection, face recognition, etc.

Thank you!

- axel.mertens@esat.kuleuven.be
- georgio.nicolas@esat.kuleuven.be
- sergi.rovira@upf.edu

- Preprint: https://eprint.iacr.org/2024/559
- Code: https://github.com/icip-24/img-processing-fhe

Nils Fleischhacker, Kasper Green Larsen, and Mark Simkin, *How to compress encrypted data*, Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I (Carmit Hazay and Martijn Stam, eds.), Lecture Notes in Computer Science, vol. 14004, Springer, 2023, pp. 551–577.

Rasoul Akhavan Mahdavi, Abdulrahman Diaa, and Florian Kerschbaum, *HE is all you need: Compressing FHE ciphertexts using additive HE*, CoRR **abs/2303.09043** (2023).