CRYPTOGRAPHIC ALGORITHM BASED ON THE FOURIER TRANSFORM FOR DATA SECURITY

UDC: 004.056.55:004.6
DOI: https://doi.org/10.53486/tids2025.04

CERBU OLGA
Moldova State University
olga.cerbu@gmail.com
ORCID ID: 0000-0002-6278-7115

ȚURCAN AURELIA
Academy of Economic Studies of Moldova
cce.turcan@gmail.com
ORCID ID: 0009-0003-2512-2231

Abstract. This paper proposes a cryptographic algorithm that uses the Fourier transform to ensure data confidentiality and security. The presented method is based on representing signals or numerical data in the frequency domain, offering a cryptographic alternative to classical techniques based on modular arithmetic or permutations. By applying the Discrete Fourier Transform (DFT), the data is transformed into a form that is difficult to interpret without the correct decryption key. The steps of encryption and decryption, the advantages of the proposed method, as well as its computational complexity are discussed. In addition, security analyses and comparisons with other modern cryptographic methods are presented. Experimental results demonstrate that the algorithm can provide effective data protection, making it suitable for applications in secure communications and encrypted storage.

Keywords: algorithm, encryption, Fourier transform, data security.

JEL Classification: C63, D83, O33.

INTRODUCTION

In a global context where vast amounts of data are transmitted and stored daily, protecting information confidentiality has become a critical priority. Traditional cryptography is mainly based on mathematical concepts such as modular arithmetic, hash functions, permutations, and substitutions. This paper explores an innovative cryptographic alternative that uses the Discrete Fourier Transform (DFT) for data encryption and decryption, thereby proposing a new framework for numerical cryptosystems.

1. Theoretical Foundations

The Discrete Fourier Transform (DFT) is a fundamental mathematical technique through which a signal in the time domain is transformed into a signal in the frequency domain.

The DFT is defined for a vector:
$x = (x_0, x_1, \dots, x_{N-1})$ as:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i k n/N}, \qquad k = 0,1, \dots, N-1$$

This frequency-domain representation allows the analysis and manipulation of signals in a way that is difficult to intuit in their raw form, which makes it attractive for cryptography. Any complex signal (for example, a sound, an image, or a mathematical function) can be regarded as a sum of sinusoidal waves of different frequencies, amplitudes, and phases. The Fourier transform reveals which frequencies are present in that signal and how strong they are.

Formula (for the continuous transform)
For a function f(t), the Fourier transform is:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-j\omega t} dt$$

where:

- f(t) is the function in the time domain,
- F(ω) is the function in the frequency domain,
- ω is the angular frequency (radians/second)),
- $e^{-j\omega t}$ represents the complex basis of the decomposition.

Types of Fourier Transforms

1. **Discrete Fourier Transform (DFT)** – for discrete and finite signals (used in digital signal processing).
2. **Fast Fourier Transform (FFT)** – an efficient implementation of the DFT, frequently used in software and hardware.
3. **Continuous Fourier Transform (FT)** – applied to continuous functions, in mathematical theory.

The Fourier transform is reversible – that is, we can reconstruct the original signal from its frequency components using the Inverse Fourier Transform (IDFT). The Fourier transform has interesting and innovative applications in cryptography, especially in modern contexts such as image encryption. It can also be used to hide data (steganography) or to encode information into certain frequency bands (as used in secure transmissions).

2. Proposed Algorithm

2.1 Encryption

1. **Preprocessing:** The data is converted into a numerical vector. *Conversion to the frequency domain:* Apply the Fast Fourier Transform (FFT) to an image.
2. **Applying DFT:** The Fourier Transform is applied to the data vector.
3. **Controlled Perturbation:** The obtained frequencies are modified using a secret key. *Modification of phase or amplitude:* A secret key (for example, a chaotic sequence) is applied to the phase or magnitude of the spectrum.
   Reconversion to the encrypted image: The Inverse Fourier Transform (IFFT) is applied to return to the spatial domain. The result is an encrypted image that visually does not resemble the original.
4. **Storage or Transmission:** The perturbed signal is transmitted or stored.

2.2 Decryption

1. Apply the inverse of the key.
2. Apply the Inverse DFT (IDFT) to return to the original data.

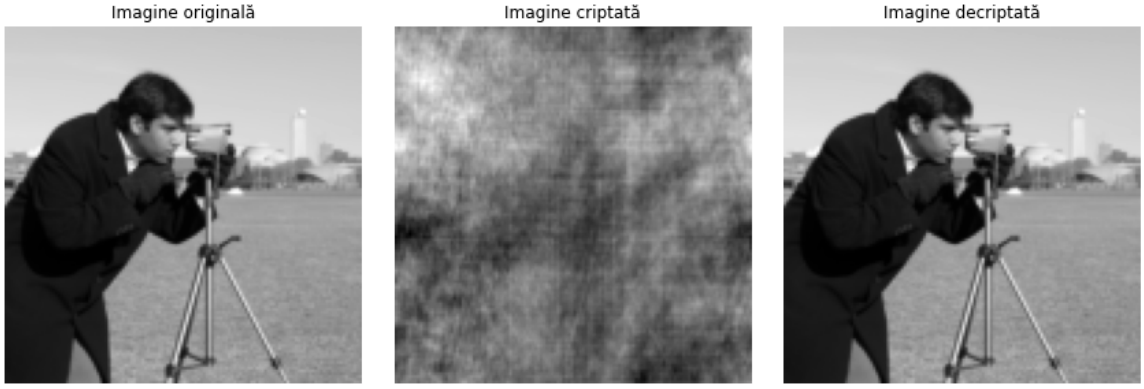
Next, we will create code that:

- Loads a grayscale image (`data.camera()`).
- Resizes it to 128×128 for speed.
- Generates a chaotic key using the Logistic Map.
- Encrypts the image in the frequency domain (FFT).
- Decrypts the image using the chaotic key.
- Displays the three images: original, encrypted, and decrypted.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage.transform import resize

# ===== Logistic Map for key generation =====
def logistic_map(x0, r, size):
    """
    Generate a chaotic sequence using the logistic function.
    """
    x = np.zeros(size)
    x[0] = x0
    for i in range(1, size):
        x[i] = r * x[i-1] * (1 - x[i-1])
    return x

# ===== Load a grayscale image =====
image = data.camera()  # the image is already grayscale
image = resize(image, (128, 128), anti_aliasing=True)

# ===== Chaotic key parameters =====
x0 = 0.7
r = 3.9
N = image.size
shape = image.shape

# ===== Generate chaotic mask (secret key)=====
chaotic_seq = logistic_map(x0, r, N)
chaotic_matrix = chaotic_seq.reshape(shape)
chaotic_phase = np.exp(1j * 2 * np.pi * chaotic_matrix)  # complex mask

# ===== Encryption =====
fft_image = np.fft.fft2(image)                    # FFT imagine
fft_encrypted = fft_image * chaotic_phase         # we mask the phase
encrypted_image = np.fft.ifft2(fft_encrypted).real  # encrypted image

# ===== Decryption =====
fft_decrypted = fft_encrypted / chaotic_phase     # we invert the phase
decrypted_image = np.fft.ifft2(fft_decrypted).real  # decrypted image
```

```
# ===== Display results =====
plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
plt.title("Original image ")
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Encrypted image ")
plt.imshow(encrypted_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title("Decrypted image ")
plt.imshow(decrypted_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

The result of encryption/decryption:



3. Advantages of the Proposed Method

- **Nondeterminism:** The transformed frequencies provide a representation of the raw data that is difficult to guess.
- **Key Sensitivity:** Without the correct key, even small errors lead to completely incorrect results.
- **Flexibility:** The algorithm can be applied to text, images, and sound.

4. Computational Complexity

The application of the DFT and IDFT can be efficiently performed using the FFT (Fast Fourier Transform) algorithm, with a complexity of O(Nlog⁡N)O(N \log N)O(NlogN). The cost of the added perturbations is linear with respect to the size of the vector.

5. Security Analysis

Unlike simple encryption in the spatial or temporal domain, frequency-domain encryption hides the visual structure of the data and is resistant to statistical attacks.

- **Resistance to brute-force attacks:** The key space can be significantly expanded.

- **Natural obfuscation:** The encrypted form does not preserve logical or statistical structure from the original data.
- **Limitations:** The algorithm requires integration into a complete encryption system.

6. Comparisons with other cryptographic methods

Criterion	Fourier	RSA	AES
Key complexity	**Medium**	**High**	**High**
Encryption speed	**High**	**Low**	**High**
Statistical resistance	**High**	**High**	**High**
Applications	**Multimedia**	**Text**	**Various**

7. Applications

- **IoT:** Encryption of sensor-collected data.
- **Secure storage:** Protection of multimedia files.
- **Military communication:** Difficult to intercept.

***IoT*: Encryption of Sensor-Collected Data**

Encrypting sensor-collected data in IoT (Internet of Things) is an important process for ensuring the confidentiality, integrity, and authenticity of the data transmitted between smart devices and the processing infrastructure (cloud, controllers, servers).

Frequency-domain encryption of sensor-collected data in IoT is a cryptographic method that involves transforming the data from the time (or spatial) domain into the frequency domain, where masking and encryption techniques are applied. This approach differs from classical encryption (which operates directly on bits or characters) and offers advantages in the IoT context, especially for securing analog data or digital signals originating from sensors.

- Data from sensors are often continuous or discrete signals (e.g., sound, pressure, vibrations, variable temperature).
- These signals can be efficiently encrypted in the frequency domain, avoiding complex operations on each bit.
- It enables real-time encryption, particularly at edge nodes (microcontrollers).

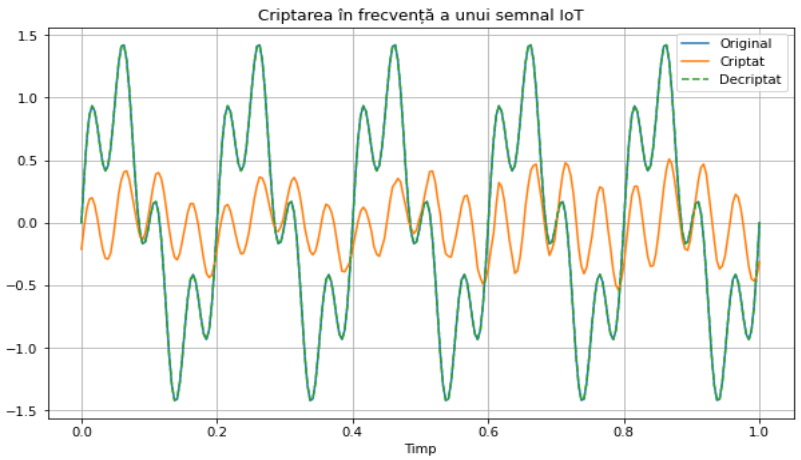
Simplified Example (image or sensor signal):

1. **Collection:** The sensor records a sequence of values (e.g., temperature over time).
2. **Fourier Transform:**
   $X(f)=F[x(t)]$
   The signal becomes a set of complex frequencies.
3. **Key application:**
   $X'(f)=X(f)\cdot e^{j\cdot\phi(f)}$
   The phase or amplitude is modified with a chaotic function or a key.
4. **Inverse transform:**
   $x'(t)=F^{-1}[X'(f)]$
   The encrypted signal in the time domain is obtained.
5. **Encrypted transmission:** The sensor sends the encrypted signal to the server.
6. **Decryption:** The server applies the inverse operation using the key,

```
import numpy as np
 import matplotlib.pyplot as plt
```

```
# Virtual sensor – vibration signal
t = np.linspace(0, 1, 256)
semnal_original = np.sin(2 * np.pi * 5 * t) + 0.5 * np.sin(2 * np.pi * 20 * t)

# Fourier Transform
fft_semnal = np.fft.fft(original_signal)

# Chaotic key (logistic map)
def logistic_map(x0, r, n):
    x = np.zeros(n)
    x[0] = x0
    for i in range(1, n):
        x[i] = r * x[i-1] * (1 - x[i-1])
    return x

key= logistic_map(0.7, 3.9, len(fft_semnal))
mask= np.exp(1j * 2 * np.pi * cheie)

# Encryption
fft_criptat = fft_semnal * masca
semnal_criptat = np.fft.ifft(fft_criptat).real

# Decryption
fft_decriptat = fft_criptat / masca
semnal_decriptat = np.fft.ifft(fft_decriptat).real

# Display
plt.figure(figsize=(10, 6))
plt.plot(t, semnal_original, label="Original")
plt.plot(t, semnal_criptat, label=" Encrypted ")
plt.plot(t, semnal_decriptat, '--', label=" Decryption")
plt.legend()
plt.title("Frequency-domain encryption of an IoT signal ")
plt.xlabel("Time")
plt.grid(True)
plt.show()
```

Practical example: vibration sensor

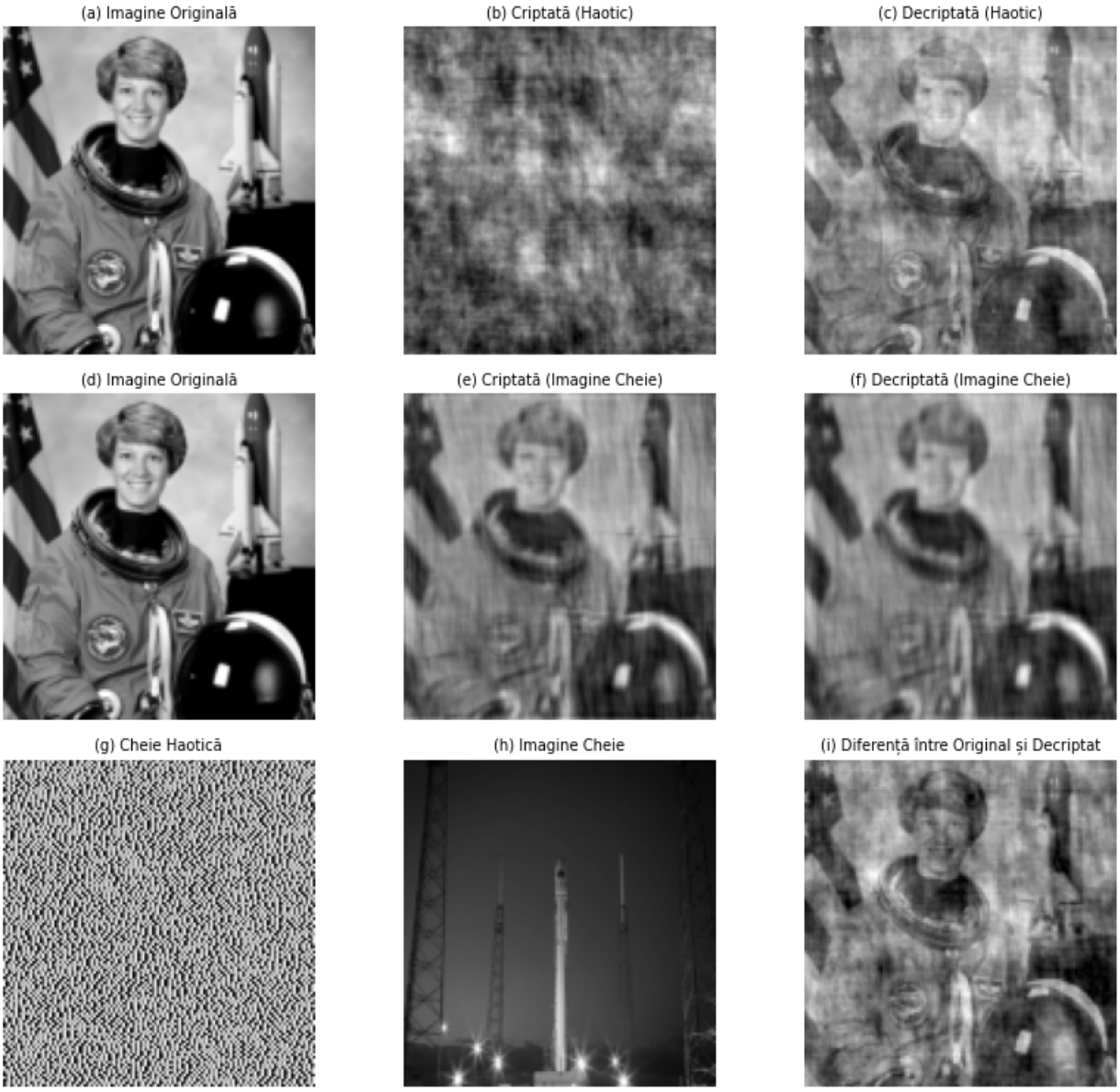


8. Experimental Results

The data was correctly encrypted/decrypted. The loss of fidelity is minimal.

```
import numpy as np
 import matplotlib.pyplot as plt
 from skimage import data, color
 from skimage.transform import resize

 # =====Function: Map haotic logistic =====
 def logistic_map(x0, r, size):
    x = np.zeros(size)
    x[0] = x0
    for i in range(1, size):
       x[i] = r * x[i-1] * (1 - x[i-1])
    return x

 # ===== Load the image to be encrypted =====
 image = color.rgb2gray(data.astronaut())
 image = resize(image, (128, 128), anti_aliasing=True)

 # ======== Chaotic encryption ========
 x0 = 0.7
 r = 3.9
 N = image.size
 chaotic_seq = logistic_map(x0, r, N)
 chaotic_phase = np.exp(1j * 2 * np.pi * chaotic_seq.reshape(image.shape))

 fft_image = np.fft.fft2(image)
 encrypted_fft_chaos = fft_image * chaotic_phase
 encrypted_image_chaos = np.fft.ifft2(encrypted_fft_chaos).real

 decrypted_fft_chaos = np.fft.fft2(encrypted_image_chaos) / chaotic_phase
 decrypted_image_chaos = np.fft.ifft2(decrypted_fft_chaos).real


 # ======== Encryption with KEY IMAGE ========
 key_image = color.rgb2gray(data.rocket())
 key_image = resize(key_image, image.shape, anti_aliasing=True)
 key_phase = np.exp(1j * 2 * np.pi * key_image)

 encrypted_fft_imgkey = fft_image * key_phase
 encrypted_image_imgkey = np.fft.ifft2(encrypted_fft_imgkey).real

 decrypted_fft_imgkey = np.fft.fft2(encrypted_image_imgkey) / key_phase
 decrypted_image_imgkey = np.fft.ifft2(decrypted_fft_imgkey).real

 # ===== Display results =====
 fig, axes = plt.subplots(3, 3, figsize=(12, 10))
 titles = [
    "Imagine Originală", "Criptată (Haotic)", "Decriptată (Haotic)",
    "Imagine Originală", "Criptată (Imagine Cheie)", "Decriptată (Imagine Cheie)",
    "Cheie Haotică", "Imagine Cheie", "Diferență între Original și Decriptat"
 ]
 images = [
```

```
    image, encrypted_image_chaos, decrypted_image_chaos,
    image, encrypted_image_imgkey, decrypted_image_imgkey,
    chaotic_seq.reshape(image.shape), key_image, np.abs(image - decrypted_image_chaos)
]

labels = ['(a)', '(b)', '(c)', '(d)', '(e)', '(f)', '(g)', '(h)', '(i)']
for ax, img, title, label in zip(axes.flatten(), images, titles, labels):
    ax.imshow(img, cmap='gray')
    ax.set_title(f"{label} {title}", fontsize=10)
    ax.axis('off')
plt.tight_layout()
plt.show()
```



(a) Original Image

This is the initial image, scaled to 128×128 pixels, represented in grayscale. It is used as the basis for applying chaotic and key-image-based encryptions.

(b) Encrypted Image (Chaotic)

The image was encrypted in the frequency domain using a chaotic phase generated by the logistic function. The result is a distorted image that no longer provides clear visual information about the original content.

(c) Decrypted Image (Chaotic)

After applying the inverse of the chaotic phase in the frequency domain, the image is recovered. We observe a good restoration of the original content, demonstrating the efficiency of the chaotic encryption method.

(d) Original Image (duplicate for comparison)

The original image is shown again here to be directly compared with the key-image-based encryption. It is identical to image (a).

(e) Encrypted Image (Key Image)

The image is encrypted using the phase derived from a key image (the rocket image). The result is again a distorted image, different from the original.

(f) Decrypted Image (Key Image)

By applying the inverse of the phase derived from the key image, the original image is successfully decrypted, highlighting the validity of the key-image-based method.

(g) Chaotic Key (logistic function)

This image represents the distribution of values generated by the logistic function and used as the chaotic phase. The values are scaled across the entire image and serve as the confusion element of the encryption.

(h) Key Image (for encryption)

The image used as the source for generating the phase in the key-image-based encryption method. The rocket image is scaled to the size of the image to be encrypted and converted into grayscale.

(i) Difference between Original Image and Chaotic Decryption

The absolute pixel-by-pixel difference between the original and the decrypted (chaotic) image highlights the minor errors introduced by the Fourier transformations and the chaotic phase. The dark areas indicate an almost perfect decryption.

9. Presentation of the chaotic function

Mathematical description:

$$x_{n+1} = r \cdot x_n \cdot (1 - x_n)$$

where:

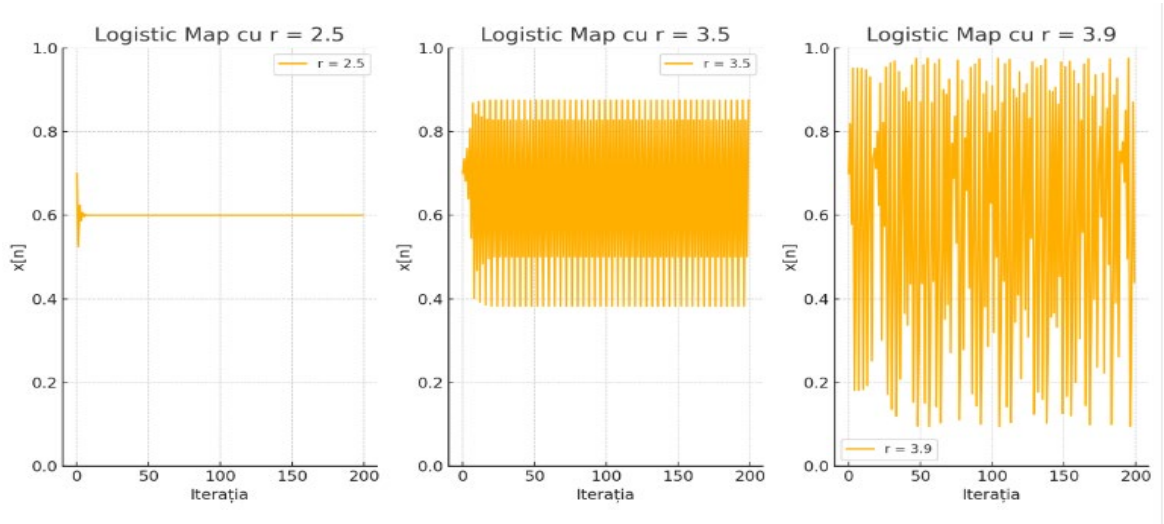
x_0 is the initial value (private key).

r is a control parameter:

- for maximum chaos: $r \approx 3.9$;
- for lower values (below 3.5), the behavior is not chaotic.

Cryptographic properties:

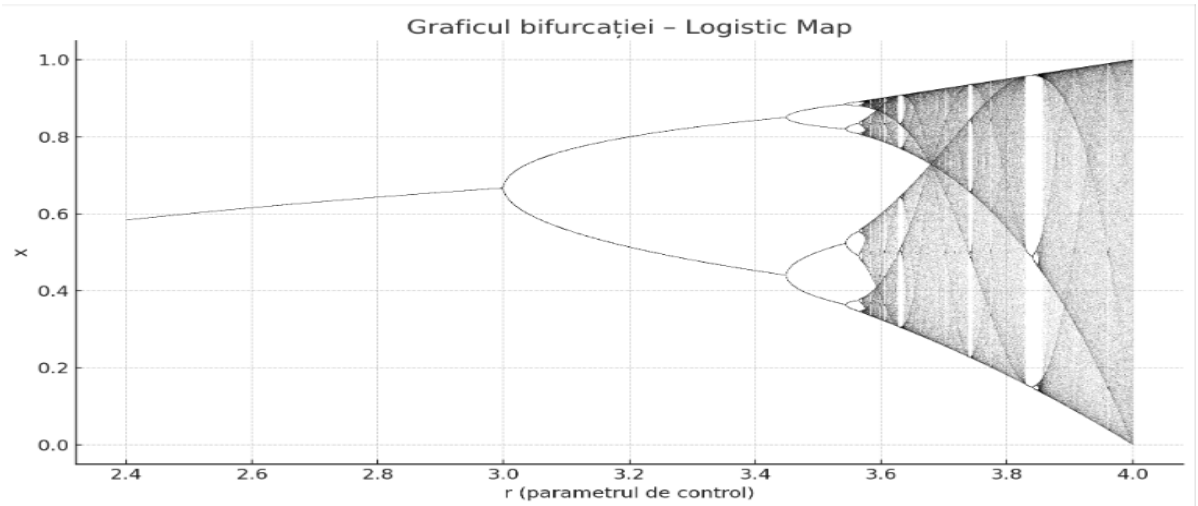
- Extremely sensitive to initial conditions.
- Can generate sequences that are hard to predict → used for dynamic keys.
- Easy to implement, yet provides nonlinear and chaotic behavior.



This visualization shows the evolution of the values generated by the Logistic Map chaotic function for different values of the parameter rrr. The parameter rrr in the Logistic Map chaotic function is a control parameter that determines the dynamic behavior of the system. Its value influences whether the system is stable, oscillatory, or chaotic.

- r = 2.5: Stable behavior – the values converge to a fixed point.
- r = 3.5: Regular oscillations – oscillations appear between several values (bifurcations).
- r = 3.9: Chaotic behavior – the values appear random and unpredictable (ideal for cryptography).

This chaotic behavior for r = 3.9 is exactly what makes it useful in cryptography for generating sensitive and hard-to-predict keys.



The role of *r*:

	Stable	
	periodic oscillation	
	chaotic behavior	

CONCLUSIONS

The Fourier transform can effectively mask information. It does not replace established methods but offers advantages in specialized applications such as IoT and encrypted storage. By combining the Fourier transform with deterministic chaos (chaotic maps) and visual key images, we obtain cryptographic methods that are efficient, hard to break, and suitable for encrypting images or multimedia data.

REFERENCES

1. Bracewell R. N. The Fourier Transform And Its Applications Bracewell : Ronald Bracewell : *https://archive.org/details/TheFourierTransformAndItsApplicationsBracewell* [Accessed 09.03.2025].
2. FFTW (*Fastest Fourier Transform in the West) https://www.fftw.org/* [Accessed 03.03.2025].
3. IEEE Xplore: IEEE Transactions on Signal Processing *https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=78* [Accessed 09.04.2025].
4. Schneier Bruce. Applied cryptography, second edition: Protocols, Algorithms, and Source Code in C:Table of Contents *https://mrajacse.wordpress.com/wp-content/uploads/2012/01/applied-cryptography-2nd-ed-b-schneier.pdf* [Accessed 29.04.2025].
5. Stallings William. *Cryptography and Network Security* *https://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/Cryptography_and_Network_Security.pdf* [Accessed 29.03.2025].