19L720 – PROJECT WORK 1

IMAGE ENCRYPTION USING AES AND CHAOTIC SYSTEM

DHILIP J (20L109)
TAMIZHARASAN M (20L148)
NAVEEN KUMAR M (21L411)
PREM N (21L412)

Dissertation submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ELECTRONICS AND COMMUNICATION ENGINEERING

Of Anna University

NOVEMBER 2023

ELECTRONICS AND COMMUNICATION ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE– 641 004.

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641004

IMAGE ENCRYPTION USING AES AND CHAOTIC SYSTEM

Bona fide record of work done by

PREM N (21L412)

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ELECTRONICS AND COMMUNICATION ENGINEERING

of Anna University

……..…	……..……
Dr. T. KESAVAMURTHY	**Dr. V. KRISHNAVENI**
Faculty Guide	Head of the Department

Certified that the candidate was examined in the viva-voice examination held on

……

…….

(Internal Examiner)

CONTENTS

CHAPTER	PAGE NO
ACKNOWLEDGEMENT	iv
SYNOPSIS	v
LIST OF FIGURES	vi
LIST OF TABLES	vii
1. INTRODUCTION	1
1.1 NEED FOR THE CURRENT STUDY	2
1.2 PROBLEM STATEMENT	2
1.3 OBJECTIVE	2
2. LITERATURE SURVEY	3
3. METHODOLOGY	4
3.1 ENCRYPTION	4
3.2 DECRYPTION	4
3.3 CHAOTIC SYSTEMS	4
3.4 TYPES OF CHAOTIC SYSTEMS	5
3.4.1 LORENTZ SYSTEM	5
3.4.2 ROSSLER'S SYSTEM	5
3.4.3 CHAU'S SYSTEM	5
3.4.4 CHEN'S SYSTEM	6
3.5 EXISTING METHODOLOGY	6
3.5.1 AES ALGORITHM	6
3.5.2 CHAOS SYTEM	6
3.5.3 EXISTING IMAGE ENCRYPTION METHOD	7
3.6 PROPOSED METHODOLOGY	8
3.7 IMAGE ENCRYPTION ALGORITHM	9
3.8 IMAGE DECRYPTION ALGORITHM	11
3.9 TEST PARAMETERS OF SYSTEMS	12
3.9.1 VARAIANCE	12
3.9.2 STANDARD DEVIATION	12

3.9.3 CORRELATION 12
3.9.4 ENTROPHY 13
3.10 RUNGA KUTTA METHOD 13
3.11 RUNGA KUTTA OUTPUT 14
4. IMPLEMENTATION AND RESULTS
4.1 TABULATION OF INITIAL STATE 15
4.2 SIMULATION RESULTS (CHAOTIC SYSTEMS) 16
4.2.1 LORENTZ SYSTEM 16
4.2.2 ROSSLER SYSTEM 17
4.2.3 CHENS SYSTEM 18
4.2.4 CHAUS SYSTEM 19
4.2.5 CHUA'S SYSTEM HIGH VARIANCE 20
4.3 COMPARISION BETWEEN SYSTEM PARAMETERS 20
4.4 IMAGE READ OF SAMPLE IMAGE 21
4.5 PIXEL VALUES 21
4.6 PROGRAM OUTPUTS 22
4.6.1 INITIAL STATE GERNERATOR OUTPUT 22
4.6.2 OUTPUT FROM SHA 256 FOR THE INPUT 23
4.7 384-BIT KEY FOR AES SYSTEM(image) 23
4.8 384-BIT KEY FOR 4 ROUNDS OPERATION 24
4.8.1 IMAGE TO 384 BLOCKS 24
4.9 IMAGE ENCRYPTION 24
4.10 IMAGE DECRYPTION 25
4.11 ENCRYPTION TEST 26
4.12 NIST TEST 27
4.12.1 NIST TEST RESULT 31
5. SOFTWARE ENCRYPTION SYSTEM 32
5.1 CHAOS ENCRYPTION SYSTEM 32
6. CONCLUSION 35
6.1 FUTURE SCOPE 36
6.2 APPENDIX 37
6.3 REFERENCES 50

ACKNOWLEDGEMENT

We would like to extend our sincere thanks to Dr. Prakasan K, Principal, PSG College of Technology, for his kind patronage.

We also wish to express our sincere thanks to Dr. V. Krishnaveni, Professor and Head-of-the Department, of Electronics and Communication Engineering.

We are grateful for the support extended by our Programme Coordinator, Dr. K. V. Anusuya, Associate Professor (CAS), Department of Electronics and Communication Engineering.

We are grateful for the support extended by our Faculty, Dr. J Ramesh, Associate Professor (CAS), Department of Electronics and Communication Engineering.

We sincerely owe our gratitude to Dr. K. Prakasan, Principal, PSG College of Technology, Coimbatore, for providing us with the best teachers and facilities.

It is our privilege to place on record our sincere thanks to Dr. V. Krishnaveni, Professor and Head of the Department, Department of ECE, PSG College of Technology, Coimbatore, for her support and guidance.

We would like to place our everlasting gratitude and indebtedness to our Project guide Dr.T.Kesavamurthy, Professor, Department of ECE for his excellent advice, guidance and continued assistance throughout the course of this project.

We express our sincere gratitude to our Tutor Dr. S. Hemachitra, Assistant Professor, PSG College of Technology, Coimbatore, for her encouragement and support towards this project work. We kindly acknowledge the timely help and valuable suggestions of our Professors, research scholars and non-teaching staff, at PSG College of Technology, Coimbatore. Our sincere and heartfelt thanks are to our classmates for their moral support and encouragement to complete the work.

SYNOPSIS

With the proliferation of digital media and the increasing need for secure data transmission, the demand for image encryption has grown substantially. Image encryption plays a vital role in safeguarding sensitive visual content, protecting it from unauthorized access and manipulation. However, encrypting images poses unique challenges due to their large size and complex data structures, making traditional encryption techniques less efficient and often inadequate.

To address these issues, this project proposes an innovative solution that leverages four chaos systems for initial key generation and combines them with AES-based encryption/decryption using a 384-bit key. Chaos systems' deterministic yet unpredictable behaviour ensures the generation of a robust 384-bit initial key, which serves as the foundation for securing images using the Advanced Encryption Standard (AES). By exploring the feasibility and effectiveness of this approach, rigorous testing will validate the system's resistance against cryptographic attacks. The outcome of this research aims to significantly contribute to the field of data security and cryptography, presenting an effective and scalable image encryption method to safeguard sensitive visual data in today's digital age.

Further the improvement of the project to next level, implementing the proposed algorithm in the real world social media network to increases the security in the real world social media applications (What’s up, Facebook). Data theft is a real time issue in current world to avoid the theft of the personal information (Addhaar, PAN and etc) to be avoidable.

LIST OF FIGURES

FIGURE	NAME	PG. NO
3.1	BLOCK DIAGRAM OF ENCRYPTION AND DECRYPTION	4
3.5.3	EXISTING METHOD BLOCK DIAGRAM	7
3.6.1	GENERAL ARCHITECTURE OF PROPOSED CRYPTOSYSTEM	8
3.7.1	AES ENCRYPTION PROCESS	9
3.7.2	S BOX-1	11
3.8.1	AES DECRYPTION PROCESS	11
3.8.2	INVERSE S BOX	12
4.2.1(i)	3D VIEW OF LORENTZ SYSTEM	16
4.2.1(ii)	AUTOCORRELATION LORENTZ SYSTEM	16
4.2.1(iii)	STATISTICAL PARAMETERS	16
4.2.2(i)	3D VIEW OF ROSSLERS SYSTEM	17
4.2.2(ii)	AUTOCORRELATION ROSSLERS SYSTEM	17
4.2.2(iii)	STATISTICAL PARAMETERS	17
4.2.3(i)	3D VIEW OF CHEN SYSTEM	18
4.2.3(ii)	AUTOCORRELATION CHEN SYSTEM	18
4.2.3(iii)	STATISTICAL PARAMETERS	18
4.2.4(i)	3D VIEW OF CHUA SYSTEM	19
4.2.4(ii)	AUTOCORRELATION CHUA SYSTEM	19
4.2.4(iii)	STATISTICAL PARAMETERS	19
4.2.4.1	CHUAS SYSTEM HIGH VARIANCE CONDITION	20
4.4	SAMPLE IMAGE AND ITS PROPERTIES	21
4.5	PIXEL REPRESENTAION OF SAMPLE IMAGE	21
4.6.1(i)	PHYSICAL KEY GEN FOR TEXT	23
4.6.1.(ii)	FINAL 384 BIT KEY AT TIME t = 100	22
4.6.1.(iii)	FINAL 384 BIT KEY AT TIME t = 10	22
4.6.2(i)	PHYSICAL KEY GEN FOR IMAGE	23
4.7.1(i)	FINAL 384 BIT KEY AT TIME t = 100	23
4.7.2.(ii)	FINAL 384 BIT KEY AT TIME t = 10	23
4.8	384 BIT KEY FOR 4 ROUND OPERATIONS	24
4.8.1	IMAGE TO 384 BLOCKS	24

4.9.1	SAMPLE IMAGE 1	24
4.9.2	SAMPLE IMAGE 2	25
4.10	IMAGE DECRYPTION	25
4.11.1	SAMPLE IMAGE 1	27
4.11.2	SAMPLE IMAGE 2	27
5.1	CHAOS ENCRYPTION SYSTEM	30

LIST OF TABLES

TABLE NO	NAME	PG. NO
4.1	INITIAL STATE PARAMETERS	19
4.3	COMPARISON BETWEEN SYSTEM PARAMETRS	20
4.12.1	NIST TEST REULTS	31

CHAPTER 1

INTRODUCTION

In the era of digital communication and information exchange, the need for robust and secure encryption methods has never been more critical. With the proliferation of data breaches, cyberattacks, and unauthorized access, ensuring the confidentiality and integrity of sensitive information has become a paramount concern. This project, titled "Image Encryption using AES and Chaotic System," seeks to address these challenges by proposing a novel encryption technique that combines the proven security of the Advanced Encryption Standard (AES) with the unpredictable dynamics of Chaotic Systems. Firstly, a strong chaos-based digital pseudo-random number generator using a chaotic system is proposed to generate high-quality keys. Secondly, a robust algorithm is suggested to encrypt and decrypt images. The latter ensures the data confusion and diffusion properties. Key generator provides high-quality random number sequences. As a consequence, the proposed cryptosystem can be used for image encryption and decryption in real-time applications.

In several fields such as commerce, medicine, military and even personal lives (google drives), cloud storage digital images represent a multimedia technology that contains secret information. Using public or shared digital networks, images are vulnerable to potentially more destructive attacks such as replay or human-based attacks, brute-force, and statistical attacks. Thus, designing an effective cryptosystem to protect secret images in storing and moving is a challenge. A good cryptosystem is established on a good key generator and good encryption architecture. Recently, many cryptographers have been designed pseudo-random number generators (PRNGs) based on chaotic systems. PRNGs based on chaotic systems have several significant benefits against other generators, such as the deterministic random number generation, the high sensitivity to the initial condition, the long periodicity, and the large space of random numbers. In the literature, many chaotic systems have been investigated such as Lorenz, Chen, Arnold, Skew Tent, Logistic, and Arnold's map. The results obtained by chaos-based image encryption have better performance and are more secure compared to other traditional systems like the advanced encryption system, which is largely used nowadays.

Traditionally, key generation and image encryption algorithms are complex and expensive to execute while taking into consideration the issue related to the security level. Thereby, effective implementation is required to achieve high performance.

The main goal is to perform image encryption and decryption with low computational complexity and to achieve high level security and high performance in terms of frequency and throughput. The main contributions are as follows: Designing a strong chaos-based digital PRNG (Key Generator) to generate high-quality keys using four different chaotic systems, unlike the literature where most researchers have used only one chaotic system and some others two chaotic systems. Designing a robust encryption and decryption architecture based on S-box-substitution confusion and XOR and permutation diffusion. Our architecture ensures high-level security for images with low computational complexity.

1.1 NEED FOR THE CURRENT STUDY

The project "Image Encryption using Chaos System" is of paramount importance in today's digital landscape, where the security and privacy of sensitive information have become critical concerns. With the exponential growth of digital images being shared and transmitted across various platforms, there is an urgent need to protect this valuable visual data from unauthorized access and potential breaches. The use of chaos-based encryption algorithms offers a promising approach to safeguarding images, as chaos systems provide a high level of randomness and unpredictability, making them resistant to decryption attacks. By developing an efficient and secure image encryption algorithm, this project aims to empower individuals, organizations, and industries to protect their intellectual property, personal data, and sensitive information. Moreover, as data privacy regulations become more stringent, the project's outcomes can contribute to compliance efforts and ensure that image-related data is handled with the utmost confidentiality. As technology advances and cyber threats continue to evolve, the significance of this project in bolstering digital security and preserving the integrity of images cannot be overstated, making it a valuable asset for data protection and secure communication in today's fast-paced and interconnected world.

1.2 PROBLEM STATEMENT

In the era of pervasive digital communication and data sharing, ensuring the security and confidentiality of sensitive visual information has become a critical challenge. The widespread use of digital images across various domains, including personal communications, healthcare, finance, and government, necessitates effective image encryption techniques to protect against unauthorized access, tampering, and potential data breaches. Traditional encryption methods may not offer sufficient resistance to sophisticated decryption attacks, making it imperative to explore innovative solutions. The problem at hand is to develop a robust and efficient image encryption algorithm based on chaos systems that can ensure a high level of randomness, data integrity, and security, thereby addressing the pressing need for image protection and privacy in today's fast-paced digital landscape. The proposed solution should be capable of encrypting various image formats, optimizing performance, and providing a user-friendly interface, ultimately contributing to a safer and more secure digital environment for image communication and storage.

1.3 OBJECTIVES

The project "Image Encryption using Chaos System" aims to develop a robust and efficient encryption algorithm based on chaos systems to enhance data privacy and security by safeguarding digital images from unauthorized access and potential breaches. The primary objectives include maintaining image integrity during encryption and decryption, supporting common image formats, optimizing performance, and providing a user-friendly interface. Comprehensive documentation and educational materials will be provided to facilitate understanding and responsible use. The project will undergo rigorous testing and validation to ensure correctness and security, addressing ethical considerations to promote responsible image encryption practices. Ultimately, the project seeks to contribute significantly to data security efforts and create a safer digital landscape for image sharing, storage, and communication

CHAPTER 2

LITERATURE SURVEY

[1] This paper, a novel image encryption algorithm is proposed based on the combi nation of the chaos sequence and the modified AES algorithm. In this method, the encryption key is generated by Arnold chaos sequence. Then, the original image is encrypted using the modified AES algorithm and by implementing the round keys produced by the chaos system. The proposed approach not only reduces the time complexity of the algorithm but also adds the diffusion ability to the proposed algorithm, which make the encrypted images by the proposed algorithm resistant to the differential attacks. The key space of the proposed method is large enough to resist the brute-force attacks. This method is so sensitive to the initial values and input image so that the small changes in these values can lead to significant changes in the encrypted image.

[2] In this paper, a robust chaos-based stream cipher (CBSC) is proposed. The novelty of this work is that it addresses all challenges confronting chaos based cryptography. The PCBSC (proposed CBSC) has a robust synchronization circuit that mitigates the effect of channel noise, a perturbation block that over comes the dynamical degradation, a robust encryption scheme, and an efficient control parameters' genera tor that generates strong keys. According to the complexity evaluation, the improved chaotic map provides good statistical properties. This can be confirmed by the obtained high values of the statistical metrics (largest Lyapunov exponent, approximate entropy, permutation entropy, and sample entropy) used for the evaluation

[3] This papers discussed the unified classification between the Lorenz and the Chen attractors, both in theory and in simulation. In fact, this unified system is likely to be the simplest chaotic system that bridges the gap between the Lorenz and the Chen systems, and contributes to a better understanding of the correlation between the Lorenz attractor and the Chen attractor.

[4] This paper discussed the question of the equivalence of various Lorenz like systems and the possibility of universal consideration of their behaviour. in view of the possibility of reduction of such systems to the same form with the help of various transformations. In the present paper the differences and similarities in the analysis of the Lorenz, the Chen and the Lu systems are discussed. It is shown that the Chen and the Lu systems stimulate the development of new methods for the analysis of chaotic systems. Open problems are discussed.

[5] This paper introduces a simple and effective chaotic system using a combination of two existing one-dimension (1D) chaotic maps (seed maps). Simulations and performance evaluations show that the proposed system is able to produce many 1D chaotic maps with larger chaotic ranges and better chaotic behaviours compared with their seed maps Using a same set of security keys, this algorithm is able to generate a completely different encrypted image each time when it is applied to the same original image. Experiments and security analysis demonstrate the algorithm's excellent performance in image encryption and various attacks.

CHAPTER 3

METHODOLOGY

3.1 ENCRYPTION

Encryption is the method by which information is converted into secret code that hides the information's true meaning. The science of encrypting and decrypting information is called cryptography. In computing, unencrypted data is also known as plaintext, and encrypted data is called cipher text. The formulas used to encode and decode messages are called encryption algorithms.

3.2 DECRYPTION

Decryption is the process of transforming data that has been rendered unreadable through encryption back to its unencrypted form. In decryption, the system extracts and converts the garbled data and transforms it to texts and images that are easily understandable not only by the reader but also by the system. Decryption may be accomplished manually or automatically. It may also be performed with a set of keys or passwords.

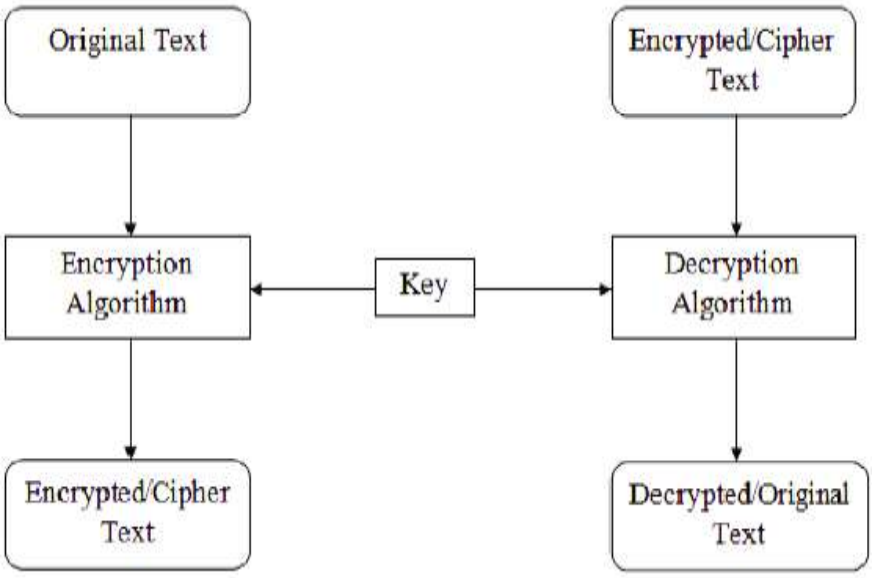


Fig.3.1 Block Diagram of Encryption and Decryption

3.3 CHAOTIC SYSTEM:

A chaotic system refers to a deterministic dynamical system that exhibits chaotic behaviour. Chaos in this context doesn't imply randomness rather, it refers to a type of complex and unpredictable behaviour that arises from deterministic equations. Chaotic systems are characterized by extreme sensitivity to initial conditions, which means that small differences in the starting conditions can lead to vastly different outcomes over time. Chaotic systems have been observed and studied in various fields, including physics, mathematics, biology, and economics.

3.4 TYPES OF CHAOTIC SYSTEM

- Lorenz System
- Rossler System
- Chua's System
- Chen System

3.4.1 LORENTZ SYSTEM

The Lorenz system is a set of three ordinary differential equations that describe a simplified model of atmospheric convection. The Lorenz system is one of the most famous examples of a chaotic dynamical system and played a significant role in the development of chaos theory.

The differential equations for Lorenz system,

- $dx/dt = \sigma(y - x)$
- $dy/dt = x(\rho - z) - y$
- $dz/dt = xy - \beta z$

Parameters: Sigma = 10.0 , Rho = 28.0 , Beta = 8.0 / 3.0

3.4.2 ROSSLER SYSTEM:

The Rossler system is a set of three coupled nonlinear ordinary differential equations that describes a chaotic dynamical system. The Rossler system serves as an example of how relatively simple mathematical equations can lead to highly complex and unpredictable behaviour. These attractors have complex shapes and exhibit the hallmark features of chaotic systems, such as unpredictability and sensitivity to initial conditions.

The differential equations for Rossler system,

- $dx/dt = -y - z$
- $dy/dt = x + a * y$
- $dz/dt = b + z * (x - c)$

Parameters: a = 0.2 , b = 0.2 , c = 5.7

3.4.3 CHAU'S SYSTEM:

Chua's system is known for its chaotic behaviour, including the presence of attractors that exhibit a range of complex and seemingly random patterns. The chaotic nature of the system arises from the nonlinearity introduced by the piecewise linear function f(x). It is a simple electronic circuit that exhibits classic chaotic behaviour. This means roughly that it is a "nonperiodic oscillator" it produces an oscillating waveform.

The differential equations for Chau's system,

- $dx/dt = \alpha(y - x - f(x))$
- $dy/dt = x - y + z$
- $dz/dt = -\beta y$
- $f(x) = m0*x + ((m1-m0)*(|x+1| - |x-1|))$

Parameters: alpha = 15.6 , Beta = 28.0 , m0= -8 / 7 , m1 = -5 / 7

3.4.4 CHEN SYSTEM:

The Chen system exhibits sensitive dependence on initial conditions and demonstrates complex and unpredictable dynamics. The Chen system is known for its chaotic behaviour, characterized by its sensitive dependence on initial conditions, irregular oscillations, and complex attractors in phase space.

The differential equations for Chen system,

- dx/dt = a*(y - x)
- dy/dt = (c - a)x - xz + c*y
- dz/dt = xy – bz

Parameters: a = 35.0 , b = 3.0 , c = 28.0

3.5 EXISTING METHODOLOGY:

3.5.1 AES ALGORITHM

The standard AES is an advanced encryption which has been introduced in 2000 by NIST. The data length in AES is 128 bits, i.e., 16 bytes. However, the key can acquire different lengths (for example, 128, 192, 256 bits). AES has 10, 12 and 14 rounds for 128-bit, 192-bit and 256-bit keys, respectively. Figure 1 shows the block diagram of AES algorithm.

AES has four main operational blocks:

1. Substitute byte transformation: An S-box is used to substitute each data block byte with another block.

2. Shift transformation of rows: Each row of the state matrix is given a cyclic shift to the right side according to its location.

3. Mix Transformation of Columns: It is a matrix multiplication operation where each column of the state matrix is multiplied by that of the fixed matrix.

4. Add Round Key Transformation: XOR operation is performed between the new state matrix and the round key one.

3.5.2 CHAOS SYSTEM

Chaos theory is a branch of mathematics which investigates the extremely complicated systems. In these systems, applying small (seemingly ignorable) changes in the input results in the significant changes in the output.

Chaos system has the following features:

1. Sensitivity to the initial value: Small changes in the initial value lead to a totally different sequence which is achieved through repetitive computations on a chaos map with the parameters.
2. Sensitivity to the parameters: Small changes in the parameters yield a totally different sequence which is achieved through repetitive computations on a chaos map with the

input values.

3. Randomness: The generated chaos sequences using the chaos maps are mostly pseudorandom sequences and their structures are very complex for analysis and prediction. If an unauthorized person does not know the correct control parameters and initial values, he cannot guess the chaos sequence. In other words, chaos systems can improve the safety of the image encryption systems.

3.5.3 EXISTING IMAGE ENCRYPTION METHOD

Three important factors must be considered for designing an algorithm:

- The algorithm must be simple enough to be evaluated and analyzed easily and completely.
- An encryptor must provide security margin more than the required value against the known attacks.
- Well-known, well-examined and reliable instruments and ideas must be used for the design.

According to the above-mentioned points, using the combination of modified AES algorithm and Arnold chaos mapping, an image encryption algorithm is proposed here which is an efficient one from both security and speed aspects.

The overall structure of the standard AES algorithm. Some modifications have been made to make the proposed method suitable for the image encryption. These terms include two modifications to the original AES encryption algorithm: The first modification is the replacement of proposed propagation operations to permutation operation in the standard encryption algorithm, and the second modification is to replace the linear transformation operation with the column integration operation. The block diagram of the proposed encryption method. The details of the proposed algorithm are explained step by step in the following

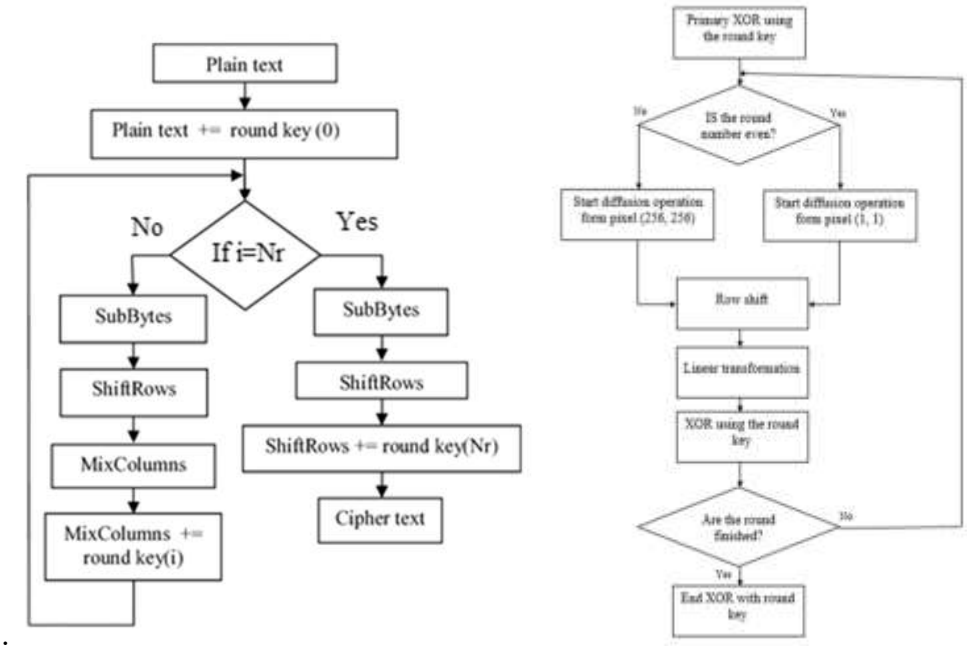


Fig 3.5.3. existing method block diagram

3.6 PROPOSED METHODOLOGY

A strong cryptosystem is based on a strong key generator and robust encryption architecture. The proposed image cryptosystem is presented in below block diagram. Increasing the size of a key will improve the complexity of robustness and it not easy for the attacker to get the actual information. Increasing key size will increase the time of process of attack. It contains the following principal modules

1. DPRNG used for high-quality key generation (384-bit key)
2. Robust encryption/decryption architecture that performs the confusion and diffusion of images.

GENERAL ARCHITECTURE OF PROPOSED CRYPTOSYSTEM

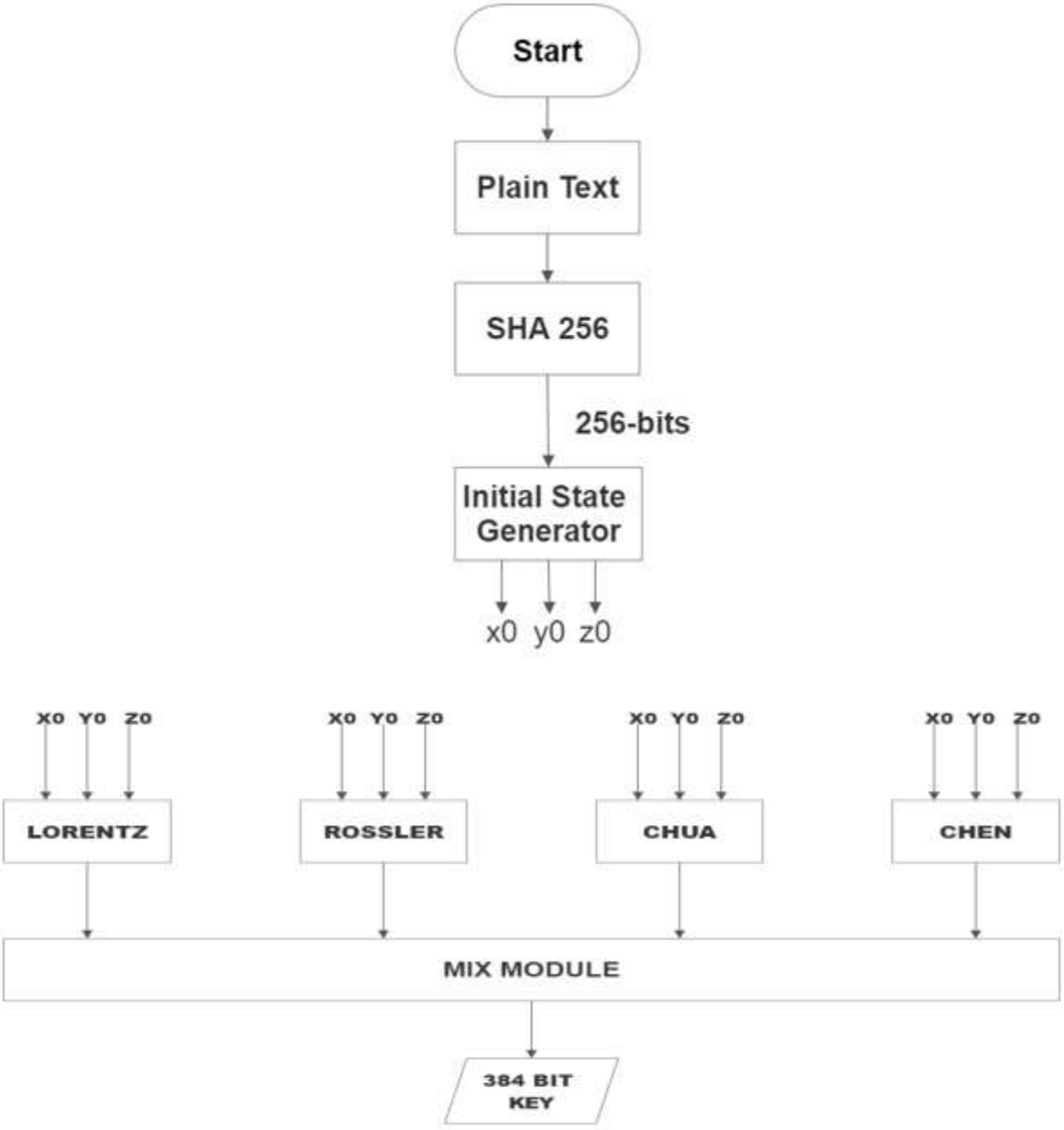


Fig.3.6.1. General Architecture of proposed Cryptosystem

A complex chaos-based DPRNG architecture is proposed to increase the key complexity, entropy, randomness, sensitivity and key space and to avoid determinism, correlation, and key dependence. Hence, the four chaotic systems are functionally combined all together. The overall system includes three principal modules an initial state generator (ISG), a complex chaotic design (CCD) and a random-number generator. The system has one 256 bits input that acquires an initial key and it outputs 384 random bits at each time. The initial key should be fully related to the original image to be encrypted. In our work, the initial key is generated using SHA-256.

Therefore, the variables of the initial state are computed separately using Equation,

- $k_i = k_1|k_2|k_3 \ldots k_{32}$
- $x_0 = \frac{k_1 \oplus k_2 \cdots k_{10}}{2^8}$
- $y_0 = \frac{k_{11} \oplus k_{12} \cdots k_{21}}{2^8}$
- $z_0 = \frac{k_{22} \oplus k_{23} \cdots k_{32}}{2^8}$

3.7 AES ENCRYPTION PROCESS

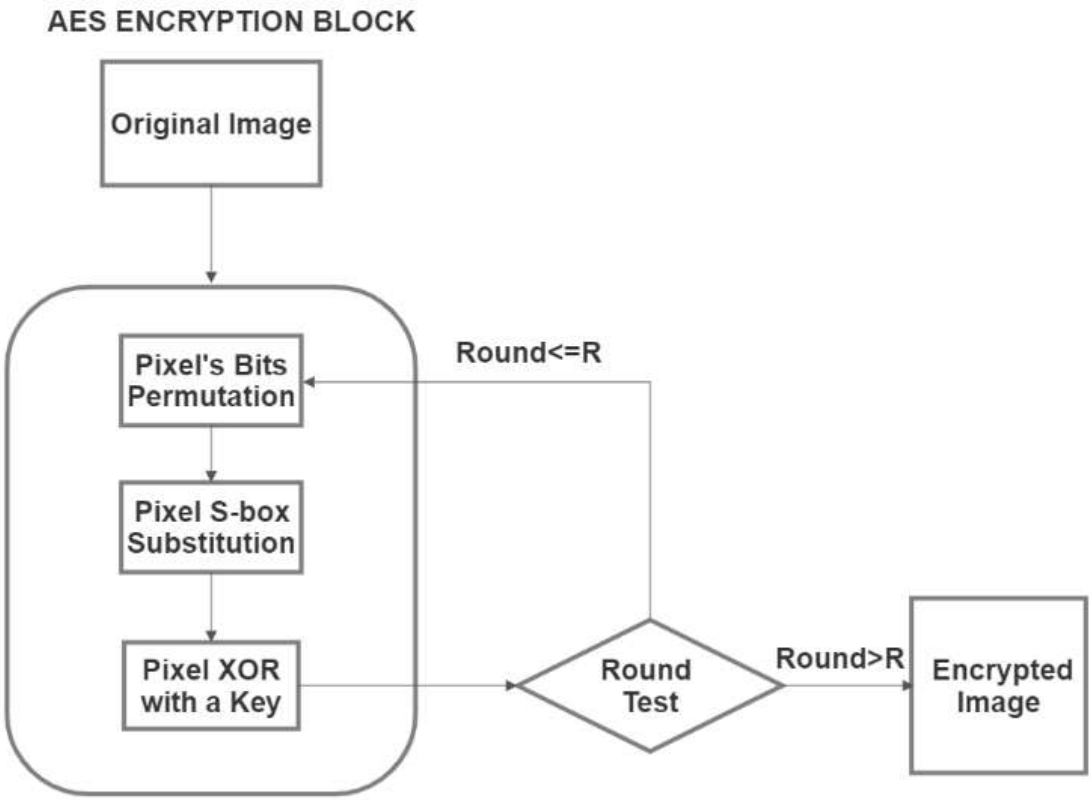


Fig 3.7.1 AES Encryption Process

In the above block diagram show the data confusion-diffusion is proposed as encryption architecture. The overall system performs image encryption mainly by the following three processes:

Process 1: Pixel bits permutation, which guarantees the property of diffusion,

Process 2: Pixel substitution, which ensures the property of confusion,

Process 3: XOR operation between pixels and a key, which ensures the property of diffusion.

The encryption steps are respectively the following:

Step 1: Read an I original image with any size S = N–M–O, where N and M are the images dimensions and O is the number of layers. For the colour image O = 3, the image is decomposed into red, blue and green components. Then, each component is encrypted separately using the encryption system.

Step 2: Execution of Process 1. The pixel bits are permuted by cycling the right shift or cycling the left shift. A condition fully related to the encryption key generated by the DPRNG is used to perform the permutation of pixel bits.

The process is as follows

If the encryption key generated by the DPRNG is even, then the pixel bits are permuted by cycling the right shift by two bits

- Pixel = Pixel $\gg 2 I_{ij} = P_7 P_6$

If the encryption key is odd, then the pixel bits are permuted by cycling the left shift by two bits

- Pixel = Pixel $\ll 2$

Step 3: Execution of Process 2. The image pixels are substituted using two different S-boxes: S-box 1, and S-box 2. The same condition used for pixel's bits permutation is used again to perform the substitution of a pixel

If the encryption key generated by the DPRNG is even, then the pixel is substituted using S-box 1

Subsituted pixel = S − box(Permuted pixel)

If the encryption key generated by the DPRNG is odd, then the pixel is substituted using S-box 2, as presented in Eq. 13.

The S-box is a 256-case substitution table. Let Iij be an 8-bit coded image per pixel and X and Y binary numbers obtained from the pixel of the image

- $I_{ij} = P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0$
- X = $P_3 P_2 P_1 P_0$
- Y = $P_7 P_6 P_5 P_4$

Each pixel of the image is substituted by the state of an S-box which corresponds to the intersection of X with Y. For example, I_{ij} =$(128)_{10}$ = $(10000000)_2$, so X = 0000 and Y = 1000. The value of the image pixel is substituted by the value of the state S = 53.

Step 4: Execution of Process 3. The permuted and substituted pixels are XORed with the key stream. As a result, an intermediate encrypted image (IEI) is obtained.

- IEI = DPRNS $\oplus$ I(i, j)
  To increase complexity, R rounds of encryption can be performed in a loop. In this case, the system processes all last steps again (R rounds) in order to produce the final encrypted image. In our work, we perform only one round of encryption.

Table 1 S-box 1

S-box		X = 4-bit															
	X/Y	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	0000	137	140	253	68	34	133	135	175	12	35	120	119	232	74	78	233
	0001	41	240	31	46	69	37	127	136	79	20	184	193	207	3	24	157
	0010	36	221	73	236	57	167	144	77	118	215	220	226	249	108	156	172
	0011	255	45	165	201	248	49	198	163	58	80	189	30	242	94	237	254
	0100	17	23	114	235	247	18	152	180	15	16	44	204	100	141	224	244
	0101	158	109	149	208	81	199	42	166	39	28	105	32	54	103	150	21
	0110	256	65	178	25	203	125	187	251	112	123	145	174	6	89	87	90
Y = 4-bit	0111	138	82	111	121	241	102	104	110	47	211	106	214	86	98	117	197
	1000	61	107	179	92	188	128	22	75	142	146	246	195	33	93	225	9
	1001	76	143	162	238	181	53	116	200	229	59	52	63	239	139	177	202
	1010	124	194	96	132	51	161	252	155	38	192	50	29	67	1	56	97
	1011	147	227	115	206	148	216	62	186	40	70	171	230	2	151	48	173
	1100	13	223	55	130	219	72	85	185	218	183	213	14	7	83	191	228
	1101	153	169	4	101	205	26	126	10	160	84	182	250	210	60	5	99
	1110	245	11	88	209	243	64	134	190	19	131	95	234	170	212	27	66
	1111	159	164	231	196	43	71	176	129	217	122	91	8	222	154	168	113

3.7.2 S BOX-1

3.8 IMAGE DECRYPTION ALGORITHM

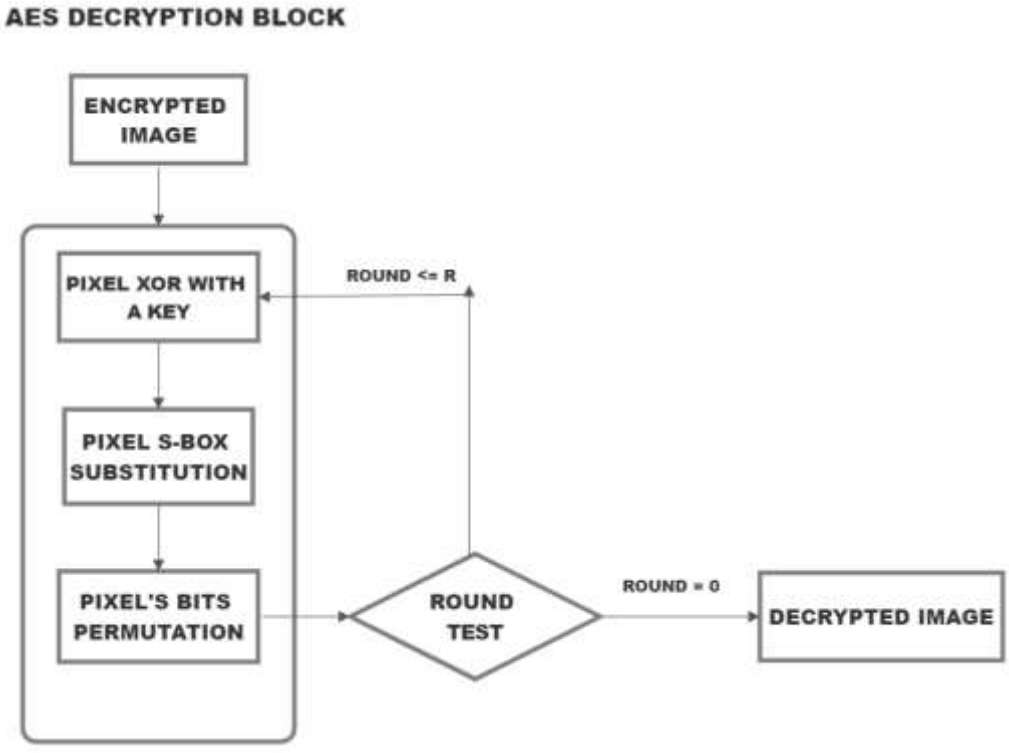


Fig3.8.1 Image Decryption Process

The decryption system translates an encrypted image back to the plain image. It is the inverse algorithm of the corresponding encryption algorithm. Thus, it requires the initial secret key and the encrypted image in order to perform decryption processing. The initial secret key permits generating the same encryption key by utilizing the DPRNG. At the substitution step, both inverse S-box 1 and S-box 2 are used. Table 2 represents the inverse S-box of S-box 1. The general architecture of the decryption system is shown in above block diagram.

Table 2 Inverse S-box 1

S-box		X=4-bit															
	X/Y	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	0000	173	188	29	210	222	108	204	251	143	215	225	8	192	203	72	73
	0001	64	69	232	25	95	134	65	30	99	213	238	89	171	59	18	91
	0010	140	4	9	32	21	168	88	184	16	86	244	74	49	19	120	190
	0011	53	170	164	154	149	92	194	174	36	56	153	221	128	182	155	229
	0100	97	239	172	3	20	185	245	197	34	13	135	144	39	14	24	57
	0101	84	113	205	217	198	124	110	226	109	111	250	131	141	61	234	162
	0110	175	125	223	76	211	117	93	118	90	122	129	45	81	119	114	104
Y=4-bit	0111	255	66	178	150	126	40	11	10	115	249	105	160	101	214	22	133
	1000	247	195	233	163	5	230	6	23	0	112	157	1	77	136	145	38
	1001	106	137	176	180	82	94	189	70	208	253	167	46	31	80	240	216
	1010	165	146	55	241	50	87	37	254	209	236	186	47	191	107	7	246
	1011	158	98	130	71	148	218	201	26	199	183	102	132	58	231	206	169
	1100	27	161	139	243	127	54	85	151	51	159	100	75	212	179	28	83
	1101	227	220	121	237	202	123	41	181	248	200	196	42	33	252	193	78
	1110	142	43	177	207	152	187	242	12	15	235	67	35	62	147	156	17
	1111	116	60	228	79	224	138	68	52	44	219	103	166	2	63	48	96

3.9 TEST PARAMETERS OF SYSTEMS

3.9.1 VARIENCE

The term variance refers to a statistical measurement of the spread between numbers in a data set. More specifically, variance measures how far each number in the set is from the mean (average), and thus from every other number in the set.

Variance is often depicted by this symbol: σ^2.

Variance (σ^2) = $\Sigma(xi - \bar{x})^2 / N$

3.9.2 STANDARD DEVIATION

Standard deviation is a statistic that measures the dispersion of a dataset relative to its mean and is calculated as the square root of the variance.

The standard deviation is calculated as the square root of variance by determining each data point's deviation relative to the mean

Standard Deviation (σ) = $\sqrt{[\Sigma(xi - \bar{x})^2 / N]}$.

3.9.3 CORRELATION

Correlation is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate).

$r = (\Sigma[(Xi - X)(Yi - \bar{Y})]) / [\sqrt{(\Sigma(Xi - \bar{X})^2)}\ \sqrt{(\Sigma(Yi - \bar{Y})^2)}]$

r is the Pearson correlation coefficient,

Xi and Yi are the individual data points of variables X and Y, respectively,

$\bar{X}$ and $\bar{Y}$ are the means of variables X and Y, respectively,

Σ indicates the sum of the values, and

√ represents the square root.

3.9.4 ENTROPY

Entropy is a measure of the number of ways a system can be arranged, often taken to be a measure of "disorder" (the higher the entropy, the higher the disorder).

Entropy(X) = -Σ P(xi) * log2(P(xi))

3.10 RUNGE-KUTTA

The Runge-Kutta method involves breaking down a continuous differential equation into discrete steps that can be computed numerically. The most common form of the Runge-Kutta method is the fourth-order Runge-Kutta (RK4) method, which provides a good balance between accuracy and computational efficiency.

- Differential Equation,

dy/dx=f(x,y)

- In the differential equation dy/dx, we need to find y with respect to x.
- But in our case the signal x,y,z are variables with respect to time 't'.

dy/dx → converted → dx/dt, dy/dt, dz/dt

RUNGE KUTTA FORMULA

- $y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
- $k_1 = hf(x_0, y_0)$
- $k_2 = hf\left(x_0 + \left(\frac{1}{2}\right)h, y_0 + \left(\frac{1}{2}\right)k_1\right)$
- $k_3 = hf\left(x_0 + \left(\frac{1}{2}\right)h, y_0 + \left(\frac{1}{2}\right)k_2\right)$
- $k_4 = hf(x_0 + h, y_0 + k_3)$

3.11 RUNGE KUTTA OUTPUT

```
C:\Users\HP\PycharmProjects\Phase1Project\venv\Scripts\python.exe
sigma=  10.0 rho=  28.0 beta=  2.6666666666666665
Initial Condition//   x0= 1   y0= 1   z0= 1
t_values[i]= 0.0
k1= [ 0.          0.26       -0.01666667]
k2= [ 0.013       0.25878333 -0.01514444]
k3= [ 0.01228917  0.2605373  -0.01509741]
k4= [ 0.02482481  0.26086553 -0.01350379]
y[:, i + 1]  [1.01256719 1.2599178  0.98489097]
```

CHAPTER 4

RESULTS

4.1 TABULATION (INITIAL STATE)

Parameters	Arguments	Initial Condition	At time instants
Manual Calculation	a = 10 , b= 28 , c= 2.6667	x_0= 1, y_0=1, z_0 = 1	t_0 = 0.0
Program Input	a = 10 , b= 28 , c= 2.6666666666666665	x_0= 1, y_0=1, z_0 = 1	t_0 = 0.0

Parameters	K1	K2	K3	K4
Manual Calculation	$k_1(x)$ =0.0 $k_1(y)$ =0.26 $k_1(z)$ =-0.016667	$k_2(x)$ =0.013 $k_2(y)$ =0.25878 $k_2(z)$ =-0.0151447	$k_3(x)$ =0.012289, $k_3(x)$ =0.260518, $k_3(x)$ =-0.015147	$k_4(x)$ =0.024829, $k_4(x)$ =0.260869, $k_4(x)$ =-0.01349
Program Output	$k_1(x)$ =0.0 $k_1(y)$ =0.26 $k_1(z)$ =-0.0166667	$k_2(x)$ =0.013 $k_2(y)$ =0.25878333 $k_2(z)$ =-0.01514444	$k_3(x)$ =0.01228917, $k_3(x)$ =0.2605373, $k_3(x)$ =-0.01509741	$k_4(x)$ =0.02482481, $k_4(x)$ =0.26086553, $k_4(x)$ =-0.01350379

Parameters	$x_1 = x_0 + \Delta x$	$y_1 = y_0 + \Delta y$	$z_1 = yx_0 + \Delta y$
Manual Calculation	1.0125668	1.25991	0.98447
Program Output	1.01256719	1.2599178	0.98489097

Parameters	$t_0 = 0.0$	$t_1 = 0.01$	…….	t_{10} =0.99
Manual Calculation	x_0= 1, y_0= 1, z_0 =1	x_0= 1.0125668, y_0=1.25991, z_0 = 0.98447		
Program Output	x_0= 1, y_0= 1, z_0 =1	x_0= 1.01256719, y_0=1.2599178, z_0 = 0.98489097		x_0= -9.378615807236297, y_0=-8.35705995529234, z_0 = 29.362403750125736

4.2 SIMULATION RESULTS (CHAOTIC SYSTEMS)

4.2.1 LORENTZ'S SYSTEM

The differential equations for Lorenz system,

- $dx/dt = \sigma(y - x)$
- $dy/dt = x(\rho - z) - y$
- $dz/dt = xy - \beta z$

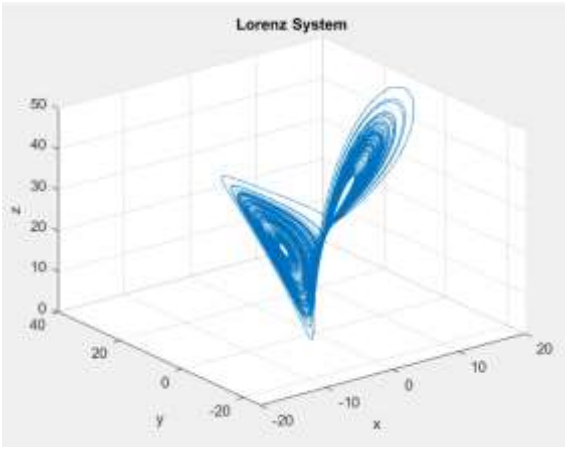


Fig 4.2.1(i) 3D view of Lorentz system

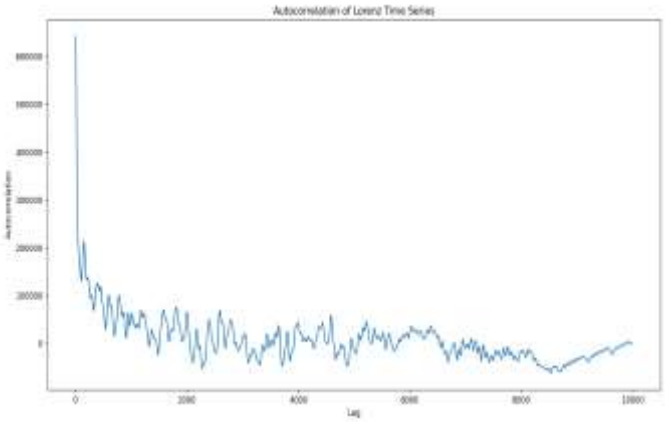


Fig 4.2.1(ii) Autocorreletion Lorentz system

lorentz_test

```
C:\Users\HP\AppData\Local\Programs\Python\Pytho
Variance Test:
Variance: 62.3906

Standard Deviation Test:
Standard Deviation: 7.8988

Correlation Test:
Correlation: 0.1591

Randomness Test:
Difference Variance: 0.1629
The time series appears to be random.

Entropy Test:
Entropy: 6.3224
```

Fig4.2.1(iii) Statistical Parameters

4.2.2 ROSSLER'S SYSTEM

The differential equations for Rossler system,

- dx/dt = -y - z
- dy/dt = x + a * y
- dz/dt = b + z * (x - c)

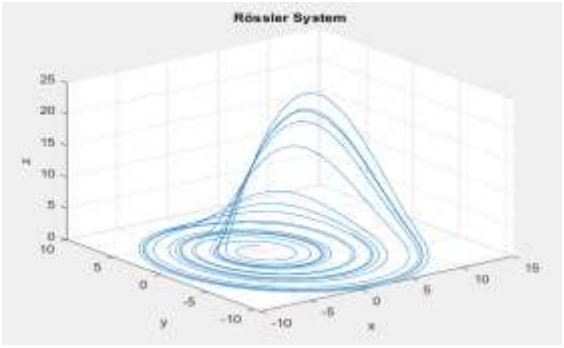


Fig 4.2.2(i) 3D view of Rossler's system

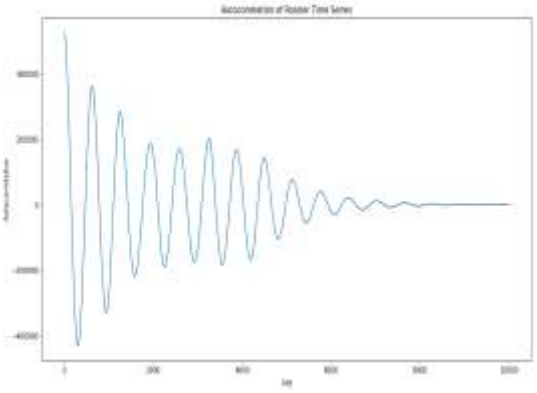


Fig 4.2.12(ii) Autocorreletion Rossler's system

```
rosslers test
C:\Users\HP\AppData\Local\Programs\Pytho
Variance Test:
Variance: 52.2654

Standard Deviation Test:
Standard Deviation: 7.2295

Correlation Test:
Correlation: 0.5059

Randomness Test:
Difference Variance: 0.0076
The time series appears to be random.

Entropy Test:
Entropy: 5.6323

Process finished with exit code 0
```

Fig4.2.2(iii) Statistical Parameters

4.2.3 CHEN'S SYSTEM

The differential equations for Chen system,

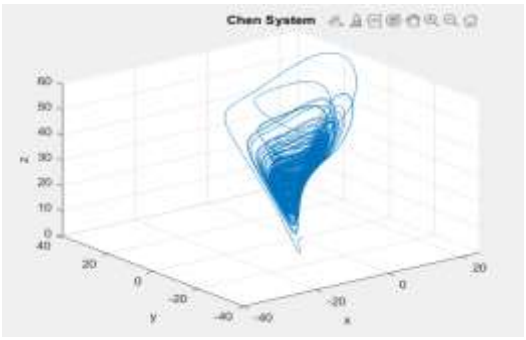


Fig 4.2.3(i) 3D view of Chen system

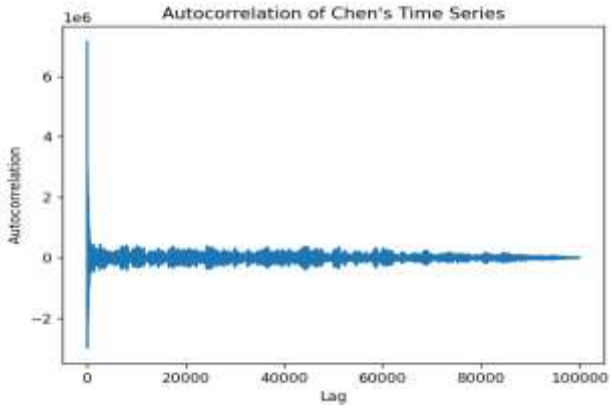


Fig 4.2.3(ii) Autocorreletion Chen system

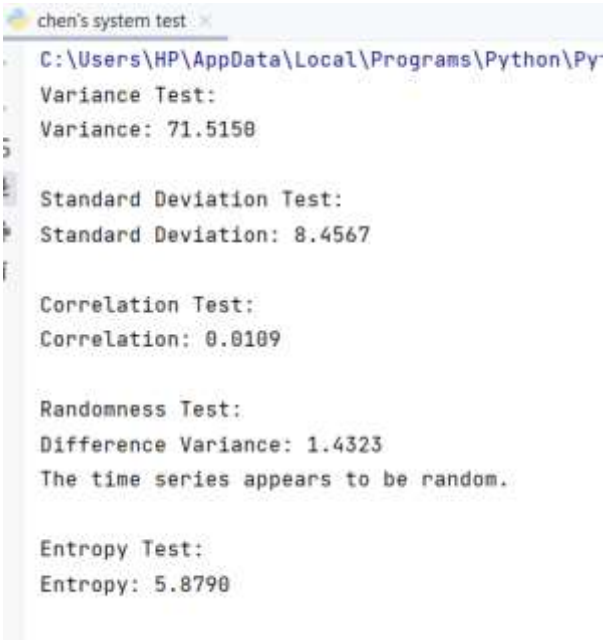


Fig4.2.1(iii) Statistical Parameters

4.2.4 CHUA'S SYSTEM

The differential equations for Chau's system,

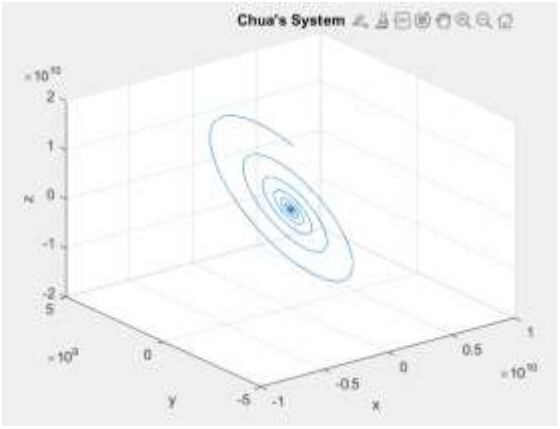


Fig 4.2.4(i) 3D view of Chua system

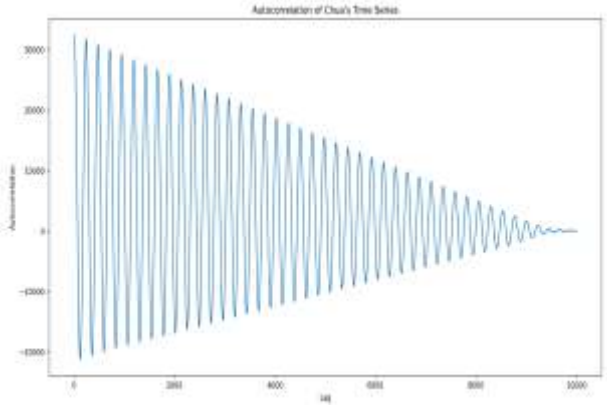


Fig 4.2.4(ii) Autocorreletion Chua system

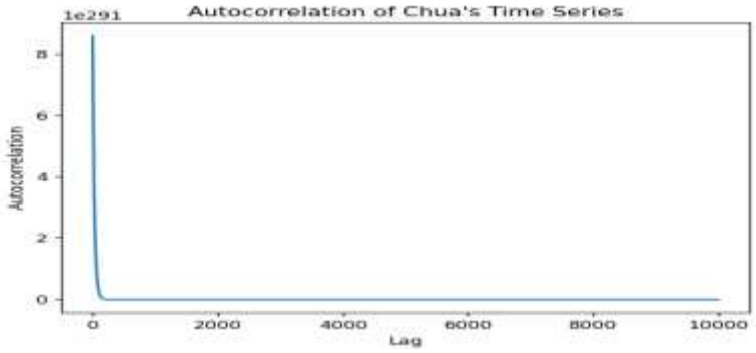
```
if __name__ == "__main__":
    # Adjust the parameters here
    alpha = 15.6
    beta = 28  ## By varing the beta from 25 to 52 periodic doubling // from 30 to 32 boundary crises
    m0 = -8/7
    m1 = -5/7
```

Fig.4.2.4(iii) Parameter Constraints

```
chuas system test
C:\Users\HP\AppData\Local\Programs\Python\Python311\pytho
Variance Test:
Variance: 2.7542

Standard Deviation Test:
Standard Deviation: 1.6596

Correlation Test:
Correlation: 0.0996

Randomness Test:
Difference Variance: 0.0020
The time series appears to be random.

Entropy Test:
Entropy: 6.3393
```

Fig4.2.4(iii) Statistical Parameters

4.2.4.1 Chua's system high variance condition




```
# Adjust the parameters here
alpha = 15.6
beta = 27  ## By varing the beta from 25 to 52 periodic doubling // from 30 to 32 boundary crises
m0 = -8/7
m1 = -5/7
```

```
chuas system test
C:\Users\HP\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\HP\Downloads\chuas system test.py"
Variance Test:
Variance: 906698682213561442678321848594178104407698189902262686897426832386163115862999089797772323325636985134688821512192O.0000

Standard Deviation Test:
Standard Deviation: 3011143648236316287163539145914636995839816911754358685696.0000

Correlation Test:
Correlation: 1.0000

Randomness Test:
Difference Variance: 10766936981542285875147015559769849281209871229131088194088179118878895742628233585855326447602430014922254831257 6.0000
The time series appears to be random.

Entropy Test:
Entropy: 0.1711
```

Fig 4.2.4.1 Chua's system high variance condition

4.3 COMPARISION BETWEEN SYSTEM PARAMETERS

Parameters	Lorentz system	Rosslers ststem	Chen's system	Chua's system
Variance	62.3906	52.2654	71.5150	2.7548
Standard deviation	7.8988	7.2295	8.4567	1.660
Correlation	0.1591	0.5059	0.0109	0.5390
Randomness test	0.1629	0.0076	1.4323	0.0034
Entropy test	6.3224	5.6323	5.8790	6.4502

4.4 IMAGE READ SAMPLE IMAGE

Fig 4.4 Sample image and its properties

4.5 PIXEL VALUES

	A	B	C	D	E	F	G	H
1	#010101	#000000	#000201	#010302	#000201	#000201	#010101	#000000
2	#010101	#010101	#000201	#000201	#000201	#000201	#010101	#000000
3	#000201	#000201	#000201	#000201	#000201	#020403	#020202	#000000
4	#000201	#000201	#000201	#000201	#000302	#020403	#010302	#000000
5	#010101	#010101	#000201	#000201	#000201	#000201	#000201	#000201
6	#010101	#010101	#000201	#000201	#000201	#000201	#000201	#000201
7	#010101	#010101	#010101	#010101	#000201	#000201	#000201	#000201
8	#010101	#010101	#010101	#010101	#000201	#000201	#000201	#010302
9	#010101	#010101	#010101	#010101	#010101	#010101	#000201	#000201
10	#010101	#010101	#010101	#010101	#010101	#010101	#000201	#000201
11	#000201	#000201	#010101	#010101	#010101	#010101	#010101	#010101
12	#000201	#000201	#000201	#010101	#010101	#010101	#010101	#010101
13	#000201	#000201	#000201	#000201	#000201	#000201	#000201	#000201
14	#000201	#000201	#000201	#000201	#000201	#000201	#000201	#000201
15	#010101	#010101	#010101	#010101	#000201	#000201	#000201	#000201
16	#010101	#010101	#020001	#010101	#010101	#000201	#000201	#000201
17	#010101	#020202	#000000	#000000	#010101	#010101	#010101	#010101
18	#010101	#010101	#000000	#000000	#010101	#010101	#010101	#010101
19	#000000	#000000	#010101	#010101	#000201	#000201	#000201	#010302
20	#000000	#000000	#010101	#010101	#000201	#000201	#000201	#000201
21	#000000	#000000	#010101	#010101	#010101	#010101	#010101	#020202
22	#000000	#000000	#010101	#010101	#010101	#010101	#020202	#020202
23	#000000	#000000	#000000	#000000	#010101	#010101	#010101	#000000
24	#000000	#000000	#000000	#000000	#010101	#010101	#010101	#010101
25	#000000	#000000	#000100	#000000	#010100	#020100	#020100	#020100

Fig 4.5 Pixel representation of sample image

4.6 PROGRAM OUTPUTS

4.6.1 INITIAL STATE GERNERATOR OUTPUT

OUTPUT FROM SHA 256 FOR THE INPUT(helloworld)

```
384_bit key
C:\Users\HP\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\HP\Desktop\Project Phase 1\Python Program\38
Enter 1 for keyword as input 0 for image as input: 1
Please enter the keyword: helloworld
Physical_key 936a185caaa266bb9cbe981e9e85cb78cd732b0b3280eb944412bb6f8f8f07af
['10010011', '01101010', '00011000', '01011100', '10101010', '10100010', '01100110', '10111011', '10011100', '1011111
147
152
128
Result (in binary): 01001010
Result (in decimal): 74
Result (in binary): 00000010
Result (in decimal): 2
Result (in binary): 11010101
Result (in decimal): 213
X0:  0.2901960784313726
Y0:  0.00784313725490196
Z0:  0.8352941176470589
key length= 384 bits
final_key:  a87271c070cdd84025555e411992723cdbf6d4c0051136410ff68541ea0686bb32f6a23cc1b77541974eae4151f570bc
```

Fig 4.6.1(i) Physical key gen for word

384-BIT KEY FOR AES SYSTEM(Text= helloworld)

TEXT AS INPUT(t=100)

```
384_bit key
key length= 384 bits
final_key:  791754be6ad5f9c85a93f5c02e0ddd3deb9cc6be069b94c01b981fc1b6a7dd3d2983093d44e7ef41570085415084333e
```

Fig 4.6.1(ii) Final 384 bit key at time t = 100

TEXT AS INPUT(t=10)

```
384_bit key
key length= 384 bits
final_key:  c93ed340f037ca40b0879fc082e6643ce4d9c6bfd4481d41de4f14c81f33153a9bf5663e674b8a414762d441e2b5983d
```

Fig 4.6.1(iii) Final 384 bit key at time t = 10

4.6.2 OUTPUT FROM SHA 256 FOR THE INPUT(image)
"C:\Users\HP\Desktop\Project Phase 1\sampleimage.jpg"

```
384_bit key
C:\Users\HP\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\HP\Desktop\Project Phase 1\Python Progr
Enter 1 for keyword as input 0 for image as input: 0
Enter the location for the image file: "C:\Users\HP\Desktop\Project Phase 1\sampleimage.jpg"
Physical_key cd26ec6931459c94e9417f8478c3ed68f4392e6ae76b6673f220427eaa5a2646
['11001101', '00100110', '11101100', '01101001', '00110001', '01000101', '10011100', '10010100', '11101001', '01
205
127
107
Result (in binary): 10111010
Result (in decimal): 186
Result (in binary): 10101011
Result (in decimal): 171
Result (in binary): 00000000
Result (in decimal): 0
X0:  0.7294117647058823
Y0:  0.6705882352941176
Z0:  0.0
key length= 384 bits
final_key:  c93ed340f037ca40b0879fc0a615653ce4d9c6bfd4481d41de4f14c826a9143a9bf5663e674b8a414762d441b9b6983d
```

Fig 4.6.2 Physical key gen for image

4.7 384-BIT KEY FOR AES SYSTEM(image)

4.7.1 IMAGE AS INPUT(t=100)

```
384_bit key
key length= 384 bits
final_key:  c93ed340f037ca40b0879fc082e6643ce4d9c6bfd4481d41de4f14c81f33153a9bf5663e674b8a414762d441e2b5983d
```

Fig 4.7.1(i) Final 384 bit key at time t = 100

4.7.2 IMAGE AS INPUT(t=10)

```
384_bit key
key length= 384 bits
final_key:  a4bc1dbf31a5bcc0650f37c180a7b03ee69e2dc05a09c7c819195ec15b44583e608ef53c5a31bb418410a141985612c8

Process finished with exit code 0
```

Fig 4.7.2(ii) Final 384 bit key at time t = 10

4.8 384-BIT KEY for 4 Rounds operation

t=100
7c052441e64a5ec093d54dc0a58b2b3cb1a99dc0c90ec1c0bafe9cc06638babab249533f9ffe4b4197e5ea4028abd33d

t=200
d18e0741b7be543ed22e4f4083e3953c7345b73f195591c0ca54d8c0845515baba5c494070d4d8412e1f1b4281189bbc

t=300
f9013c40c36030c113890c40fb319a3c3bfb6940bcbc87c014f512c0606772b3e6742a4115861242c559e741d1cd9abc

t=400
f64400c0285092bfaef250c04d299a3cabd7e03f4ccf33c0ca0544407bb2c434ce31e13e0251a74189aee541b5289abc

4.8.1 Image to 384 blocks

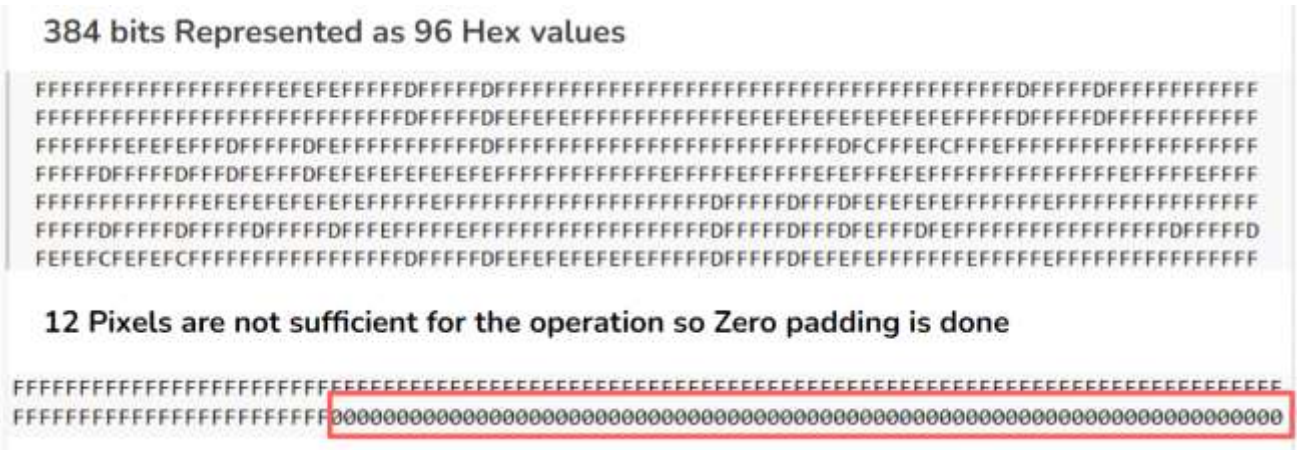


Fig 4.8.1 Image to blocks

4.9 IMAGE ENCRYPTION

4.9.1 Sample Image 1

ORIGINAL IMAGE **ENCRYPTED IMAGE**

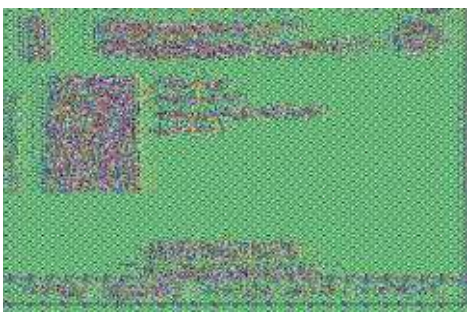


Fig 4.9.1 Sample Image 1

4.9.2 Sample Image 2

ORIGINAL IMAGE **ENCRYPTED IMAGE**

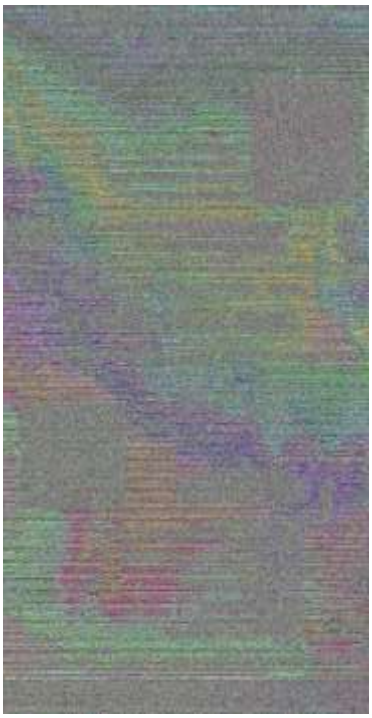


Fig 4.9.2 Sample Image 2

4.10 IMAGE DECRYPTION

ENCRYPTED IMAGE **DECRYPTED IMAGE**

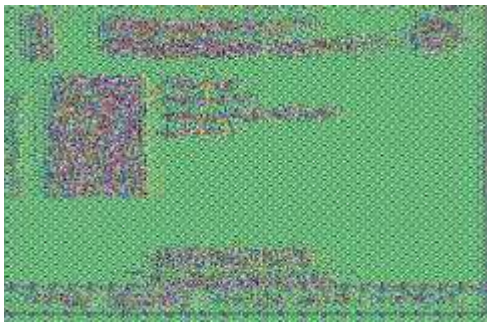


Image ID	
Dimensions	396 x 259
Width	396 pixels
Height	259 pixels
Horizontal resolution	96 dpi
Vertical resolution	96 dpi
Bit depth	24
Size:	56.0 KB (57,346 bytes)
Size on disk:	60.0 KB (61,440 bytes)

Dimensions	396 x 259
Width	396 pixels
Height	259 pixels
Horizontal resolution	96 dpi
Vertical resolution	96 dpi
Bit depth	24
Size:	56.0 KB (57,346 bytes)
Size on disk:	60.0 KB (61,440 bytes)

Fig 10 Image Decryption

4.11 ENCRPTION TEST

1. NPCR (Normalized Pixel Change Rate):

- A high NPCR value is desired for a secure image encryption algorithm.

- Typically, a good encryption algorithm will yield an NPCR value close to 99.61% or higher.

- Higher NPCR values indicate that the encryption algorithm is changing a significant portion of the pixel values, which is important for security. [7]

$$\mathrm{NPCR} = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N} D(i,j)}{M * N} 100\%$$

$$D(i,j) = 0, \quad \text{if} \quad C1(i,j) = C2(i,j)$$

$$D(i,j) = 1, \quad \text{if} \quad C1(i,j) \neq C2(i,j)$$

where:

M and N are the width and height of the encrypted interferogram.

C1(i,j): is the interferogram encrypted before a pixel change.

C2(i,j): is the encrypted interferogram after a pixel change.

D(i,j): is a bipolar network.

2. UACI (Unified Average Change Intensity):

- A low UACI value is desired for a secure image encryption algorithm.

- Typically, a good encryption algorithm will yield a UACI value close to 33.46% or lower.

- Lower UACI values indicate that the encryption algorithm is making subtle changes in pixel values, which is important for security

$$\mathrm{UACI} = \frac{1}{M * N}\left[\frac{\sum_{i=1}^{M} \sum_{j=1}^{N} C1(i,j) - C2(i,j)}{255}\right] 100\%$$

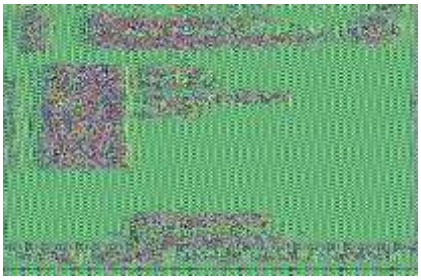
where:

M and N are the width and height of the encrypted interferogram.

C1(i,j): is the interferogram encrypted before a pixel change.

C2(i,j): is the encrypted interferogram after a pixel change.

4.11.1 Sample Image 1



NPCR: 100.0000%
UACI: 33.4112%

Fig 4.11.1 Sample Image 1

4.11.2 Sample Image 2

NPCR: 99.9998%
UACI: 28.8998%

Fig 4.11.2 Sample Image 2

All the test images will pass the both NPCR and UACI test. Hence the proposed encryption will give a perfect encryption result on every images.

4.12 NIST

NIST stands for the National Institute of Standards and Technology, a federal agency within the United States Department of Commerce. NIST is one of the nation's oldest physical science laboratories and is tasked with promoting and maintaining measurement standards, conducting research, and providing technology-related guidance and support to a wide range of industries and government agencies.

NIST's primary mission is to advance measurement science, standards, and technology to enhance economic security and improve the quality of life for the American people. It plays a crucial role in developing and maintaining standards and guidelines in various fields, including:

1. **Information Technology:** NIST publishes guidelines and best practices for cybersecurity, such as the NIST Cybersecurity Framework, and standards like the NIST Special Publications that help organizations secure their information systems and data.

2. **Measurement and Metrology:** NIST is responsible for developing and maintaining measurement standards for a wide range of physical quantities. These standards are critical for ensuring consistency and accuracy in measurements across various industries.

3. **Manufacturing and Engineering**: NIST provides guidance and support for manufacturing processes and quality control, helping businesses maintain and improve their manufacturing capabilities.

4. **Physical Sciences**: NIST conducts research in physics, chemistry, and materials science, contributing to advancements in a wide range of scientific disciplines.

5. **Biotechnology:** NIST supports the biotechnology and pharmaceutical industries by providing measurement standards and research in areas like genomics and proteomics.

6. **Public Safety and Emergency Services**: NIST works on improving public safety by developing standards for emergency communication systems and disaster response technologies.

NIST released a Statistical test for the Pseudorandom Number Generators for cryptographic applications 800-22

The distribution used to perform statistical test:

1. **Standard Normal distributions**

   $z = (x - \mu)/\sigma$, where x is the sample test statistic value, and μ and σ2 are the expected value and the variance of the test statistic.

2. **Chi-Square ($\chi 2$) distributions**

$$\chi 2 = \Sigma \left((O_i - e_i)^2 / e_i \right)$$ where oi and ei are the observed and expected frequencies of occurrence of the measure

1. Frequency (Monobit) Test: This test checks if the proportion of ones and zeros in the sequence is approximately equal. It calculates the total count of ones and zeros and compares it to the expected values for a truly random sequence.

2. Frequency Test within a Block: This test divides the sequence into fixed-length blocks and examines each block's proportion of ones. It checks if the distribution of ones within these blocks deviates from the expected values for randomness.

3. Runs Test: The Runs Test analyzes the runs or consecutive sequences of ones and zeros in the bit sequence. It assesses the number of runs and their lengths to detect patterns or clumping in the data.

4. **Longest Run of Ones in a Block Test:** Similar to the Runs Test, this test looks at the length of the longest run of ones within each block. Deviations from expected values indicate a lack of randomness.

5. Binary Matrix Rank Test: This test divides the sequence into overlapping submatrices and calculates their rank. It checks for linear dependencies by analyzing the rank of these submatrices.

6.Discrete Fourier Transform (Spectral) Test: This test applies the Discrete Fourier Transform to the sequence and examines the resulting spectral components. It looks for significant frequencies that can indicate non-randomness.

7. Non-overlapping Template Matching Test: In this test, predefined templates are compared against the bit sequence to find occurrences of these templates. The frequency of template matches is analyzed to assess the randomness of the sequence.

8. Overlapping Template Matching Test: Similar to the non-overlapping version, this test uses overlapping templates to check for occurrences in the bit sequence.

9. Maurer's "Universal Statistical" Test: This test assesses the unpredictability of the sequence by measuring the number of bits required to predict the next bit. A higher number of bits required indicates higher unpredictability.

10. Random Excursions Test: The Random Excursions Test examines the number of excursions from zero in the sequence. It looks for patterns or irregularities in these excursions.

11. Random Excursions Variant Test: A variant of the Random Excursions Test, this test focuses on a specific count of excursions and examines deviations from the expected distribution.

12. Serial Test: The Serial Test checks for correlations between adjacent bits within the sequence by analyzing pairs of bits. It evaluates whether the sequence exhibits a predictable pattern.

13. Linear Complexity Test: This test measures the linear complexity of the sequence, which is the degree to which it can be predicted by a linear recurrence equation. Higher complexity indicates greater unpredictability.

14. Approximate Entropy Test: The Approximate Entropy Test assesses the complexity of the sequence by comparing the frequency of repeating patterns of a certain length. Lower entropy values indicate more regularity.

15. Cumulative Sums (Cusum) Test: This test examines the cumulative sums of the bits in the sequence. It detects deviations from expected cumulative sums, which could suggest non-randomness.

These tests collectively help evaluate the quality of random number generators and ensure that their output sequences exhibit the characteristics of true randomness.

4.12.1 NIST TEST REULTS

P-value should be greater than 0.01

NAME OF THE TEST	P-VALUE
The Frequency (Monobit) Test	0.83825
Frequency Test within a Block	0.9281
The Runs Test	0.2601
Tests for the Longest-Run-of-Ones in a Block	0.9087
The Binary Matrix Rank Test	0.47452
The Discrete Fourier Transform (Spectral) Test	0.7787
The Non-overlapping Template Matching Test	0.09297
The Approximate Entropy Test	0.60125
The Cumulative Sums (Cusums) Test	0.95256
The Random Excursions Test	0.3287
The Random Excursions Variant Test.	0.4153

During the NIST test analysis for Pseudorandom Number for the 384 bit key sequence, all the 11 test will give a maximum result than the defined P-value of 0.01. Hence the Pseudorandom Number generated from the four Chaos system will give a good result of random sequence, hence it is usable for the further AES encryption system.

CHAPTER – 5

SOFTWARE ENCRYPTION SYSTEM

5.1 CHAOS ENCRYPTION SYSTEM

Fig 5.1 Chaos Encryption system

1. **Imports:**
    - The code starts by importing necessary libraries:
        - tkinter as tk: This imports the tkinter library as tk, which is used for creating the graphical user interface (GUI).
        - from tkinter import filedialog: This imports the filedialog module from tkinter, which is used to create a file dialog for selecting an image.
        - import subprocess: This library is used to run external Python scripts.
        - from PIL import Image, ImageTk: These modules from the Pillow (PIL) library are used for image handling and display.
2. **Function Definitions:**
    - open_file_dialog():
        - This function is called when the "Select Image" button is clicked.
        - It opens a file dialog for the user to select an image file.

- The selected image's file path is displayed on the GUI using the result_label and is also written to a text file named "image_path.txt."

- run_script(script_file):
    - This function is used to run external Python scripts.
    - It takes the name of a script file as an argument.
    - It uses subprocess.run(["python", script_file]) to execute the specified script.
    - If the script file is not found, an error message is displayed on the GUI.

3. **Create the Main Application Window:**
    - An instance of the Tk class is created and assigned to the variable app. This is the main window for the GUI.
    - The window's title is set to "Chaos Encryption System."
4. **Window Configuration:**
    - The window's size is set to 600x550 pixels using the app.geometry() method.
    - Resizing of the window is disabled using app.resizable(False, False) to maintain a fixed window size.
5. **Background Image:**
    - An image (assumed to be named "background.png") is loaded using Pillow (PIL).
    - The image is resized to fit the window dimensions while preserving its aspect ratio.
    - The resized image is displayed as the background of the GUI using a Label widget called bg_label.
6. **Result Label:**
    - A Label widget called result_label is created to display the path of the selected image. Initially, it displays no text.
7. **Buttons:**
    - Six buttons are created for different actions:
        - "Select Image" button: Calls open_file_dialog() to select an image file.
        - "Initial State" button: Runs the "initial_state_generation.py" script.
        - "Key Generation" button: Runs the "key_generation.py" script.
        - "Image to Blocks" button: Runs the "image_to_blocks.py" script.
        - "Image Encryption" button: Runs the "Image_Encryption.py" script.

- "Image Decryption" button: Runs the "Image_Decryption_1.py" script.

8. **Button Styling and Layout:**
    - The buttons are configured with a larger font, specific dimensions (width and height), and a white background color.
    - Buttons are arranged vertically using the pack method, with a specified amount of padding (pady) between them.
9. **Main Event Loop:**
    - The main event loop is started using app.mainloop(). This loop keeps the GUI application running and responsive to user interactions.

CHAPTER 6

CONCLUSION

In conclusion, the project "Image Encryption using Chaos System" has successfully developed a robust and efficient image encryption algorithm based on chaos systems, addressing the pressing need for enhanced data security and privacy in the digital era. By leveraging chaos systems, the algorithm ensures the confidentiality of sensitive visual information and protects digital images from unauthorized access and potential breaches. Number of test are performed to check the robustness of the algorithm. The compatibility with common image formats, optimized performance, and user-friendly interface make it a practical and versatile solution for various applications. Comprehensive documentation, extensive testing, and ethical considerations further enhance the algorithm's effectiveness and responsible use. Overall, the project's outcomes contribute significantly to data security efforts, creating a safer and more secure digital landscape for image sharing, storage, and communication across diverse domains.

FUTURE SCOPE

The future scope of the project "Image Encryption using Chaos System" includes the following potential areas of development and expansion:

1. Advanced Encryption Techniques: Explore and implement more advanced chaos-based encryption techniques to further enhance the security and robustness of the algorithm.

2. Cloud-Based Image Encryption: Investigate the feasibility of adapting the algorithm for image encryption in cloud computing environments to secure images during cloud storage and transmission.

3. Quantum Image Encryption: Research and develop quantum-based image encryption methods to leverage quantum properties for even higher levels of security.

4. Real-Time Image Encryption: Optimize the algorithm to support real-time image encryption for applications like video streaming and live image communication.

5. Integration with Image Processing: Integrate the encryption algorithm with image processing techniques to offer combined functionalities, such as image compression and encryption.

6. Blockchain and Image Authentication: Explore the integration of blockchain technology for image authentication, ensuring the integrity and provenance of encrypted images.

7. Cross-Platform Compatibility: Extend the algorithm's compatibility to support encryption across different platforms and devices, including mobile devices and IoT devices.

8. Machine Learning and AI: Investigate the application of machine learning and artificial intelligence techniques to strengthen encryption and predict potential vulnerabilities.

9. Post-Quantum Cryptography: Research and adapt the algorithm to be resistant against potential future quantum attacks on conventional cryptographic systems.

10. Standardization and Industry Adoption: Work towards standardizing the chaos-based encryption algorithm and promoting its adoption in the industry to enhance overall data security practices.

11. Performance Optimization: Continuously improve the algorithm's performance to ensure faster encryption and decryption times without compromising security.

12. Collaborative Research: Collaborate with academic institutions and industry partners to share findings, conduct joint research, and exchange ideas for further advancements.

The future scope of the project offers numerous opportunities for expanding the encryption algorithm's capabilities, exploring emerging technologies, and addressing evolving data security challenges in the digital landscape. As technology evolves, there will be continuous scope for research and innovation to strengthen image encryption and ensure secure image sharing, storage, and communication.

APPENDIX

1.1 IMAGE READ PROGRAM BLOCK

```
from PIL import Image
import sys
import openpyxl
import subprocess

# Check if a command-line argument is provided
if len(sys.argv) > 1:
    image_path = sys.argv[1]
    try:
        # Open the image file
        image = Image.open(image_path)

        #gettin pixel values
        pixels = list(image.getdata())
        width, height = image.size
        workbook = openpyxl.Workbook()
        sheet = workbook.active

        for row in range(height):
            for col in range(width):
                pixel = pixels[row * width + col]
                pixel_hex = '#{:02x}{:02x}{:02x}'.format(*pixel[:3])
                #pixel_hex = '#{:02x}{:02x}{:02x}'.format(*pixel[:3])
                sheet.cell(row=row + 1, column=col + 1, value=pixel_hex)

        workbook_path = "pixel_values.xlsx"
        workbook.save(workbook_path)
        print("Pixel values saved to", workbook_path)

        image.show() # Display the image
        n = int(input('press 1 for data view data 0 to exit : '))
        if n : subprocess.Popen([workbook_path], shell=True)

        #for pixel in pixels:
        #    print(pixel)

    except IOError:
        print("Unable to open or display the image.")
else:
    print("No image path provided.")
```

1.2 INITIAL STATE GENERATOR BLOCK

```
from func import sha_256_list,sha_256_img,sha_256_text

def xor_numbers(numbers):
    result = int(numbers[0], 2)
    for binary_num in numbers[1:10]:
        num = int(binary_num, 2)
        result ^= num
    return bin(result)[2:].zfill(8)  # Convert the result back to 8-bit
binary

numbers_list = []
f = int(input("Enter 1 for keyword as input 0 for image as input : "))
if f :
    message = input("Please enter the key word: ")
```

```
    print("Physical_key",sha_256_text(message))
    numbers_list = sha_256_list(message)
else :
    image_path = input("Please enter the image path : ")
    try:
        with open(image_path, 'rb') as file:
            image_data = file.read()
        print("Physical_key",sha_256_img(image_data))
        numbers_list = sha_256_list(image_data)
    except FileNotFoundError:
        print("Error: The file does not exist.")
    except Exception as e:
        print("Error:", e)



result_binary_x = xor_numbers(numbers_list)
result_binary_y = xor_numbers(numbers_list[10:])
result_binary_z = xor_numbers(numbers_list[:11])


print("Result (in binary):", result_binary_x)
print("Result (in decimal):", int(result_binary_x, 2))

print("Result (in binary):", result_binary_y)
print("Result (in decimal):", int(result_binary_y, 2))

print("Result (in binary):", result_binary_z)
print("Result (in decimal):", int(result_binary_z, 2))


#divide the number by 2^8 to get the initial value

in_integer_x = int(result_binary_x, 2)
initial_value_x = in_integer_x/((28)-1)
x0 = initial_value_x
print("X0: ", initial_value_x)

in_integer_y = int(result_binary_y, 2)
initial_value_y = in_integer_x/((28)-1)
y0 = initial_value_y
print("Y0: ", initial_value_y)

in_integer_z = int(result_binary_z, 2)
initial_value_z = in_integer_x/((28)-1)
z0 = initial_value_z
print("Z0: ", initial_value_z)
```

1.3 384 BIT KEY GENERATOR BLOCK

```
import numpy as np
from scipy.integrate import solve_ivp
import os
import hashlib
import numpy as np
```

```
import initial_state_generator_block as ini_state

# Rossler system
def rossler(t, state, a, b, c):
    x, y, z = state
    dxdt = -y - z
    dydt = x + a * y
    dzdt = b + z * (x - c)
    return [dxdt, dydt, dzdt]

# Lorenz system
def lorenz(t, state, sigma, rho, beta):
    x, y, z = state
    dxdt = sigma * (y - x)
    dydt = x * (rho - z) - y
    dzdt = x * y - beta * z
    return [dxdt, dydt, dzdt]

# Chen's system
def chen(t, state, a, b, c):
    x, y, z = state
    dxdt = a * (y - x)
    dydt = (c - a) * x - x * z + c * y
    dzdt = x * y - b * z
    return [dxdt, dydt, dzdt]

def chua(t, state, alpha, beta, m0, m1):
    x, y, z = state
    dxdt = alpha * (y - x) - m0 * x - ((m1-m0)* abs(x+1) - abs(x-1))/2
    dydt = x - y + z
    dzdt = -beta * y
    return [dxdt, dydt, dzdt]

def runge_kutta(f, t_span, y0, args=(), h=0.01):
    t_values = np.arange(t_span[0], t_span[1] + h, h)
    y = np.zeros((len(y0), len(t_values)))
    y[:, 0] = y0

    for i in range(len(t_values) - 1):
        k1 = h * np.array(f(t_values[i], y[:, i], *args))
        k2 = h * np.array(f(t_values[i] + h / 2, y[:, i] + k1 / 2, *args))
        k3 = h * np.array(f(t_values[i] + h / 2, y[:, i] + k2 / 2, *args))
        k4 = h * np.array(f(t_values[i] + h, y[:, i] + k3, *args))
        y[:, i + 1] = y[:, i] + (k1 + 2 * k2 + 2 * k3 + k4) / 6

    return t_values, y

def keyfinal(t_values,sol):
```

```
    x,y,z=sol
    step_counter=0
    for i in range(len(t_values)):
        if i % 100 == 0:
            t = t_values[i]
            x_val = x[i]
            y_val = y[i]
            z_val = z[i]
            step_counter += 1
    key=[x_val,y_val,z_val]
    return key

# System parameters for Rossler system
a_rossler, b_rossler, c_rossler = 0.2, 0.2, 5.7

# System parameters for Lorenz system
sigma_lorenz, rho_lorenz, beta_lorenz = 10.0, 28.0, 8.0 / 3.0

# System parameters for Chen's system
a_chen, b_chen, c_chen = 35.0, 3.0, 28.0

# System parameters for Chua's system
a_chua, b_chua, m0_chua, m1_chua = 15.6, 28.0, -1.143, -0.714

# Initial conditions for all systems
x0, y0, z0 =  ini_state.x0,ini_state.y0,ini_state.z0

# Time span for integration
t_span = (0, 100)

# Numerically solve the differential equations using the Runge-Kutta method (runge_kutta)
t_values_r, sol_r = runge_kutta(rossler, t_span, [x0, y0, z0], args=(a_rossler, b_rossler, c_rossler))
t_values_l, sol_l = runge_kutta(lorenz, t_span, [x0, y0, z0], args=(sigma_lorenz, rho_lorenz,
beta_lorenz))
t_values_c, sol_c = runge_kutta(chen, t_span, [x0, y0, z0], args=(a_chen, b_chen, c_chen))
t_values_ch, sol_ch = runge_kutta(chua, t_span, [x0, y0, z0], args=(a_chua, b_chua, m0_chua,
m1_chua))

key1=keyfinal(t_values_r, sol_r )
key2=keyfinal(t_values_l, sol_l)
key3=keyfinal(t_values_c, sol_c)
key4=keyfinal(t_values_ch, sol_ch)

sol=np.array([[np.float32(key1[0]),np.float32(key2[0]),np.float32(key3[0]),np.float32(key4[0])],[np.flo
at32(key1[1]),np.float32(key2[1]),np.float32(key3[1]),np.float32(key4[1])],[np.float32(key1[2]),np.flo
at32(key2[2]),np.float32(key3[2]),np.float32(key4[2])]])
finalkey=sol.tobytes().hex()
print("key length=",len(finalkey)*4,"bits")
```

```
print("final_key: ",finalkey)
```

1.4 IMAGE TO BLOCKS

```
from PIL import Image

# Open the color image

input_file_path = 'image_path.txt'

# Read the hex strings from the text file into an array
image_path = ""
with open(input_file_path, 'r') as file:
    for line in file:
        image_path = line.strip()

image = Image.open(image_path)

# Convert the image to a NumPy array
image_data = image.load()

# Initialize an empty list to store the hexadecimal values
hex_values = []
hex_string = ""
count = 0

print(image.height, image.width)

# Iterate through each pixel and convert it to a hexadecimal value
for y in range(image.height):
    for x in range(image.width):
        pixel = image_data[x, y]
        hex_value = "{:02X}{:02X}{:02X}".format(pixel[0], pixel[1],
pixel[2])
        hex_string += hex_value
        count += 1

        # If we have processed 16 pixels, add hex_string to hex_values and
reset
        if count == 16:
            hex_values.append(hex_string)
            hex_string = ""
            count = 0

print(len(hex_values), count)

# If there are fewer than 16 pixels, add zero-padding
while count < 16:
    hex_string += "000000"
    count += 1
    # You can also break out of the loop if you want exactly 16 values,
    # even if there are fewer pixels available

# Append the last hex string to hex values
hex_values.append(hex_string)

print(hex_values[-1])

with open('hex_values.txt', 'w') as file:
```

```
    for hex_value in hex_values:
        if (len(hex_value) == 96):
            file.write(hex_value + '\n')

print("Hex values saved to hex_values.txt")
```

1.5 IMAGE ENCRYPTION

```
import numpy as np
from PIL import Image

# Specify the input file path with hex strings
input_file_path = 'hex_values.txt'

# Read the hex strings from the text file into an array
hex_strings = []
with open(input_file_path, 'r') as file:
    for line in file:
        hex_strings.append(line.strip())

s_box = [
    137, 140, 253, 68, 34, 133, 135, 175, 12, 35, 120, 119, 232, 74, 78,
233,
    41, 240, 31, 46, 69, 37, 127, 136, 79, 20, 184, 193, 207, 3, 24, 157,
    36, 221, 73, 236, 57, 167, 144, 77, 118, 215, 220, 226, 249, 108, 156,
172,
    255, 45, 165, 201, 248, 49, 198, 163, 58, 80, 189, 30, 242, 94, 237,
254,
    17, 23, 114, 235, 247, 18, 152, 180, 15, 16, 44, 204, 100, 141, 224,
244,
    158, 109, 149, 208, 81, 199, 42, 166, 39, 28, 105, 32, 54, 103, 150,
21,
    0, 65, 178, 25, 203, 125, 187, 251, 112, 123, 145, 174, 6, 89, 87, 90,
    138, 82, 111, 121, 241, 102, 104, 110, 47, 211, 106, 214, 86, 98, 117,
197,
    61, 107, 179, 92, 188, 128, 22, 75, 142, 146, 246, 195, 33, 93, 225, 9,
    76, 143, 162, 238, 181, 53, 116, 200, 229, 59, 52, 63, 239, 139, 177,
202,
    124, 194, 96, 132, 51, 161, 252, 155, 38, 192, 50, 29, 67, 1, 56, 97,
    147, 227, 115, 206, 148, 216, 62, 186, 40, 70, 171, 230, 2, 151, 48,
173,
    13, 223, 55, 130, 219, 72, 85, 185, 218, 183, 213, 14, 7, 83, 191, 228,
    153, 169, 4, 101, 205, 26, 126, 10, 160, 84, 182, 250, 210, 60, 5, 99,
    245, 11, 88, 209, 243, 64, 134, 190, 19, 131, 95, 234, 170, 212, 27,
66,
    159, 164, 231, 196, 43, 71, 176, 129, 217, 122, 91, 8, 222, 154, 168,
113
]

a = []
for i in range(0, 256):
    a.append(i)

for i in a:
    if (i not in s_box):
        print(i)
```

```
permutation_table = [20, 7, 36, 13, 23, 33, 3, 27, 17, 46, 9, 41, 44, 22,
12, 29, 18, 1, 32, 14,
                     26, 8, 4, 11, 38, 45, 19, 2, 30, 21, 47, 34, 5, 10,
15, 6, 25, 24, 31, 37, 0,
                     42, 43, 35, 40, 28, 16, 39]

input_file_path = 'key_strings.txt'

# Read the hex strings from the text file into an array
key_string = []
with open(input_file_path, 'r') as file:
    for line in file:
        key_string.append(line.strip())

Encrpyted_strings = []


def substitute_bits(input_8bit, s_box):
    return s_box[input_8bit]


for data_string in hex_strings:
    if (len(data_string) != 96):
        print(data_string, len(data_string))

    original_384_bit = [int(data_string[i:i + 2], 16) for i in range(0,
len(data_string), 2)]

    for i in range(len(key_string)):
        key_str = key_string[i]

        key_bytes = [int(key_str[i:i + 2], 16) for i in range(0,
len(key_str), 2)]

        actual_value = sum(key_bytes) % 2

        if actual_value % 2 == 1:
            shifted_data_string = data_string[2:] + data_string[:2]
        else:
            shifted_data_string = data_string[-2:] + data_string[:-2]

        original_384_bit = [int(shifted_data_string[i:i + 2], 16) for i in
range(0, len(shifted_data_string), 2)]

        try:
            permuted_384_bit = [original_384_bit[index] for index in
permutation_table]
        except IndexError:
            print(len(original_384_bit))
            print(original_384_bit)

        substituted_384_bit = [substitute_bits(value, s_box) for value in
permuted_384_bit]

        result = 0

        byte = ''.join(format(x, '08b') for x in key_bytes)

        s_384_bit = ''.join(format(x, '08b') for x in substituted_384_bit)
        result = ""
```

```
        for i in range(len(byte)):
            if (byte[i] == "0" and s_384_bit[i] == "0" or byte[i] == "1"
and s_384_bit[i] == "1"):
                result += "0"
            else:
                result += "1"

        result_hex = ''.join([hex(int(result[i:i + 4], 2))[2:].upper() for
i in range(0, len(result), 4)])
        data_string = result_hex

    Encrpyted_strings.append(result_hex)

# Specify the output file path
output_file_path = 'Encrpyted_strings.txt'

# Write the hex strings to a text file
with open(output_file_path, 'w') as file:
    for hex_string in Encrpyted_strings:
        file.write(hex_string + '\n')

print(f"Encrpyted_Hex_strings saved to '{output_file_path}'")

input_file_path = 'Encrpyted_strings.txt'

# Read the hex strings from the text file into an array
hex_values = []
with open(input_file_path, 'r') as file:
    for line in file:
        hex_values.append(line.strip())

input_file_path = 'image_path.txt'

# Read the hex strings from the text file into an array
image_path = ""
with open(input_file_path, 'r') as file:
    for line in file:
        image_path = line.strip()

# Replace this with the path to your original image
original_image_path = image_path

# Open the original color image
original_image = Image.open(original_image_path)

# Define the width and height of the original image
original_width, original_height = original_image.size

# Initialize an empty list to store the hexadecimal values

hex_string = ""
# Create a new image using the dimensions of the original image
reconstructed_image = Image.new('RGB', (original_width, original_height))

# Iterate through the hexadecimal values and reconstruct the image
pixel_data = []
pixel_count = 0

for hex_string in hex_values:
    for i in range(0, len(hex_string), 6):  # Read six characters at a time
        if pixel_count < original_width * original_height:
```

```
            hex_color = hex_string[i:i + 6]
            r = int(hex_color[0:2], 16)
            g = int(hex_color[2:4], 16)
            b = int(hex_color[4:6], 16)
            pixel_data.append((r, g, b))
            pixel_count += 1

# Put the pixel data into the reconstructed image
reconstructed_image.putdata(pixel_data)

# Display the reconstructed image
reconstructed_image.show()

reconstructed_image.save(r"Encrpyted.jpg")
```

1.6 IMAGE DECRYPTION

```
import numpy as np
from PIL import Image

# Specify the input file path with hex strings


input_file_path = 'Encrpyted_strings.txt'

# Read the hex strings from the text file into an array
Encrpyted_strings = []
with open(input_file_path, 'r') as file:
    for line in file:
        Encrpyted_strings.append(line.strip())

permutation_table = [20, 7, 36, 13, 23, 33, 3, 27, 17, 46, 9, 41, 44, 22,
12, 29, 18, 1, 32, 14,
                     26, 8, 4, 11, 38, 45, 19, 2, 30, 21, 47, 34, 5, 10,
15, 6, 25, 24, 31, 37, 0,
                     42, 43, 35, 40, 28, 16, 39]

input_file_path = 'key_strings.txt'

# Read the hex strings from the text file into an array
key_string = []
with open(input_file_path, 'r') as file:
    for line in file:
        key_string.append(line.strip())

inverse_s_box = [
    173, 188, 29, 210, 222, 108, 204, 251, 143, 215, 225, 8, 192, 203, 72,
73,
    64, 69, 232, 25, 95, 134, 65, 30, 99, 213, 238, 89, 171, 59, 18, 91,
    140, 4, 9, 32, 21, 168, 88, 184, 16, 86, 244, 74, 49, 19, 120, 190,
    53, 170, 164, 154, 149, 92, 194, 174, 36, 56, 153, 221, 128, 182, 155,
229,
    97, 239, 172, 3, 20, 185, 245, 197, 34, 13, 135, 144, 39, 14, 24, 57,
    84, 113, 205, 217, 198, 124, 110, 226, 109, 111, 250, 131, 141, 61,
234, 162,
    175, 125, 223, 76, 211, 117, 93, 118, 90, 122, 129, 45, 81, 119, 114,
104,
    255, 66, 178, 150, 126, 40, 11, 10, 115, 249, 105, 160, 101, 214, 22,
```

```
133,
    247, 195, 233, 163, 5, 230, 6, 23, 0, 112, 157, 1, 77, 136, 145, 38,
    106, 137, 176, 180, 82, 94, 189, 70, 208, 253, 167, 46, 31, 80, 240,
216,
    165, 146, 55, 241, 50, 87, 37, 254, 209, 236, 186, 47, 191, 107, 7,
246,
    158, 98, 130, 71, 148, 218, 201, 26, 199, 183, 102, 132, 58, 231, 206,
169,
    27, 161, 139, 243, 127, 54, 85, 151, 51, 159, 100, 75, 212, 179, 28,
83,
    227, 220, 121, 237, 202, 123, 41, 181, 248, 200, 196, 42, 33, 252, 193,
78,
    142, 43, 177, 207, 152, 187, 242, 12, 15, 235, 67, 35, 62, 147, 156,
17,
    116, 60, 228, 79, 224, 138, 68, 52, 44, 219, 103, 166, 2, 63, 48, 96
]

inverse_permutation_table = [0] * len(permutation_table)

for index, value in enumerate(permutation_table):
    inverse_permutation_table[value] = index


def inverse_substitute_bits(output_8bit, inverse_s_box):
    return inverse_s_box[output_8bit - 1]


# Convert the encrypted result_hex back to a list of bytes


Decrpyted_strings = []

for data_string in Encrpyted_strings:

    for i in range(len(key_string) - 1, -1, -1):

        try:
            encrypted_bytes = bytes.fromhex(data_string)
        except ValueError:
            pass

        # Step 1: Undo the XOR operation with the key
        decrypted_substituted_384_bit = []

        key_bytes = bytes.fromhex(key_string[i])
        actual_value = sum(key_bytes) % 2
        for i, byte in enumerate(key_bytes):
            xor_value = byte ^ encrypted_bytes[i]
            decrypted_substituted_384_bit.append(xor_value)

        # Step 2: Undo the inverse S-box substitution
        decrypted_original_384_bit = [
            inverse_substitute_bits(value, inverse_s_box) for value in
decrypted_substituted_384_bit]

        decrypted_original_384_bit = [decrypted_original_384_bit[index] for
index in inverse_permutation_table]

        # Step 3: Undo the shifting
        if actual_value % 2 == 1:
            decrypted_original_384_bit = decrypted_original_384_bit[-1:] +
```

```
decrypted_original_384_bit[:-1]
        else:
            decrypted_original_384_bit = decrypted_original_384_bit[1:] +
decrypted_original_384_bit[:1]

        # Convert the decrypted values back to a hexadecimal string
        decrypted_data_string = ''.join(format(byte, '02X') for byte in
decrypted_original_384_bit)
        data_string = decrypted_data_string

    Decrpyted_strings.append(decrypted_data_string)

# Specify the output file path
output_file_path = 'Decrpyted_strings.txt'

# Write the hex strings to a text file
with open(output_file_path, 'w') as file:
    for hex_string in Decrpyted_strings:
        file.write(hex_string + '\n')

print(f"Decrpyted_Hex_strings saved to '{output_file_path}'")

input_file_path = 'Decrpyted_strings.txt'

# Read the hex strings from the text file into an array
hex_values = []
with open(input_file_path, 'r') as file:
    for line in file:
        hex_values.append(line.strip())

input_file_path = 'image_path.txt'

# Read the hex strings from the text file into an array
image_path = ""
with open(input_file_path, 'r') as file:
    for line in file:
        image_path = line.strip()

# Replace this with the path to your original image
original_image_path = image_path

# Open the original color image
original_image = Image.open(original_image_path)

# Define the width and height of the original image
original_width, original_height = original_image.size

# Initialize an empty list to store the hexadecimal values

hex_string = ""
# Create a new image using the dimensions of the original image
reconstructed_image = Image.new('RGB', (original_width, original_height))

# Iterate through the hexadecimal values and reconstruct the image
pixel_data = []
pixel_count = 0

for hex_string in hex_values:
    for i in range(0, len(hex_string), 6):  # Read six characters at a time
        if pixel_count < original_width * original_height:
            hex_color = hex_string[i:i + 6]
```

```
            r = int(hex_color[0:2], 16)
            g = int(hex_color[2:4], 16)
            b = int(hex_color[4:6], 16)
            pixel_data.append((r, g, b))
            pixel_count += 1

# Put the pixel data into the reconstructed image
reconstructed_image.putdata(pixel_data)

# Display the reconstructed image
reconstructed_image.show()

reconstructed_image.save(r"Decrpyted.jpg")
```

1.7 CHOAS ENCRYPTION SYSTEM

```
import tkinter as tk
from tkinter import filedialog
import subprocess
from PIL import Image, ImageTk  # Import the necessary Pillow modules

def open_file_dialog():
    file_path = filedialog.askopenfilename()
    if file_path:
        result_label.config(text=f"Selected Image: {file_path}")
        with open("image_path.txt", "w") as file:
            file.write(file_path)

def run_script(script_file):
    try:
        subprocess.run(["python", script_file])
        # Print a message in the command window
        print(f"Script '{script_file}' has been executed.")
    except FileNotFoundError:
        result_label.config(text="Script file not found")

# Create the main application window
app = tk.Tk()
app.title("Image Processing System")

# Set the window size
app.geometry("600x550")  # Set your desired width and height

# Disable window resizing
app.resizable(False, False)bg_image = Image.open("background.png")
bg_image = bg_image.resize((600, 550), Image.ANTIALIAS)  # Set the window size as the new size
bg_photo = ImageTk.PhotoImage(bg_image)
```

```
# Create a label to display the background image
bg_label = tk.Label(app, image=bg_photo)
bg_label.place(relwidth=1, relheight=1)

# Create a label to display the selected image path
result_label = tk.Label(app, text="")
result_label.pack()
button_font = ('Helvetica', 16)
button_width = 20
button_height = 2

button_bg_color = "white"

open_button = tk.Button(app, text="Select Image", command=open_file_dialog, font=button_font,
width=button_width, height=button_height, bg=button_bg_color)
initial_state_button = tk.Button(app, text="Initial State", command=lambda:
run_script("initial_state_generation.py"), font=button_font, width=button_width,
height=button_height, bg=button_bg_color)
key_generation_button = tk.Button(app, text="Key Generation", command=lambda:
run_script("key_generation.py"), font=button_font, width=button_width, height=button_height,
bg=button_bg_color)
image_to_blocks = tk.Button(app, text="Image to Blocks", command=lambda:
run_script("image_to_blocks.py"), font=button_font, width=button_width, height=button_height,
bg=button_bg_color)
image_encryption_button = tk.Button(app, text="Image Encryption", command=lambda:
run_script("Image_Encryption.py"), font=button_font, width=button_width, height=button_height,
bg=button_bg_color)
image_decryption_button = tk.Button(app, text="Image Decryption", command=lambda:
run_script("Image_Decryption_1.py"), font=button_font, width=button_width,
height=button_height, bg=button_bg_color)

# Add the 6th button
image_to_blocks_button = tk.Button(app, text="Image to Blocks", command=lambda:
run_script("image_to_blocks.py"), font=button_font, width=button_width, height=button_height,
bg=button_bg_color)

# Pack buttons to the GUI
open_button.pack(pady=10)
initial_state_button.pack(pady=10)
key_generation_button.pack(pady=10)
image_to_blocks_button.pack(pady=10)
image_encryption_button.pack(pady=10)
image_decryption_button.pack(pady=10)

# Start the main event loop
app.mainlo
```

REFERENCES

1. Mohamed Gafsi, Mohamed Ali Hajjaji Jihene Malek, Abdellatif Mtibaa.: FPGA hardware acceleration of an improved chaos-based cryptosystem for real-time image encryption and decryption. Journal of Ambient Intelligence and Humanized Computing, vol 14, pp. 7001–7022, (2023) https://doi.org/10.1007/s12652-021-03555-5

2. Merah Lahcene · Chaib Noureddine · Pascal Lorenz · Ali-Pacha Adda.: Securing information using a proposed reliable chaos-based stream cipher: with real-time FPGA-based wireless connection implementation. Nonlinear Dyn, vol. 111, pp. 801–830, (2023) https://doi.org/10.1007/s11071-022-07824-6

3. K. Saranya , K. N. Vijeyakumar.: A Low Area FPGA Implementation of Reversible Gate Encryption with Heterogeneous Key Generation, Circuits, Systems, and Signal Processing, vol. 40, pp. 3836–3865, (2021) https://doi.org/10.1007/s00034-021-01649-1

4. An image encryption method based on chaos system and AES algorithm Alireza Arab1 · Mohammad Javad Rostami1 · Behnam Ghavami1 .The Journal of Supercomputing (2019) 75:6663–6682 https://doi.org/10.1007/s11227-019-02878-7

5. On differences and similarities in the analysis of Lorenz, Chen, and Lu systems G.A. Leonov, N.V. Kuznetsov http://dx.doi.org/10.1016/j.amc. 2014.12.132 0096-3003/ 2015 The Authors. Published by Elsevier Inc

6. Bridge The Gap Between The Lorenz System And The Chen System JINHU LÜ, GUANRONG CHEN, DAIZHAN CHENG, and SERGEJ CELIKOVSKYInternational Journal of Bifurcation and Chaos, Vol. 12, No. 12 (2002) 2917–2926 https://doi.org/10.1142/S021812740200631X

7. Number of Pixel Change Rate and Unified Average Changing Intensity for Sensitivity Analysis of Encrypted inSAR Interferogram by Riad Saidi1 , Nada Cherrid , Tarek Bentahar, Hicham Mayache, Atef Bentahar.

BOOK AS REFERENCE

1. **Digital Image Processing**, 3rd edition Book by **Rafael C. Gonzalez and Richard E. Woods**

2. **Linear Algebra and Its Applications** (4th Edition) Textbook by **David C Lay, Judi J. McDonald, and Steven R Lay**

3. **Cryptography and Network Security**: Principles and Practice Book by **William Stallings**