The Scientific Temper (2024) Vol. 15 (4): 3113-3121
Doi: 10.58414/SCIENTIFICTEMPER.2024.15.4.19
E-ISSN: 2231-6396, ISSN: 0976-8653
https://scientifictemper.com/

RESEARCH ARTICLE

ECDS: Enhanced cloud data security technique to protect data being stored in cloud infrastructure

Aruljothi Rajasekaran*, **Jemima Priyadarsini R.**

Abstract

The rapid adoption of cloud computing has transformed IT resource management by providing scalable, flexible, and cost-effective solutions. Despite these benefits, cloud computing presents critical security challenges, particularly in protecting sensitive data during transmission and storage. This paper introduces the enhanced cloud data security (ECDS) technique, a new approach aimed at strengthening data protection within cloud infrastructures. ECDS incorporates substitution and permutation methods to secure data and utilizes a combination of encryption strategies to ensure that encrypted data remains inaccessible to unauthorized users. ECDS is a symmetric cryptographic system that uses the same key for encryption and decryption. It is 256-bit block cipher encryption and it uses 312-but keys. The ECDS is implemented in Python and compared against DES and blowfish encryption techniques. Extensive testing and performance analysis reveal that ECDS significantly enhances security and efficiency compared to traditional encryption methods. This paper contributes to the ongoing efforts to secure cloud computing environments for safeguarding sensitive data in the cloud.

Keywords: Cloud computing security, Data protection, Cryptographic, Symmetric encryption, Data encryption.

Introduction

Cloud computing offers a technological solution that provides on-demand access to a variety of configurable computing resources—such as servers, storage, applications, and services—via the internet. This system enables both businesses and individuals to utilize scalable and flexible IT resources without requiring significant upfront investments in hardware and infrastructure (I. Sudha *et al.*, 2024). The pay-as-you-go model of cloud computing enhances collaboration, streamlines operations, and brings cost efficiencies, allowing users to adjust resources according to their needs (Pronika *et al.*, 2021). Despite these advantages, cloud computing also introduces several security challenges, particularly concerning data protection, privacy, and compliance. Protecting data during transmission and storage is critical, as breaches and leaks can have severe repercussions (Ravinder *et al.*, 2024). The multi-tenant nature of cloud environments can result in data segregation issues, potentially causing data from different clients to mix. There are also risks associated with unauthorized access and insider threats, where malicious actors or even employees could exploit system vulnerabilities (Prabhu K *et al.*, 2024). Furthermore, the complexity of managing and securing the numerous interfaces and APIs involved in cloud services necessitates robust monitoring mechanisms to prevent exploitation (Selvaraj R *et al.*, 2023).

With the increasing sophistication of data breaches and cyber-attacks, safeguarding sensitive information in the cloud is more crucial than ever (S. F. Esmaeili *et al.*, 2024). Researchers have been exploring advanced cryptographic methods and innovative security models to address these concerns. The "Enhanced Cloud Data Security (ECDS)" technique represents a new approach aimed at bolstering data protection within cloud infrastructures. ECDS uses substitution and permutation techniques to secure data both in transit and at rest. It also employs a blend of encryption methods to ensure that encrypted data remains inaccessible to unauthorized users. By integrating these advanced security measures, ECDS enables to protection of sensitive information from both internal and external

Department of Computer Science, Bishop Heber College (Autonomous), Tiruchirapalli, Tamil Nadu. Affiliated with Bharathidasan University, Tiruchirapalli, India.

***Corresponding Author:** Aruljothi Rajasekaran, Department of Computer Science, Bishop Heber College (Autonomous), Tiruchirapalli, Tamil Nadu. Affiliated with Bharathidasan University, Tiruchirapalli, India., E-Mail: aruljothi07@gmail.com

How to cite this article: Rajasekaran, A., Priyadarsini, J. R. (2024). ECDS: Enhanced cloud data security technique to protect data being stored in cloud infrastructure. The Scientific Temper, **15**(4):3113-3121.

Doi: 10.58414/SCIENTIFICTEMPER.2024.15.4.19

Source of support: Nil

Conflict of interest: None.

© *The Scientific Temper. 2024*

Received: 10/10/2024 **Accepted:** 06/11/2024 **Published** : 20/12/2024

threats, including unauthorized access by malicious insiders and external hackers (Selvaraj R *et al*., 2023a). This paper details the implementation of the ECDS technique, highlighting its effectiveness in enhancing data security in cloud environments. Through extensive testing and performance analysis, the ECDS technique has shown significant improvements in security and efficiency over traditional encryption methods.

The subsequent sections of this paper delve into the specific methodologies employed by the ECDS technique, assess its performance, and compare its effectiveness with existing security solutions. This study aims to contribute to the ongoing efforts to secure cloud computing environments and establish a reliable framework for protecting sensitive data in the cloud.

Related works

This section reviews related works pertinent to the proposed ECDS technique. An improved Blowfish algorithm for data encryption, combined with an elliptic-curve-based algorithm for key security and an MD5-based digital signature for data integrity, was proposed to enhance overall security measures by Hossein Abroshan (2021). This hybrid solution showed notable improvements in execution time, throughput, and memory consumption. It also provides robust protection against common cyber threats and ensures that the encryption process is scalable, allowing it to efficiently handle large datasets.

A novel method introduced by Huthaifa A. *et al*. (2022) integrates XOR operations with genetic algorithms to improve data encryption and decryption, focusing on protecting the encryption key from loss or theft. This approach has proven effective in a Java-based application, enhancing security measures by introducing unpredictability into the encryption process, which makes it more challenging for unauthorized parties to decrypt the data. Additionally, by splitting the encrypted data and keys into two separate files stored on different cloud platforms, the method further strengthens security by distributing risk.

A dual-layer security approach using symmetric encryption (AES and Blowfish) and steganography was proposed by Faluyi B I *et al*. (2022). In this method, data is encrypted and then hidden within an image using the least significant bit (LSB) technique, which effectively protects cloud-stored data. This technique not only secures the data but also conceals its existence, making it a highly stealthy security measure. The implementation of this approach in Python demonstrated its feasibility and adaptability to various cloud environments and platforms.

A review of hybrid cryptographic models by Sherief Murad *et al*. (2022) explored the integration of symmetric and asymmetric encryption, which provides enhanced security and performance for cloud data compared to using individual cryptographic methods. These hybrid models combine the speed and efficiency of symmetric encryption with the robust key management capabilities of asymmetric encryption, resulting in more resilient security frameworks. The study also emphasized the necessity for continuous improvement and adaptation of these models to address the evolving security threats in cloud computing.

A method for securing data in public and private clouds by separating encryption, key generation, and storage services was presented by Dr. Nisha Jebaseli *et al*. (2022). This model, known as symmetric encryption as a service (SEaaS), demonstrated strong security and efficient performance in real-time cloud environments. By decentralizing key management, this approach reduces the risk of key exposure and enhances data privacy. The proposed techniques were rigorously tested and showed compatibility with existing cloud infrastructures, with minimal impact on performance.

A hybrid cryptographic model combining elliptic curve cryptography (ECC) and Diffie-Hellman key exchange (DHKE) was introduced by R. Sabitha *et al*. (2023) to enhance cloud data protection. This approach significantly improves both data security and operational efficiency through its three key stages: authorization, key generation, and data encryption ensuring a comprehensive security protocol. The use of ECC reduces computational overhead, resulting in a faster and more efficient encryption process, which is essential for real-time cloud applications.

A hybrid cryptographic method using hyper-elliptic curve cryptography (HECC) to enhance cloud storage security was proposed by N. Krishnamoorthy *et al*. (2023). Their method demonstrated improvements in encryption efficiency, reduced computational load, and increased overall security. The key evaluation and verification steps ensure that the encryption keys are robust and reliable. This approach effectively addresses major concerns such as secure data transfer and optimal resource utilization within cloud architectures, making it a versatile solution for various cloud environments.

A comprehensive examination of various cryptographic techniques, including symmetric, asymmetric, homomorphic encryption, and multi-party computation (MPC), was conducted to maintain data confidentiality, integrity, and authenticity in cloud computing by Rambabu Nalagandla *et al*., (2023). The study emphasized the need for advanced cryptographic algorithms to counter the risks posed by quantum computing. It also highlighted key management challenges and the computational intensity of these methods, indicating the necessity for efficient and scalable cryptographic solutions. To address emerging threats, the adoption of post-quantum cryptography was suggested as a means to future-proof data security.

A hybrid security model was introduced that combines the cat swarm optimization algorithm for task scheduling with a quantum key distribution protocol (QKDP) to enhance cloud computing security by Jayachandran R. *et al*., (2024).

This model improved resource allocation, reduced energy consumption, and ensured secure key exchanges, offering a robust solution for cloud environments. The ICS-TS algorithm enhances task scheduling by minimizing make-span time and increasing throughput. By integrating quantum cryptography, the model provides secure key distribution, making it highly resilient to cyber-attacks. Experimental results demonstrated the model's effectiveness in securing cloud data while optimizing resource utilization. Table 1 shows the comparison of related work discussed in this section using some common parameters like objective, methods, encryption technique used, findings, advantages and limitations.

Methodology

The proposed ECDS technique is designed to convert plaintext into ciphertext using a combination of binary conversion, key generation, grey code transformation, matrix manipulation, and repeated permutations. This methodology details the step-by-step process to ensure data security through this encryption method.

Steps in the Proposed Methodology

Step 1: Data acquisition and conversion

Objective: Convert the plaintext (PT) data into a format suitable for encryption.

- Input: Plaintext (PT).
- Process: Convert each character in the plaintext to its ASCII binary representation, turning each character into a sequence of 0s and 1s.
- Output: Binary data ready for further processing.

Step 2: Block segmentation

- *Objective*

Prepare the binary data for encryption by dividing it into manageable blocks.

Table 1: Comparison of existing works in cloud data security

Authors year	*Objective*	*Methods used*	*Encryption techniques*	*Key findings*	*Advantages*	*Limitations*
(Hossein Abroshan, 2021)	Enhance cloud security while maintaining performance	Combination of improved Blowfish algorithm and elliptic-curve-based key encryption	Symmetric and asymmetric encryption	Reduced encryption time, improved throughput and memory consumption	Efficient encryption and decryption	Key management complexity
(Huthaifa A. *et al*., 2022)	Enhance data security in cloud computing	XOR operations with genetic algorithms	Symmetric encryption	Effective encryption and decryption	Robust security	Implementation complexity
(Faluyi B I *et al*., 2022)	Enhance cloud data security using cryptography and steganography	Combination of AES, Blowfish, and LSB steganography	Symmetric encryption and steganography	Improved data security and privacy	Dual-layer security	Implementation complexity
(Sherief Murad *et al*., 2022)	Enhance cloud security and performance	Hybrid cryptographic models review	Symmetric and asymmetric encryption	Improved security and performance	Leverages strengths of both encryption types	Continuous optimization required
(Dr. Nisha Jebaseli *et al*., 2022)	Ensure data security in hybrid cloud environments	Hybrid cryptographic techniques	Symmetric encryption	Secure data transfer and storage	Secure key management	Implementation complexity
(R. Sabitha *et al*., 2023)	Enhance data protection in cloud storage	Hybrid cryptography model using ECC and DHKE	Symmetric and asymmetric encryption	Enhanced security, secure key management	Efficient, secure key transfers	Implementation complexity
(N. Krishnamoorthy *et al*., 2023)	Improve cloud storage security	Hybrid cryptographic approach using HECC	Symmetric and asymmetric encryption	Enhanced data security and performance	Improved encryption efficiency	Resource-intensive
(Rambabu Nalagandla *et al*., 2023)	Safeguard cloud-stored data using advanced cryptographic techniques	Various encryption methods, including homomorphic encryption and MPC	Symmetric and asymmetric encryption	Enhanced data confidentiality, integrity, and authenticity	Robust protection	High computational demands
(Jayachandran R. *et al*., 2024)	Enhance data security in cloud computing	Hybrid security model combining improved Cat Swarm Optimization algorithm and Quantum Key Distribution Protocol (QKDP)	Symmetric and asymmetric encryption	Improved resource allocation, reduced energy consumption, secure key transfer	Optimized resource usage, robust security	Computational complexity

- Process: Divide the binary data into 256-bit blocks. Each block will be processed independently in the following steps.
- Output: A series of 256-bit blocks.

Step 3: Key generation

- *Objective*

Generate cryptographic keys necessary for the encryption process.

- 256-bit Key (K1):
    - Process: Generate a 256-bit binary key (K1) to be used for initial XOR operations.
    - Output: Key K1.
- 56-bit Key (K2):
    - Process: Generate a 56-bit binary key (K2), which will be divided into subkeys for matrix permutation and repetition control.
    - Output: Key K2.
- The total key size is 312-bit

Step 4: Grey code transformation

- *Objective*

Enhance data complexity by transforming binary segments into grey code.

- Process: Split each 256-bit block into 8-bit segments. Convert each segment into its grey code equivalent, where each successive value differs by only one bit.
- Output: A 256-bit block represented in grey code.

Step 5: One's complement calculation

- *Objective*

Further obscure the data by inverting the bits.

- Process: Merge the grey code segments back into a single 256-bit block. Calculate the one's complement by inverting each bit (changing 0s to 1s and vice versa).
- Output: A complemented 256-bit block.

Step 6: XOR operation

- *Objective*

Use cryptographic key K1 to modify the complemented block.

- Process: Perform an XOR operation between the complemented block and the 256-bit key (K1). The XOR operation produces a new 256-bit block where each bit is the result of the XOR between corresponding bits of the operands.
- Output: An XORed 256-bit block.

Step 7: Decimal conversion

- *Objective*

Convert binary data into a format suitable for matrix operations.

- Process: Divide the XORed 256-bit block into 8-bit segments and convert each segment into its decimal equivalent.
- Output: A list of 32 decimal values.

Step 8: Matrix formation

- *Objective*

Structure the data into a matrix for permutation operations.

- Process: Create a 6x6 square matrix (SM). Populate the matrix with the decimal values, filling from left to right, row by row. Fill any remaining cells with a filler character, typically 'X'.
- Output: A populated square matrix (SM).

Step 9: Matrix permutation

- *Objective*

Increase data complexity through matrix permutations.

Subkey1 for columnar transposition

- Process: Use the first 48 bits of key K2 (subkey1) to determine the order of columns for permutation.
- Output: Permuted matrix based on columnar transposition.

Subkey2 for Repetition control

- Process: Use the remaining 8 bits of key K2 (subkey2) to determine the number of times the permutation process should be repeated.
- Output: Final permuted matrix after the specified number of repetitions.

Step 10: Read the values from the matrix row by row character conversion

- *Objective*

Read and convert the processed data back into a readable format.

- Process: Convert each decimal value in the final permuted matrix back to its ASCII character equivalent.
- Output: A sequence of ASCII characters.

Step 11: Form Final Ciphertext

- *Objective*

Combine the final characters to produce the encrypted output.

- Process: Combine the ASCII characters derived from the matrix to form the final ciphertext.
- Output: The final encrypted ciphertext.

This process outlines a complex encryption scheme involving binary operations, matrix manipulations, and multiple keys to ensure data security.

ECDS Technique Procedures

The encryption process involves several steps to convert plaintext into ciphertext using binary conversion, key generation, and matrix manipulation. It starts with converting the plaintext (PT) into its ASCII binary representation. The binary data is divided into 256-bit blocks, and a 256-bit binary key (K1) is generated for XOR operations to enhance security. Each 256-bit block is divided into 8-bit segments, converted into grey codes (where successive values differ by one bit), and merged back into a 256-bit block. This block undergoes one's complement (inverting all bits) and is then XORed with K1, producing a new 256-bit block. This block is converted into 32 decimal values, which fill a 6x6 matrix (SM) row by row. Any remaining cells are filled with 'X'. A 56-bit key (K2) is generated and split into subkey1 (48 bits) and subkey2 (8 bits). Subkey1 determines the column order for matrix permutation, while subkey2 specifies the permutation count. The matrix undergoes columnar transposition based on subkey1, repeated as per subkey2, adding encryption complexity. Finally, each decimal in the matrix is converted back to its ASCII equivalent, forming the final ciphertext.

This multi-step encryption ensures the data is highly secure, obfuscating it against unauthorized access and interpretation through systematic binary conversion, key-based XOR operations, and matrix manipulations.

Pseudo-code of ECDS

```
INPUT: Plaintext (PT) data
    binary data = convertToBinary(PT)
    for each 256-bit block in binaryData:
        K1 = generate256BitKey()
        greyCodeBlocks = []
        for each 8-bit block in 256-bit block:
            greyCodeBlocks.append(convertToGreyCode(8-bit block))
        mergedBlock = mergeBlocks(greyCodeBlocks)
        complementedBlock=onesComplement(mergedBlock)
        XORedBlock = xorBlocks(complementedBlock, K1)
        decimalValues = convertToDecimal(XORedBlock)
        SM = createMatrix(6, 6)
        fillMatrix(SM, decimalValues, 'X')
        K2 = generate56BitKey()
        subkey1 = first48Bits(K2)
        subkey2 = last8Bits(K2)
        permutedMatrix = columnarTranspose(SM, subkey1)
        for i = 1 to subkey2:
            permutedMatrix=columnarTranspose(permutedMatrix, subkey1)
        cipherText = ""
        for each decimal in permutedMatrix:
            cipherText += convertToASCII(decimal)
        finalCipherText += cipherText
    OUTPUT: finalCipherText
```

Experiment with Sample Data

Let's take a sample 256-bit plaintext and carry out the encryption steps one by one. The plaintext considered for the sample experiment is "Welcome, this is a sample text message", which is 256 bits long when converted to binary. The below details walk through the steps, providing explanations along the way.

Step 1: User's data is taken for encryption

Plaintext (PT): "Welcome, this is a sample text message"

Step 2: Convert PT data to ASCII binaries

Now, convert each character in the plaintext to its ASCII binary representation.

W: 01010111 e: 01100101 l: 01101100 c : 01100011 o: 01101111
m: 01101101 e: 01100101 ,: 00101100 t : 01110100 h: 01101000
i: 01101001 s: 01110011 i: 01101001 s : 01110011 a: 01100001
s: 01110011 a: 01100001 m: 01101101 p : 01110000 l: 01101100
e: 01100101 t: 01110100 e: 01100101 x : 01111000 t: 01110100
m: 01101101 e: 01100101 s: 01110011 s : 01110011 a: 01100001 g: 01100111 e: 01100101

Step 3: Consider the 256-bit block of data for encryption

Now, the first 256 bits from the binary representation (32 characters * 8 bits each).

0101011101100101011011000110001101101111011011010110 0101 0010110001110100011010000110100101110011011010010111 0011 0110000101110011011000010110110101110000 011011000110010101110100011001010111100001110100 0110110101100101 0111001101110011011000010110011101100101

Step 4: Generate a 256-bit binary key K1

Let's assume K1 is:

11001010 10100101 10010101 11100010 11010010 01110111 10011001 10010110 10101010 01100001 11100101 10010101 10010101 11000011 10101010 11110110 11011001 11010100 10100101 10111001 10011010 11101001 10110101 10101001 11010101 11001001 10101001 10101101 11010101 10011001 11010100 10101001

Step 5: Split PT into 8-bit blocks and find the grey code for each block

The plaintext 356-bit block is split into 8-bit segments and convert each segment into its grey code equivalent.

For example:

- Binary: 01010111
- Grey Code: 01111100

Perform this conversion for all 8-bit blocks. The grey codes are:

01111100 01011011 01010010 01010010 01010000 01011110 01011011 00111010 01011010 01011100 01011101 01001110 01011101 01001110 01010011 01001110 01010011 01011110 01001100 01011110 01011011 01011011 01011011 01001100 01011010 01011100 01011011 01001110 01001110 01010011 01011100 01011010

Step 6: Now find the 1's complement

The grey code blocks of 256-bit are involved in calculating the one's complement.

- *Grey coded block*

01111100 01011011 01010010 01010010 01010000 01011110 01011011 00111010 01011010 01011100 01011101 01001110 01011101 01001110 01010011 01001110 01010011 01011110 01001100 01011110 01011011 01011011 01011011 01001100

01011010 01011100 01011011 01001110 01001110 01010011 01011100 01011010

- *1's complement block*

10000011 10100100 10101101 10101101 10101111 10100001 10100100 11000101 10100101 10100011 10100010 10110001 10100010 10110001 10101100 10110001 10101100 10100001 10110011 10100001 10100100 10100100 10100100 10110011

10100101 10100011 10100100 10110001 10110001 10101100 10100011 10100101

Step 7: Find XOR with the 256-bit key K1

Perform XOR between the complemented block and the key K1.

- *Complemented block*

10000011 10100100 10101101 10101101 10101111 10100001 10100100 11000101 10100101 10100011 10100010 10110001 10100010 10110001 10101100 10110001 10101100 10100001 10110011 10100001 10100100 10100100 10100100 10110011

10100101 10100011 10100100 10110001 10110001 10101100 10100011 10100101

- *Key K1*

11001010 10100101 10010101 11100010 11010010 01110111 10011001 10010110 10101010 01100001 11100101 10010101 10010101 11000011 10101010 11110110 11011001 11010100 10100101 10111001 10011010 11101001 10110101 10101001

11010101 11001001 10101001 10101101 11010101 10011001 11010100 10101001

- *XOR result*

01001001 00000001 00111000 01001111 01111101 11010110 00111101 01010011 00001111 11000010 01000111 00100100 00110111 01110010 00000110 01000111 01110101 01110101 10010110 00011000 00111110 01001101 00010001 00011010

01110000 01101010 00011101 00011100 01100100 00110101 01100111 00001100

Step 8: Convert the XORed 256-bit block to decimal values

Convert each 8-bit block to its decimal equivalent.

- *Decimal values*

[73, 1, 56, 79, 125, 214, 61, 83, 15, 194, 71, 36, 55, 114, 6, 71, 117, 117, 150, 24, 62, 77, 17, 26, 112, 106, 29, 28, 100, 53, 103, 12]

Step 9: Form a 6x6 square matrix (SM)

Create a 6×6 matrix and fill it with the decimal values, left to right, row by row. Fill remaining cells with 'X'.

73	1	56	79	125	214
61	83	15	194	71	36
55	114	6	71	117	117
150	24	62	77	17	26
112	106	29	28	100	53
103	12	X	X	X	X

Step 10: Generate a 56-bit key K2

The K2 is:

K2: 00001001 00100101 00110110 00111001 00100100 00010001 00000010

Subkey1: 00001001 00100101 00110110 00111001 00100100 00010001

Decimal equivalents are 9, 37, 54, 57, 36, 17

Subkey2: 00000010 à 2

Step 11: Permute matrix based on subkey1 using columnar transposition

The subkey1 translates to column order [9, 37, 54, 57, 36, 17].

- *Before permutation*

73	1	56	79	125	214
61	83	15	194	71	36
55	114	6	71	117	117
150	24	62	77	17	26
112	106	29	28	100	53
103	12	X	X	X	X

- *After 1st permutation*

73	61	55	150	112	103
214	36	117	26	53	x
125	71	117	17	100	x
1	83	114	24	106	12
56	15	6	62	29	x
76	194	71	77	28	x

Step 12: Repeat permutation based on subkey2

The generated subkey2 is 2, so repeat the permutation 2 times. One transposition is already completed in the previous step. The second transposition is as follows.

- *After 2nd permutation*

73	214	125	1	56	76
103	x	x	12	x	x
112	53	100	106	29	28
61	36	71	83	15	194
55	117	117	114	6	71
150	26	17	24	62	77

Step 13: Read the values from the matrix row by row and convert each decimal in the matrix to an ASCII character code values from the matrix:

73 214 125 1 56 76 103 x x 12 x x 112 53 100 106 29 28 61 36 71 83 15 194 55 117 117 114 6 71 150 26 17 24 62 77

Convert to ASCII character code:

73:I 214:╥ 125:} 1:ɸ 56:8 76:L 103:g x x 12: ♀ x x 112:p 53:5 100:d 106:j 29: ↔ 28: ∟ 61:= 36:$ 71:G 83:S 15: ☼ 194: ┬ 55:7 117:u 117:u 114:r 6: ♠ 71:G 150: û 26: → 17: ◄ 24: ↑ 62:> 77:M

Step 14: Output the final ciphertext

Combine all ASCII characters to form the final ciphertext.

Ciphertext: "I ╥ } ɸ 8 L g x x ♀ xxp5dj↔∟=$GS☼┬7uur♠Gû→◄↑>M"

This sample demonstration shows the step-by-step encryption process with the given plaintext, resulting in a secure ciphertext through a series of systematic steps, including binary conversion, grey code transformation, XOR operations, and matrix permutations.

Results and Discussion

The encryption procedure described above exhibits conceptual similarities with established encryption algorithms like data encryption standard (DES) and Blowfish but also presents unique differences in its approach and complexity.

DES is a symmetric key algorithm designed for encrypting electronic data. Developed in the 1970s, it encrypts data in 64-bit blocks using a 56-bit key. DES employs 16 rounds of a Feistel network, where each round involves a series of substitutions and permutations (using S-boxes and P-boxes) to transform plaintext into ciphertext. Although DES is known for its simplicity and efficiency, its relatively short key length renders it susceptible to brute-force attacks, making it unsuitable for many modern applications.

Blowfish, created by Bruce Schneier in 1993, is another symmetric-key block cipher but offers greater flexibility and security than DES. Like DES, Blowfish encrypts data in 64-bit blocks, but it supports key lengths ranging from 32 bits to 448 bits, providing enhanced security. Blowfish utilizes a Feistel network with 16 rounds, similar to DES, but with a more complex key schedule and larger S-boxes. Blowfish is recognized for its speed and effectiveness in software implementations, as well as its robustness against various cryptanalytic attacks.

The proposed ECDS encryption process described here adopts a unique approach. It starts by converting plaintext into ASCII binaries and processes data in 256-bit blocks, significantly larger than the 64-bit blocks used by DES and Blowfish. The process involves generating a 256-bit key (K1) for initial encryption, followed by grey code conversion and the calculation of one's complement, introducing layers of complexity not present in standard DES or blowfish. Additionally, the ECDS method incorporates matrix manipulation. A 6x6 matrix is populated with decimal values derived from XOR operations between the one's complement of the grey-coded plaintext and the key K1.

Further security is introduced by permuting this matrix multiple times based on a second 56-bit key (K2), which is divided into subkeys for columnar transposition and repeated permutations. This matrix-based approach is distinct from the round-based Feistel networks in DES and blowfish. The performance of the proposed method is compared with existing methods based on the time taken for encryption and decryption. Table 2 shows the encryption time comparison between the proposed and existing methods.

Table 2: Encryption time comparison

Data size	DES	Blowfish	ECDS
	Milliseconds		
100	130	90	75
200	260	180	150
300	390	270	225
400	520	360	300
500	650	450	375

Table 3 shows the decryption time taken by the three encryption techniques in milliseconds.

Table 3: Decryption time comparison

Data size	DES	Blowfish	ECDS
	Milliseconds		
100	125	85	70
200	255	175	145
300	385	265	220
400	515	355	295
500	645	445	370

The DES encryption took the longest among the three algorithms, which is expected due to its older design and the relatively high number of rounds in its Feistel network. The Blowfish encryption was faster than DES, which is due to its more modern design and efficient implementation. The ECDS encryption process is faster than DES and blowfish. Hence, the proposed ECDS performs well based on the time taken for encryption and decryption.

The DES and blowfish follow well-established cryptographic principles; the ECDS encryption method introduces novel steps like grey code conversion and matrix permutations. These steps potentially enhance security through added complexity. The ECDS encryption process blends elements of traditional block ciphers with innovative techniques to aim for enhanced security. The efficiency of the ECDS is evaluated using a hacking tool called ABC Hackman tool. This tool is used to evaluate the

Table 4: Comparison of security levels

S. No	*Techniques*	*Security level (%)*
1	DES	81
2	Blowfish	85
3	ECDS	89

security level of the encryption algorithms. The proposed ECDS is incorporated into the hackman tool evaluation to measure the security level. The following mathematical formula determines the level of security for the encryption techniques:

Let C represent the size of the ciphertext recorded in the Cloud storage. H refers to the plaintext that was hacked and obtained from the text that was encrypted using the ABC Hackman system.

To calculate the amount of security provided by the techniques of encryption,

S ß C – H …(1)

Here, S is the count of texts that are dissimilar to the simple text. Next, the security level (SP) is determined as percentage by using the following formula:,

SPßS/C*100 …(2)

Table 4 presents a comparison of the security levels between the current and stated encryption method.

The security levels are assessed using the ABC Hackman Tool, which evaluates the extent of data retrieval from the encrypted text. The results show that DES achieved a security level of 81%, reflecting its older design and vulnerability to brute-force attacks due to its 56-bit key length. Blowfish, with its variable key lengths ranging from 32 to 448 bits, demonstrated an improved security level of 85%, benefiting from a more modern and robust cryptographic design. The ECDS technique surpassed both, achieving a security level of 89%. This higher security level is due to ECDS's innovative approach, which includes grey code transformation, matrix manipulations, and multiple permutations, adding complexity and enhancing data protection. The comparative analysis highlights ECDS's effectiveness in providing superior security for cloud data encryption, making it a promising solution for protecting sensitive information in cloud environments.

Conclusion

The enhanced cloud data security (ECDS) technique presents significant advancements in cloud computing security by incorporating novel approaches such as grey code transformation, matrix manipulations, and hybrid cryptographic algorithms. These innovations address critical challenges in protecting sensitive data within cloud environments. Comparative analysis with established encryption methods like DES and blowfish shows that ECDS not only enhances security levels but also improves performance efficiency. ECDS achieves a higher security rating of 89%, compared to 81% for DES and 85% for blowfish, demonstrating its robustness against data breaches and unauthorized access. The promising results of ECDS, confirmed through extensive testing and performance analysis, highlight its potential as a dependable solution for protecting sensitive information in the cloud. Future research should focus on refining the technique further and exploring its applicability across different cloud environments to ensure its adaptability and resilience against emerging security threats.

References

Abroshan, H. (2021). A hybrid encryption solution to improve cloud computing security using symmetric and asymmetric cryptography algorithms. International Journal of Advanced Computer Science and Applications, 12(6), 31-37. https://dx.doi.org/10.14569/IJACSA.2021.0120604

Esmaeili S. F., M. Yadollahzadeh-Tabari and A. A. Pouyan. (2024). Improving security in cloud computing using colonial competition algorithm. IEEE International Symposium on Artificial Intelligence and Signal Processing, Iran: 1-8. https://doi.org/10.1109/AISP61396.2024.10475223

Faluyi Bamidele Ibitayo, Oguntuase RianatAbimbola and Makinde Bukola Oyeladun. (2022). Securing Cloud Computing Contents with Cryptography and Steganography. International Journal of Science and Engineering Applications, 11(6): 76 – 88. https://ijsea.com/archive/volume11/issue6/IJSEA11061002.pdf

Huthaifa A. Al Issa, Mustafa Hamzeh Al-Jarah, Ammar Almomani. (2022). Encryption and Decryption Cloud Computing Data Based on XOR and Genetic Algorithm. International Journal of Cloud Applications and Computing, 12(1): 1-10. https://doi.org/10.4018/IJCAC.297101

Jayachandran, R., & Malathi, D. (2024). Enhanced cloud security model with hybrid encryption approach for advanced data security in cloud computing. International Journal of Intelligent Systems and Applications in Engineering, 12(4s), 432-439. https://ijisae.org/index.php/IJISAE/article/view/3804

Jebaseli, N., & Banu, A. F. (2022). A new approach for maintaining data security using cryptography in a hybrid cloud environment. Journal of Algebraic Statistics, 13(2): 214-224. https://www.publishoa.com/index.php/journal/article/view/159/147

Krishnamoorthy, N., & Umarani, S. (2023). Implementation of cloud computing data security based on hybrid elliptical curve cryptography. International Journal of Intelligent Systems and Applications in Engineering, 11(10s), 483-488. https://ijisae.org/index.php/IJISAE/article/view/3303

Murad, S., & Rahouma, K. H. (2022). Hybrid cryptography for cloud security: Methodologies and designs. Springer Digital Transformation Technology: 129-140. https://doi.org/10.1007/978-981-16-2275-5_7

Nalagandla, R., & Pattanaik, O. (2023). Cloud data security: Advanced cryptography algorithms. EasyChair Preprints, (10784). https://easychair.org/publications/preprint/mD3B

Prabhu K and P. Sudhakar. (2024). A Comprehensive Survey: Exploring Current Trends and Challenges in Intrusion Detection and Prevention Systems in the Cloud Computing Paradigm. *IEEE International Conference on Intelligent Data*

Communication Technologies and Internet of Things, India: 351-358. https://doi.org/10.1109/IDCIoT59759.2024.10467700

Pronika and S. S. Tyagi. (2021). Secure Data Storage in Cloud using Encryption Algorithm. IEEE International Conference on Intelligent Communication Technologies and Virtual Mobile Networks, India: 136-141. https://doi.org/10.1109/ICICV50876.2021.9388388

Ravinder, M. S. Khan and J. Singh. (2024). Security Challenges in Mobile Cloud Computing. IEEE Karachi Section Humanitarian Technology Conference (KHI-HTC), Pakistan: 1-9. https://doi.org/10.1109/KHI-HTC60760.2024.10482107

Sabitha, R., Sydulu, S. J., Karthik, S., & Kavitha, M. S. (2023). Secure data storage on cloud using hybrid cryptography methods. EasyChair Preprints, (10126). https://easychair.org/publications/preprint/DRFZ

Selvaraj, R., & Sundaram, M. S. (2023). ECM: Enhanced confidentiality method to ensure the secure migration of data in VM to cloud environment. The Scientific Temper, 14(03), 902–908. https://doi.org/10.58414/SCIENTIFICTEMPER.2023.14.3.53

Selvaraj, R., & Sundari, M. S. (2023a). EAM: Enhanced authentication method to ensure the authenticity and integrity of the data in VM migration to the cloud environment. The Scientific Temper, 14(01), 227–232. https://doi.org/10.58414/SCIENTIFICTEMPER.2023.14.1.29

Sudha I, C. Donald, S. Navya, G. Nithya, M. Balamurugan and S. Saravanan. (2024). A Secure Data Encryption Mechanism in Cloud Using Elliptic Curve Cryptography. IEEE International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics, India: 1-5. https://doi.org/10.1109/IITCEE59897.2024.10467407