The Scientific Temper (2025) Vol. 16 (2): 3783-3791

E-ISSN: 2231-6396, ISSN: 0976-8653

Doi: 10.58414/SCIENTIFICTEMPER.2025.16.2.10

https://scientifictemper.com/

RESEARCH ARTICLE

ECE cipher: Enhanced convergent encryption for securing and deduplicating public cloud data

Priya Nandhagopal*, Jayasimman Lawrence

Abstract

Cloud computing offers scalable and cost-effective storage solutions, but concerns over data security, unauthorized access, and storage inefficiencies remain significant. Data deduplication is crucial in reducing storage costs by eliminating redundant copies, yet traditional encryption methods hinder deduplication by generating unique ciphertexts for identical plaintexts, leading to increased storage requirements. To address these challenges, this paper presents ECEcipher, an advanced symmetric block cipher encryption technique that integrates convergent encryption for secure deduplication while ensuring strong data security. It uses a 196-bit encryption key, generated from the plaintext data, and applies substitution and permutation operations for enhanced security. Unlike conventional encryption, ECEcipher dynamically determines encryption rounds, making it harder to break. Performance evaluation shows ECEcipher outperforms DES and Blowfish in encryption speed and efficiency, making it ideal for real-time cloud applications. Additionally, ECEcipher supports deduplication without compromising security, ensuring optimized storage utilization. Security analysis using the ABC Universal Hackman tool confirms higher resistance to brute-force and dictionary attacks.

Keywords: Cloud security, Deduplication, Convergent encryption, Data confidentiality, Block cipher encryption.

Introduction

Cloud computing offers a robust and scalable environment for data storage, allowing users to store large volumes of information efficiently. Organizations, particularly those handling massive datasets, benefit from cloud storage by reducing the need for costly on-premise infrastructure (Arockiam L. *et al*., 2013). Enterprises, especially small-scale businesses, can optimize storage costs by renting cloud storage resources instead of investing in expensive servers. The scalability of cloud storage enables users to adjust storage capacity dynamically, ensuring flexibility and cost efficiency. One of the major advantages of cloud storage is its reliability, as data stored in the cloud can be retrieved on demand. Additionally, cloud storage protects data from physical damage caused by natural disasters, ensuring business continuity. Storage can be either dedicated or shared, with shared storage options available at a minimal cost in public cloud environments.

Despite these benefits, cloud storage faces a significant challenge—data duplication. Data deduplication is a common practice in cloud storage that aims to eliminate redundant copies of files stored by multiple users (Sabeerath K., *et al*., 2024a). Since many users store identical files, cloud service providers (CSPs) implement deduplication techniques to minimize storage space and improve efficiency. Deduplication helps reduce storage costs and bandwidth consumption by ensuring that only a single instance of a file is stored, while duplicate copies are linked to the original data. However, this process poses a serious security risk, as traditional encryption methods prevent deduplication by generating unique ciphertexts for identical plaintexts. If different users encrypt the same file with different keys, the cloud system treats them as separate files, increasing storage redundancy.

To address this issue, convergent encryption (CE) has emerged as an effective solution for enabling secure deduplication in cloud storage (Selvaraj R. *et al*., 2023a).

Department of Computer Science, Bishop Heber College, Tiruchirappalli, Affiliated to Bharathidasan University, Tamil Nadu, India.

***Corresponding Author:** Priya Nandhagopal, Department of Computer Science, Bishop Heber College, Tiruchirappalli, Affiliated to Bharathidasan University, Tamil Nadu, India., E-Mail: priya.phdbhc@gmail.com

How to cite this article: Nandhagopal, P., Lawrence, J. (2025). ECE cipher: Enhanced convergent encryption for securing and deduplicating public cloud data. The Scientific Temper, **16**(2):3783-3791.

Doi: 10.58414/SCIENTIFICTEMPER.2025.16.2.10

Source of support: Nil

Conflict of interest: None.

Received: 22/12/2024 **Accepted:** 09/01/2025 **Published** : 20/03/2025

Convergent encryption ensures that identical plaintexts produce the same ciphertext, allowing the cloud system to detect duplicate data and store only one encrypted version of the file, thereby optimizing storage efficiency. In this approach, a cryptographic key is derived from the data itself, meaning that only users who possess the original file can generate the same encryption key to decrypt it. This mechanism allows cloud storage providers to perform deduplication while ensuring data confidentiality (Sabeerath K., *et al*., 2024b). However, convergent encryption also presents security vulnerabilities, particularly against brute-force and dictionary attacks, where an attacker can precompute ciphertexts for commonly stored files and attempt to match encrypted data with known plaintexts. This creates a trade-off between storage efficiency and data security (Selvaraj R. *et al*., 2023b). To strengthen security, advanced encryption methods must integrate robust cryptographic techniques while maintaining support for deduplication.

To enhance cloud data security, this paper introduces ECEcipher, an advanced encryption technique designed to fortify data protection in cloud environments. ECEcipher is a symmetric block cipher convergent encryption method specifically developed to secure data stored in public cloud infrastructures. The encryption process is offered as a cloud-based service, with the required encryption keys retrieved from a dedicated cloud key management system. The proposed encryption approach employs substitution and permutation techniques to ensure strong data protection. It processes 64-bit plaintext and encrypts it using a 196-bit encryption key, significantly enhancing data confidentiality. By integrating advanced encryption mechanisms, ECEcipher aims to address the limitations of traditional encryption methods, ensuring both high security and storage efficiency.

Related Work

Cloud storage faces challenges in data security and privacy, particularly in ensuring confidentiality and trust in service providers. To address this, Arockiam *et al*. (2013) propose a hybrid symmetric encryption algorithm to protect cloud-stored data from unauthorized access. Their method enhances security and storage efficiency by leveraging symmetric encryption techniques. The proposed method encrypts data before transmission and storage, reducing risks from external threats and insider attacks. Integrating access control measures further minimizes unauthorized data exposure. The study analyzes the encryption and decryption processes, demonstrating improved cloud security. Their findings confirm that this technique offers a robust, efficient solution for protecting cloud-based information while ensuring optimal performance.

Ensuring cloud data security is a major challenge, requiring multi-layered protection to prevent unauthorized access. To enhance confidentiality, Arockiam *et al*. (2014) propose a dual-layered approach combining encryption and obfuscation. While encryption secures data, obfuscation makes it unreadable to attackers, ensuring stronger protection. Even if encrypted data is accessed, deciphering it remains extremely difficult without authorization. The authors argue that encryption alone has vulnerabilities, making obfuscation essential for better resilience against cryptographic attacks. Their strategy effectively addresses growing concerns over cloud data privacy, offering a comprehensive security solution. The authors confirm that even if intercepted, encrypted data remains inaccessible, reinforcing confidentiality and integrity in cloud storage environments.

Ensuring data confidentiality in cloud storage is challenging, especially with third-party providers managing data. To address this, da Rocha *et al*. (2020) propose a client-side encryption solution using Intel's SGX within a trusted execution environment (TEE). This method ensures that data is encrypted before uploading, preventing unauthorized access, even if the cloud provider is compromised. By integrating their approach with Cryptomator, a widely used client-side encryption tool, they demonstrate its feasibility and effectiveness in enhancing cloud security. The SGX-based encryption strengthens confidentiality and access control, protecting sensitive data from breaches. The paper emphasizes the importance of trusted execution environments in modern cloud security, offering a scalable and practical solution that maintains both security and usability.

Securing smart system data in cloud environments is a growing challenge, requiring robust encryption mechanisms. To address this, Qureshi *et al*. (2022) conducted a comprehensive survey of encryption techniques for cloud-based smart systems. Their study provides a comparative analysis of various methods, highlighting strengths, limitations, and applications. The authors use graphical workflows to simplify encryption processes, helping researchers and practitioners evaluate effective security strategies. By consolidating encryption techniques, the survey serves as a valuable reference for enhancing cloud security. The paper emphasizes the need for balancing security, efficiency, and computational cost, ensuring optimal data protection in cloud infrastructures.

Securing cloud-stored data is increasingly important due to rising cyber threats and unauthorized access risks. To address this, Aruljothi Rajasekaran *et al*. (2024) introduce the enhanced cloud data security (ECDS) technique, designed to strengthen data protection in cloud infrastructures. ECDS focuses on mitigating cloud storage vulnerabilities by implementing advanced security measures that ensure confidentiality and integrity. While specific technical details are not fully outlined, the authors emphasize robust

encryption protocols and access control mechanisms to safeguard sensitive cloud data. The authors stress the need for enhanced security frameworks to prevent unauthorized access, breaches, and insider threats. By integrating strong encryption techniques, ECDS enhances cloud data confidentiality, making it more resistant to cyberattacks. The study contributes to ongoing cloud security research, providing a framework for secure data storage, privacy protection, and reliable access control in modern cloud environments.

Ensuring cloud data security is crucial due to the risks of unauthorized access and data breaches. Aslam *et al*. (2024) propose a hybrid encryption model that combines symmetric encryption for speed and asymmetric encryption for secure key management, enhancing data confidentiality and efficiency. Their study evaluates encryption and decryption times, demonstrating improved security with minimal performance impact. Additionally, the research highlights user education as vital in enhancing cloud security, recommending training programs on encryption best practices. Further analysis confirms the model's resistance to brute-force and man-in-the-middle attacks. Aslam *et al*. (2024) conclude that their approach provides a secure and efficient encryption framework, ensuring strong cloud data protection while maintaining optimal performance for modern cloud applications. Table 1 shows a comparison of the related works discussed in this paper.

Methodology

The proposed ECEcipher is a symmetric block convergent encryption technique designed to enhance data security in cloud storage. It employs a secret key for both encryption and decryption, ensuring that only authorized users with the correct key can access the original data. Unlike traditional symmetric encryption methods that use a fixed number of processing rounds for all plaintext inputs, ECEcipher introduces a dynamic round selection mechanism to strengthen security.

Key Features of ECEcipher

Dynamic Encryption Rounds

Conventional encryption techniques use a predetermined number of encryption and decryption rounds, making them susceptible to cryptanalysis. Attackers can analyze patterns in ciphertext to infer plaintext. In contrast, ECEcipher dynamically determines the number of encryption and decryption rounds based on the key, ensuring greater unpredictability and resistance to attacks.

Enhanced Key Structure

The encryption technique employs a 200-bit convergent key, which is later transformed into a 196-bit encryption key through a systematic process:

- The original 200-bit key is generated from the plaintext data.
- The last 8 bits of the key are split into two 4-bit segments.
- The first 4-bit segment is discarded, leaving a 196-bit final key for encryption.
- The remaining 4-bit segment (Subkey1) determines the number of encryption rounds for a given plaintext.
- The same process is followed during decryption to maintain consistency.

Key Generation Mechanism

The encryption key consists of 25 character length digest value from the plaintext, each 8-bit in length, making it highly secure and difficult to predict. The dynamic nature of key-based round selection further complicates brute-force attacks.

Proposed ECEcipher

The proposed ECEcipher convergent encryption method processes 64-bit plaintext as input and produces 64-bit

Table 1: Comparison of related works

Reference	*Encryption type*	*Security focus*	*Methodology*	*Performance impact*	*Application*
Arockiam *et al*. (2013)	Hybrid symmetric encryption	Data confidentiality & efficiency	Symmetric encryption for secure storage	Efficient & secure	Cloud data storage
Arockiam *et al*. (2014)	Encryption & obfuscation	Multi-layered security	Combining encryption & obfuscation	Improved security, slight overhead	Cloud privacy protection
da Rocha *et al*. (2020)	Client-side encryption (sgx)	Prevent unauthorized access	SGX-based Trusted Execution Environment	Minimal performance loss	Securing cloud-stored data
Qureshi *et al*. (2022)	Survey on various techniques	Comparative study of encryption	Comparative analysis & graphical workflows	Dependent on technique used	General cloud-based encryption
Aruljothi Rajasekaran *et al*. (2024)	ECDS - advanced encryption	Cloud data protection	Enhanced security frameworks	Improved security & control	Cloud infrastructure security
Aslam *et al*. (2024)	Hybrid Encryption (Symmetric & Asymmetric)	Cloud security with user awareness	Combining symmetric & asymmetric encryption	Strong security without major performance loss	Cloud data confidentiality

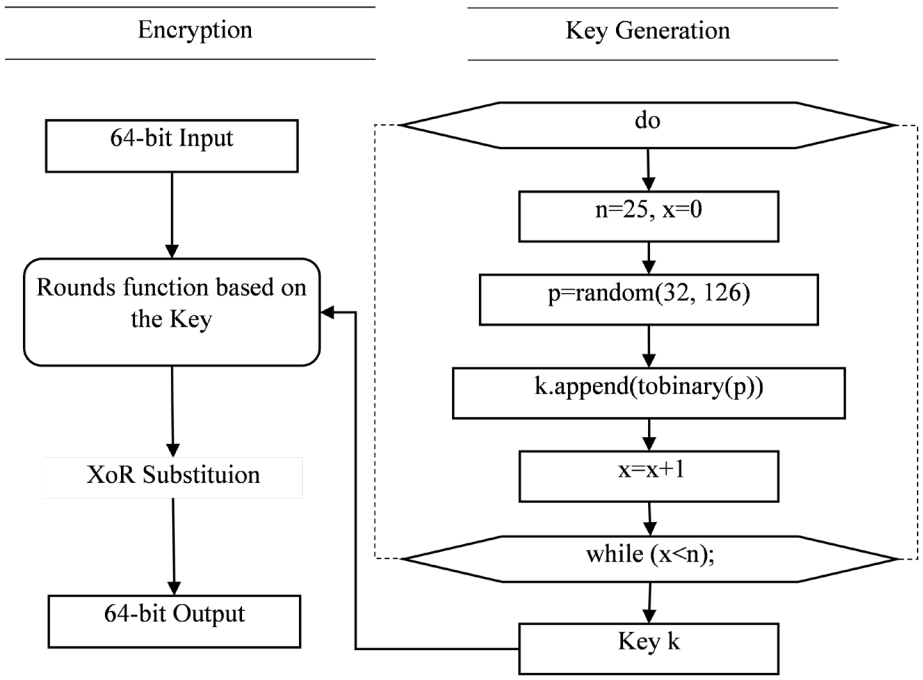

Figure Block Diagram of Encryption and Key Generation in PUCSCipher

Figure 1: ECEcipher block diagram

ciphertext as output. Figure 1 shows the block diagram of ECEcipher. The encryption process integrates two fundamental cryptographic techniques: substitution and transposition. Unlike traditional encryption methods with a fixed number of rounds, ECEcipher dynamically determines the number of encryption rounds based on the key, enhancing security against cryptanalysis.

Dynamic Round Execution

- The number of encryption rounds is not predetermined but is dynamically derived from Subkey1 of the encryption key.
- This dynamic variation ensures that different plaintext inputs undergo varying encryption rounds, making it difficult for attackers to infer patterns.

Key Structure and Subkey Functions

The 196-bit encryption key is divided into four subkeys, each playing a distinct role in the encryption process:

- *Subkey1*

Determines the number of encryption rounds for a given plaintext.

- *Subkey2*

Controls the bit permutation (transposition), ensuring a rearrangement of bits before further processing.

- *Subkey3*

A 64-bit key, which is further divided into two 32-bit segments for additional cryptographic operations.

- *Subkey4*

After all rounds, a 64-bit output is derived. It is XoRed with the fourth subkey.

Step-by-Step Encryption Process

- *Bit Permutation (Transposition)*
  - The 64-bit plaintext undergoes permutation based on Subkey2, effectively shuffling the bit positions to introduce diffusion.
- *Splitting of Data*
  - After permutation, the 64-bit data is divided into two 32-bit halves.
  - Similarly, Subkey3 (64-bit) is also divided into two 32-bit subkeys.
- *XOR Operation*
  - Each 32-bit half of plaintext is XoRed with the corresponding 32-bit subkey from Subkey3.
- *Bit Swapping*
  - The resulting two 32-bit halves are swapped to enhance confusion, making it difficult to trace

patterns in the encryption process.

- *Merging and Round Completion*
  - The swapped halves are merged back into a 64-bit block.
  - This completes one round of encryption.
  - The process repeats for the dynamically determined number of rounds.

By implementing substitution, permutation, XOR operations, and dynamic round selection, ECEcipher significantly strengthens encryption security, making it resistant to cryptanalysis attacks. The decryption process follows the same logic in reverse, ensuring accurate retrieval of the original plaintext.

The encryption process continues for the number of rounds dynamically determined by Subkey1. Each round follows the structured sequence of permutation, XOR operations, swapping, and merging to enhance data security.

Once all encryption rounds are completed:

- The final 64-bit processed data undergoes an XOR operation with Subkey4.
- This final XOR operation further obfuscates the data, strengthening resistance against cryptanalysis.
- The resulting 64-bit output is the ciphertext, which is securely stored or transmitted.

By incorporating dynamic round execution and multi-layered encryption transformations, ECEcipher ensures robust confidentiality, making it significantly more secure than conventional block cipher techniques.

ECEcipher Encryption Procedure

The ECEcipher encryption and decryption process follows a structured set of steps to ensure secure data transformation. Below is a detailed breakdown of the encryption procedure:

Procedure for ECEcipher Encryption

- *Input Data Acquisition*
  - The user's data is taken as input plaintext (PTEXT).
- *Binary Conversion*
  - Convert the plaintext (PTEXT) into its binary representation.
- *Block Division*
  - The input plaintext is divided into 64-bit blocks, as ECEcipher processes 64-bit data blocks at a time.
- *Key Retrieval*
  - Obtain a 196-bit convergent encryption key (KEY) from the plaintext.
- *Round Determination*
  - Extract the last four bits from the 196-bit KEY to determine the number of encryption rounds.
- *Matrix Formation*
  - Convert the 64-bit plaintext block into an 8×8 matrix (MAT).
- *Subkey Extraction - SKEY2*
  - Retrieve the first 64-bit subkey (SKEY2) from the 196-bit KEY.
- *Decimal Conversion*
  - Convert SKEY2 (64-bit) into eight decimal values.
- *Column Labeling*
  - Assign these eight decimal values as labels to the top of each column in the matrix.
- *Column-Based Bit Rearrangement*
  - Read bits column-wise based on the ascending order of the decimal values assigned to the columns.
- *Splitting into Two Halves*
  - The 64-bit rearranged data is split into two 32-bit halves by separating even and odd positional bits.
- *Subkey Extraction - SKEY3*
  - Retrieve the second 64-bit subkey (SKEY3) from the 196-bit KEY.
- *Splitting SKEY3*
  - Divide SKEY3 into two 32-bit subkeys.
- *XOR Operation*
  - Perform the XOR operation between the two 32-bit plaintext halves and their corresponding 32-bit subkeys.
- *Merging the Blocks*

Merge the two processed 32-bit blocks into a 64-bit block, arranging bits from both halves.

- *Round Execution*
  - Steps 6 to 15 are repeated for the number of rounds determined in Step 5.
  - The output of each round serves as the input for the next round.
- *Final XOR Operation*
  - Once all rounds are completed, the final 64-bit result is XoRed with the fourth subkey (SKEY4).

Ciphertext Generation

- The final 64-bit result from step 17 is the ciphertext (CTEXT), ready for secure storage or transmission.

Experiment of Proposed ECEcipher

To demonstrate the functionality of the ECEcipher encryption technique, an experiment was conducted using hospital data as sample input. The encryption procedure follows the proposed methodology, ensuring the secure

transformation of sensitive information.

Encryption Procedure

The encryption process begins with user-provided plaintext data, which is then processed as binary data. The first 64-bit block of the input is extracted and encrypted using the ECEcipher approach.

Step 1: User's data are taken as input plain text (P_{TEXT})

Sample input data of user

PNAme	*PMR No*	*DOB*	*Hospital name*	*Disease*	*Amount*
Raj S	IP9475	10/4/2002	Kumars	hunger	12000

P_{TEXT} → RajSIP947510/4/2002Kumarshunger12000

Step 1: Input plaintext (PTEXT)

- 64-bit Plaintext block: RajSIP94
- This is the original user data block considered for encryption.

Step 2: Convert plaintext to binary

Each character is converted into its binary ASCII equivalent.

- *Binary Representation*

01010010 01100001 01101010 01010011 01001001 01010000 00111001 00110100 00110111 00110101 00110001 00110000 00101111 00110100 00101111 00110010 00110000 00110000 00110010 01001011 01110101 01101101 01100001 01110010 01110011 01101000 01110101 01101110 01100111 01100101 01110010 00110001 00110010 00110000 00110000 00110000

- Each character of the plaintext is converted into its 8-bit binary ASCII equivalent.

Step 3: Extract the First 64-bit Block

- *First 64-bit Block for Encryption*

0101001001100001011010100101001101001001010100000011100100110100

- The encryption process works on 64-bit blocks at a time.

Step 4: Generate a 196-bit Encryption Key

- *Generated 196-bit Key*

011001101011001101011100000011100110101001000111101111000001010001100110011110011101001011010110011001010010101110111010001100011011001111001011110000010100101011101110100011000110110011110010111100000010100101011101110100011000110110011110010111100000010100101011101110100000000001010101111110

- This key is generated for convergent encryption.

Step 5: Determine the Number of Encryption Rounds by Subkey1

- Last 4 Bits of Key (Subkey1): 1110
- Number of Rounds: 14 (Binary 1110 converted to decimal 14)
  - The encryption process will execute 14 rounds.

Step 6: Convert Plaintext Block into an 8×8 Matrix

- *Matrix Representation of the 64-bit Block*

0 1 0 1 0 0 1 0
0 1 1 0 0 0 0 1
0 1 1 0 1 0 1 0
0 1 0 1 0 0 1 1
0 1 0 0 1 0 0 1
0 1 0 1 0 0 0 0
0 0 1 1 1 0 0 1
0 0 1 1 0 1 0 0

Step 7-8: Extract First 64-bit Subkey (Subkey2)

Subkey2: 0110011010110011010111000000111001101010010001110110101010110001

- *Converted to Decimal Values*

[102, 179, 92, 62, 57, 78, 100, 113]

Step 9-10: Permutation Based on Column Sorting

- Columns are reordered based on the decimal values of Subkey2.
- The ascending order of these values determines the new column arrangement.

Step 11: Split into Two 32-bit Blocks

- Left Half (L1): First 32 bits of permuted matrix
- Right Half (R1): Last 32 bits of permuted matrix

Step 12-13: Extract Second 64-bit Subkey (Subkey3) and Split into 32-bit Keys

Subkey3 (64-bit): 1010111011001101010110000110111100111001110101010110101001101101

- *Split into Two 32-bit Keys*
  - K1: 10101110110011010101100001101111
  - K2: 00111001110101010110101001101101

Step 14: XOR Operation with Subkeys

Left Half XOR K1: L1′ = L1 ⊕ K1

Right Half XOR K2: R1′ = R1 ⊕K2

Step 15: Swap Blocks

- New Left Half: R1′
- New Right Half: L1′

Step 16: Repeat for 14 Encryption Rounds

This process repeats for 14 rounds, with the intermediate results of each round being used as input for the next round.

Step 17: Extract Final Subkey (Subkey4)

Subkey4 (64-bit): 0110101001110001010100110110111001011101101100101101010110110111

- The final subkey is used for the last transformation.

Step 18: Perform Final XOR with Subkey4

- *Final Ciphertext*

110010101100100001011010111110111000101111110100111011010110101

- This is the secure encrypted form of the input plaintext.
- *Binary to ASCII Decimal:*

[202, 200, 90, 251, 139, 244, 57, 173]

The final ciphertext: ╩╦Z√i╩9¡

Decryption Procedure

The authorized users decrypt the encrypted data using the same key used in the encryption process. In the above encryption process, a 196-bit is used, and the same key is used to decrypt the data. The process of decryption is the reverse procedure of encryption.

Results and Discussion

The ECEcipher encryption algorithm and its key generation process have been implemented using C#.Net and deployed as a cloud service on the myASP.net platform-as-a-service (PaaS) environment. The efficiency of ECEcipher is assessed based on three key metrics: encryption time, decryption time, and security level.

To evaluate its security robustness, the encrypted data is subjected to attack simulations using the ABC Universal Hackman security analysis tool. This tool is installed on the cloud server and is designed to test encryption resilience by executing dictionary attacks and brute-force attacks to retrieve plaintext from encrypted data. The retrieved plaintext is compared with the original plaintext to determine the retrieval percentage, which serves as a benchmark for measuring the security strength of ECEcipher compared to existing encryption techniques.

Encryption Performance Evaluation

The encryption efficiency of ECEcipher is analyzed by measuring the time taken to encrypt data. Table 2 presents a comparative analysis of the encryption time required by ECEcipher, DES, and blowfish encryption algorithms. The results indicate that ECEcipher achieves faster encryption than both DES and blowfish when processing 64-bit data blocks.

Table 2 presents the encryption time required for ECEcipher, DES, and blowfish across different file sizes ranging from 10 to 50 KB. The results indicate that ECEcipher consistently requires less time for encryption compared to DES and Blowfish. For example, when encrypting a 10 KB file, DES requires 7 ms, Blowfish takes 4 ms, while ECEcipher completes the encryption in just 3 ms. As the file size increases, a similar trend is observed—ECEcipher maintains the lowest encryption time, taking only 18 ms for a 50 KB file, whereas DES requires 35 ms, which is nearly twice as long.

The significance of this result lies in ECEcipher's enhanced efficiency. The reduction in encryption time is attributed to its optimized encryption structure, which dynamically selects the number of rounds based on the key rather than using a fixed-round approach like traditional algorithms. Faster encryption makes ECEcipher an ideal choice for real-time cloud storage applications, secured messaging systems, and financial transactions, where speed is critical without compromising security.

Decryption Performance Evaluation

Similarly, the decryption efficiency is assessed based on the time taken to decrypt encrypted data. Table 3 highlights the decryption time of ECEcipher, DES, and blowfish for different data sizes. The results indicate that ECEcipher outperforms other encryption techniques in terms of decryption speed.

Table 3 illustrates the decryption time required to recover the original plaintext from encrypted data using ECEcipher, DES, and blowfish. The results confirm that ECEcipher maintains the lowest decryption time across all file sizes, ensuring faster data retrieval. For instance, when decrypting a 10 KB file, ECEcipher takes 3 ms, while DES requires 6 ms, and blowfish takes 4 ms. As file size increases, ECEcipher continues to outperform its counterparts, requiring just 17 ms for a 50 KB file, whereas DES demands 35 ms.

The key takeaway from these results is that ECEcipher not only encrypts data faster but also decrypts it efficiently, ensuring minimal processing delays in cloud-based environments. This is crucial for applications where quick access to encrypted data is necessary, such as secure cloud storage, e-commerce transactions, and medical data retrieval systems. Additionally, low decryption overhead makes ECEcipher an excellent choice for devices with limited computational power, such as IoT devices, mobile phones, and embedded systems.

Table 2: Encryption time comparison of ECEcipher with existing techniques

Size (KB)	*DES (ms)*	*Blowfish (ms)*	*ECEcipher (ms)*
10	7	4	**3**
20	14	9	**7**
30	21	13	**11**
40	28	18	15
50	35	22	**18**

Table 3: Decryption time comparison of ECEcipher with existing techniques

Size (KB)	*DES (ms)*	*Blowfish (ms)*	*ECEcipher (ms)*
10	6	4	**3**
20	13	8	**6**
30	20	12	10
40	27	17	**14**
50	35	21	**17**

Impact of Number of Rounds on Computation Time

ECEcipher employs a dynamic round selection mechanism, unlike traditional encryption algorithms where the number of encryption rounds remains fixed. The number of encryption rounds in ECEcipher is dynamically determined based on the encryption key, ensuring greater unpredictability and enhanced security.

Table 4 presents a comparative analysis of the computation time for different encryption rounds across multiple data sizes. The results illustrate that ECEcipher efficiently handles encryption and decryption with minimal computational overhead.

Table 4 examines the computation time required for ECEcipher across different numbers of encryption rounds. Unlike traditional encryption techniques, where the number of rounds is fixed, ECEcipher dynamically determines the number of encryption rounds based on the encryption key, enhancing both efficiency and security. The results indicate that while higher rounds increase computation time, ECEcipher still maintains an efficient processing rate. For example, encrypting a 10 KB file with 5 rounds takes only 1.2 ms, whereas 15 rounds require 3.5 ms. Even for larger file sizes, such as 50 KB, the increase in computation time remains controlled, with 5 rounds taking 6.1 ms and 15 rounds requiring 17.3 ms.

This adaptive encryption approach allows users to balance security and performance based on their specific requirements. Fewer rounds ensure fast encryption for low-risk applications, while more rounds provide enhanced security for highly sensitive data. This makes ECEcipher highly scalable and adaptable for different security needs, such as banking transactions, secure file transfers, and government data protection. The ability to increase encryption rounds without significantly affecting performance gives ECEcipher a distinct advantage over traditional encryption methods.

Security Strength Analysis

The security strength of ECEcipher is assessed using the ABC Hackman Tool, which evaluates encryption robustness against brute-force and dictionary attacks. This tool attempts to extract the original plaintext from encrypted ciphertext stored in the cloud and calculates the percentage of successful retrieval.

Table 4: Computation time comparison for different rounds

Size (KB)	*5 Rounds (ms)*	*8 Rounds (ms)*	*10 Rounds (ms)*	*15 Rounds (ms)*
10	1.2	1.9	2.3	3.5
20	2.5	3.7	4.6	6.9
30	3.5	5.8	7.0	10.6
40	4.8	7.5	9.2	14.1
50	6.1	9.6	11.6	17.3

Table 5: Security level comparison of ececipher with existing techniques

S. No.	*Encryption technique*	*Security level (%)*
1	DES	82
2	Blowfish	88
3	ECEcipher	90

To measure the level of security, the following parameters are considered:

- Ed: Total amount of encrypted text stored in cloud storage.
- Ex: Amount of original text retrieved by ABC Hackman after an attack.

The deviation between encrypted text and retrieved text is computed using the following formula:

$$DNm = Ed - Ex$$

The security level percentage (Sl) is then calculated as:

$$SI = \frac{DNm}{Ed} * 100$$

Where Sl represents the security level percentage of the encryption technique.

Table 5 provides a comparative analysis of the security levels of ECEcipher, DES, and Blowfish encryption algorithms.

Table 5 presents the security strength of ECEcipher compared to DES and Blowfish, evaluated using the ABC Hackman tool. This tool analyzes encryption resilience against brute-force and dictionary attacks by attempting to recover the original plaintext from encrypted data stored in the cloud. The security level is determined by measuring the percentage of original text that remains protected after the attack. The results demonstrate that ECEcipher achieves the highest security level of 90%, outperforming DES (82%) and Blowfish (88%).

These findings confirm the superior encryption strength of ECEcipher, making it highly resistant to common cryptographic attacks. The higher security level is attributed to ECEcipher's key-based dynamic round selection, substitution-permutation structure, and advanced XOR transformations. This ensures that even if an attacker gains access to the encrypted text, extracting meaningful data remains computationally infeasible. As a result, ECEcipher is well-suited for high-security applications, including military communications, financial data security, and enterprise cloud storage.

Conclusion

Cloud environments provide scalable data storage solutions, but ensuring data security remains a critical challenge for users. The proposed ECEcipher encryption technique enhances public cloud security by enabling users to

encrypt data before transmission, preventing unauthorized access. ECEcipher is a symmetric block cipher encryption method that utilizes a 196-bit encryption key, generated within a cloud key service and securely distributed to users. The encryption and decryption process maintains low computation time, making it faster than traditional encryption techniques such as DES and blowfish. Security analysis results confirm that ECEcipher provides superior protection against brute-force and dictionary attacks, achieving a higher security level than existing techniques. The proposed encryption method has been successfully implemented as a cloud service and deployed in a cloud-based environment, ensuring real-world applicability. Comparative analysis, presented in tables and figures, demonstrates that ECEcipher significantly outperforms existing encryption algorithms in terms of efficiency and security. Future research will focus on developing an enhanced symmetric encryption technique for private cloud environments, further improving confidentiality and data protection mechanisms in enterprise cloud systems.

Acknowledgment

We sincerely acknowledge the Head of the department, Dr. J. James Manoharan, and Dr. J. Princy Merlin, Principal of the institution, for providing the facility to complete this paper Successfully.

References

Arockiam, L., & Monikandan, S. (2013). Data security and privacy in cloud storage using hybrid symmetric encryption algorithm. *International Journal of Advanced Research in Computer and Communication Engineering*, *2*(8), 3064-3070.

Arockiam, L., & Monikandan, S. (2014, January). Efficient cloud storage confidentiality to ensure data security. In *2014 International Conference on Computer Communication and Informatics* (pp. 1-5). IEEE.

Arockiam, L., Monikandan, S., & Parthasarathy, G. (2017). Cloud computing: A survey. *Journal of Computer and Communication Technology: Vol*, *8*(1), 4. https://core.ac.uk/download/pdf/480907559.pdf

Aruljothi Rajasekaran, & Jemima Priyadarsini R. (2024). ECDS: Enhanced Cloud Data Security Technique to Protect Data Being Stored in Cloud Infrastructure: Data Security in Cloud Infrastructure. *The Scientific Temper*, *15*(04), 3113–3121. https://doi.org/10.58414/SCIENTIFICTEMPER.2024.15.4.19

Aslam, J. M., & Kumar, K. M. (2024). Enhancing cloud data security: User-centric approaches and advanced mechanisms. *The Scientific Temper*, *15*(01), 1784–1789. https://doi.org/10.58414/SCIENTIFICTEMPER.2024.15.1.29

da Rocha, M., Valadares, D. C. G., Perkusich, A., Gorgonio, K. C., Pagno, R. T., & Will, N. C. (2020). *Secure cloud storage with client-side encryption using a trusted execution environment*. arXiv preprint arXiv:2003.04163. https://doi.org/10.48550/arXiv.2003.04163

Qureshi, M. B., Qureshi, M. S., Tahir, S., Anwar, A., Hussain, S., Uddin, M., & Chen, C.-L. (2022). Encryption Techniques for Smart Systems Data Security Offloaded to the Cloud. *Symmetry*, *14*(4), 695. https://doi.org/10.3390/sym14040695

Sabeerath K., & Manikandasaran S. Sundaram. (2024a). BTEDD: Block-level tokens for efficient data deduplication in public cloud infrastructures. *The Scientific Temper*, *15*(03), 2507–2514. https://doi.org/10.58414/SCIENTIFICTEMPER.2024.15.3.16

Sabeerath K., & Manikandasaran S. Sundaram. (2024b). ESPoW: Efficient and secured proof of ownership method to enable authentic deduplicated data access in public cloud storage. *The Scientific Temper*, *15*(04), 3165–3172. https://doi.org/10.58414/SCIENTIFICTEMPER.2024.15.4.25

Selvaraj, R., & Sundaram, M. S. (2023a). 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. (2023b). 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