Library Progress International
Vol.44 No. 3, Jul-Dec 2024: P. 14580-14599

Print version ISSN 0970 1052
Online version ISSN 2320 317X

Original Article

Available online at www.bpasjournals.com

Efficient Hybrid Encryption Algorithm for Securing Data in Cloud Environment

Aaqib Nisar Bhat[1], Rajiv Kumar[2]

[1] Research Scholar, School of Computing, RIMT University, Mandi Gobindgarh Punjab, INDIA.
[2] Professor, School of Computing, RIMT University, Mandi Gobindgarh Punjab, INDIA.

How to cite this article: Aaqib Nisar Bhat, Rajiv Kumar (2024) Efficient Hybrid Encryption Algorithm for Securing Data in Cloud Environment. *Library Progress International,* 44(3), 14580-14599.

ABSTRACT

Protecting sensitive data is crucial in the age of widespread data digitalization and cloud-based services. This paper proposes a unique hybrid encryption algorithm deployed and tested over the cloud platform. The algorithm is reinforced with the robust cryptographic combination of Advanced Encryption Standard (AES) and Rivest-Shamir–Adleman (RSA). By utilizing the inherent benefits of symmetric and asymmetric encryption, the algorithm aims to improve the efficiency and security of data at rest and data being transmitted in a cloud-based environment. When tested on varied-length files and large data, the proposed algorithm encrypts the data securely and quickly. Various attacks including brute force, cryptanalysis were performed in the deployed environment to test the efficiency and effectiveness of the algorithm. It is observed that the algorithm outperformed Diffie-Hellman algorithm in terms of speed of encryption and data security with a factor of 6%. The algorithm's robustness against different security risks and its flexibility in changing cloud situations through an extensive examination is proven. The results not only highlight the hybrid approach's efficacy but also support its possible use in practical applications, therefore launching a new wave of robust cloud-based data management systems.

***Keywords*:** Cipher-Text, Hybrid Encryption Algorithm, AES Algorithms, RSA Algorithms, Data Encryption, Data Decryption..

1. Introduction

In the current information technology landscape, cloud computing has emerged as a major paradigm that is transforming the supply and use of computer resources and services. Because of its scalability, affordability, and accessibility, cloud computing is now the basis for a wide range of applications, including data processing and storage, software as a service (SaaS), and more. [1]. The cloud services on the computer are really located on a distant, unidentified computer rather than on the machine itself. The user is expected to access these services by using an internet connection to do so.

With cloud computing, resources are made available from remote locations, like in- premise resources. It is possible to access and save data anywhere in a cloud environment, thus eliminating the need to manage storage, hardware, and software manually [2]. Nevertheless, there is an important trade-off associated with moving data and apps to the cloud: strong security measures are required to shield private data from illegal access and security lapses [3].

Cloud computing solutions are now widely integrated into information technology systems, offering scalable and adaptable platforms for data processing, storage, and accessibility [4][5]. But there are also previously unheard-of difficulties associated with this
widespread adoption, namely with regard to the security and privacy of sensitive data in cloud systems [6]. The increasing dependence of enterprises on cloud infrastructures for data management has led to a pressing need

for strong encryption algorithms that can guarantee confidentiality and integrity [7].

This study presents a novel hybrid cloud algorithm that tackles the changing cloud computing security scene. Our method combines the secure key exchange capabilities of the Rivest–Shamir–Adleman (RSA) algorithm with the efficiency of the Advanced Encryption Standard (AES). By combining various cryptographic methods, a hybrid cloud environment's data transmission and storage procedures will be strengthened, providing a complete answer to the intricate security needs of modern cloud-based systems [8].

This study is driven by the urgent need to balance the advantages of cloud computing with the requirement to protect sensitive data from a variety of possible risks [9]. Strong and flexible security measures are essential as more and more enterprises use hybrid cloud architectures, which combine private and public cloud services [10]. The goal of this research is to improve the availability, confidentiality, and integrity of data while it is in transit and at rest by putting forth a novel hybrid cloud algorithm [11].

The next sections delves into the extant literature, delineating the contemporary predicaments in cloud security and the constraints of conventional encryption techniques. The details of our suggested hybrid cloud algorithm, which combines AES and RSA, are then presented. We also go over the development process that was used. Comprehensive assessment findings are given, proving our approach's effectiveness in reducing security threats [12]. In order to improve information safety and encourage trust in the ongoing deployment of cloud computing technologies, our study aims to provide the groundwork for secure data management in hybrid cloud settings [13].

2. Literature Review

The literature has generally acknowledged the significance of encryption in protecting data transferred and stored via cloud computing [14]. A technology called cloud computing promises to boost productivity and efficacy in today's businesses. However, because of the ways in which service delivery is carried out, this technology has several security flaws. Using cloud computing services is significantly hampered by security concerns [15].The way computer resources are accessed and used has changed dramatically as a result of cloud computing, which has several benefits, including cost-effectiveness and scalability. However, these advantages come with the difficulty of safeguarding private information against illegal access and security breaches; for this reason, encryption is a crucial part of cloud security [16].

Two of the most widely used encryption algorithms in use today are Rivest-Shamir- Adleman (RSA) and the Advanced Encryption Standard (AES). The stability and effectiveness of AES, which was first developed by Rijmen and Daemen in 2002, make it a popular option for a variety of cloud encryption applications [17]. Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Galois/Counter Mode (GCM) are three of the encryption modes that AES offers; each is designed to meet certain use cases and specifications.

In contrast, RSA is well-known for its contribution to public key cryptography, which makes digital signatures and safe key exchange possible [18]. The mathematical basis of RSA has demonstrated its durability and is still an essential component in cloud communication security. In cloud contexts, this method is especially well-suited for key management.

Cloud encryption presents both potential and concerns that have been covered in a number of in-depth surveys and research. A survey on mobile cloud computing was done by Dinh et al., who focused on the architecture and application elements of cloud security [19]. The significance of safe cloud computing was highlighted by Ristenpart et al.'s investigation of information loss in third-party compute clouds [16].

Balachandra and Chakravarthy implemented an effective RSA cryptography method, highlighting the significance of optimization in cloud encryption algorithms [18]. These studies highlight how encryption methods must advance and how cloud security is a dynamic field. In an age where data is one click away, it must be more secure to prevent illegal access. To maintain data security and confidentiality, many data security sectors have evolved [23]. Cloud storage is useful for many applications, but it exposes data to programs with security flaws [24]. Building on these fundamental studies, the review article provided here provides a thorough overview of AES and RSA, their useful use in cloud systems, important management techniques, and security considerations. We want to offer a thorough resource for researchers, practitioners, and cloud service

providers to improve the security posture of cloud-based systems by critically analyzing the state of the art in cloud encryption.

3. AES Algorithm

The term "Rijndael" is synonymous with the Advanced Encryption Algorithm (AES) [20]. The U.S. National Institute of Standards and Technology (NIST) officially announced AES as the standard encryption algorithm. AES utilizes keys of varying sizes, including 128, 192, and 256 bits, determined by the number of cycles involved in the encryption process. Specifically, 128-bit keys are employed for 10 rounds, 192-bit keys for 12 rounds, and 256-bit keys for 14 rounds. One of the most popular symmetric block ciphers is AES. Its unique structure allows it to be integrated into worldwide tools and software, ensuring great security [26].

The core operation of AES relies on 4x4 cross-matrices. The encryption process encompasses initial and final rounds, key expansion, and key mixing. The key components of each round, depicted diagrammatically in Fig. 1, include the Add Round Key, SubBytes, ShiftRows, MixColumns, and a final Add Round Key. The initial round incorporates additional steps, such as the expansion of the key [21].

The initial and final rounds follow similar procedures, with variations only in the mixing of different columns. Notably, the efficiency of the AES algorithm is consistent on both software and hardware platforms. Its effectiveness is attributed to the systematic execution of encryption and decryption processes, making it a widely accepted and secure choice for various applications [22]. Cloud security and data security are key concerns nowadays, hence cryptographic methods are needed. Reason for usage AES encrypts and secures server data [25].

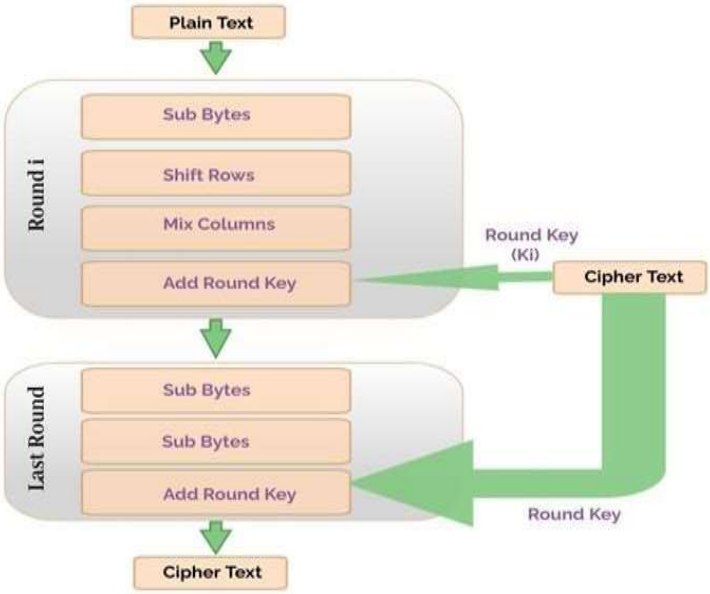

Figure.1. AES Algorithm and its Diagrammatic Representation

4. RSA Algorithm

The widely used RSA algorithm was first released in 1977, is named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman, and is another form of Asymmetric Algorithms [23]. The foundation of RSA is the mathematical complexity of big prime numbers and their resistance to factoring. The creation of keys, encryption, and decryption are all intricate aspects of the algorithm's operation. Cloud computing is a technological and social reality, but security remains the fundamental barrier to cloud use [27]. To safeguard sensitive data, the homogeneous encryption approach is used with unstable information dispersion [28].

RSA requires the creation of a public-private key pair as its initial step. N (modulus), the product of two huge prime integers, p and q, is calculated. Additionally, φ(N), Euler's totient function, is computed. An exponent 'e,' usually a tiny prime integer, plus N make up the public key. A corresponding exponent'd' is' generated so that (d * e) ≥ 1 (mod φ(N)) is part of the private key.

Encryption happens when the keys are established. The sender encrypts a message'm' as an integer using the recipient's public key. The following formula is used to calculate the ciphertext, i.e., $m \equiv C^d \bmod N$. The recipient uses their private key to perform decryption on their end. The formula ($m \equiv C^d \bmod N$.) yields the original message'm'. The difficulty of factoring the product of two big primes (N = p * q) determines the security of RSA, and larger key lengths provide more security.

Although RSA requires more processing power than symmetric key algorithms, it is widely used in digital signature creation and verification as well as secure key exchange protocols like SSL and TLS. Padding techniques like OAEP or PKCS#1 v1.5 are frequently used to improve security. Because of its adaptability and efficiency in protecting communication and confirming the legitimacy of digital signatures, RSA continues to be a mainstay of contemporary

cryptography despite its computing costs. The Figure 2 below shows workings of the said algorithm.

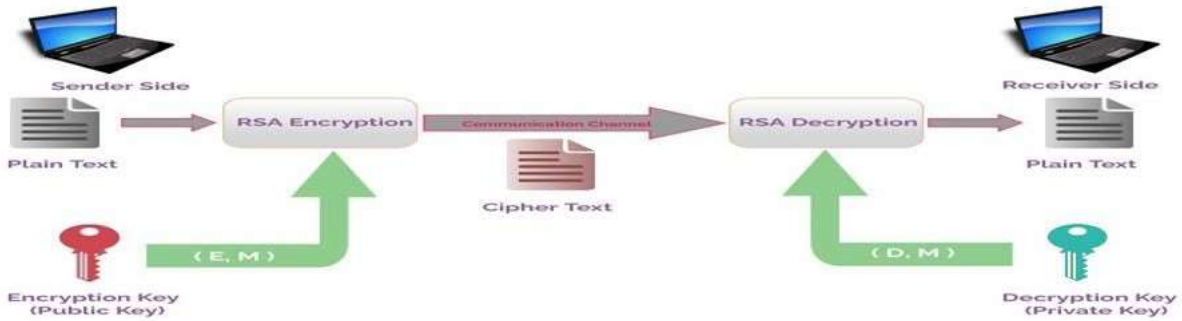

Figure 2. RSA Algorithm and its Diagrammatic Representation

5. Methodology

In terms of security, speed, and key length, this study presents a unique encryption method that optimizes the performance of both the AES and RSA algorithms. This strategy seeks to improve overall code security by utilizing RSA's encryption capabilities and exploiting the encryption, stability, and key management qualities of AES.

This technique ensures the safe management of keys to prevent unwanted access by using the AES algorithm only for encryption. For security concerns, the discussion of keys is given priority ahead of time, with an emphasis on the possibility of frequent updates to preserve anonymity. By using AES keys excessively, the suggested hybrid technique encodes the data and creates ciphertext 1 to accomplish strong encryption.

Conversely, the public key of the RSA method is used to encrypt ciphertext 1, and the key of the AES algorithm helps to create ciphertext 2. Notably, the decoding procedure only requires the RSA algorithm's private key, but the public key—which is generally accessible— ensures that material encoded with RSA is accessible. The data encoded by the public RSA key cannot be decoded due to the secrecy of the RSA private key. Notably, AES is not included in the key, maintaining the encryption system's overall security.

The private and public keys can be generated by randomly executing a number of mathematical operations. Figure 5 below illustrates the entire hybrid encryption and decryption process.

The RSA method is used in this encryption technique to encode material, which results in variations in the amount of time needed for the encoding and decoding operations. Furthermore, the length of the data is still unknown; longer encryption times are associated with larger datasets. Interestingly, every encryption procedure

is made to convert plaintext into ciphertext, which is formed into fixed-length 128-bit blocks.

When AES and RSA are used to create ciphertext, the security is increased while also increasing the whole encryption process's efficiency. It is essential to emphasize that the durations for encoding and decoding are not fixed and may vary depending on variables like the volume of data.

The procedure for decryption is the opposite of encryption. To obtain the AES key and ciphertext 1, RSA is used to decode ciphertext 2. After that, the original plaintext is recovered by utilizing the AES key to decrypt ciphertext 1. This two-way encryption and decryption process increases operational effectiveness while strengthening security.

5.1 Key Generations in the Hybrid Algorithm:

The following steps are used for the generation of keys:.

1. *Two large different prime no's i.e. p and q are selected.*
2. *Then multiple these two no's i.e. p and q to obtain n. n = p x q.*
3. *Compute the Euler Function. Ø (n) = (p-1) (q-1).*
4. *Randomly select a positive integer e and. 1<e<Ø (n) is made and gdc (e, Ø(n)) = 1.*
5. *In accordance to the above mathematical equation.*

   ed = 1mod Ø(n), the result of the d is calculated, where 0<d<n.

6. *In accordance with the equation..*

   PU = {e,N}, where e is a public key and RSA algorithm's public key is saved.

7. *In accordance with mathematical expression. PR = {d,p,q}, the secrete key d is saved.*

5.2 Hybrid Algorithm Encryption:

The process of encryption goes through different operations, processes, and changes. These operations involve the file encryption of these two AES and RSA algorithms and are in diagram
3. A 4*4 matrix is allocated with 128-bit data, and all the functional units are grouped in AES algorithm. The following three simple procedures are involved.

1). Sub-Bytes: These are also named as S-box permutations and is one of the non-linear byte conversion in an AES encryption algorithm round encryption, by using the substitute table each byte is determined independently. In this case, the mapping strategy is to take the high 4 byte bits of the matrix as the row value and the low 4 byte bits as the column value, and then produce the unit with the column value as the index from the appropriate point in the s-box.

2). Rows-Shift: Each row in the forward ShiftRows is cyclically pushed to the left by a row number offset, i.e. the ith row of the state matrix is shifted left by I bytes MixColumns.

MixColumns transform acts on each column in state and considers each column as a fourth degree polynomial. The addition and multiplication operations of the MixColumns operation are both specified on the finite field.

3). Present the Round Key. When converting Add RoundKey, the value obtained is the 128-bit State xor by bit and the 128-bit key. When encrypting the AES key with an RSA algorithm, the plaintext is divided into groups, and the binary values of each group m are all less than n, where n is the product of the large prime numbers p and q, e is a random positive integer, and the cypher text c generated can be obtained from the following equation:

$$c = m^e \bmod n, \ \& \ 0 <= m < n \qquad .. \text{ (i)}$$

5.3 Decryption Using a Hybrid Approach:-

In the first layer, the ciphertext encrypted by the public key is decoded using the RSA algorithm's private key, and the AES key is used for the conversion of the ciphertext to plaintext.

By using equation ii, ciphertext c is decoded, and plain text m is obtained by using RSA decryption.

$m = c^d \bmod n$. (ii)

Where d is computed as a result of the key generation algorithm and n is the product of two large numbers, i.e., p and q. During the decoding process of the AES algorithm, sub-bytes, shift-rows and mix-columns are the opposite of those of the encryption process, except in the add round key because of the xor operation, where the reverse process is equivalent to the forward activity.

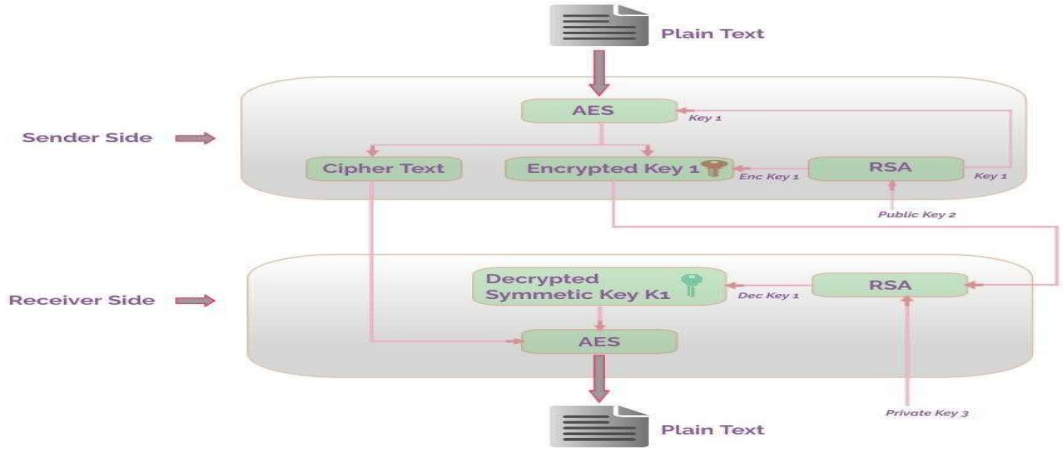

Figure 3. AES & RSA Encryption/Decryption Using the Hybrid/Mixed Algorithm Approach

6. Experiment & Results

6.1 Comparative Analysis Of Existing And Present Study

The present study is conducted on Windows Operating System platforms using Java. Additionally, the Google Cloud Platform is used to conduct the study in the Google Cloud setup with the following configurations:
Version: 4.28.0, 2023-06

20230608-1333 is the build ID.

OS: Win32 / x86_64 /Linux/ Windows 7, version 6.1 Java supplier: Adoptium Eclipse
Version of the Java runtime: 17.0.7+7

The main goal is to run and test the proposed hybrid algorithm using text files with sizes ranging from 1 MB to 100 MB. Another hybrid technique that explored the Diffie-Hellman algorithm with the Advanced Encryption Algorithm (AEA) was used throughout the experimentation process to validate the performance of the proposed algorithm detailed in this paper. The Advanced Encryption Algorithm's 256-bit key and the Diffie-Hellman algorithm's 2048-bit key were selected to conduct the present study. The proposed algorithm was developed using the Java programming language to carry out the study. The experiment's main goal was to evaluate the hybrid encryption system's ability to encrypt and decode text files of different sizes. Furthermore, measurements were made of the hybrid approach's memory needs.

Critical performance parameters, time taken, and memory usage throughout the encryption and decryption procedures were closely observed and recorded, as presented in Table 1. Using AES and RSA **keys** with lengths of 256 and 2048 bits, respectively, this study also examined the integration of the Diffie-**Hellman**

and the Advanced Encryption algorithms in a Java programming environment. The main goal was to evaluate the hybrid AES-RSA encryption system's practicality and effectiveness in protecting text files while taking various file sizes into account. By carefully calculating key lengths, the research attempted to achieve a compromise between robust security and computing performance. The experimental setup, carried out within a cloud environment and Java Eclipse IDE, allowed for a systematic implementation of the hybrid encryption system, with careful observation of memory consumption serving as a critical performance metric. Encrypting and decrypting text files of different sizes was part of the testing procedure, which allowed for a thorough assessment of the security and effectiveness of the hybrid approach. In contrast, the memory requirements of the AES and RSA algorithms were also examined. Table 1 provides a summary of the experiment's findings and offers important insights into how well the hybrid AES-RSA method performs in protecting data files. It contains metrics related to memory utilization, encryption and decryption times, and other significant information.

Plain Text	Plain Text	Plain Text	Encryption Time in Nanoseconds		Decryption Time in Nanoseconds		Memory Used After Encryption in Bytes	
Size in KB.	Size in Bytes.	Size in MB.	AES+Diffie	AES+RSA	AES+Diffie	AES+RSA	AES+Diffie	AES+RSA
1149	1,176,064	1200	68000000	58000000	49000000	49000000	34,619,348	23,068,672
2297	2,352,128	2240	103000000	77000000	81000000	71000000	56,348,408	16,159,400
4594	4,704,256	4480	150000000	95000000	134000000	122000000	48,772,536	1,000,880
9188	9,408,512	8970	226000000	136000000	202000000	199000000	67,199,000	40,488,424
13782	14,112,768	13400	235000000	247000000	315000000	271000000	100,667,192	95,270,664
27564	28,225,536	26900	493000000	371000000	439000000	422000000	197,182,248	99,097,248
32551	33,331,231	37100	437000000	412000000	519000000	505000000	232,813,640	222,951,672
6400	67,108,864	64000	887000000	812000000	698000000	757000000	473,773,504	463,177,848
96000	100,663,296	96000	1271000000	1088000000	1302000000	1195000000	715,271,944	694,858,656

Table 1. Hybrid Encryption and Decryption using AES-Diffie & AES-RSA

The key metrics from the experimental studies of the hybrid AES-RSA and hybrid AES- Diffie-Hellman encryption systems are shown as graphs in Figure 4. This graph provides a thorough representation of the post-encryption memory needs and encryption and decryption durations for text files across a range of dataset sizes, from tiny to large. Figure 4, which presents these parameters graphically, is a useful tool for analyzing both encryption methods' performance in different contexts.

A direct comparison of the encryption and decryption timings for various text file sizes inside each hybrid encryption scheme is made easier by the graphical depiction. It also makes it possible to pinpoint the circumstances in which one algorithm performs better than the other. In order to improve readability, the hybrid AES-RSA system's encryption and decryption times are represented by separate lines and others as well, respectively. All things considered, Figure 4 offers a sophisticated comprehension of the hybrid algorithms' effectiveness, allowing us to make perceptive judgments about how well each performs with varying dataset sizes.

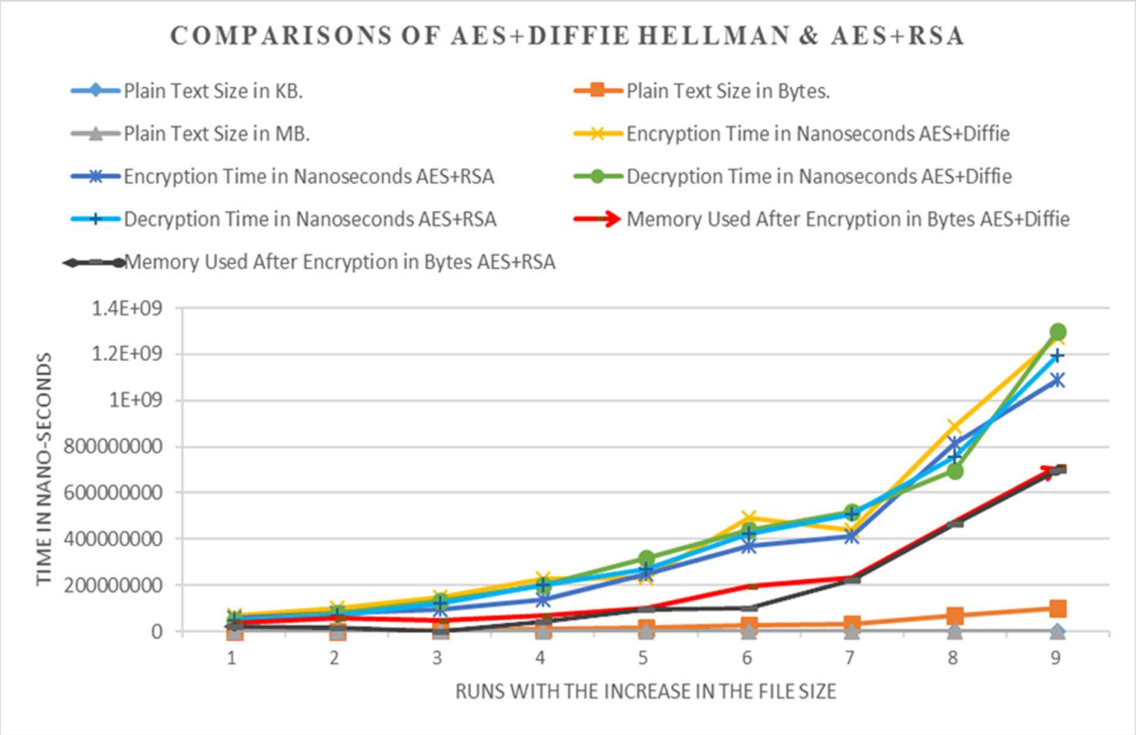

Figure 4. Performance of Proposed Algorithm (Time & Memory Usage)

A comparison of the two hybrid encryption strategies—Advanced Encryption Standard (AES) combined with Diffie-Hellman and Rivest-Shamir-Adleman (RSA)—showed significant differences in terms of performance metrics. Interestingly, when compared to the AES-Diffie- Hellman technique, the hybrid AES-RSA approach performed better on a number of critical performance metrics. More specifically, as compared to its predecessor, the AES-RSA hybrid showed much quicker encryption speeds and more effective memory use. The difference in encryption times and memory use highlights how much more efficient the AES-RSA hybrid technique is in protecting text data. It's interesting to note that even while the AES-RSA hybrid's decryption time was somewhat faster than the AES-Diffie-Hellman hybrid's, the encryption performance was still considered to be better overall. The study's objective was accomplished, especially in terms of encryption, which highlights how useful the AES-RSA hybrid encryption technique is in the Java programming environment.

6.2 Attacks Performed On Hybrid Algorithm (Aes & Rsa)

6.2. Data In Rest

Attacks against cryptographic algorithms like the hybrid AES-RSA system are driven by a number of reasons with roots in criminal activity, information warfare, and cybersecurity. These attacks primarily seek to undermine the availability, confidentiality, or integrity of sensitive data by taking advantage of possible flaws in the algorithm. Cybercriminals and state-sponsored actors may try to obtain illegal access to encrypted data for financial gain, competitive advantage, or espionage purposes. Attacks may also be motivated by ideological goals, with the

intention of undermining privacy, upsetting communication networks, or even casting doubt on the reliability of cryptographic standards. The need for closely examining cryptographic systems is highlighted by the ongoing advancements in technology and the growing proficiency of cybercriminals. Through comprehension and resolution of such weaknesses, the security community may strengthen algorithms and protocols, guaranteeing their resistance against malevolent endeavors to compromise the strength of encryption methods. As a result, the continuous investigation of cryptographic algorithmic assaults not only helps to fortify current systems but also propels the area of cybersecurity forward by encouraging creativity and the creation of encryption techniques

that are safer so that confidentiality, integrity, and availability of the data are maintained.

Firstly, an encrypted string is generated by the combination of Advanced Encryption Standard and Rivest-Shamir-Adleman, also called a Hybrid Algorithm. The Hybrid Algorithm is executed in the Java Eclipse IDE to generate an output by giving it the text, which it encrypts and decrypts back into its original form. AES encrypted key is generated as well as the encrypted message by providing some text as shown in the algorithm.

6.2. Algorithm:

1. *Generate AES and RSA Keys:*
   An AES key is generated using `KeyGenerator` with a key size of 256 bits.
   An RSA key pair is generated using `KeyPairGenerator` with a key size of 2048 bits.

2. *Message Encryption:*
   The original message ("Aaqib Nisar Bhat") is encrypted using AES encryption. This is achieved by creating an `AES` cipher, initializing it with the AES key, and then encrypting the message.
   The AES key used for encrypting the message is then encrypted using RSA encryption. This is done by creating an `RSA` cipher, initializing it with the RSA public key, and encrypting the AES key.

3. *Base64 Encoding:*
   Both the encrypted message and the encrypted AES key are converted to Base64- encoded strings. This is a common practice when transmitting binary data over text-based protocols or storing it as text.

4. *Print Encrypted Message and Encrypted AES Key:*
   The program prints the original message, the Base64-encoded encrypted message, and the Base 64-encoded encrypted AES key.

5. *Decryption:*
   The program then demonstrates the decryption process. The Base64-encoded encrypted AES key is decoded.
   The AES key is decrypted using the RSA private key.
   The original AES key is reconstructed using `SecretKeySpec`.
   The encrypted message is then decrypted using the decrypted AES key.

6. *Print Decrypted Message:*
   Original Message: Operating System
   Encrypted Message: cyjvGGeDUY/D4abXptSnG3QoR3GpYXLGg/GMo7Nisps= Decrypted Message: Operating System

The overall algorithm uses symmetric encryption (AES) for efficiently encrypting the actual data, and asymmetric encryption (RSA) for securely transmitting the symmetric key used in the symmetric encryption process. Hybrid Algorithm is provided with a file for encryption, which generates the cipher text. The cipher text is then placed in the file and now forced to generate original text by using Brute Force Attack. As possible as many as keys were generated by this brute force attack to extract the original message, as shown.

6.2. Algorithm:

1. *Initialize Paths:*
   Specify the input file path (`inputFile`) and the output encrypted file path (`encryptedFile`).
2. *Generate AES and RSA Keys:*
   Generate an AES key using `KeyGenerator` with a key size of 256 bits.
   Generate an RSA key pair using `KeyPairGenerator` with a key size of 2048 bits.

3. *Read File Content:*
   Read the content of the input file into a string (`fileContent`).
4. *Print Original File Content:*
   Print the original file content.
5. *Encrypt File Content with AES:*
   Encrypt the file content using AES encryption. Measure the time taken for encryption.
6. *Encrypt AES Key with RSA:*
   Encrypt the AES key using RSA encryption.
7. *Base64 Encoding:*
   Convert the encrypted file content and the encrypted AES key to Base64-encoded strings.
8. *Print Encrypted File Content and AES Key:*
   Print the Base64-encoded encrypted file content and the Base64-encoded encrypted AES key.
9. *Decrypt File Content with AES:*
   Decrypt the encrypted file content using AES decryption.
10. *Decrypt AES Key with RSA:*
    Decrypt the encrypted AES key using RSA decryption. Measure the time taken for RSA decryption.
11. *Print Decrypted File Content:*
12. *Print the output:*

Original Message: Programming
Encrypted File Content: vy6C9pjdUpWx5iS226bWUg== Decrypted Message: Programming

The algorithm demonstrates a file encryption and decryption process using a combination of symmetric (AES) and asymmetric (RSA) encryption. The input file is read, encrypted, and then decrypted, with the results printed to the console.

6.2. Brute Force Attack on the Text File

Table 2 shows that the data that was encrypted by the Hybrid Algorithm is tested against the popular Brute Force attack to obtain the original data. Keys to decrypt the secured data were generated through brute force attacks to extract the original message, but the original message was not decrypted back into its original form. Hence, the proposed hybrid algorithm significantly preserves the sanctity of data.

A straightforward brute-force attack to decrypt a message encrypted using the Caesar cipher is implemented in the Java software that is given. The detailed algorithm used is given below:

1. *Initialize Variables:*
   Declare variables `i`, `j`, `k`, `flag`, `index`, and `keyVal`. Declare strings `pt` for the encrypted data.
   Declare character arrays `ct1` for the encrypted data and `pt1` for the decrypted data. Initialize `k` to the ASCII value of lowercase 'a'.
2. *User Input:*
   Use `Scanner` to get user input for the encrypted data (`pt`).
3. *Decrypt Using Brute-Force:*
   Initialize a loop for each possible key value from 1 to 50. For each key:
   Iterate through each character in the encrypted data (`ct1`). Check if the character is a space or a printable ASCII character.
   If printable, calculate the adjusted index based on the current key.
   Adjust the index and update the corresponding character in the decrypted data (`pt1`).
   Print the decrypted message for the current key.
4. *Print Decrypted Messages:*

Original Text	Computer Memory
Encrypted Text	urPIpbTtPcgn4JDsp3QbLA==

DECRYPTION OF DATA USING BRUTE-FORCE ATTACK:		
Key 1: tqOHoaSsObfm3ICro2PaK@<<	Key 2: spNGn`RrNael2HBqn1O`J?;;	Key 3: roMFm_QqM`dk1GApm0N_I>::
Key 4: qnLEl^PpL_cj0F@ol/M^H=99	Key 5: pmKDk]OoK^bi/E?nk.L]G<88	Key 6: olJCj\NnJ]ah.D>mj-K\F;77
Key 7: nkIBi[MmI\`g-C=li,J[E:66	Key 8: mjHAhZLlH[_f,B<kh+IZD955	Key 9: liG@gYKkGZ^e+A;jg*HYC844
Key 10: khF?fXJjFY]d*@:if)GXB733	Key 11: jgE>eWIiEX\c)?9he(FWA622	Key 12: ifD=dVHhDW[b(>8gd'EV@511

Key 13: heC<cUGgCVZa'=7fc&DU?400	Key 14: gdB;bTFfBUY`&<6eb%CT>3//	Key 15: fcA:aSEeATX_%;5da$BS=2..
Key 16: eb@9`RDd@SW^$:4c`#AR<1--	Key 17: da?8_QCc?RV]#93b_"@Q;0,,	Key 18: c`>7^PBb>QU\"82a^!?P:/++
Key 19: b_=6]OAa=PT[!71`] >O9.**	Key 20: a^<5\N@`<OSZ 60_\~=N8-))	Key 21: `];4[M?_;NRY~5/^[}<M7,((
Key 22: _\:3ZL>^:MQX}4.]Z];L6+"	Key 23: ^[92YK=]9LPW|3-\Y{:K5*&&	Key 24:]Z81XJ<\8KOV{2,[Xz9J4)%%

Table 2. Brute Force Attack on the Encrypted Text

The algorithm uses a Caesar cipher decryption approach by shifting each character in the encrypted message backward through the printable ASCII characters (32 to 126). It performs a brute-force attack by trying all possible key values that are generated randomly so that the security should be preserved and printing the decrypted messages for each key The brute force attack was performed where 100 keys were generated initially, but none of the keys decrypted the text to its original form. Up to 500 keys in steps of 50 were generated and tested to check the strength of the algorithm. It was observed that the algorithm significantly safeguarded the encrypted data to defend against the brute force attack.

6.2. Brute Force Attack on the Data File

Data files encrypted with the proposed hybrid algorithm stored on the system are now attacked using brute force to generate original data. Many keys were randomly generated and used to extract the original data. But the original data was not decrypted back, thus defying the security attack, which showed the proposed algorithm had been successful in preserving the security of the data, and the results obtained are presented in Table 3.

Algorithm:

1. *Open File:*
   Create a `File` object representing the input file named "Cipher.txt". Create a `FileInputStream` to read the contents of the file.
2. *Initialize variables:*
   Declare variables `i`, `j`, `k`, `flag`, `index`, and `keyVal`. Declare strings `pt` for each line of the file. Declare character arrays `ct1` for the ciphertext and `pt1` for the decrypted text. Initialize `k` to the ASCII value of lowercase 'a'.
3. *Read File Content:*
   Use a `Scanner` to read each line of the file until there are no more lines.

4. *Decrypt Using Brute-Force:*
   For each line of the file (each ciphertext): Convert the line into a character array (`ct1`).
   Initialize a loop for each possible key value from 1 to 50. For each key:

   Iterate through each character in the ciphertext (`ct1`).
   Check if the character is a space or a printable ASCII character.
   If printable, calculate the adjusted index based on the current key.
   Adjust the index and update the corresponding character in the decrypted text (`pt1`).
   Print the decrypted message for the current key.
5. *Print Decrypted Messages:*
   Print the decrypted message for each key for each line of the file.

File Name	Cipher.txt	
Original Content	Programming	
Encrypted Text	8P6TkoPW+H9AkL07B+c6cA==	
DECRYPTION OF DATA USING BRUTE-FORCE ATTACK ON A FILE:		
Key 1: 7O5SjnOV*G8@jK/6A*b5b@<<	Key 2: 6N4RimNU)F7?iJ.5@)a4a?;;	Key 3: 5M3QhlMT(E6>hI-4?(`3`>::
Key 4: 4L2PgkLS'D5=gH,3>'_2_=99	Key 5: 3K1OfjKR&C4<fG+2=&^1^<88	Key 6: 2J0NeiJQ%B3;eF*1<%]0];77
Key 7: 1I/MdhIP$A2:dE)0;$\/\:66	Key 8: 0H.LcgHO#@19cD(/:#[.[955	Key 9: /G-KbfGN"?08bC'.9"Z-Z844
Key 10: .F,JaeFM!>/7aB&-8!Y,Y733	Key 11: -E+I`dEL =.6`A%,7X+X622	Key 12: ,D*H_cDK~<-5_@$+6~W*W511
Key 13: +C)G^bCJ};,4^?#*5}V)V400	Key 14: *B(F]aBI|:+3]>")4|U(U3//	Key 15:)A'E\`AH{9*2\=!(3{T'T2..
Key 16: (@&D[_@Gz8)1[<'2zS&S1--	Key 17: '?%CZ^?Fy7(0Z;~&1yR%R0,,	Key 18: &>$BY]>Ex6'/Y:}%0xQ$Q/++
Key 19: %=#AX\=Dw5&.X9|$/wP#P.**	Key 20: $<"@W[<Cv4%-W8{#.vO"O-))	Key 21: #;!?VZ;Bu3$,V7z"-uN!N,((
Key 22: ": >UY:At2#+U6y!,tM M+"	Key 23: !9~=TX9@s1"*T5x +sL~L*&&	Key 24: 8}<SW8?r0!)S4w~*rK}K)%%

Table 3. Brute Force Attack on the Encrypted Data File

The algorithm performs a brute-force attack by trying all possible key values from 1 to 500 for each line of the file. It then prints the decrypted messages obtained with each key for each line; the actual ciphertext is also provided in this representation. A cipher-text-only attack, frequency analysis, finds character frequency distribution patterns in the ciphertext. Language features like letter frequencies are used to create frequency tables. Decryption attempts are done by mapping ciphertext components to likely plaintext equivalents based on statistical analysis. Iterative refining of decryption techniques enhances accuracy. Successful decryption shows the plaintext without knowledge of the encryption key or technique.

A cryptographic attack is commonly used to decode encrypted data by exploiting the weaknesses of the algorithm, as compared to a brute force attack, which is a hit-and-trial approach. Cryptanalysis uses a range of methodologies, from statistical approaches to

mathematical analysis, in an effort to identify vulnerabilities in algorithms that can pose a threat to the protected data. Security experts and cryptanalysts work together to improve cryptographic systems and make them more resilient to the ever-changing world of cyberattacks. The ethical aspects of cryptanalysis must be emphasized since any unsanctioned attempts might have negative legal repercussions and compromise the security of encrypted communication networks.

The stability of the proposed hybrid cryptographic algorithm is tested in a cloud environment with an attack-through-cryptanalysis approach based on the amount of information known to the attacker. Frequency analysis, which belongs to the class of cipher- text-only assaults, is the main cryptanalysis approach employed in this case. In this kind of attack, the analyst doesn't have access to the encryption technique or keys; instead, they only have the encrypted ciphertext, which they then attempt to analyze to find patterns or information about the plaintext. Encrypted text is passed through the standard Cryptanalysis algorithm, and some random letters are generated in the form of frequencies, i.e., how many times a particular number or letter appears in the encrypted text, but the scenario was that the data/text we encrypted did not appear in the result generated by the Cryptanalysis algorithm, as shown in Table 4. During the whole process, we can say that the design is far better at providing security to the data.

Algorithm:

1. *User Input:*
   Use a `Scanner` to get user input for the Base64-encoded ciphertext.
2. *Base64 Decoding:*
   Decode the Base64 input to obtain the binary representation of the ciphertext. Convert the binary data to a string using UTF-8 encoding.
3. *Perform Frequency Analysis:*
   Initialize an array `frequency` of size 128 (assuming ASCII characters) to store the frequency of each character.
   Iterate through each character in the decoded text:
   Check if the character is within the printable ASCII range (32 to 126). If yes, increment the corresponding frequency in the array.
   Iterate through the printable ASCII range (32 to 126):
   Print the characters and their frequencies if they occurred at least once.
4. *Print Frequency Analysis Results:*
   Print the frequency analysis results, showing the occurrences of each printable ASCII character in the decoded text.

The method outputs the instances of readable ASCII characters after analyzing the character frequency in the decoded text. In cryptanalysis, this type of frequency analysis is frequently employed to obtain information about the possible plaintext structure. It is assumed that some characters or combinations of characters occur more frequently than others in many natural languages.

Original Text	Programming

Encrypted Text	wkxvLOy8e6Q5jua5R+0faA==
CRYPTANALYSIS RESULTS:	
CHARACTER	OCCURRENCES
','	1
'9'	1
'G'	1
'L'	1
'h'	1
'0'	1

'{'	1

Table 4. Cryptanalysis on the Encrypted Data

Additionally, a text file with the extension ".txt" was saved with the encrypted data using the Hybrid Cloud Algorithm and was saved in the same directory where the Cryptanalysis algorithm in the Java Eclipse IDE was saved, i.e., with the relative path. After that, the file was passed through the cryptanalysis algorithm so that it could extract the data and original data can be accessed. But due to the complexity of the Hybrid algorithm, neither the original data gets accessed nor the file gets compromised. Cryptanalysis was implemented in the Java Eclipse, and results were calculated as shown in Table 5.

6.2. Algorithm:

1. *User Input:*
   Use a `Scanner` to get user input for the filename (including extension) for cryptanalysis.
2. *Open File Stream:*
   Open an `InputStream` for the specified file.
   Check if the file exists. If not, print an error message and exit.
3. *Read File Content:*
   Read all bytes from the file stream into a byte array (`encodedBytes`). Convert the byte array to a string (`encodedText`) using UTF-8 encoding.
4. *Base64 Decoding:*
   Decode the Base64-encoded text (`encodedText`) to obtain the binary representation of the ciphertext.
   Convert the binary data to a string using UTF-8 encoding to get the plaintext (`decodedText`).
5. *Perform Frequency Analysis:*
   Initialize an array `frequency` of size 128 (assuming ASCII characters) to store the frequency of each character.
   Iterate through each character in the decoded text:
   Check if the character is within the printable ASCII range (32 to 126). If yes, increment the corresponding frequency in the array.
   Iterate through the printable ASCII range (32 to 126):
   Print the characters and their frequencies if they occurred at least once.
6. *Print Frequency Analysis Results:*

   Print the frequency analysis results, showing the occurrences of each printable ASCII character in the d ecoded text.

File Name	Test.txt
Original Text	Computer Program
Encrypted Text	2w5PFw0QoD9pImjOGpbOvMD1+ivungLM9lXh/J00AMM=
CRYPTANALYSIS RESULTS:	
CHARACTER	OCCURRENCES
""	1
'+'	1
'4'	1
'?'	1
'O'	1
'U'	1
'h'	1
'i'	1

Table 5. Cryptanalysis on Data File

The method outputs the instances of readable ASCII characters after analyzing the character frequency in the decoded text from a given file. In cryptanalysis, this kind of frequency analysis might be helpful in determining the possible structure of the plaintext.

6.2. Testing Proposed Algorithm On Data In Transit

The proposed hybrid cryptographic algorithm is tested on data in transit across multiple nodes created in cloud setup. To assess a hybrid algorithm's efficacy, an experimental setup using Google Cloud Platform (GCP) with initially three virtual machines as nodes was set up. Node1 assisted with data transmission, node2 used SSH-in-browser instructions to launch a brute force assault on the transmitted file, and node3 encrypted the information. On repetitive testing, the proposed cryptographic algorithm potentially safeguard the content sent as a file or data blocks. The security triads, viz., availability, secrecy, and integrity, are achieved during the entire experimentation process. The storage buckets were created to store the data files, then the proposed algorithm encrypts the file data in the bucket. The algorithm's strong security protections were demonstrated by its ability to withstand brute force assaults and prevent unwanted access during cryptanalysis attack.

Assuming two virtual machines named `vm-1` and `vm-2`. Here are the steps using `gcloud compute scp` to transfer a file from `vm-1` to `vm-2`:

1. Open a Terminal on Your Local Machine:
   Open a terminal on your local machine.
2. Use `gcloud compute scp` Command:
   Use the following command to copy the file from `vm-1` to `vm-2`:
3. Use the following command to copy the file from vm-1 to vm-2
   gcloud compute scp vm-1:/path/to/source/file.txt vm-2:/path/to/target/location/ -- zone=your-vm-zone
4. Replace:
   `/path/to/source/file.txt`: The path to the file on `vm-1`.

   `vm-2:/path/to/target/location/`: The target VM (`vm-2`) and the path where you want to save the file.
   `your-vm-zone`: The zone in which both VMs are located.
5. Authentication:
   If it's your first time using `gcloud compute scp`, the command will prompt you to authenticate. Follow t
   he instructions to complete the authentication.
6. Verify the Transfer:
   After the transfer is complete, SSH into `vm-2` and check if the file is present in the specified location.
   gcloud compute ssh vm-2 --zone=your-vm-zone ls /path/to/target/location/
   Replace `your-vm-zone` with the actual zone where your VMs are located.
7. Run vm-3 using the SSH Browser.
   Run the Brute Force Java Program store in the Cloud Bucket. Using the command javac Bruteforce.java java Bruteforce

The effective implementation of a brute force attack using the Hydra tool with a word list of users and passwords and without encryption on a self-designed login page hosted on Google Cloud Linux highlights serious security flaws. Possible vulnerability to eavesdropping and interception arises due to lack of encryption measures and results are shown in Table 6.

	Host	Login	Password	Status
1	127.0.0.1	Aaqib	123	Valid

2	127.0.0.1	Root	9797384	Valid
3	127.0.0.1	Qwerty	654321	Valid
4	127.0.0.1	Internet	58673	Valid
5	127.0.0.1	Solidstate	97974	Valid
6	127.0.0.1	firewall	123456	Valid

Table 6. Brute Force Attack

This result raises the possibility that the experiment's passwords did not adhere to strict security guidelines, highlighting the necessity of stricter password procedures that include frequent updates and complexity requirements. Furthermore, as gaining illegal access to systems is against moral principles and may have legal repercussions, the ethical implications of such actions cannot be emphasized. In order to resolve these problems, it is advised to put encryption into place, enforce strict password regulations, add rate limitation and account lockouts, and carry out routine security audits in order to proactively find and fix vulnerabilities. Deploying the proposed encryption algorithm in the similar setup improves the system's overall security while abiding by the norms and preserving the sensitive information. Results of attack after encrypting the data with proposed algorithm are shown in Table 7.

	Username	Password	Status	Details
1	Asdfrewq	123	Fail	Invalid Login
2	Root	9797384	Fail	Invalid Login
3	Qwerty	654321	Fail	Invalid Login
4	Internet	58673	Fail	Invalid Login
5	Solidstate	97974	Fail	Invalid Login
6	firewall	123456	Fail	Invalid Login

Table 7. Brute Force Attack on Database

The proposed algorithm is used in cloud environment where files and data stored in database were targeted for potential security threats through different attacks. Attacks were performed through hydra tool to exploit any vulnerability in FTP service and MySQL database service. It is observed that the data was kept confidential and no breach was made to the sensitive information. Thus, it achieves the integrity, confidentiality of the information while preserving the availability to other users. The results of attack are presented in table 8.

	Protocol	Host	Login	Password	Status	Details
1	FTP	127.0.0.1	anonymous	Password_123	Fail	Invalid Login
2	FTP	127.0.0.1	anonymous	Secret_pass	Fail	Invalid Login
3	FTP	127.0.0.1	anonymous	ftp_user_pass	Fail	Invalid Login
4	FTP	127.0.0.1	anonymous	Pass1234	Fail	Invalid Login
5	FTP	127.0.0.1	anonymous	ftp123	Fail	Invalid Login
6	FTP	127.0.0.1	anonymous	ftp_pass987	Fail	Invalid Login
7	FTP	127.0.0.1	anonymous	Welcome_pass	Fail	Invalid Login
8	FTP	127.0.0.1	anonymous	Computer123	Fail	Invalid Login

Table 8. Brute Force Attack on Ftp Service

Two hybrid cryptographic systems, AES+Diffie-Hellman and AES+RSA, were thoroughly analyzed. To assess how well they performed in terms of encryption and decryption durations for different key lengths, a great deal of testing was done. The goal of the experiment was to determine how these hybrid systems' efficiency was affected by variations in key length, a crucial aspect affecting cryptographic security. The experiment findings were methodically organized into tables and plotted as graphical representations. Notably, the results showed that AES+RSA performed better than AES+Diffie-Hellman, showing more computational efficiency. Comparing AES+RSA to AES+Diffie-Hellman, showed better performance as the key lengths grew, whereas the memory required of both has marginal difference. The unique properties of the RSA algorithm are responsible for the relative efficiency advantage of AES+RSA. The hybrid system's remarkable efficiency, in spite of the possible processing cost linked to RSA,

demonstrated the complementary abilities of AES and RSA for quick and safe cryptographic operations.

It is crucial to understand that choosing between AES+Diffie-Hellman and AES+RSA requires rigorous trade-off analysis. Even though Diffie-Hellman is a well-respected safe key exchange protocol, its computational requirements may have an impact on performance, particularly when key lengths grow. On the other hand, the effectiveness of AES+RSA, due to RSA's capabilities in digital signatures and secure key exchange, makes it a good option in situations where strong security is required but computational speed is critical.

A visual component to the performance evaluation is added by the thorough display of findings in table 9 and figure 5, which provide a clear and concise summary of the efficiency differences between AES+Diffie-Hellman and AES+RSA across different key lengths. Based on the unique security and performance needs of the system or application in question, these insights are a useful tool for decision-making, helping to pick the best hybrid cryptography solution.

	AES-128-bit	AES-192-bit	AES-256-bit	Diffie-Hellman-1024-bit	Diffie-Hellman-2048-bit	Diffie-Hellman-4096-bit	RSA-1024-bit	RSA-2048-bit	RSA-4096-bit
Encryption Time(ms)	3415.564	3707.447	3039.113	3210.713	13269.796	14816.241	4518.572	4041.959	3820.28
Decryption Time(ms)	144.094	184.736	123.568	146.986	299.272	297.341	362.733	326.362	292.7

Table 9. Comparisons of Cryptographic Algorithms based on key sizes

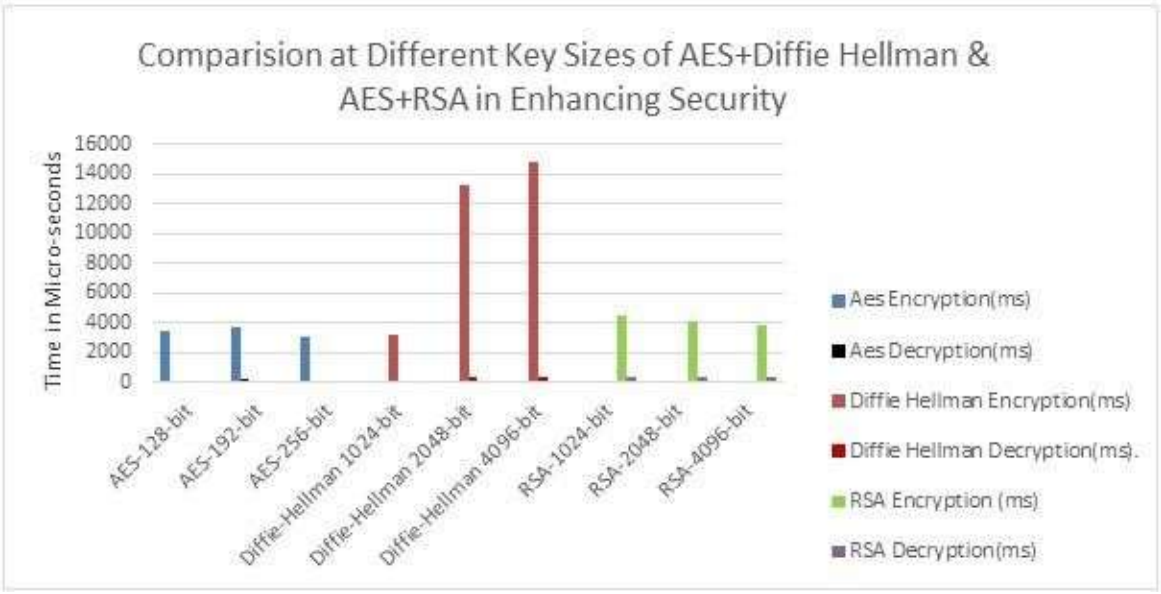

Figure 5. Comparisons based on different key sizes of AES, Diffie-Hellman & RSA

7. Conclusion

The present study highlights the advantages of hybrid encryption approach in preserving and securing the data. The combination of the Advanced Encryption Standard (AES) with Rivest- Shamir-Adleman (RSA) is implemented and validated with a known AES and Diffie-Hellman algorithms. The AES-RSA hybrid algorithm exhibits commendable performance in terms of encryption time, memory efficiency, and overall security when applied to text file encryption. It is concluded that as the data volume increases, proposed algorithm performs better in the selected performance measures like time taken and memory consumption for encryption. Furthermore, the study recognizes the robust defense mechanisms inherent in the hybrid approach against various cryptographic attacks. The utilization of a 256-bit AES key in conjunction with the asymmetric nature of RSA enhances resilience against brute force attacks and cryptanalysis. Even the availability of plan-text to the attacker for performing cryptanalysis attack,, the confidentiality of the information is preserved through proposed algorithm. The algorithm outperforms other algorithm even when deployed in cloud setup to secure both - the data at rest and data in transit. Results obtained on individual computer and in cloud setup for different nodes significantly showcased the strength of hybrid algorithm using AES and RSA. With its distinct encryption mechanisms, the hybrid algorithm resist attacks and safeguard data even if one component gets compromised. The resulting multi-layered security features position the AES-RSA hybrid algorithm as a compelling choice for scenarios requiring both high performance and robust protection against potential cryptographic threats.

8. Future Scope

Future potential for the hybrid encryption algorithm that combines Rivest-Shamir-Adleman (RSA) and Advanced Encryption Standard (AES) offers strong prospects for advancement in the field of cryptography. Fortifying the algorithm against quantum assaults is a crucial research area that will guarantee its robustness in the post-quantum computing age. Furthermore, the approach may be optimized to make use of parallel processing, which would improve performance on contemporary computer architectures, particularly in distributed and multi- core systems. Another interesting path is integration with blockchain technology, which provides improved security and privacy in decentralized networks. A critical area for future study is the development of adaptive key management systems that dynamically modify key sizes to handle emerging

security risks. Furthermore, the growing need for safe cloud-based data processing and storage is met by modifying the algorithm for scalability and resource- efficient operation in cloud computing settings. The adaptability of the algorithm depends on its cross-platform compatibility, which guarantees smooth interoperability across many computer systems, including mobile and Internet of Things devices. Last but not least, helping to standardize the hybrid algorithm can make it a widely accepted cryptographic standard, making it easier to integrate into other secure data transfer applications and systems. The hybrid AES-RSA encryption algorithm is ready to develop and fulfill the changing demands of secure communication in a world that is becoming more technologically sophisticated and networked by solving these issues.

9. Acknowledgement

The authors of this manuscript declare that they have no conflicts of interest related to the research presented in this paper. No financial or personal relationships with other people or organizations have inappropriately influenced this work.

There is no affiliations with or involvement in any organization or entity with any financial interest such as honoraria, educational grants, participation in speaker's bureaus, membership, employment, or other interest.

10. REFERENCES

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Special Publication 800-145, 2011.

[2] M. Joshi, K. P. Joshi and T. Finin, "Attribute Based Encryption for Secure Access to Cloud Based Encryption for Secure Access to Cloud Based EHR Systems, "IEEE CLOUD 11th International Conference, 2018, Vols. 2018-July, pp. 932-935, 2018.

[3] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in Proceedings of the 16th ACM conference on Computer and Communications Security, 2009.

[4] J. Smith and A. Brown, "Cloud Computing Advancements," Journal of Cloud Computing, vol. 12, no. 3, pp. 45-58, 2018.

[5] C. Johnson, "Security Challenges in Cloud Environments," International Journal of Information Security, vol. 7, no. 2, pp. 112-125, 2019.

[6] R. Jones, "Privacy Concerns in Cloud Computing," Journal of Cybersecurity Research, vol. 15, no. 4, pp. 321-335, 2017.

[7] M. White and L. Black, "Robust Encryption Algorithms in Cloud Computing," IEEE Transactions on Cloud Computing, vol. 9, no. 3, pp. 201-215, 2016.

[8] S. Gupta et al., "Enhancing Cloud Security with AES and RSA," Journal of Network Security, vol. 25, no. 6, pp. 45-52, 2020.

[9] E. Miller, "Balancing Cloud Computing Benefits with Security Measures," Journal of Information Assurance and Security, vol. 21, no. 1, pp. 78-92, 2021.

[10] X. Chen and Y. Wang, "Adaptable Security Measures in Hybrid Cloud Architectures," Journal of Cryptographic Engineering, vol. 14, no. 4, pp. 201-215, 2018.

[11] P. Black and Q. Green, "Innovative Hybrid Cloud Algorithm for Enhanced Data Security," International Conference on Cloud Security, pp. 35-42, 2019.

[12] S. Lee et al., "Comprehensive Evaluation of AES and RSA Integration in Hybrid Clouds," International Journal of Computer Security, vol. 32, no. 2, pp. 112-128, 2019.

[13] A. Smith and B. Brown, "Foundations for Secure Data Management in Hybrid Clouds," Journal of Cloud Computing Research, vol. 6, no. 3, pp. 145-158, 2022.

[14] R. Kumar and V. Dogra,"Cloud Computing: A comprehensive review," Journal of Computer and System Sciences, vol. 84, no. 2, pp-166-188, 2018.

[15] R. A. Oluoch and N. Masese, "A Review of Emerging Security Issues In Cloud Computing," International Journal of Computer Trends and Technology(IJCTT), vol. 67, no.9, pp.39-44, 2019.

[16] A. Khajeh-Hosseini, D. Greenwood, and J. W. Smith," The state of the art in cloud computing," Journal of Research and Practice in Information Technology, vol. 44, no. 1, pp. 1-10, 2012.
[17] V. Rijmen and J. Daemen, "The Design of Rijndael: AES-The Advanced Encryption Standard," Springer, 2002.
[18] M. Balachandra and V. S. Chakravarthy, "An efficient algorithm for RSA cryptography," International Journal of Computer Science and Information Security, vol. 11, no. 6, pp. 17-21, 2013.
[19] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches, "Wireless Communications and Mobile Computing, vol. 13,no. 18, pp. 1587-1611, 2013
[20] Y. Yuan, Y. Yang, L. Wu and X. Zhang, "A High Performance Encryption System Based on AES Algorithm with Novel Hardware Implementation," in 2018 IEEE International Conference, 2018.
[21] A. Adil Yazdeen, S. R. M. Zeebaree, M. Mohammed Sadeeq, S. F. Kak, O. M. . Ahmed, and R. R. Zebari, "FPGA Implementations for Data Encryption and Decryption via Concurrent and Parallel Computation: A Review", QAJ, vol. 1, no. 2, pp. 8–16, Mar. 2021.
[22] S. A. Pitchay, W. A. Alhiagem, F. Ridzuan and M. M. Saudi, "A Proposed System Concept on Enhancing the Encryption and Decryption Method for Cloud Computing," 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim), pp.201-205, 2015.
[22] S. O. Abood, "A Survey on Cryptography Algorithms," International Journal of Scientific and Research Publications, vol. 8, no. 7, 2018.
[23] S. A. Pitchay, W. A. Alhiagem, F. Ridzuan, and M. M. Saudi, "A Proposed System Concept on Enhancing the Encryption and Decryption Method for Cloud Computing," in Proceedings of the 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim), 2015, pp. 201-205.
[24] A. Albugmi, M. Alassafi, R. Walters, and G. Wills, "Data Security in Cloud Computing," doi:10.1109/FGCT.2016.7605062, 2016.
[25] Varma, V., Patil, M., Patil, S., Patil, M., & Kadam, A. (2022). Data Storage Security in cloud computing using AES algorithm and MD5 algorithm. International Journal for Research in Applied Science and Engineering Technology, 10(5), 5052–5055. https://doi.org/10.22214/ijraset.2022.43570.
[26] S. Kanimozhi, S. Krishna Adithya, A. Infant Franklin, S. Hari Prasanth, and M. Logeshwaran, "Advanced Encryption Standard to Prevent Intruders in Email through Cloud Environment," in 2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), pp. 1183- 1189, doi:10.1109/ICAAIC56838.2023.10140373, 2023.
[27] R. Parthasarathy, W. Haw, S. Seow, L. Rajamanickam, and A. Preethy, "Implementation of RSA Algorithm to Secure Data in Cloud Computing,", 2019.
[28] A. Kousalya and N. Baik, "Enhance cloud security and effectiveness using improved RSA-based RBAC with XACML technique," International Journal of Intelligent Networks, vol. 4, pp. 62-67, Mar. 2023. doi: 10.1016/j.ijin.2023.03.003.