Kurdistan Journal of Applied Research (KJAR)
Print-ISSN: 2411-7684 | Electronic-ISSN: 2411-7706

Original Article

Check for updates

A Hybrid Approach to Cloud Data Security Using ChaCha20 and ECDH for Secure Encryption and Key Exchange

Rebwar Khalid Muhammed [a*], **Zryan Najat Rashid** [b], **Shaida Jumaah Saydah** [c]

[a] Department of Network, Computer Science Institute, Sulaimani Polytechnic University, Sulaymaniyah, Iraq.
[b] Department of Computer Network, Technical College of Informatics, Sulaimani Polytechnic University, Sulaymaniyah, Iraq.
[c] Ministry of Education, Kirkuk Education Department of Kurdish Studies, Hawazen Preparatory School for Girls, Kirkuk, Iraq.

Submitted: 27 November2024
Revised: 7 February 2025
Accepted: 6 March 2025

*** Corresponding Author:**
rebwar.khalid@spu.edu.iq

Keywords: Cloud Computing, Data Security, ChaCha20 Encryption, Elliptic Curve Diffie-Hellman (ECDH), Key Transfer

How to cite this paper: R. K. Muhammed, Z. N. Rashid, S. J. Sadyah, "A Hybrid Approach to Cloud Data Security Using ChaCha20 and ECDH for Secure Encryption and Key Exchange", KJAR, vol. 10, no. 1, pp: 66-82, June 2025, doi: 10.24017/science.2025.1.5

Abstract: Cloud computing has transformed data storage and processing by offering on-demand resources and global accessibility. However, this convenience introduces significant security risks due to the reliance on third-party services, raising concerns about data confidentiality and integrity. This research proposes a hybrid encryption model that combines the high-speed ChaCha20 algorithm for data encryption with the Elliptic Curve Diffie-Hellman (ECDH) protocol for secure key exchange. The model ensures robust data protection in Cloud environments by generating a ChaCha20 key, encrypting it with ECDH, and securely storing encrypted key fragments in the cloud for later reassembly and decryption. This approach enhances security during data transmission and storage while mitigating the common vulnerabilities found in single-algorithm solutions. The study evaluates and compares the performance of ChaCha20 with ECDH against Rivest-Shamir-Adleman (RSA) with advanced encryption standard (AES) and Blowfish with Elliptic-Curve Cryptography (ECC). The results show that ChaCha20 with ECDH provides the fastest encryption time of 2ms and a key generation time of 15.8ms, with moderate memory usage. By contrast, RSA with AES is slower but offers consistent memory usage, while Blowfish with ECC balances speed and memory efficiency. The proposed hybrid model outperforms traditional encryption methods in both speed and security, making it suitable for modern cloud applications requiring scalability and high performance. Future research could focus on optimizing this model for resource-constrained environments, such as IoT and mobile.

1. Introduction

The internet service industry, encompassing areas like cloud computing, represents a rapidly evolving model for large-scale infrastructure [1]. Cloud computing has emerged as a transformative force across multiple industries, fundamentally changing how businesses manage and process data [2]. The cloud computing model and its distribution architecture are built upon the Internet. Its main goal is to store sensitive information quickly and securely. Cloud computing enables global access to a centralized collection of resources, including servers, storage, networks, services, and applications, via the Internet from any location in the world [3].

There are risks associated with cloud computing. Despite many organizations already storing sensitive information in the cloud, large companies are hesitant to migrate due to security concerns [4]. The rapid growth of sensitive data stored on cloud platforms has significantly heightened its

vulnerability to security breaches [5]. Therefore, the main issue with storage in cloud services is data security. To safeguard sensitive data, users should encrypt their data before sending it to cloud storage and implement access control mechanisms using cryptographic methods. There are various techniques available to secure data, with cryptographic algorithms being particularly effective [6]. Recently, numerous cloud computing encryption techniques are being researched in both industrial and academic fields. Additionally, securing files in the cloud and safeguarding private information is a crucial task. Preserving privacy is the method used to protect private information in the cloud. To achieve this, various security approaches are employed, including key generation, encryption, and decryption. Consequently, various privacy preservation methods have been utilized in existing research to protect sensitive data [7].

The most important technique for ensuring confidentiality and the accessibility of information for those who receive it while protecting it from attackers is the use of encryption. Cryptographic techniques additionally protect private messages from being accessed by the public or by third parties. The encryption and decryption procedures often involve users using a secret key to encrypt messages before transferring them over communication channels. Using the secret key, the sender decrypts the message that was sent. Cryptography provides a number of security techniques that protect data privacy and prevent the manipulation of information, in addition to other benefits. Nowadays, security procedures make wide use of cryptography's important security benefits [8]. In addition, algorithms for cryptography can be classified into primary categories. The key cryptography can be either symmetric or asymmetric.

Symmetric key cryptography is where both the sender and receiver of the message use the same key for data transmission encryption and decryption in the symmetric algorithm, also known as shared key cryptography. Data Encryption Standard (DES), Triple Data Encryption Standard, Advanced Encryption Standard (AES), Blowfish, and ChaCha20 are several examples of different symmetric algorithms [9- 11] .

Asymmetric key cryptography are where two keys are used in public key cryptography, commonly referred to as the asymmetric algorithm. These use the "private key" and the "public key". The sender uses the public key to encrypt the plaintext during the data transfer, creating ciphertext in the process. The receiver then decrypts the ciphertext using their private key. Common types of asymmetric algorithms include Rivest-Shamir-Adleman (RSA), Elliptic Curve Diffie-Hellman (ECDH), and Elliptic-Curve Cryptography (ECC) [9, 10, 12].

To develop a more effective and secure encryption system, hybrid cryptography combines both public key (asymmetric) and shared key (symmetric) cryptography approaches. This strategy develops new algorithms using the advantages of both approaches. A protocol that combines symmetric and asymmetric cryptographic algorithms, using each one's advantages to achieve the highest levels of safety and efficiency, is known as a hybrid cryptosystem. During the process, the public and private keys remain entirely securely, thus hybrid encryption is thought to be highly reliable. Hybrid cryptography is implemented by utilizing unique session keys in combination with symmetric encryption for secure data transfer. Public key encryption is used to encrypt a randomly generated symmetric key. Secure communication is then made available by the sender using their private key to decrypt the symmetric key. After recovering the symmetric key, it is used to decrypt the message [8, 13]. Hybrid cryptography combines symmetric and asymmetric cryptographic techniques, incorporating three keys. The goal of the three-key hybrid method is to enhance security over conventional two-key asymmetric or single-key symmetric techniques.

Data security is a significant concern as it can be compromised through various internal and external methods. To safeguard data transmission over the Internet, various encryption techniques are utilized. One such method is ChaCha20 encryption, which generates a key immediately after the input file is uploaded. Using the same key for both encryption and decryption, ChaCha20 uses symmetric key encryption due to this. If this key is compromised by a third party, they can easily decrypt and re-encrypt the file, making it challenging for the user to detect any unauthorized access. While ChaCha20 is considered to be a secure algorithm, if the single key becomes known, its security may be threatened. In contrast, ECDH utilizes asymmetric key encryption, involving two keys: a public key for encryption

and a private key for decryption. This dual-key approach provides a higher level of security, as it is more difficult for hackers to compromise both keys at the same time. Additionally, ECDH is recognized for delivering the same level of security as other algorithms while utilizing smaller key sizes.

A system that guarantees cloud data security, decreases computational costs, and decreases the time required for encryption and decoding must be developed. Our suggested model increases security and efficiency by utilizing the advantages of both the ChaCha20 and ECDH algorithms. A cloud data center's security is largely comparable to that of a conventional data center [8, 12]. It is essential to protect cloud computing from various threats. Based on our study, the following points highlight the key contributions:

- This study introduces an innovative approach to cloud data security by combining the ChaCha20 encryption algorithm with the ECDH key exchange protocol. This integration strengthens both encryption reliability and secure key exchange processes, effectively addressing the vulnerabilities present in traditional encryption methods.
- By utilizing ChaCha20, recognized for its high performance and robust security guarantees, this study significantly enhances encryption strength. The efficiency of ChaCha20 ensures that the data remains secure while reducing computational overhead. ECDH's capability to generate shared secrets over an insecure channel enhances the overall security of the encryption process, safeguarding against eavesdropping and other potential attacks.

The second section reviews related works, emphasizing recent studies and updated reference papers. The third section details the materials and methodologies used in this study, including an in-depth discussion of several algorithms. The fourth section introduces the proposed system, highlighting its components and functionality. Section five presents the results, showcasing the system's performance. Section six offers a comprehensive discussion, comparing this study's findings with prior research. Finally, the paper concludes by summarizing the key insights and implications derived from the results.

2. Related Works

Since sensitive data and applications must be protected and cyber-security threats are becoming more common, data security in cloud computing is a major concern. Numerous research studies have introduced innovative solutions aimed at improving data security within the cloud environment. Mahalle and Shahade [14] highlight that data security in cloud computing is a significant concern, prompting the exploration of various encryption techniques. Their study proposes a hybrid approach that combines homomorphic encryption with the Blowfish algorithm to enhance data security, thereby improving the confidentiality and integrity of information kept in the cloud. Zaineldeen *et al*. [15] propose a new hybrid encryption method that ensures safe communication among the user and the server by combining the AES and Enhanced Homomorphic Cryptosystem (EHC) algorithms. Almoysheer *et al*.[16] propose a hybrid model that integrates both symmetric and asymmetric cryptography techniques. This approach employs Blowfish encryption to secure cloud data while using ECC for generating and managing encryption keys. Anjana [17] present a research paper focused on secure cloud storage, utilizing RSA encryption, MD5 hashing, and Blowfish encryption techniques. Their proposed method enhances the data security of cloud computing by encrypting data before uploading and generating hash values to ensure data integrity. Deshpande e*t al.* [18] implemented a thorough examination of cloud security from a cryptography perspective, focusing on the application of different encryption algorithms such as AES, ECC, RSA, Secure Hash Algorithm (SHA), and the Diffie-Hellman key exchange. In cloud computing environments, asymmetric cloud encryption methods such as RSA are widely used to improve data security.

As a means of advancing data storage technologies and meeting the expanding needs for data storage, the study also looks at the convergence of edge and cloud computing. Muthulakshmi and Anithaashri [19] present a research paper that integrates the AES with code-based cryptography to efficiently encrypt data in cloud systems. AES is a widely recognized encryption algorithm known for its

speed and security, offering an additional layer of protection for sensitive information stored in the cloud. Chaloop and Abdullah [20] proposed utilizing hybrid encryption techniques for file encryption that combine RSA and AES. They introduced the fundamental concepts of both AES and RSA algorithms to ensure the protection of personal and enterprise data, while also evaluating their advantages and disadvantages. The study demonstrated that the hybrid encryption algorithm optimized data security, key management, and efficiency. Abualkas and Bhaskari [21] proposed a method that would encrypt the AES key using ECC, before using that key to both encrypt and decrypt the actual data. This approach adds another level of protection because ECC encryption protects the AES key. To further improve security, the system will also leverage key management strategies including rotation and splitting. Table 1 offers a detailed overview of the selected studies, emphasizing the cryptographic techniques relevant to cloud data security.

Table 1: Overview of selected studies on cryptographic techniques for cloud data security.

No	Cipher Algorithms	Description	Varieties of Data Employed	Year
1	Homographic and Blowfish [14]	The study proposes a hybrid homomorphic-Blowfish algorithm to enhance cloud data security	Cloud data	2019
2	AES and EHC [15]	The paper proposes a hybrid AES-EHC approach to improve cloud data transfer security and efficiency	Cloud data	2020
3	Blowfish and ECC [16]	The study introduces a Blowfish-ECC hybrid for secure and authenticated cloud SaaS.	Cloud data	2021
4	RSA, Blowfish and MD5 [17]	The proposed hybrid RSA, Blowfish, and MD5 encryption enhances cloud data security	Cloud Computing	2022
5	SHA, RSA, ECC, AES, Diffie-Hellman, Edge Computing [18]	The paper explores RSA, ECC, AES, and SHA to strengthen cloud security	Cloud data	2023
6	AES [19]	The study introduces McEliece-AES encryption for quantum-resistant cloud security	Cloud data	2024
7	AES, RSA [20]	The study proposes a hybrid AES-RSA encryption for fast and secure data transmission	Data	2024
8	ECC, AES [21]	The study proposes a hybrid ECC-AES encryption to enhance cloud storage security.	Online Data Storage	2024

With the growing adoption of cloud computing, organizations face several risks that must be addressed to protect sensitive data and maintain system integrity. Some privacy and security-related concerns deemed critical for cloud computing include [12]:

- **Malicious insiders**: An individual who is authorized to access an organization's network and data but uses such access in a way that compromises the integrity and confidentiality of the company's data and information systems is known as a malevolent insider. Many organizations recognize this threat due to its difficulty regarding detection and the significant impact it can have on the organization.
- **Account or service hijacking**: Software vulnerabilities and fraud combine to create this threat. In certain situations, an attacker can access cloud resources that are sensitive, giving them the ability to steal confidential data and passwords.
- **Hypervisor vulnerabilities**: A hypervisor is a critical component of virtualization. However, hypervisors are known to have significant security vulnerabilities, and available remedies are often limited and proprietary.
- **Insecure interfaces and application programming interfaces (APIs):** Organizations may be vulnerable to security risks such as password reuse, unauthorized access, the transmission of private data, clear text authentication, strict access control, and invalid authorizations if they employ poorly designed interfaces and APIs.
- **Cyber-attacks:** Network hacking and cyberattacks have grown in seriousness in recent years.

Table 2 evaluates each algorithm based on key size, security strength, performance, and typical use cases. The comparison highlights both well-established cryptographic techniques and newer algorithms from our system, such as ChaCha20 and ECDH, showcasing their suitability for various environments and applications.

Table 2: Comparison for benchmarking security levels of cryptographic techniques.

Algorithm	Key Size	Security Strength	Performance	Common Use Cases
AES [22]	128, 192, 256 bits	Strong; widely adopted	High efficiency in both hardware and software	Data encryption at rest and in transit
RSA [23]	2048, 3072, 4096 bits	Secure with large key sizes	Slower; used for small data blocks	Primarily for key exchange and digital signatures.
Blowfish [24]	32–448 bits	Moderate; outdated for modern needs	Fast but limited by 64-bit block size	Primarily for legacy systems.
ECC [25]	256, 384, 521 bits	High; small key sizes for strong security	Efficient, especially in resource-constrained devices	Secure communications and key exchange.
ChaCha20 (Our System) [11]	256 bits	Strong; secure against known attacks	Very fast in software, suitable for low-power devices	Best for lightweight and resource-constrained environments.
ECDH (Our System) [25]	256, 384, 521 bits	High; relies on ECC's strength	Efficient for secure key exchange	Used for securely exchanging keys over insecure channels.

3. Materials and Methods

The study involves employing a hybrid cryptographic approach combining ChaCha20 and ECDH, discussed in this section with detailed explanations of each algorithm in dedicated subsections. Following that, the methods for secure encryption, key exchange, and decryption are outlined. For further clarification, all methods are explained, demonstrating their integration in cloud security. Additionally, the encryption and decryption processes are illustrated using scenarios and practical programs addressing specific security challenges.

3.1. Advanced Encryption Standard

Since AES is a symmetric key cryptography method, it encrypts and decrypts data using the same cryptographic key. Data can be securely encrypted and decrypted using the same key according to this type of encryption. AES uses block cypher encryption, which encrypts data by permuting, modifying, and substituting. Each state of the data is composed of up of 128-bit pieces arranged in a matrix structure, with rows and columns acting as the keys. Every component in the matrix is referred to as a cell. Through its multiple variations, each of which uses a different number of rounds, AES provides varying security levels. AES encryption uses key sizes in sizes of 128, 192, and 256 bits [26].

3.2. Rivest-Shamir-Adleman

In 1977, Ronald Rivest, Adi Shamir, and Leonard Adelman introduced the RSA algorithm, the most widely used asymmetric key cryptosystem. This encryption and authentication system has been integral to various cryptographic applications, including email security, banking, e-commerce, and digital signatures on the Internet. The algorithm's security relies on the computational challenge of factoring large integers into their prime components. The RSA process involves three primary stages: key generation, encryption, and decryption [27].

3.3. Elliptic Curve Cryptography

A newer form of public key cryptography, ECC provides a higher security per bit than other cryptography techniques currently in use today. Elliptic curves are cubic curves in mathematics that are topologically equal to tori. Although they receive their name from the elliptic integral, they are not directly connected to the ellipse. The fundamental universal elliptic curve used in cryptography, known as the Weierstrass normal form, has the formula $y^2 = x^3 + ax + b$. Different values for a and b define

curves of this type. The visualization of the curve can be made to extend, compress, or pinch off to form two distinct parts by changing these parameters. In practice, curves used in cryptography are frequently designed using extremely large integer values for a and b [28]

3.4. Blowfish

Blowfish is a symmetric block cipher that utilizes a Feistel network consisting of 16 rounds of encryption and decryption operations. It processes data in 64-bit blocks and supports variable key lengths ranging from 1 to 448 bits. The cipher employs 18 32-bit subkeys, known as P-boxes, and four 8-bit S-boxes, each containing 256 entries, for substitution. Blowfish operates in two main stages: key expansion and data encryption. During key expansion, the key is transformed into multiple subkeys. In the data encryption phase, the plaintext undergoes 16 rounds of operations, including key-dependent permutations and substitutions influenced by both the key and input data. This design ensures a high level of security and efficiency [29]

3.5. ChaCha20 Algorithm

This algorithm makes use of 20 rounds, a 512-bit block size, and a 256-bit secret key. It is built in stream cypher methods. Each round consists of sixteen XOR operations, sixteen additions modulo 232, and sixteen rotation operations (ARX operations). During the data encryption and decryption operations, the state of the matrix $4x4$, which has sixteen entries computed by the QRF function, is used by the method [11].

Stream cypher methods are based on the generation of key stream elements using predetermined, dynamic states with initial values. The key stream elements combine with the plaintext parts during the encryption process to create the ciphertext or encrypted data. For a variety of reasons, a number of encryption algorithms that make use of stream cypher approaches have been proposed to guarantee data confidentiality [30]. Using stream cypher techniques, the ChaCha20 encryption algorithm was created to ensure data secrecy in a range of data security applications, such as smartphones, Google Chrome on smart devices, and security protocols like OpenSSL and OpenSSH [11]. Several ARX logical operations, such as addition, rotation, and XOR operations, are essential to the algorithm's architecture. These 32-bit word ARX procedures have been shown to be effective for both hardware and software implementations. For instance, the algorithm's 16 rounds comprise 16 XOR, 16 modulo addition (mod 2^{32}), and 16 rotation operations [31]. A 256-bit secret key, $K = (k_0, k1, k2, \ldots, k7)$, is required for the ChaCha20 cypher algorithm. At each stage, it processes data in 32-bit words. The state of the algorithm is arranged into a $4x4$ matrix of 16 elements, and there are 32-bit words for each element. The following parts are placed in rows: $X_0, X_1, X_2, X_3, X_4, X_5, X_6 X_7, X_8 X_9, X_{10}, X_{11}, X_{12} X_{13}, X_{14}, X_{15},$.

During encryption, the state-run matrix is initialized with predefined constant values (e.g. X_0=0x61707865, X_1=0x3320646e, X_2=0x79622d32, and X_3=0x6b206574), while other elements are filled with parts of the secret key ($X_4 = k_0, X_5 = k_1, X_6 = k_2, X_7 = k_3, \ldots . X_{11} = k_7$) Additionally, the matrix includes a 32-bit counter ($X_{12} = C_0$) and three 32-bit nonce values ($X_{13} = n_0, X_{14} = n_1, X_{15} = n_2$)[32] .

Four 32-bit words (a, b, c, and d) are put into the ChaCha20 cypher algorithm's Quarter Round Function (QRF). It performs a sequence of operations that modifies the input words, resulting in updated values for aaa, bbb, ccc, and ddd, each still consisting of 32 bits. The sequence of operations is outlined in Algorithm 1.

The keystream generation process is outlined in Algorithm 2. The current status matrix [Table 3] illustrates this. The state of the matrix X is first established using values from counter $C0$, nonce numbers ($n0, n1, n2$), secret key K, and unchanged values. Following 20 iterations of the Quarter Round Function (QRF), the 16 components of the state-run matrix X go through processing to create the Z Keystream. After this, the keystream is used to both encrypt and decrypt the input data [33].

Algorithm 1:

```
Quarter Round Function QRF (a, b, c, d)
{
   a = d XOR a
   d = a + b
   d = (d) << 16
   c = b XOR c
   b = c + d
   b = (b) << 12
   a = d XOR a
   d = a + b
   d = (d) << 8
   c = b XOR c
   b = c + d
   b = (b) << 7
   Return a, b, c, d
}
```

Algorithm 2: ChaCha20 Key Stream Procedure

```
Input: State matrix X (nonce values n0,n1,n2), secret key K, 32-bit counter C0
{
   Y1 ← X
   For(i = 1; i <= 10; i++) {
      // Column rounds
      QRF(X0, X4, X8, X12)
      QRF(X5, X9, X13, X1)
      QRF(X10, X14, X2, X6)
      QRF(X15, X3, X7, X11)

      // Diagonal rounds
      QRF(X0, X5, X10, X15)
      QRF(X1, X6, X11, X12)
      QRF(X2, X7, X8, X13)
      QRF(X3, X4, X9, X14)
   }
   Z ← X + Y1
   Return Z Key Stream
}
```

Table 3: Elements of (4x4) state matrix.

0x61707865	0x3320646e	0x79622d32	0x6b206574
k0	k1	K2	K3
k4	k5	k6	k7
C0	n0	n1	n2

3.6. Key Exchange Protocol Elliptic Curve Diffie Hellman

The Diffie-Hellman key exchange algorithm is a public-key cryptography approach that requires the completion of a number of complex mathematical operations. An algorithm's security is determined by the difficulty of computing the discrete logarithm problem modulo a prime number. There is a discrete logarithm problem when p is a prime number, y can be any integer, and q is a primitive root of p. It is computationally difficult to find x such that $y = qx\ (mod\ p)$. This data raises an issue. For example, Alice and Bob decide on a prime number n and an integer q that is a primitive root of n; n and q are not private. Rather, these are both known to the general public. In this algorithm, Alice generates a random integer x and sends the result of her calculation $X = q^x \mathrm{mod}\ n$ to Bob. Bob, in turn, generates a random integer y and sends his result $Y = q^x \mathrm{mod}\ n$ to Alice. Alice then computes $K=Y^x \bmod n$ $K = Y_X \mathrm{mod}\ n$, and Bob computes $K'=X^y \bmod n$. If the calculations are correct, K=K′, this indicates that both parties have successfully derived the same shared secret key for symmetric encryption.

The symmetric key K can be calculated more easily and computationally efficiently using the ECDH technique [34, 35]. Regarding the elliptic group in the addition operation of the elliptic curve

equation $y2 \equiv x3 + ax + b\ (mod\ p)$, Alice and Bob accepted a set of integers a and b prime numbers and a starting point $G(x, y)$. The basis point, $G(x, y)$, was chosen because the addition operation makes use of the elliptic group, and because the prime values a and b are the parameters that Alice and Bob accept.

Neither the sender nor the recipient creates the domain parameter since it is difficult to implement and takes a lot of work to determine the number of lines on a curve. In the equation $T(p, a, b, G, n, h)$, the elliptic curve parameter domain above Fp is defined by a and b, where p is the field in which the curve is specified. The number of points in a group elliptic $Ep\ (a, b)$ divided by n is the cofactor of the elliptic curve equation, or h. At generator point G are the group construction components. The smallest value positive integer is n $G = 0$, since n is the prime order of G.

Algorithm 3: ECDH Key Generation

Input: Domain parameters (p,a,b,G,n,h)
Output: Private key (x,y), Public keys (PA,PB)
{
1. Select integer x,y ∈ [1,n-1]
2. Employer A calculates PA = x.G, sends to B
3. Employer B calculates PB = y.G, sends to A
4. Employer A computes K = x.PB = x(y.G)
5. Employer B computes K' = y.PA = y(x.G)
}

The Diffie-Hellman technique is built on an elliptic curve and a known ECDH is used in the next step of the message encryption process. It solves the following issue: two parties (generally Alice and Bob) want to share information securely while making sure that a third party is unable to decrypt their messages. Algorithm 4 describes the encryption procedure used by ECDH.

Algorithm 4: ECDH Encryption

Input: Domain parameters, Private keys (x,y), Public keys (PA,PB), plaintext M
Output: Ciphertext C
{
1. Calculate S = y.PA = x.PB
2. Calculate C = M + S
}

The message that is encrypted is restored to its original form in this decryption procedure. According to the description in Algorithm 5, the decryption procedure entails subtracting the cypher point using the shared secret key.

Algorithm 5: ECDH Decryption

Input: Domain parameters (p,a,b,G,n,h), shared secret S, ciphertext C
Output: Plaintext M
{
1. Calculate M = C - S
2. Return plaintext M
}

3.7. Mitigating Risks in Key Splitting and Reassembly for ChaCha20 Encryption Systems

To mitigate the risks associated with the key splitting and reassembly process in the shown ChaCha20 encryption system, the following measures should be implemented:

1. **Encrypting Key Fragments**
   Each key fragment should be encrypted before transmission or storage. This adds an additional layer of security, ensuring that even if a fragment is intercepted, it remains unusable without the encryption key.

2. **Secure Transmission Channels**
   All communications involving key fragments should employ end-to-end encryption (e.g., TLS) to prevent interception during transmission. This ensures the confidentiality and integrity of the fragments.
3. **Randomized Fragmentation and Reassembly**
   Cryptographically secure randomness should be used during the fragmentation and reassembly process. This makes the process non-deterministic and resistant to prediction or reverse engineering by attackers.
4. **Redundancy Mechanisms**
   Introducing redundancy in the key fragmentation process ensures that the system is resilient to single-point failures. For instance, multiple copies of fragments could be created and securely distributed.
5. **Side-Channel Protections**
   Employ countermeasures such as constant-time operations during key splitting and reassembly to eliminate patterns observable through side-channel attacks (e.g., timing or power analysis).

Including such a comprehensive security analysis strengthens the overall robustness and credibility of the ChaCha20-based encryption system, ensuring that both key management and data integrity are maintained even under adversarial conditions.

3.8. Proposed System

The proposed system enhances the data security of cloud storage by implementing a robust encryption and key exchange mechanism. It leverages ECDH for secure key exchange and employs the ChaCha20 encryption algorithm to protect sensitive data during transmission and storage.

The diagram illustrates the workflow of securely exchanging cryptographic keys and encrypted files between two parties (Alice and Bob) via a cloud service. The process can be summarized as follows:

1. **Public Key Generation and Upload**:
   - Both Alice and Bob generate their unique public and private key pairs.
   - They upload their public keys to the cloud, enabling the secure exchange.
2. **Public Key Retrieval**:
   - Alice and Bob download each other's public keys from the cloud.
3. **Key Exchange and Shared Secret Generation**:
   - Using ECDH, each party combines their private key with the other's public key to derive a shared secret key, ensuring that the key is never transmitted over the network.
4. **Data Encryption**:
   - Alice encrypts a file using a randomly generated ChaCha20 key.
   - The ChaCha20 key is then encrypted with the shared secret key and uploaded to the cloud along with the encrypted file.
5. **Data Decryption**:
   - Bob retrieves the encrypted file and the encrypted ChaCha20 key from the cloud.
   - He decrypts the ChaCha20 key using the shared secret key.
   - Finally, Bob uses the decrypted ChaCha20 key to decrypt the file and access its original contents.

Key Features:

- **End-to-End Security**: ECDH ensures a secure key exchange, while ChaCha20 provides robust encryption for the data.
- **Cloud Integration**: The cloud serves as a medium for public key exchange and file storage, simplifying the process while maintaining security.

- **Performance and Scalability**: The system's encryption mechanisms are efficient, making it suitable for secure communication and data storage in various use cases, as shown in figure 1.

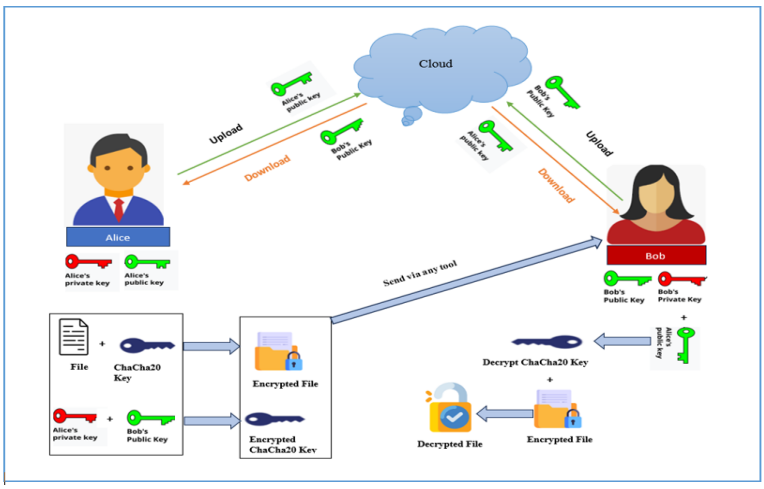

Figure 1: Cloud-based secure key exchange workflow.

A secure file transmission mechanism using the ChaCha20 symmetric encryption algorithm is integrated with the ECDH asymmetric key exchange between two parties, Alice and Bob. This process combines the efficiency of fast symmetric encryption with the security of public-private key cryptography, ensuring the secure transfer of files over a cloud-based system. The process, clearly illustrated in Figure 1, is explained through the following steps:

Step 1: Encryption and Upload Process

The process begins with the encryption of the original file using the ChaCha20 stream cipher. ChaCha20 is widely recognized for its speed and efficiency, making it suitable for environments that require resource conservation, such as cloud-based systems. The algorithm generates a symmetric encryption key used to encrypt the file, which ensures the confidentiality of the data during both transmission and storage.

Once encrypted, both the encrypted file and the ChaCha20 key are uploaded to the cloud. The encryption ensures that even if the cloud storage or transmission channels are compromised, unauthorized access to the file is prevented.

Step 2: Key Generation and Exchange

In parallel with the file encryption, Alice and Bob generate their respective public-private key pairs. This asymmetric key pair ensures secure key exchange, which is crucial for decrypting the ChaCha20 key later.

- **Alice's Role:** Alice generates her public and private keys, and shares her public key with Bob. Using her private key, she encrypts the ChaCha20 symmetric key, preparing it for secure transmission to Bob.
- **Bob's Role:** Similarly, Bob generates his key pair and shares his public key with Alice. He will later use his private key to decrypt the ChaCha20 key encrypted by Alice.

This exchange ensures that only Bob, with his private key, can decrypt the ChaCha20 key, even if the key is intercepted during transmission. This layer of security guarantees the integrity and confidentiality of the key exchange process, preventing unauthorized parties from gaining access to the ChaCha20 key.

Step 3: Decryption and File Retrieval

Once Bob downloads the encrypted file and the encrypted ChaCha20 key from the cloud, he uses his private key to decrypt the ChaCha20 key. After successfully decrypting the key, Bob applies the ChaCha20 algorithm to decrypt the original file.

This dual-layered encryption process—using ChaCha20 for the file and public-private keys for the key—ensures the file's confidentiality throughout its lifecycle, from encryption to cloud storage and eventual decryption. This hybrid approach, combining symmetric and asymmetric encryption techniques, provides both speed and security.

Step 4: Security and Efficiency in Cloud-Based Systems

ChaCha20's symmetric encryption algorithm provides a robust layer of security by encrypting the file, while its performance makes it ideal for cloud environments where scalability and efficiency are critical. The integration of asymmetric cryptography for key exchange adds another layer of security, ensuring that even if the ChaCha20 key is intercepted, it cannot be decrypted without the appropriate private key. This mutual authentication between Alice and Bob ensures that only the intended recipient can access the key, defending the integrity of the system against data interceptions and man-in-the-middle attacks. The technique achieves a compromise between security and performance by using public-private key pairs for key exchange and ChaCha20 for encryption. Cloud storage, enhanced by secure key transmission, ensures that files remain safe from unauthorized access both during transmission and while stored in the cloud.

3.9. Strategies to Ensure Resilience of the Proposed System Against Real-World Attacks

The proposed system incorporates robust mitigation strategies to address potential vulnerabilities and strengthen its security framework against common real-world attacks:

- **Brute Force Resistance**
  - Cryptographic key lengths adhere to the current standards for computational infeasibility.
  - The system utilizes 256-bit keys for both ChaCha20 and ECDH, ensuring resistance to exhaustive search attacks.
  - High-entropy key generation is emphasized to further enhance security.
- **Side-Channel Attack Mitigation**
  - The implementation employs secure coding practices, including constant-time algorithms and noise injection techniques.
  - Hardware-level protections are integrated to counter timing, power analysis, and electromagnetic side-channel attacks.
  - Regular implementation-level analyses are conducted to proactively identify and eliminate vulnerabilities.
- **Replay Attack Prevention**
  - Unique session identifiers, timestamps, and nonce values are used in the encryption and key exchange processes.
  - These mechanisms ensure that intercepted messages or keys cannot be reused, effectively thwarting replay attacks.
- **Comprehensive Security Testing**
  - The system undergoes rigorous security testing, including simulated attack scenarios and penetration testing.
  - These tests validate the system's ability to maintain integrity, confidentiality, and authentication, even under active adversarial conditions.
  - Continuous improvements are guided by the results of these evaluations.

By implementing these strategies, the proposed system demonstrates resilience against brute force, side-channel, and replay attacks. These measures align with cryptographic security best practices, ensuring the system's practical applicability and credibility in securing sensitive cloud data.

4. Results

The proposed encryption system's performance metrics are displayed in Table 4's results. It provides secure key exchange via the combination of the ECDH and ChaCha20 encryption algorithms. The evaluation is based on different file types, including PDF, JPG, MP3, MP4, and PPTX files of varying sizes.

Key generation time averages 15.8ms, with negligible variance across file types, indicating that the ECDH key exchange process is efficient and independent of file size. Encryption times are uniformly low, averaging 2ms across all file types, reflecting the efficiency of ChaCha20. However, decryption times show more variation, with larger files such as MP4 (79 MB) taking 6ms, and smaller files like PDF and PPTX require only 1ms.

Memory usage during encryption and decryption follows a similar trend, increasing with file size. The average memory used for encryption is 148.4 MB, while decryption averages 151.55 MB. This shows that the system remains efficient even for larger files, making it suitable for cloud storage encryption where both security and performance are critical.

The consistency in encryption time and moderate variation in decryption time and memory usage demonstrate that this combined approach balances strong encryption with computational efficiency, making it a robust choice for enhancing cloud data security.

Table 4: Performance metrics of ChaCha20 and ECDH encryption and decryption for various file types.

Types	Size	Key Generation Time(ms)	Encryption Time(ms)	Memory Used for Encrption(MB)	Decryption Time(ms)	Memory Used for Decryption (MB)
PDF File	367KB	16	2	105.93	1	108.66
JPG	9328 KB	15	2	133.52	3	136.25
MP3	7240KB	17	3	125.82	6	128.54
Mp4	79792KB	15	2	267.84	1	272.40
PPTX	1743KB	16	1	107.10	1	111.91
	Average	15.8	2	148.4	2.4	151.55

Figure 2 presents a visual comparison of encryption and decryption performance across various file types, including PPTX, MP4, MP3, JPG, and PDF, using the ChaCha20 and ECDH algorithms. Key performance metrics including key generation time, encryption time, decryption time, and memory usage for both encryption and decryption are displayed in the graph. Memory usage for encryption and decryption shows a significant increase with larger files, such as the MP4 file (79 MB), which required 267.84 MB and 272.4 MB, respectively. On the other hand, smaller files like PDF (367 KB) and PPTX (1.7 MB) used much less memory, around 105-111 MB. Decryption times vary more than encryption times, with larger files (e.g., MP3, MP4) requiring longer decryption times (up to 6ms for MP3). However, encryption times remain fairly consistent across all file types, averaging 2ms.

Overall, the graph highlights the efficiency of the encryption process while demonstrating a slight increase in decryption time and memory usage for larger files, reflecting the scalability of the system in handling different data sizes.

Table 5 highlights the performance metrics of RSA and AES encryption and decryption for different file types. The average key generation time across all files is 2532.8ms, with MP3 files taking the longest at 3334ms. Encryption times range from 63ms for PPTX files to 978ms for MP4 files, while decryption times span 78ms to 1088ms. Memory usage varies significantly, with MP4 files requiring the most for both encryption (75.26 MB) and decryption (77.92 MB). In contrast, PPTX files exhibit the lowest memory usage. The results demonstrate how file size and type impact performance, particularly in encryption.

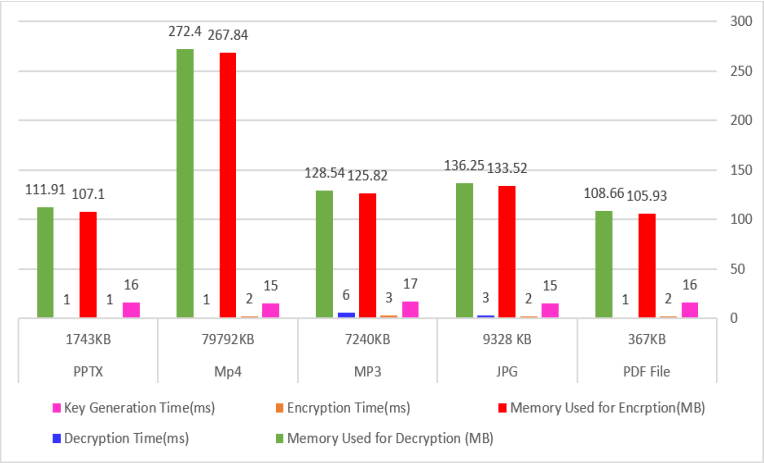

Figure 2: Graphical representation of encryption and decryption performance metrics for various file types using ChaCha20 and ECDH.

Table 5: Performance metrics of RSA and AES encryption and decryption for various file types

Types	Size	Key Generation Time(ms)	Encryption Time(ms)	Memory Used for Encrption(MB)	Decryption Time(ms)	Memory Used for Decryption (MB)
PDF File	367KB	2192	70	1.55	78	1.11
JPG	9328 KB	2212	142	6.52	173	9.11
MP3	7240KB	3334	125	4.41	139	7.07
Mp4	79792KB	2426	978	75.26	1088	77.92
PPTX	1743KB	2500	63	0.95	111	1.70
	Average	2532.8	275.6	17.74	317.8	19.38

Figure 3 illustrates the performance metrics for encryption and decryption processes across various file types, including PPTX, MP4, MP3, JPG, and PDF files. Key generation time, encryption time, decryption time, and memory usage for encryption and decryption are compared. MP4 files, with the largest size (79,792KB), require the highest decryption time (1,088ms) and significant memory for decryption (77.92MB). MP3 and JPG files demonstrate higher key generation times of 3,334ms and 2,212ms, respectively. In contrast, the smaller PDF file (367KB) has the lowest encryption time (70ms) and minimal memory requirements for encryption (1.55MB). PPTX files show moderate performance across all metrics. These results highlight the variations in computational overhead depending on file size and format, impacting encryption-decryption efficiency.

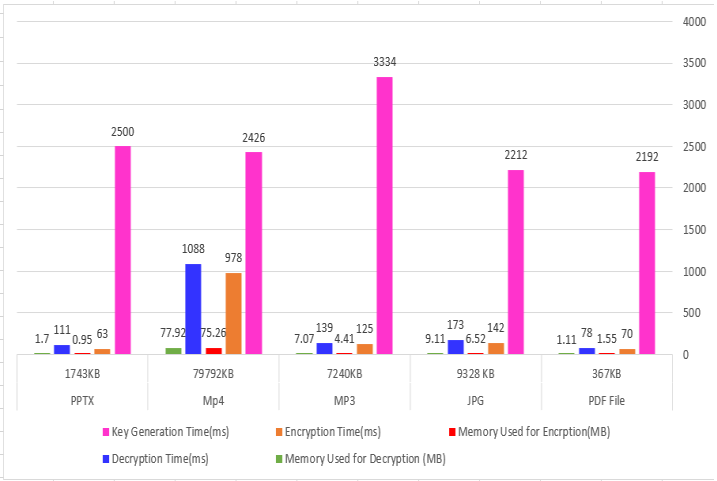

Figure 3: Graphical representation of encryption and decryption performance metrics for various file types using RSA and AES.

Table 6 presents the performance metrics of Blowfish and ECC encryption and decryption for various file types. Key generation times range from 354ms to 379ms, with an average of 366.6ms. Encryption time varies significantly, with the highest recorded for JPG files (127ms) and the lowest for PDF files (14ms). Memory usage for encryption and decryption remains consistent, averaging 74 MB and 87.4 MB, respectively. Decryption times follow a similar trend, averaging 66.4ms across all file types. Overall, the performance demonstrates efficient processing for both algorithms, making them suitable for different file formats.

Table 6: Performance metrics of Blowfish and ECC encryption and decryption for various file types.

Types	Size	Key Generation Time(ms)	Encryption Time(ms)	Memory Used for Encrption(MB)	Decryption Time(ms)	Memory Used for Decryption (MB)
PDF File	367KB	354	14	62	11	62
JPG	9328 KB	376	127	88	124	115
MP3	7240KB	356	110	82	94	103
Mp4	79792KB	379	74	74	75	88
PPTX	1743KB	368	29	64	28	69
	Average	366.6	70.8	74	66.4	87.4

The graphical representation in Figure 4 illustrates the encryption and decryption performance metrics for various file types using the Blowfish and ECC algorithms. Metrics such as key generation time, encryption time, decryption time, memory used for encryption, and memory used for decryption are compared across a range of file types (PPTX, MP4, MP3, JPG, and PDF). Each file's size significantly influences these metrics. Notably, JPG and MP4 files exhibit higher memory usage during encryption and decryption compared to smaller files like PDF. Blowfish and ECC's efficiency are reflected in the processing times and memory usage trends, highlighting their suitability for specific file types.

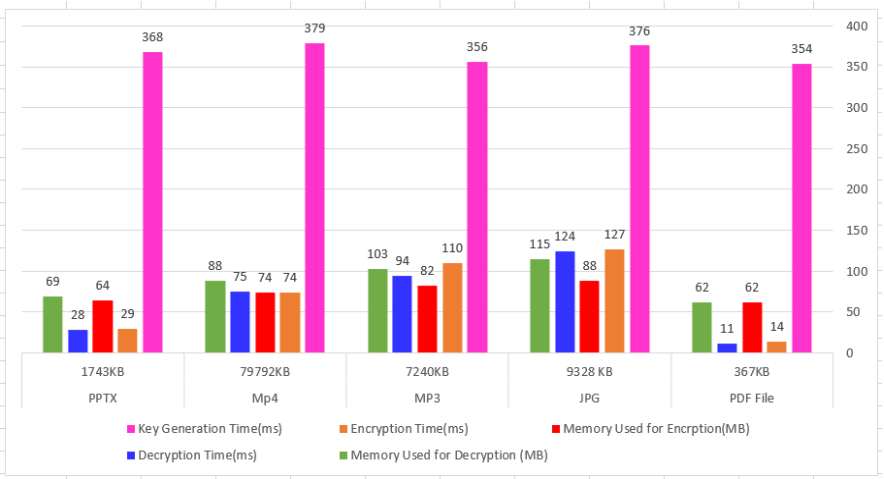

Figure 4: Graphical representation of encryption and decryption performance metrics for various file types using Blowfish and ECC.

5. Discussion

Previously, hybrid encryption methods such as Blowfish with ECC and RSA with AES were used [29, 36] . However, our proposed hybrid approach using ChaCha20 and ECDH demonstrates superior performance. A comparison of these hybrid encryption methods is presented in table 7. The ChaCha20 and ECDH method outperforms the others, achieving the fastest key generation time (15.8ms) and encryption time (2ms), with a moderate memory consumption for encryption (148.4 MB) and decryption (151.55 MB). In contrast, the RSA and AES method had the slowest performance, with key generation taking 2532.8ms and encryption 275.6ms, requiring minimal memory. The Blowfish and ECC approach offered a balanced performance, with a key generation time of 366.6ms and encryption time of 70.8ms.

These results highlight the suitability of different hybrid encryption methods for varying performance requirements in e-services.

Table 7: Performance comparison of hybrid encryption methods.

Encryption Method	Key Generation Time (Ms)	Encryption Time (Ms)	Memory Used for Encrption(MB)	Decryption Time(Ms)	Memory Used for Decryption (MB)
Hybrid Encryption ChaCha20 and ECDH (our system)	15.8	2	148.4	2.4	151.55
Hybrid Encryption RSA and AES	2532.8	275.6	17.74	317.8	19.38
Hybrid Encryption Blowfish and ECC	366.6	70.8	74	66.4	87.4

ChaCha20 offers exceptional efficiency in software, making it an excellent choice for devices lacking AES specific hardware, such as mobile and embedded systems. Despite being newer than AES, it has undergone rigorous cryptographic analysis and has no significant vulnerabilities, ensuring strong security. Its integration into widely used protocols like TLS 1.3 and adoption by industry leaders such as Google highlight its growing acceptance and reliability. While AES remains the benchmark for well-tested cryptographic standards, ChaCha20's efficiency and proven security make it a strong candidate for modern, resource-constrained, or mobile-focused systems**.** Additiona**lly,** using ECDH for key exchange introduces a computational overhead compared to symmetric-only methods. These costs become significant in resource-constrained or high-demand environments, potentially impacting scalability. To mitigate these challenges while retaining the security benefits of ECDH, the following strategies are proposed:

- **Hardware Acceleration:** Cryptographic accelerators or hardware security modules (HSMs) can offload ECDH computations from the CPU. These solutions enhance scalability by efficiently handling high volumes of cryptographic operations, and are increasingly accessible.
- **Optimized Algorithms:** Utilizing efficient elliptic curve implementations, such as Curve25519 or secp256r1, reduces the computational overhead. These curves offer faster key exchange operations without compromising security.
- **Ephemeral Key Caching:** Reusing ephemeral keys for short-lived sessions can reduce the frequency of ECDH computations. This approach is especially advantageous in scenarios like IoT, where connections are frequent but transient.
- **Hybrid Cryptographic Approaches:** Combining ECDH for initial key exchanges with symmetric encryption for subsequent communication balances scalability and security. Symmetric algorithms are computationally lighter, ensuring high performance during ongoing data exchanges.

These strategies can significantly improve the scalability of ECDH-based systems, making them viable for a wider range of applications without sacrificing security.

6. Conclusions

In the dynamic field of cloud computing, safeguarding data during transit and storage is paramount. A robust and efficient approach integrates ChaCha20 for encryption and ECDH for secure key exchange. ChaCha20 ensures high-speed encryption with strong security, while ECDH enables reliable key exchange over untrusted channels, ensuring confidentiality, integrity, and authenticity.

A comparative analysis highlights the exceptional efficiency of the ChaCha20 and ECDH hybrid system. It outperforms alternatives like Blowfish with ECC and RSA with AES in key performance metrics. ChaCha20 and ECDH achieve the fastest key generation time of 15.8ms, encryption time of 2ms, and decryption time of 2.4ms, with a minimal memory usage for encryption of 148.4 MB and decryption of 151.55 MB. In contrast, Blowfish with ECC has a key generation time of 366.6ms, an

encryption time of 70.8ms, and decryption time of 66.4ms, with a memory usage of 74 MB during encryption and 87.4 MB during decryption. In comparison, RSA with AES requires 2532.8ms for key generation, 275.6ms for encryption, and 317.8ms for decryption, with a memory consumption of 17.74 MB for encryption and 19.38 MB for decryption. While Blowfish with ECC performs better than RSA with AES in speed, it still requires significantly more time and memory compared to ChaCha20 and ECDH. These findings emphasize that the ChaCha20 and ECDH combination offers a highly optimized solution for cloud computing, balancing speed, memory efficiency, and robust encryption.

This hybrid system is ideal for modern applications requiring lightweight, high-performance cryptographic techniques. Future research could explore ways to further enhance scalability and adaptability, enabling this approach to address evolving security needs in diverse real-world scenarios. Although ChaCha20 combined with ECDH is known for its efficiency and security, challenges may arise when deploying this framework on resource-constrained devices such as IoT sensors or embedded systems. Limited computational power, restricted memory capacity, and energy consumption concerns can hinder performance. Optimization strategies, such as reducing the cryptographic overhead or adopting lightweight protocols, may be necessary to ensure smooth operation in these environments without compromising security.

Author contributions: **Rebwar Khalid Muhammed**: Investigation, Methodology and Project Administration. **Zryan Najat Rashid**: Formal analysis, Software, Writing – original draft, **Shaida Jumaah Saydah**: Supervision, Writing – review and editing, Validation.

Data availability: The data will be made available on request.

Conflicts of interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding: The authors did not receive support from any organization for the conducting of the study.

References

[1] Y. S. Abdulsalam and M. Hedabou, "Security and privacy in cloud computing: technical review," *Future Internet,* vol. 14, no. 11, pp. 1-27, Dec. 27, 2022. doi: 10.3390/fi14010011.

[2] S. Akter, K. Michael, M. R. Uddin, G. McCarthy, and M. Rahman, "Transforming business using digital innovations: the application of AI, blockchain, cloud and data analytics," *Annals of Operations Research,* vol. 308, no. 1–2, pp. 7–39, Jan. 2022, doi: 10.1007/s10479-020-03620-w.

[3] K. Sasikumar and S. Nagarajan, "Comprehensive review and analysis of cryptography techniques in cloud computing," *IEEE Access*, vol. 12, pp. 52325–52351, 2024, doi: 10.1109/ACCESS.2024.3385449.

[4] M. A. M.Sadeeq, S. R. M. Zeebaree, R. Qashi, S. H. Ahmed, and K. Jacksi, "Internet of things security: a survey," in *2018 International Conference on Advanced Science and Engineering (ICOASE)*, 2018, pp. 162–166. doi: 10.1109/ICOASE.2018.8548785.

[5] A. Orobosade, T. A. Favour-Bethy, A. B. Kayode, and A. J. Gabriel, "Cloud application security using hybrid encryption," *Communications on Applied Electronics*, vol. 7, no. 33, pp. 25–31, 2020, doi: 10.5120/cae2020652866.

[6] S. Srisakthi and A. P. Shanthi, "Design of a secure encryption model (sem) for cloud data storage using hadamard transforms," *Wireless Personal Communications*, vol. 100, no. 4, pp. 1727–1741, 2018, doi: 10.1007/s11277-018-5667-8.

[7] X.M. He, X. S. Wang, D. Li, and Y.-N. Hao, "Semi-Homogenous generalization: improving homogenous generalization for privacy preservation in cloud computing,"*Journal of Computer Science and Technology*, vol. 31, no. 6, pp. 1124–1135, 2016, doi: 10.1007/s11390-016-1687-6.

[8] K. F. Jasim, R. J. Ismail, A. A. N. Al-Rabeeah, and S. Solaimanzadeh, "Analysis the Structures of Some Symmetric Ci-pher Algorithms Suitable for the Security of IoT Devices," Cihan University-Erbil Scientific Journal, vol. 5, no. 2, pp. 13–19, 2021. https://doi.org/10.24086/cuesj.v5n2y2021.pp13-19

[9] R. K. Muhammed, K. H. Ali Faraj, J. F. G. Mohammed, T. N. Ahmad Al Attar, S. J. Saydah, and D. A. Rashid, "Automated performance analysis E-services by AES-based hybrid cryptosystems with RSA, ElGamal, and ECC," *Advances in Science, Technology and Engineering Systems Journal*, vol. 9, no. 3, pp. 84–91, Jul. 2024, doi: 10.25046/aj090308.

[10] M. Bhavitha, K. Rakshitha, and S. M. Rajagopal, "Performance evaluation of AES, DES, RSA, and paillier homomorphic for image security," in *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, IEEE, Apr. 2024, pp. 1–5. doi: 10.1109/I2CT61223.2024.10544282.

[11] Muhammed *et al.*, "Comparative analysis of AES, Blowfish, Twofish, Salsa20, and ChaCha20 for Image Encryption," *Kurdistan Journal of Applied Research*, vol. 9, no. 1, pp. 52–65, May 2024, doi: 10.24017/science.2024.1.5.

[12] P. Verma, J. Shekhar, and A. A. Preety, "A survey for performance analysis various cryptography techniques digital contents," *International Journal of Computer Science and Mobile Computing*, vol. 4, no. 1, pp. 522–531, 2015.

[13] A. Gour, S. Singh Malhi, G. Singh, and G. Kaur, "Hybrid cryptographic approach: for secure data communication using block cipher techniques," *E3S Web of Conferences*, vol. 556, p. 01048, Aug. 2024, doi: 10.1051/e3sconf/202455601048.
[14] V. S. Mahalle and A. K. Shahade, "Enhancing the data security in Cloud by implementing hybrid (Rsa & Aes) encryption algorithm," in 2014 IEEE International Conference on Power, Automation and Communication (INPAC), IEEE, Oct. 2014, pp. 146-149. doi: 10.1109/INPAC.2014.6981152.
[15] S. Zaineldeen and A. Ate, "Improved cloud data transfer security using hybrid encryption algorithm," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 20, no. 1, pp. 521–527, Oct. 2020, doi: 10.11591/ijeecs.v20.i1.pp521-527.
[16] N. Almoysheer, M. Humayun, and N. Z. Jhanjhi, "Enhancing cloud data security using multilevel encryption techniques.," *Turkish Online Journal of Qualitative Inquiry*, vol. 12, no. 3, 2021.
[17] D. A. S. Anjana, "Hybrid Cryptographic solution using RSA, Blowfish and MD5 for information security in cloud computing," *Mathematical Statistician and Engineering Applications*, vol. 71, no. 3s, pp. 1250–1268, 2022.
[18] A. G. Deshpande, C. Srinivasan, R. Raman, S. Rajarajan, and R. Adhvaryu, "Enhancing cloud security: a deep cryptographic analysis," in *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, 2023, pp. 1118–1123. doi: 10.1109/ICAICCIT60255.2023.10465863.
[19] R. M. Muthulakshmi and T. P Anithaashri, "A Robust Approach to Cloud data security using an amalgamation of aes and code-based cryptography," in *2024 International Conference on Science Technology Engineering and Management (ICSTEM)*, 2024, pp. 1–5. doi: 10.1109/ICSTEM61137.2024.10560532.
[20] S. G. Chaloop and M. Z. Abdullah, "Enhancing hybrid security approach using AES and RSA algorithms," *Journal of Engineering and Sustainable Development*, vol. 25, no. 4, pp. 58–66, Feb. 2022, doi: 10.31272/jeasd.25.4.6.
[21] Y. M. A. Abualkas and D. L. Bhaskari, "Hybrid approach to cloud storage security using ECC-AES encryption and key management techniques," *International Journal of Engineering Trends and Technology*, vol. 72, no. 4, pp. 92–100, Apr. 2024, doi: 10.14445/22315381/IJETT-V72I4P110.
[22] B. Sarkar, A. Saha, D. Dutta, G. De Sarkar, and K. Karmakar, "A Survey on the Advanced Encryption Standard (AES): A Pillar of Modern Cryptography," International Journal of Computer Science and Mobile Computing, vol. 13, no. 4, pp. 68–87, Apr. 2024, doi.org/10.47760/ijcsmc.2024.v13i04.008
[23] E. Ochoa-Jimenez, L. Rivera-Zamarripa, N. Cruz-Cortes, and F. Rodriguez-Henriquez, "Implementation of RSA signatures on GPU and CPU architectures," *IEEE Access*, vol. 8, pp. 9928–9941, 2020, doi: 10.1109/ACCESS.2019.2963826.
[24] M. Faheem, S. Jamel, A. Hassan, Z. A., N. Shafinaz, and M. Mat, "A Survey on the cryptographic encryption algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 11, 2017, doi: 10.14569/IJACSA.2017.081141.
[25] O. Popoola, M. A. Rodrigues, J. Marchang, A. Shenfield, A. Ikpehai, and J. Popoola, "An optimized hybrid encryption framework for smart home healthcare: ensuring data confidentiality and security," *Internet of Things*, vol. 27, p. 101314, Oct. 2024, doi: 10.1016/j.iot.2024.101314.
[26] Z. A. Mohammed, H. Q. Gheni, Z. J. Hussein, and A. K. M. Al-Qurabat, "Advancing cloud image security via aes algorithm enhancement techniques," *Engineering, Technology & Applied Science Research*, vol. 14, no. 1, pp. 12694–12701, Feb. 2024, doi: 10.48084/etasr.6601.
[27] S. F. Yousif, "Performance Comparison between RSA and El-Gamal algorithms for speech data encryption and decryption," *Diyala Journal of Engineering Sciences*, pp. 123–137, Mar. 2023, doi: 10.24237/djes.2023.16112.
[28] P. William, A. Choubey, G. S. Chhabra, R. Bhattacharya, K. Vengatesan, and S. Choubey, "Assessment of hybrid cryptographic algorithm for secure sharing of textual and pictorial content," in *2022 International Conference on Electronics and Renewable Systems (ICEARS)*, IEEE, Mar. 2022, pp. 918–922. doi: 10.1109/ICEARS53579.2022.9751932.
[29] P. Chinnasamy, S. Padmavathi, R. Swathy, and S. Rakesh, "Efficient data security using hybrid cryptography on cloud computing," 2021, pp. 537–547. doi: 10.1007/978-981-15-7345-3_46.
[30] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and Y. Papaefstathiou, "A survey of lightweight stream ciphers for embedded systems," *Security and Communication Networks*, vol. 9, no. 10, pp. 1226–1246, Jul. 2016, doi: 10.1002/sec.1399.
[31] M. Coutinho and T. C. Souza Neto, "Improved linear approximations to ARX ciphers and attacks against ChaCha," in *Advances in Cryptology – EUROCRYPT 2021*, A. Canteaut and F.-X. Standaert, Eds., Cham: Springer International Publishing, 2021, pp. 711–740. doi: doi.org/10.1007/978-3-030-77870-5_25.
[32] P. McLaren, W. J. Buchanan, G. Russell, and Z. Tan, "Deriving ChaCha20 key streams from targeted memory analysis," *Journal of Information Security and Applications*, vol. 48, p. 102372, 2019, doi: 10.1016/j.jisa.2019.102372.
[33] S. M. S. Reza, A. Ayob, M. M. Arifeen, N. Amin, M. H. M. Saad, and A. Hussain, "A lightweight security scheme for advanced metering infrastructures in smart grid," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 2, pp. 777–784, 2020, doi: 10.11591/eei.v9i2.2086.
[34] A. Saepulrohman and T. P. Negara, "Implementation of elliptic curve diffie-hellman (ECDH) for encoding messeges becomes a point on the gf (***pp***)," 2020.
[35] S. Madasu, P. Murugesan, H. V. Jaganathan, and S. Pamulaparthyvenkata, "Elliptic curve diffie-hellman based privacy-preserving deduplication for big data in cloud systems," in *2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS)*, IEEE, Aug. 2024, pp. 1–4. doi: 10.1109/IACIS61494.2024.10721723.
[36] A. P. Jagadeesan, K. Jain, and R. Aragona, "Performance comparison of hybrid encryption models," in *2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, IEEE, Aug. 2023, pp. 1196–1203. doi: 10.1109/ICAISS58487.2023.10250698.