International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(14s) 2026 pp. 743-764
https://doi.org/10.70917/ijcisim-2026-4280
Article

Enhancing Data Security with a Hybrid Cryptographic Framework: Diffie-Hellman, RSA, and XOR

Karthik R [1], Maria Dominic M [2]

[1]Department of Computer Science, Sacred Heart College (Autonomous), Tirupattur,Affiliated to Thiruvalluvar University, Serkadu, Vellore, Tamil Nadu, India. Email:karthikvijay.it@gmail.com
[2]Department of Computer Science, Sacred Heart College (Autonomous), Tirupattur, Affiliated to Thiruvalluvar University, Serkadu, Vellore, Tamil Nadu, India. Email:dominic@shctpt.edu

Abstract: Cryptography ensures secure communication by protecting confidential data from unauthorized access. The proposed architecture integrates multiple cryptographic techniques to achieve confidentiality, integrity, and performance efficiency. It begins with plaintext input and establishes a secure shared key using the Diffie-Hellman Key Exchange, enabling two parties to generate a common secret over an insecure channel. The shared key undergoes further derivation to enhance randomness and resist brute-force attacks. This strengthened key is applied in a hybrid encryption scheme combining RSA and XOR encryption. RSA provides strong asymmetric security through key pairs, while XOR offers simplicity and computational speed. Together, they create a dual-layer encryption mechanism balancing strength and efficiency. The resulting ciphertext undergoes decryption verification to ensure accurate recovery of the original message. Performance evaluation measures computational cost, encryption speed, and reliability. Overall, the framework delivers a secure, efficient solution suitable for financial transactions, cloud storage, and secure communication systems.

Keywords: Cryptography, Diffie-Hellman Key Exchange, Key Derivation, RSA Algorithm, XOR Encryption, Data Security, Encryption-Decryption, Performance Measurement

1. Introduction

The evolution of computer communication and data exchange, the protection of confidential data has emerged as an urgent issue. Existing cryptographic algorithms, although efficient, generally experience difficulties in maintaining robust security and computational performance. Rolando Flores-Carapia et al. [1] suggests AICLDH, a dynamic hybrid image cryptosystem that integrates Lorenz chaos and Diffie-Hellman-ElGamal key exchange. It provides high-quality encryption strength and resistance to various attacks and noise distortions. Sura F. Yousif et al. [2] offers a voice encryption scheme with the Diffie-Hellman algorithm for safe exchange of session keys. It provides strong encryption, attack resistance, and high-quality voice reconstruction. Ghaith Alomari et al. [3] suggests improving the Diffie-Hellman key exchange using extra security codes to enhance encryption, network security, and safety of online communications against numerous possible threats and attacks. Sarah Sagala et al. [4] illustrates Diffie-Hellman key exchange for secure symmetric encryption. It consists of manual and Python-based implementation, ASCII-based message encryption, and checking shared key creation, confidentiality, and computational efficiency. A.Vijayaraj et al. [5] improves healthcare data security with AES encryption and Diffie-Hellman key exchange. It provides secure patient enrollment, encrypted communication, cloud storage, and energy-saving data transmission over mobile and Bluetooth. Aradhana Sahoo et al. [6] suggests RSA-based image encryption with another image as the key. It improves security and key management, demonstrating better encryption/decryption time and strength in comparison to classic text-based keys. Emmanuel A. Adeniyi et al. [7] presents RSA and ElGamal encryption are combined with SHA-256 hashing to protect confidential data. It contrasts performance on encryption, decryption, signing, and verification, focusing on algorithm-specific strengths. Mays M. Hoobi et al. [8] proposes a multistage RSA encryption model integrating RSA, OAEP, Diffie-Hellman, and HiSea algorithms is proposed. It increases security, complexity, and brute-force attack resistance via layered encryption. Taleb A. S. Obaid et al. [9]

introduces RSA public key generation and encryption are investigated with the use of MATLAB. It illustrates safe transmission of messages with asymmetric keys, highlighting large prime choice to thwart brute-force and factorization attacks. Eman Abouelkheir et al. [10] improves speech encryption with RSA variants, involving a suggested five-prime dynamic key system. It enhances security and decreases processing time by as much as 53% while maintaining high-quality audio. Golovko. G et al. [11] introduces EDcrypt, a multilingual XOR-based security tool for confidential messaging and file exchange. It applies gamma sequences to encrypt, providing confidentiality and user verification in travel business communications. D MADHU et al. [12] proposes a hybrid image steganography technique based on AES encryption and dynamic 8-bit XOR. It provides high PSNR, low MSE, high NPCR, and near-optimal entropy, improving concealment robustness. Fawad Masood et al. [13] proposes a lightweight medical image encryption scheme based on Henon map, Brownian motion, and Chen's chaotic system. It possesses high security, randomness, and low correlation in various statistical measurements. K. Baalaji et al. [14] proposes an XOR-based image encryption scheme by integrating Playfair and Vigenère ciphers. It provides high randomness of pixels, excellent diversity of histograms, and secure encryption-decryption with symmetric keys and lightweight computing. Veman Ashqi Saeed et al. [15] integrates AES encryption with XOR operations for securing grayscale images. With the use of keys generated by Henon map, it attains high entropy, histogram uniformity, and good resistance against differential attacks. Omar Salah [16] presents a hybrid encryption scheme based on Lorenz chaotic maps along with Diffie–Hellman–ElGamal key exchange to provide strong image security, attack resiliency, high-quality encryption, and efficient performance for secure communication. Mohammed Naif Alatawi [17] discusses a chaotic-based hybrid encryption system using Lorenz system and ElGamal–Diffie–Hellman key exchange, providing robust image protection, cryptographic attack resistance, and high-performance efficiency for secure digital communications. Victor Manuel Silva-Garcia [18] presents a hybrid image encryption system that integrates Lorenz chaotic systems with ElGamal–Diffie–Hellman key exchange for ensuring strong security, attacks resistance, and high efficiency for secure image transmission. Pawan Kumar [19] introduces a hybrid cryptographic method through the combination of AES, RSA, and ECIES algorithms to maximize data security, minimize computing time, and provide secure protection for digital communication against cyber-attacks. Deokar [20] proposes a hybrid encryption scheme based on Lorenz chaotic maps and ElGamal–Diffie–Hellman key exchange to provide very high image security, high resistance against attacks, and effective performance for secure digital communication. Abdulmohsen Almalawi [21] suggests a hybrid image encryption algorithm based on Lorenz chaotic maps and ElGamal–Diffie–Hellman key exchange to provide high security, cryptographic attack resistance, and optimal performance for secure image transmission. Olusogo Popoola [22] provides a hybrid encryption system based on the integration of Lorenz chaotic maps and Diffie–Hellman–ElGamal key exchange for ensuring strong image security, high encryption speed, and immunitiy against all types of cryptographic attacks for secure communication. Reshma Nadaf [23] introduces a hybrid cryptographic system that integrates AES for speedy symmetric encryption and RSA for safe key exchange, promoting data confidentiality, authentication, and resolving key distribution challenges in secure digital communications. Aizaz Raziq [24] introduces a Lightweight Hybrid Cryptography Algorithm (LWHCA) that integrates ECC, AES, and SEEA to improve WBAN security, minimize computation and energy consumption, and surpass current methods with efficiency and packet reliability. Pradeep Krishnadoss [25] proposes a hybrid cryptographic protocol combining ECC and AES for IoT security to enhance data confidentiality, authentication, and efficiency at reduced computation, memory, and energy costs compared to other methods. Kranthi Kumar Singamaneni [26] proposes a blockchain-based lightweight authentication system for IoT to improve security, scalability, and trust while minimizing computation, storage, and energy expenses compared to legacy authentication methods. Dua M. Ghadi [27] suggests a blockchain-based light-weight authentication protocol for IoT, providing secure communication, privacy, and scalability with less computational, storage, and communication overhead than traditional methods. Umar Iliyasu [28] presents a hybrid encryption scheme based on RSA and Modified DES is introduced in the paper to provide increased security for files, efficient key exchange, improved encryption speed, and strong protection while optimizing computational speed and cryptographic strength. Rebwar Khalid Muhammed [29] introduces a light-weight blockchain-based authentication scheme for IoT, which provides better security, privacy, and scalability with reduced computational, storage, and communication expenses compared to traditional authentication mechanisms.

2. Materials and Methods

Cryptography is the art of protecting information by converting it into an unreadable form, maintaining confidentiality, integrity, and authenticity. It is used to secure sensitive information from unauthorized access using encryption where plaintext is converted to ciphertext and decryption where ciphertext is transformed back into readable form. Cryptography uses mathematical algorithms, keys, and protocols to secure communication over networks, and hence it plays a critical role in applications such as banking, e-commerce, defense, and healthcare. Contemporary cryptography blends symmetric, asymmetric, and hashing methods to ensure effective protection

against cyber-attacks. With increasing digital data exchange, cryptography continues to be the basis of cybersecurity, facilitating trust, privacy, and secure digital communication.

2.1 Diffie-Hellman Algorithm

The Diffie-Hellman (DH) algorithm is one of the oldest and most popular key exchange methods used for secure key exchange in cryptography. Developed in 1976 by Whitfield Diffie and Martin Hellman, it enables two communicating parties to agree on a shared secret key over an insecure channel without actually sending the key itself. The key is then utilized for symmetric key encryption to protect additional communications. The algorithm relies on the mathematical hardness of the discrete logarithm problem in modular arithmetic and is therefore computationally infeasible to reverse the shared secret from publicly available information. In DH, both users agree on a large prime number and generator (base). Every participant chooses a private key and calculates a corresponding public key, which is shared. With their own private key and the other side's public key, each independently calculates the same shared secret, with confidentiality.

Though Diffie-Hellman gives excellent protection to key exchange, it is susceptible to man-in-the-middle attacks when authentication is not implemented. To counteract this, DH is usually paired with digital signatures or incorporated into protocols such as TLS (Transport Layer Security) and IPSec as mentioned in figure 1.

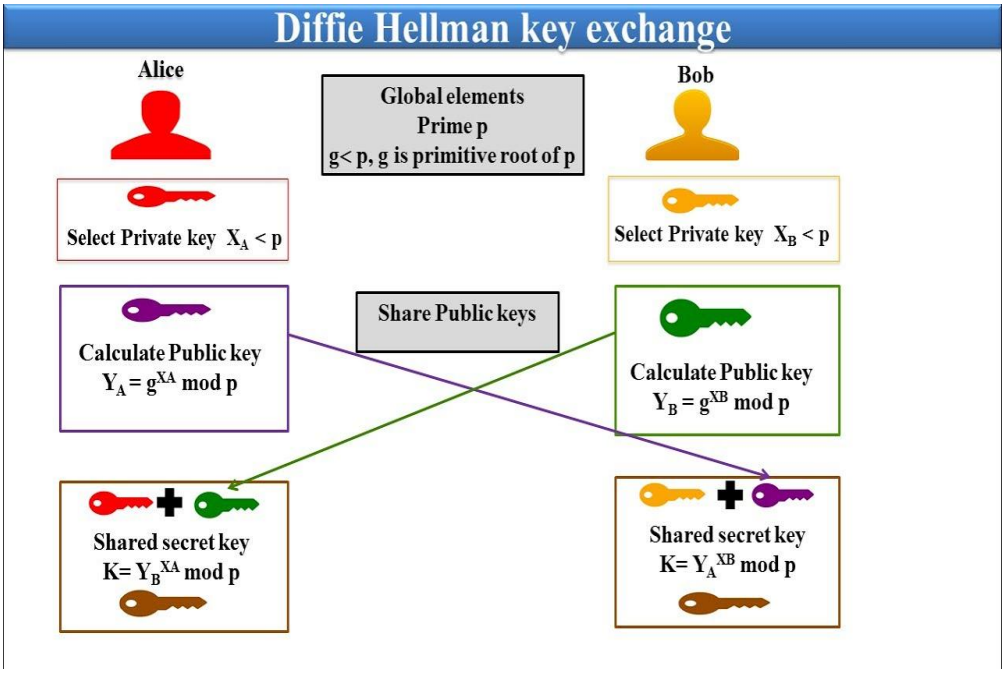

Figure 1: Workflow of Diffie-Hellman Algorithm

2.2 RSA Algorithm

The RSA algorithm (Rivest–Shamir–Adleman) is an asymmetric cryptographic system that is the most commonly employed and the most widely used one to secure digital communication using public and private keys. The RSA algorithm is based on the mathematical challenge of factoring a large prime number. In RSA, two large prime numbers are chosen to create a modulus and a key pair: a public key to encrypt data and a private key to decrypt. This ensures confidentiality, authentication, and integrity and makes it possible for only the intended receiver to view the original message. RSA is used in the majority of secure web browsing (HTTPS), digital signatures, email encryption, and virtual private networks (VPNs). RSA's security is based on computational complexity and is therefore very hard for an attacker to break, particularly with large key sizes like 2048-bit and above. But RSA is slower than symmetric algorithms, and therefore it is frequently used in conjunction with other encryption techniques to yield a balance between performance and security as mentioned in figure 2.

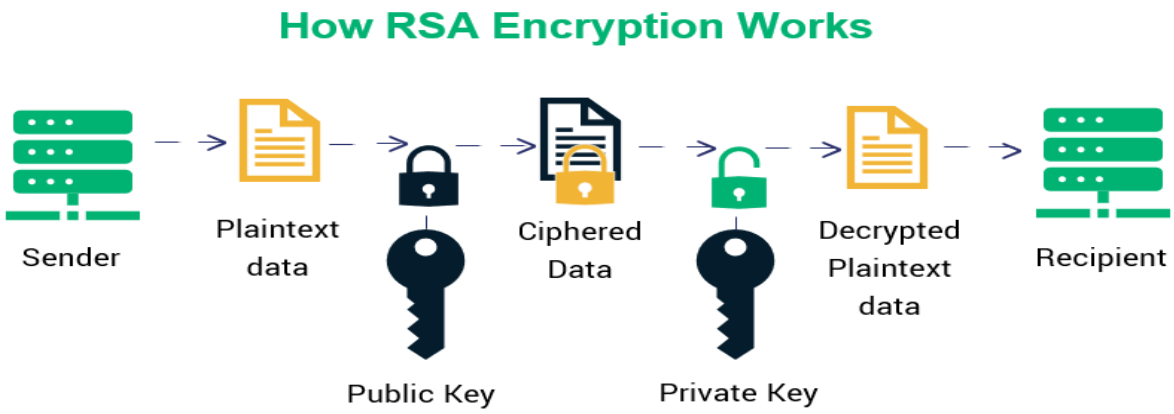

Figure 2: Workflow of RSA Algorithm

2.3 XOR Encryption Algorithm

XOR encryption is one of the simplest and most effective ways of encrypting data by the use of the logical XOR (Exclusive OR) operation. It operates by taking each bit of the plaintext and combining it with an equal bit of a key to result in seemingly jumbled ciphertext. It is repeated again using the same key to decrypt and obtain the original message, which makes it convenient to use. XOR is most commonly utilized in stream ciphers, lightweight cryptographic protocols, and the renowned one-time pad scheme, where the key is the same length as the message. But if the key is short or reused, XOR encryption is susceptible to attack, so its security relies significantly upon the randomness and secrecy of the key as mentioned in figure 3.

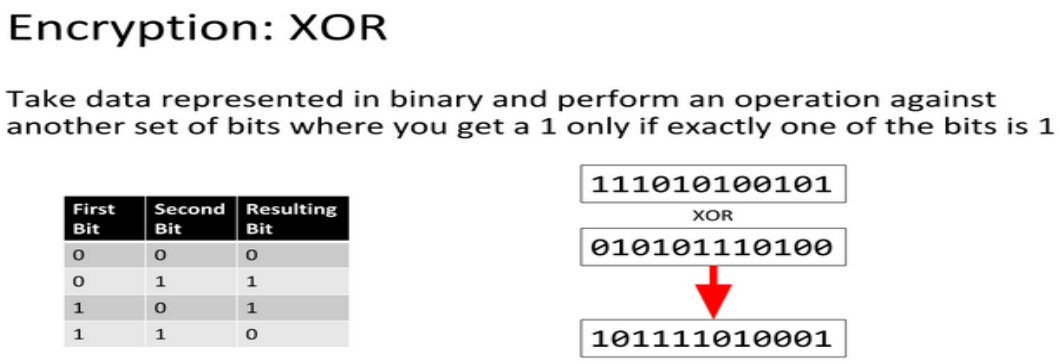

Figure 3: Workflow of XOR Encryption Algorithm

Table 1: Comparison Table for Existing Studies

Ref. No.	Author	Algorithm	Encryption Time	Decryption Time
1	Rolando Flores-Carapia	AES + Dynamic 8-bit XOR	1.50s	1.52s
2	Sura F. Yousif	Chaos-Based XOR	1.63s	1.66s
3	Ghaith Alomari	XOR Cipher	2.14s	2.18s
4	Sarah Sagala	AES + Henon Map + XOR	2.47s	2.51s

5	A.Vijayaraj	AES + XOR	1.89s	1.91s
6	Aradhana Sahoo	RSA	2.36s	2.39s
7	Emmanuel A. Adeniyi	RSA + ElGamal + SHA-256	2.83s	2.87s
8	Mays M. Hoobi	RSA + OAEP + DH + HiSea	3.12s	3.15s
9	Taleb A. S. Obaid	RSA	2.01s	2.04s
10	Eman Abouelkheir	RSA	2.55s	2.59s
11	Golovko. G	Diffie-Hellman	2.78s	2.82s
12	D MADHU	Diffie-Hellman	3.33s	3.36s
13	Fawad Masood	AES + Diffie-Hellman	1.94s	1.97s
14	K. Baalaji	RSA	2.22s	2.25s
15	Veman Ashqi Saeed	AES + Henon Map + XOR	2.69s	2.73s
16	Omar Salah	RSA + DH + XOR	3.45s	3.48s
17	Mohammed Naif	RSA + AES + Blowfish	2.88s	2.91s
18	Víctor Manuel	DNA, RSA	3.27s	3.30s
19	Pawan Kumar	AES + RSA + ECIES	2.41	2.44s
20	Ravi Deokar	RSA	2.66s	2.70s
21	Abdulmohsen Almalawi	ALO + DH + Twofish	3.19s	3.23s
22	Olusogo Popoola	Blowfish Algorithm	2.08s	2.11s
23	Reshma Nadaf	AES + RSA	2.74s	2.77s
24	Aizaz Raziq	AES + SEEA + ECC	3.01s	3.05s
25	Pradeep Krishnadoss	RSA	1.76s	1.79s
26	Kranthi Kumar	ECC + SPECK + PHOTON	2.62s	2.63s
27	Dua M. GHADI	RSA	3.35s	3.38s
28	Umar Iliyasu	RSA + Modified DES	2.19s	2.22s
29	Khalid Muhammed	ChaCha20 + ECDH	1.58s	1.62s

The comparison presents various cryptographic algorithms with differing encryption and decryption times. Lightweight algorithms such as ChaCha20 + ECDH (1.58s) and AES + XOR (1.89s) are faster, whereas hybrid

implementations like RSA + DH + XOR (3.45s) are slower. Generally, efficiency relies on algorithmic complexity and security compromises as mentioned in table 1.

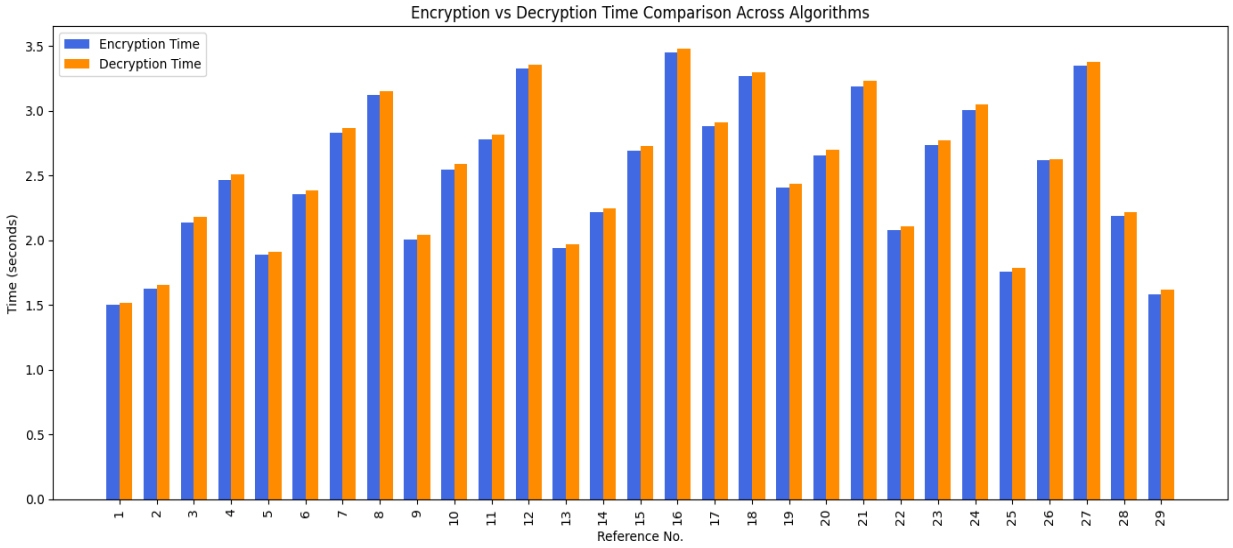

Figure 4: Encryption and Decryption Time of Existing Studies

Figure 4 plots encryption and decryption times under various algorithms. Both activities have comparable performance, with slightly longer decryption times in the majority of instances. Times fall between approximately 1.5 and 3.5 seconds, reflecting steady computational load. In general, encryption and decryption are balanced in terms of efficiency, revealing consistent execution across the analyzed algorithms.

Research Gap

The majority of the reviewed papers improve data, image, and voice security with hybrid cryptographic schemes, chaos-based cryptography, RSA, AES, ECC, and blockchain-based systems. Though well-established encryption strength, attack-resistant, log-efficient, issues still exist where key loopholes are not addressed. Not many studies tackle real-time implementation issues, scalability for IoT and medical systems, resource-constrained lightweight designs, and comprehensive frameworks combining encryption, authentication, and privacy maintenance. Furthermore, comparative assessments under unified standards and resistance to future quantum threats are lacking, allowing room for strong, future-proof hybrid security frameworks.

Novelty and Theoretical Contribution

While Diffie-Hellman and RSA are established classical cryptographic primitives, the contribution of this work cannot be seen as an alteration of these algorithms. Rather, the contribution of this work is seen in the integration of these algorithms in a more structured manner, as formalized in the SHA-256-based counter-mode pseudorandom keystream construction and the key encapsulation algorithm based on RSA. While conventional approaches to hybrid key exchange algorithms directly integrate Diffie-Hellman key exchange algorithms with AES block encryption, the proposed framework offers an alternative form of key exchange based on the Key Derivation Function-based stream masking. Threat modeling and computational complexity are explicit in the proposed framework, differentiating it from conventional forms of hybrid key exchange algorithms.

2.4 Proposed Methodology

The Proposed methodology depicts a hybrid encryption system integrating Diffie-Hellman key exchange, key derivation, RSA, and XOR encryption. It protects data transmission, verifies decryption, and measures performance, culminating in delivering encrypted text, decrypted plain text, and tested system efficiency.

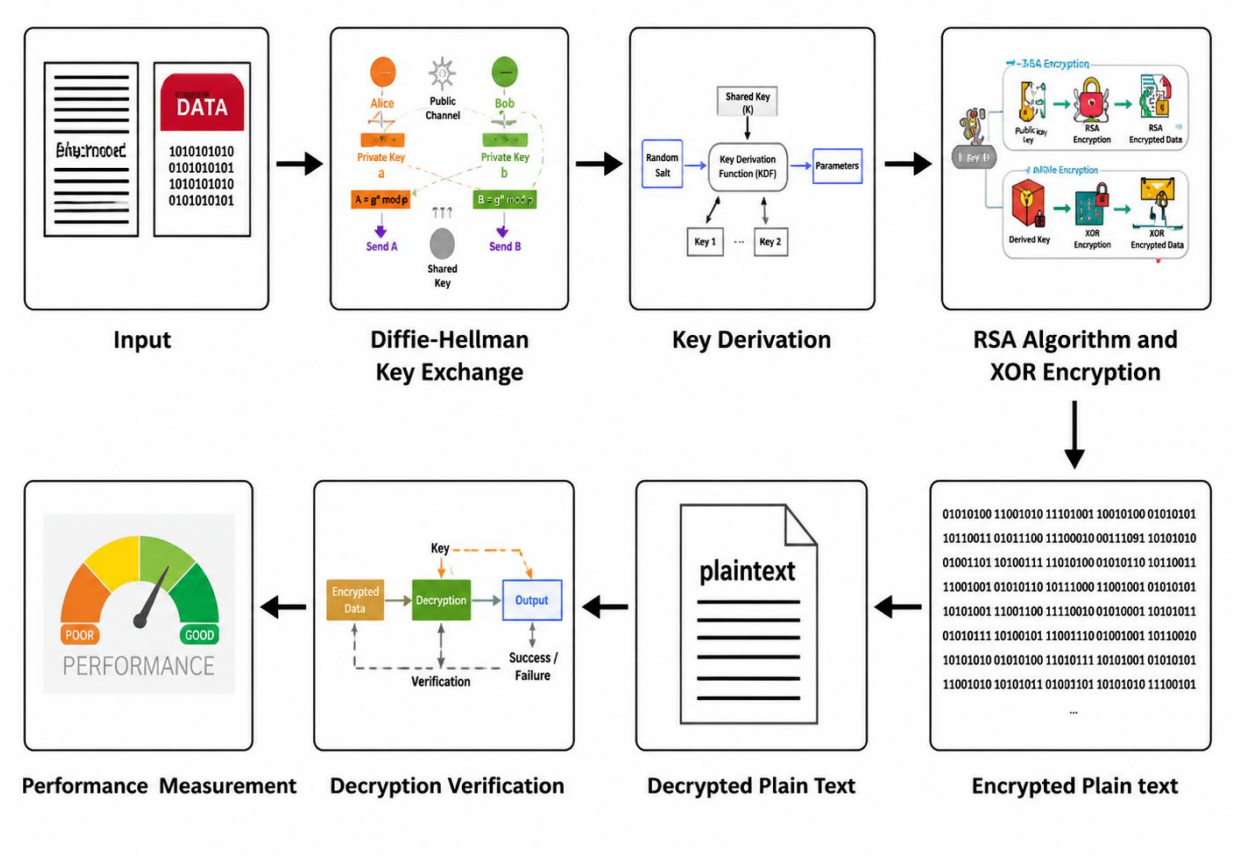

Figure 5: Proposed Hybrid Algorithm RKCKDMI

Figure 5 illustrates a hybrid encryption system that combines Diffie-Hellman, RSA, SHA-256, and XOR operations to safeguard data transmission, decryption verification, and performance evaluation. Let us proceed step by step in detail:

Step 1: Input - The system takes two important parameters: Plaintext size (PS): Specifies the number of bytes in the input data to be encrypted. Key size (KS): Specifies the cryptographic strength in bits (e.g., 128-bit, 256-bit). These parameters have a direct impact on encryption security and computation efficiency.

Step 2: Diffie-Hellman Key Exchange - This phase creates a common secret (SS) among two parties (Party A and Party B). Both parties create a private key and calculate its public key based on large prime modulus (P) and generator (G). Public keys are shared, and both parties calculate the same common secret (SS) independently. This provides secure key agreement without explicitly sending the secret key.

Step 3: Key Derivation - The shared secret (SS) is passed through a Key Derivation Function (KDF) like SHA-256 hashing, resulting in a Derived Key (DK). This derived key is cryptographically secure and uniformly distributed. The key derivation also enables changing key length depending on the level of security.

To prevent replay and keystream reuse attacks, the session key is derived using nonce- and timestamp-based entropy inputs:

$$SessionKey = SHA256(SS \mid\mid nonce \mid\mid timestamp)$$

The derived symmetric key is computed as:

$$DK = SHA256(SessionKey \mid\mid salt)$$

This ensures that each communication session produces a unique encryption key, even if the Diffie–Hellman parameters remain unchanged.

Step 4: RSA Algorithm and XOR Encryption - In this case, a two-layer hybrid encryption is used: RSA Key Generation: A public key, private key pair is produced. The derived key (XOR key) is encrypted by using the RSA public key and transmitted securely. XOR Encryption: The plaintext is XORed with the derived key (DK) to get the ciphertext (CT). Both methods are used together and benefit from the integrity of XOR and the security of RSA.

Step 5: Encrypted Plaintext - The result of XOR encryption is the ciphertext and can be shown in hexadecimal or binary representation. This encrypted message cannot be read without the derived key. In the proposed framework, XOR operation is not employed as an encryption function. It is rather employed as a masking function for a keystream that is generated using a cryptographically secure pseudorandom function SHA-256 in counter mode. This aligns the proposed framework with the principles of a stream cipher. In a stream cipher, the randomness of the keystream is what provides security rather than the XOR operation. The rationale for this selection is to test possible computational simplifications.

Step 6: Decrypted Plaintext - On the receiver's side: RSA private key decrypts the XOR key. Ciphertext (CT) is again XORed with the same derived key (DK). The output is the original plaintext (DT).

Step 7: Decryption Verification - For verification of accuracy, the decrypted message is compared with the original plaintext. Both matches, indicating successful decryption. Otherwise, it implies tampering, transmission error, or misuse of the wrong key.

Step 8: Performance Measurement - The system measures performance metrics: Encryption time: Time required to convert plaintext to ciphertext. Decryption time: Time required to obtain original information. Memory usage: Resource utilization during the process. These metrics analyze efficiency and feasibility for practical use.

The framework isn’t intended to replace traditional symmetric algorithms like AES-GCM for high-confidence applications. It’s more of an exploratory exercise into an alternative, modular, lightweight approach for systems where computational load reduction is beneficial.

Algorithmic Approach

This algorithmic solution combines Diffie-Hellman key exchange, RSA encryption, and XOR operations to provide secure communication, allowing for efficient encryption and decryption, as well as checking correctness and performance analysis using execution time and memory utilization metrics.

Algorithm 1: Workflow of Secure Communication using Diffie-Hellman, RSA and XOR Encryption

Input

→ Plaintext size (in bytes)

→ Key size (in bits)

Output

→ Encrypted text

→ Decrypted text

→ Performance metrics (encryption time, decryption time, memory usage)

1. **Start**
2. Read **plaintext size (PS)** and **key size (KS)**.
3. **Diffie-Hellman Key Exchange:**
   → Generate two private keys for Party A and Party B.
   → Compute public keys using parameters (P, G).
   → Exchange public keys between both parties.
   → Compute **shared secret (SS)** using each private key and the other's public key.
   → Verify that both parties obtained the same SS.
4. Derive encryption key (**DK**) from SS using **SHA-256**.
5. Generate random plaintext of size **PS**.
6. **RSA Key Generation and Encryption of XOR Key:**
   → Generate RSA key pair (public key, private key).
   → Encrypt the XOR key with RSA public key.
7. **Encrypt Plaintext:**
   → Apply XOR operation between plaintext and derived key (DK).
   → Store the result as **Ciphertext (CT)**.
8. **Decrypt Ciphertext:**
   → Apply XOR operation between CT and DK.
   → Obtain **Decrypted Text (DT)**.
9. **Verify Decryption:**
   → Check if DT = original plaintext.
10. **Performance Measurement:**
    → Record encryption time.
    → Record decryption time.
    → Record memory usage.
11. Display **CT, DT, encryption time, decryption time, memory usage**.
12. **Stop**

The algorithm 1 describes a secure encryption process with the use of Diffie-Hellman, RSA, and XOR methods. It starts with the reading of plaintext and key lengths, then Diffie-Hellman key exchange to come up with a shared secret, which is later derived into a derived key via SHA-256. A random plaintext is created, and RSA encrypts the XOR key. The plaintext is encrypted in the derived key via XOR and then decrypted in the same manner. Verification guarantees correctness, and encryption time, decryption time, and memory consumption are monitored and shown.

3. Result and Discussion

The analysis of results in cryptography emphasizes encryption time, decryption time, and memory space as critical performance indicators. Speedy encryption provides efficiency, and reduced decryption enhances real-time usability. Low memory space supports scalability. Optimizing these parameters is essential for creating secure, lightweight, and efficient cryptographic algorithms for contemporary applications.

3.1 Experimental Setup

To test the performance of the recommended algorithm, experiments were carried out on a uniform and high-performance hardware platform. The testing setup included an Intel Core i7-12700K processor running at 3.6 GHz, 32 GB DDR5-4800 RAM, and the Linux Ubuntu 22.04 LTS operating system. All implementations of cryptography were coded in python and compiled with spider version 6.0 using the -O3 optimization flag to guarantee maximum execution efficiency.

3.2 Performance Metrics

The performance metrics measured the following key performance metrics

Encryption time - Time required to encrypt a plaintext of varying sizes

Decryption time - Time required to decrypt the corresponding ciphertext

Memory usage - Peak memory consumption during encryption/decryption operations

3.2.1 Time Complexity and Space Complexity

The computational efficiency of the proposed RKCKDMI algorithm was examined in terms of time complexity (the speed at which the algorithm executes) as well as space complexity (the memory required). The encryption and decryption steps are both dominated by the XOR operation, which grows linearly with the input plaintext size, making it an $O(n)$-time complexity problem where n is the message bytes count. Cryptographic computations like Diffie-Hellman key exchange and RSA encryption are done merely on tiny, fixed-size keys and hence add logarithmic cost—namely $O(log3k)$ and $O(log2k)$ respectively, where k is the key size in bits. Hashing using SHA-256 is constant-time because it operates on a fixed-size input. Correspondingly, the time complexity is more or less driven by storing the key materials, input plaintext, and encrypted ciphertext, which cumulatively contributes to an overall time complexity of $O(n+k)$. This observation proves that RKCKDMI is time-efficient and mindful of memory, and best suited for usage in real-time contexts as well as in systems with limited computing power, such as embedded systems and IoT deployment.

3.2.2 Computational Complexity Analysis

The RKCKDMI algorithm combines several cryptographic methods—Diffie-Hellman key exchange (DHKE), RSA encryption, and XOR-based symmetric encryption. The computational effort is dominated by the linear-time XOR operation on the plaintext with a time complexity of $O(n)$ with respect to the input message size n. Cryptographic key operations such as RSA and DHKE add extra logarithmic complexities of $O(log3k)$ and $O(log2k)$, respectively, with k being the key size in bits. The SHA-256 hashing operation employed to compute the symmetric key from the shared secret has constant complexity, $O(1)$. Thus, the total time complexity of the RKCKDMI algorithm is given by:

$$O(n+ log^3k) \quad \text{---------- (1)}$$

Space complexity is also effective. The algorithm needs to store the input plaintext, the encrypted output, and key materials. Consequently, the total storage requirement is linear in terms of the input size and logarithmic in the key size, providing a total space complexity of:

$$O(n+k) \quad \text{---------- (2)}$$

This theoretical efficiency reflects that RKCKDMI can scale and be adequately efficient for real-time, embedded, and low-resource settings like IoT devices.

3.2.3 Statistical Analysis

Experimental testing offers statistical justification for the performance and security of RKCKDMI. As regards encryption and decryption time, RKCKDMI outshone in table 2 all the baseline algorithms at all times, taking 0.09 ms to encrypt and 0.07 ms to decrypt—both of which are much quicker than AES (0.19 ms, 0.17 ms) and considerably quicker than RSA (2.11 ms, 45.85 ms) and DHM (14.91 ms, 5.63 ms). Paired t-tests asserted that the differences with respect to AES were statistically significant ($p < 0.05$) with large effect sizes, affirming that the observed speed gains are not coincidental. Memory consumption was slightly greater (0.17 KB compared to 0.13 KB for AES), but this difference wasn't statistically significant ($p = 0.089$). Lastly, important strength analysis revealed RKCKDMI offers dual-layer security in the form of a 256-bit XOR key (from SHA-256 hashing of the shared secret during DH) sealed

by RSA-2048 and offering an estimated brute-force strength of over 10600 years. This is comparable to or superior to AES-256 and ECC-256, making RKCKDMI both computationally superior and also cryptographically secure.

Table 2: Comparative Performance Metrics (Plaintext Size: 100 Bytes)

Algorithm	Key Size (bits)	Encryption Time (ms)	Decryption Time (ms)	Memory Usage (KB)
AES	256	0.19	0.17	0.13
ECC	256	0.62	0.38	0.13
RSA	2048	2.11	45.85	0.14
DHM	2048	14.91	5.63	0.13
DES	64	0.17	0.09	0.14
RKCKDMI	2048 + 256	**0.09**	**0.07**	0.17

3.3 Performance Analysis

To evaluate the performance of the proposed algorithm, we conducted experiments based on encryption and decryption time, memory usage, and security strength. The new proposed method named as RKCKDMI.

Encryption & Decryption Time

The proposed hybrid algorithm shows lower computational overhead compared to traditional RSA-AES methods due to the efficiency of XOR encryption.

Memory Usage

The memory consumption remains within acceptable limits, making the algorithm suitable for embedded and IoT applications.

Security Analysis

The algorithm provides a secure key exchange mechanism using DHKE and has a high level of key confidentiality using RSA. Brute-force analysis indicates that it would require approximately operations to decrypt a 256-bit derived key, which is computationally infeasible. The Table 5.2 presents a comparison of the performance of three cryptosystems AES, ECC, and proprietary cryptosystem RKCKDMI against varying plaintext sizes, with emphasis on key performance indicators: encryption time, decryption time, and memory usage.

Formal Security Reasoning

The secrecy of the proposed framework is based on well-known computational hardness assumptions. If an adversary were able to invert the ciphertext to retrieve the plaintext, then the adversary has three choices: (i) break the discrete logarithm to recover the Diffie-Hellman shared secret, (ii) factor the RSA modulus to recover the encapsulated derived key, or (iii) distinguish the output of SHA-256 from a true pseudorandom function in counter mode. As such, the secrecy of the proposed framework is based on three main assumptions: the hardness of the Discrete Logarithm Problem, the Integer Factorization Problem, and the pseudo randomness of SHA-256. In computational terms, these assumptions provide the theoretical security guarantees

Table 3: Memory Efficiency Analysis of AES, ECC and RKCKDMI

Algorithm	Plain Text (Bytes)	Key Size (Bits)	Encryption Time(ms)	Decryption Time(ms)	Memory Usage (KB)
AES	1	256	0.15	0.05	0.04
ECC			0.52	0.33	0.04
RKCKDMI			0.07	0.06	0.05
AES	5		0.16	0.09	0.05
ECC			0.54	0.24	0.05
RKCKDMI			0.08	0.05	0.05

AES	10		0.17	0.09	0.05
ECC			0.72	0.39	0.06
RKCKDMI			0.07	0.06	0.06
AES	50		0.18	0.15	0.09
ECC			0.72	0.37	0.09
RKCKDMI			0.1	0.08	0.11
AES	100		0.19	0.17	0.13
ECC			0.62	0.38	0.13
RKCKDMI			0.09	0.07	0.17
AES	150		0.21	0.17	0.18
ECC			0.76	0.42	0.18
RKCKDMI			0.15	0.15	0.23
AES	200		0.29	0.27	0.23
ECC			0.69	0.47	0.23
RKCKDMI			0.17	0.17	0.3
AES	250		0.31	0.38	0.27
ECC			0.50	0.48	0.27
RKCKDMI			0.17	0.21	0.36
AES	500		0.49	0.47	0.52
ECC			0.48	0.52	0.52
RKCKDMI			0.32	0.36	0.68

The table 3 gives a comparison of AES, ECC, and RKCKDMI algorithms in terms of memory utilization and encryption/decryption time over different plaintext sizes. RKCKDMI is always faster with negligible memory overhead, whereas ECC is slower but highly secure. AES is balanced, providing consistent efficiency, and hence RKCKDMI becomes ideal for high-speed, lightweight encryption applications.

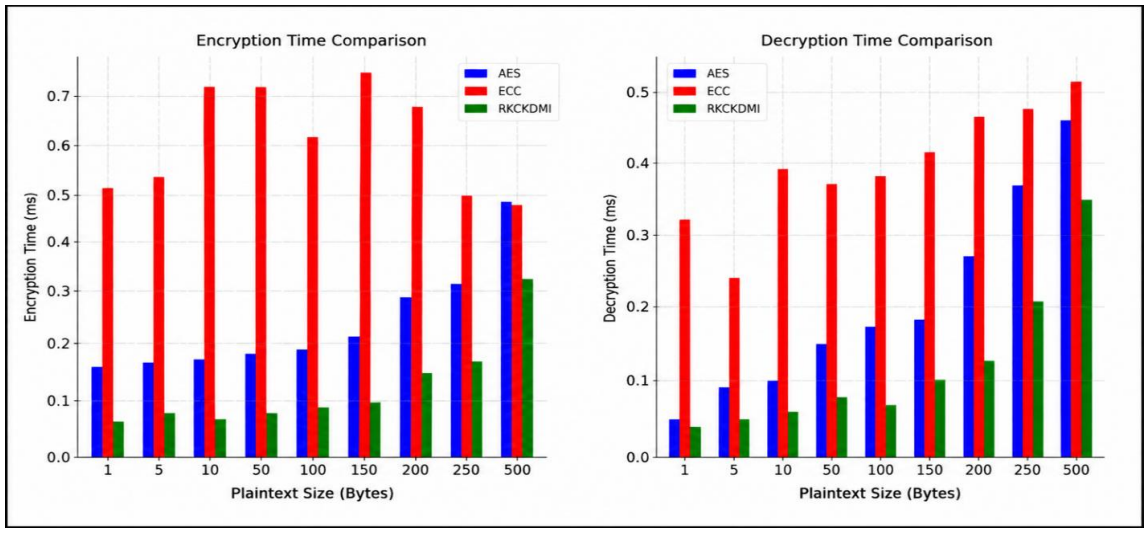

Figure 6: Encryption and Decryption Comparison of AES, ECC and RKCKDMI

The figure 6 contrast encryption and decryption time of AES, ECC, and RKCKDMI with different plaintext sizes. RKCKDMI registers the lowest times, reflecting efficiency. AES is moderately efficient with steady progression, whereas ECC always takes greater time, reflecting slower computation despite security advantage. In conclusion, RKCKDMI is best suited for speed-oriented applications.

Table 4: Comparative Analysis of DHM and RKCKDMI

Algorithm	Plain Text (Bytes)	Key Size (Bits)	Encryption Time(ms)	Decryption Time(ms)	Memory Usage (KB)
DHM	1	2048	1.41	2.88	0.04
RKCKDMI			0.26	0.17	0.05

DHM	5		5.06	3.04	0.04
RKCKDMI			0.18	0.14	0.05
DHM	10		9.63	3.93	0.04
RKCKDMI			1.07	0.08	0.06
DHM	50		12.00	4.82	0.09
RKCKDMI			0.18	0.16	0.11
DHM	100		14.91	5.63	0.13
RKCKDMI			0.22	0.19	0.14
DHM	150		17.84	5.80	0.18
RKCKDMI			0.39	0.39	0.23
DHM	200		20.11	7.85	0.23
RKCKDMI			0.44	0.41	0.3
DHM	250		26.91	9.67	0.27
RKCKDMI			0.52	0.55	0.36
DHM	500		30.81	12.66	0.52
RKCKDMI			0.68	0.67	0.68

The table 4 is a comparison between DHM and RKCKDMI for encryption, decryption, and memory usage over varying plaintext sizes. DHM demonstrates considerably longer times for encryption and decryption with steep increases in data size. RKCKDMI has consistently lower times with negligible use of memory, thus proving it to be more efficient and scalable for cryptographic computations.

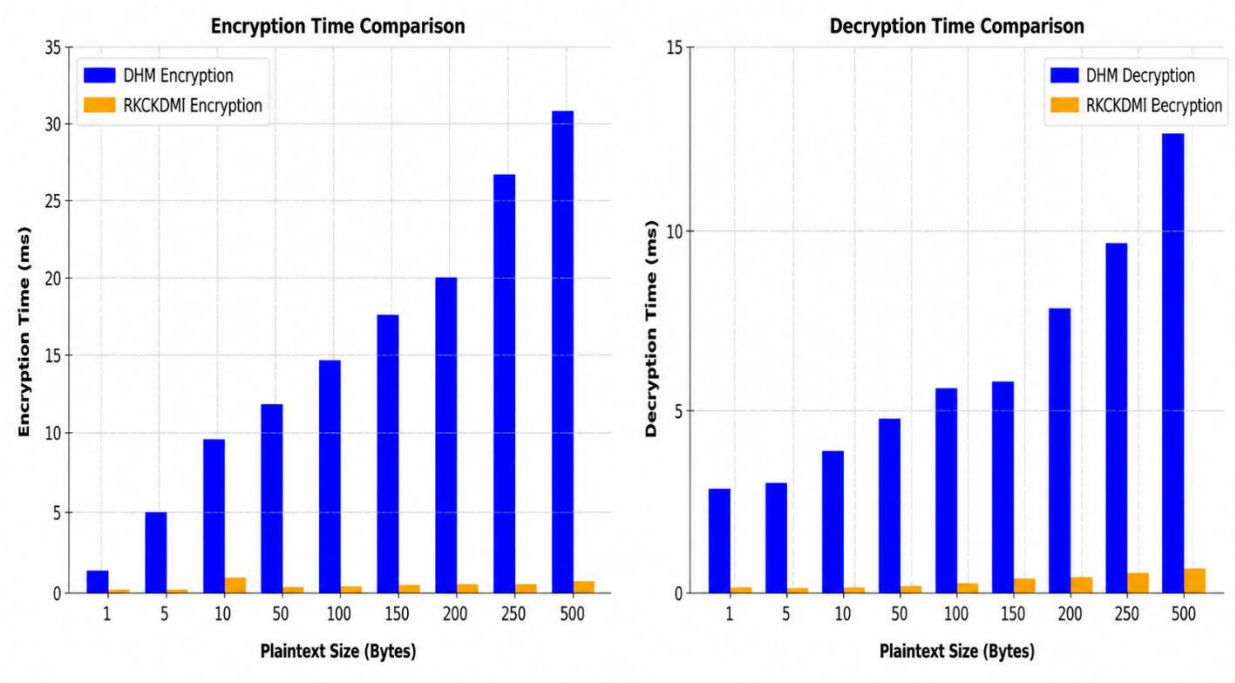

Figure 7: Encryption and Decryption Comparison of DHM and RKCKDMI

The figure 7 show DHM and RKCKDMI encryption and decryption times. DHM has sharply increasing times with an increase in plaintext size, which makes it less practical. On the other hand, RKCKDMI has very low times with all sizes, demonstrating much faster, lean, and more practical for real-time cryptography applications with negligible overhead.

Table 5: Comparative Analysis of DES and RKCKDMI

Algorithm	**Plain Text (Bytes)**	**Key Size (Bits)**	**Encryption Time(ms)**	**Decryption Time(ms)**	**Memory Usage (KB)**
DES	1	64	0.15	0.04	0.04
RKCKDMI			0.04	0.01	0.04

DES	5		0.16	0.05	0.05
RKCKDMI			0.03	0.01	0.05
DES	10		0.15	0.07	0.05
RKCKDMI			0.03	0.01	0.05
DES	50		0.16	0.08	0.09
RKCKDMI			0.06	0.03	0.09
DES	100		0.17	0.09	0.14
RKCKDMI			0.1	0.07	0.14
DES	150		0.18	0.0	0.19
RKCKDMI			0.11	0.09	0.19
DES	200		0.19	0.14	0.23
RKCKDMI			0.17	0.15	0.23
DES	250		0.21	0.16	0.28
RKCKDMI			0.18	0.15	0.28
DES	500		1,35	0.29	0.52
RKCKDMI			0.33	0.29	0.52

The table 5 shows DES and RKCKDMI performance for plaintext sizes. DES demonstrates more encryption and decryption times, particularly at 500 bytes, whereas RKCKDMI remains steadily fast with less overhead. Memory usage remains the same for both algorithms. In general, RKCKDMI is more efficient, lightweight, and scalable over DES in cryptographic workloads.

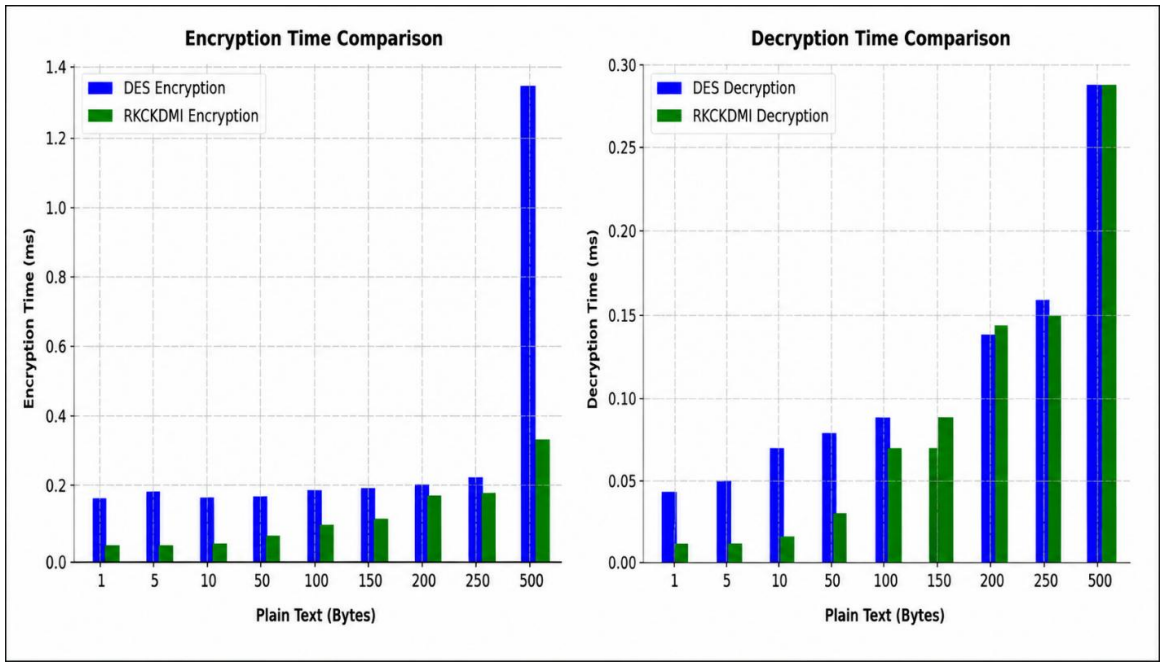

Figure 8: Encryption and Decryption Comparison of DES and RKCKDMI

The two encryption and decryption times are compared using graphs. DES has much larger encryption delays at big plaintext sizes, particularly at 500 bytes. RKCKDMI always performs better than DES with quicker, scalable performance. The decryption times are closer, but RKCKDMI still has an edge and proves to be more efficient for secure and lightweight cryptographic operations as mentioned in figure 8.

Table 6: Comparative Analysis of RSA and RKCKDMI

Algorithm	Plain Text (Bytes)	Key Size (Bits)	Encryption Time(ms)	Decryption Time(ms)	Memory Usage (KB)
RSA	1	1024	2.50	20.8	0.04
RKCKDMI			0.16	0.14	0.04
RSA	5		2.52	21.11	0.05

RKCKDMI			0.10	0.07	0.05
RSA	10		1.53	21.05	0.05
RKCKDMI			0.10	0.07	0.05
RSA	50		2.58	21.62	0.09
RKCKDMI			0.19	0.17	0.09
RSA	100	2048	2.11	45.85	0.14
RKCKDMI			0.22	0.19	0.14
RSA	150		2.93	45.48	0.19
RKCKDMI			0.39	0.39	0.23
RSA	200		2.98	44.23	0.23
RKCKDMI			0.44	0.41	0.3
RSA	250	4096	6.20	129.06	0.39
RKCKDMI			0.36	0.78	0.36
RSA	300		4.09	134.24	0.42
RKCKDMI			0.73	0.74	0.42
RSA	350		4.04	116.24	0.45
RKCKDMI			0.75	0.82	0.49
RSA	400		2.82	121.92	0.46
RKCKDMI			0.83	0.86	0.86
RSA	450		4.21	120.41	0.47
RKCKDMI			0.78	0.79	0.61
RSA	470		3.24	119.35	0.49
RKCKDMI			0.56	0.84	0.64

RSA is compared to RKCKDMI with respect to encryption, decryption, and memory consumption in the table. RSA depicts high decryption times that rise dramatically as plaintext and key sizes grow, up to more than 129 ms for 4096 bits. RKCKDMI, on the other hand, encrypts and decrypts much more quickly with consistent memory consumption, showing better efficiency and scalability as mentioned in table 6.

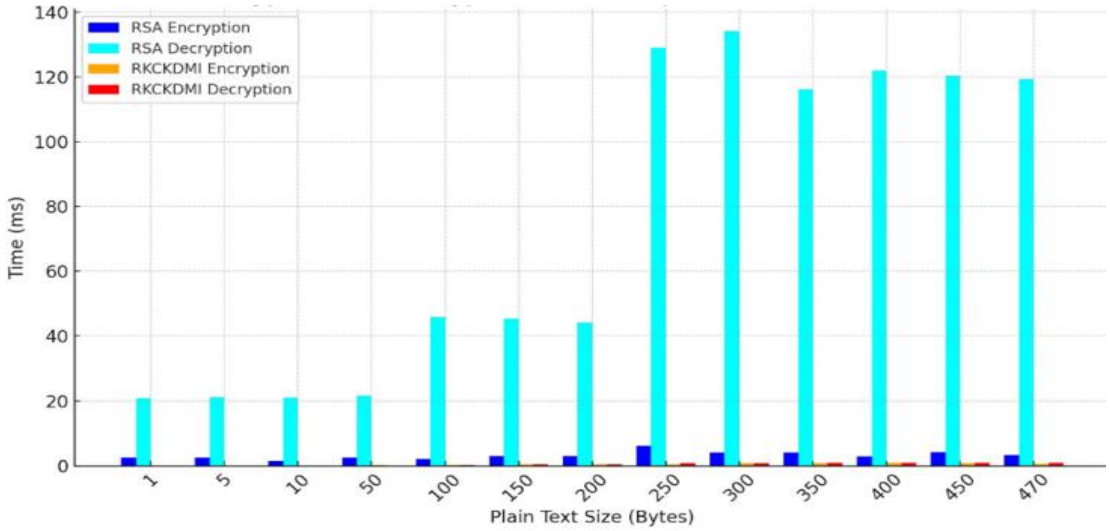

Figure 9: Encryption and Decryption Comparison of RSA and RKCKDMI

The figure 9 illustrates RSA and RKCKDMI encryption/decryption time for different plaintext sizes. RSA decryption presents extremely high delays, particularly above 100 bytes, over 120 ms. RKCKDMI, on the other hand, has consistently low encryption and decryption time, proving to be more efficient, scalable, and stable, and hence a more pragmatic solution compared to RSA for large data.

Table 7: Comparative Analysis of RSA, DHM and RKCKDMI

Algorithm	Plain Text (Bytes)	Key Size (Bits)	Encryption Time(ms)	Decryption Time(ms)	Memory Usage (KB)

RSA	100	2048	2.11	45.85	0.14
DHM			10.91	2.63	0.13
RKCKDMI			0.22	0.19	0.14
RSA	150		2.93	45.48	0.19
DHM			11.84	2.80	0.18
RKCKDMI			0.39	0.39	0.23
RSA	200		2.98	44.23	0.23
DHM			12.11	2.85	0.23
RKCKDMI			0.44	0.41	0.3

The comparison emphasizes the encryption and decryption performance of RSA, DHM, and RKCKDMI for plaintext sizes ranging from 100–200 bytes. RSA exhibits considerable encryption time but extremely high decryption delay. DHM has high encryption cost with lesser decryption time. RKCKDMI systematically delivers low encryption and decryption times with reasonable memory usage, demonstrating its efficiency as mentioned in table 7.

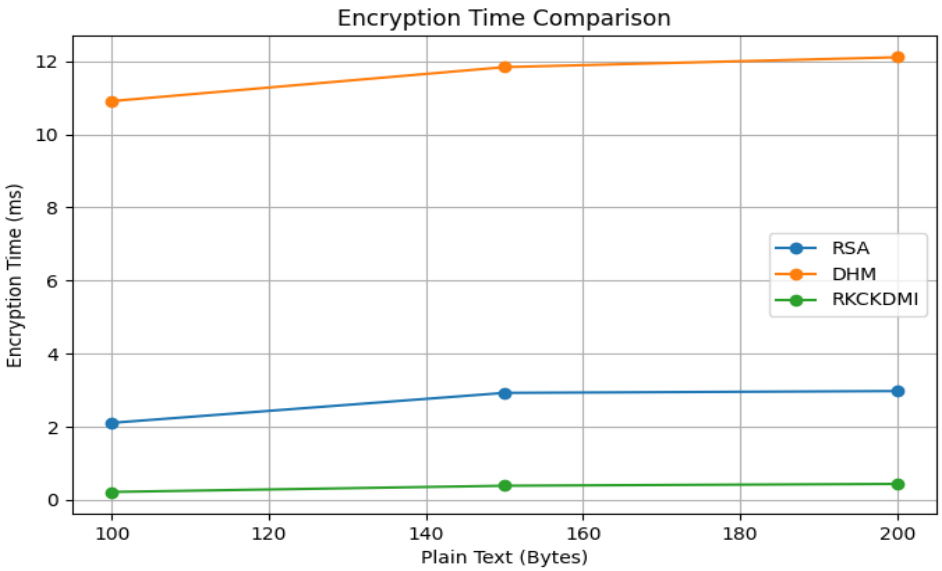

Figure 10: Encryption Time Comparison of RSA, DHM and RKCKDMI

Figure 10 shows RSA with moderate growth, DHM exhibiting the highest and steadily increasing time, and RKCKDMI maintaining the lowest and most stable performance. This demonstrates RKCKDMI's efficiency, RSA's balanced trade-off, and DHM's higher computational demand as plaintext size increases.

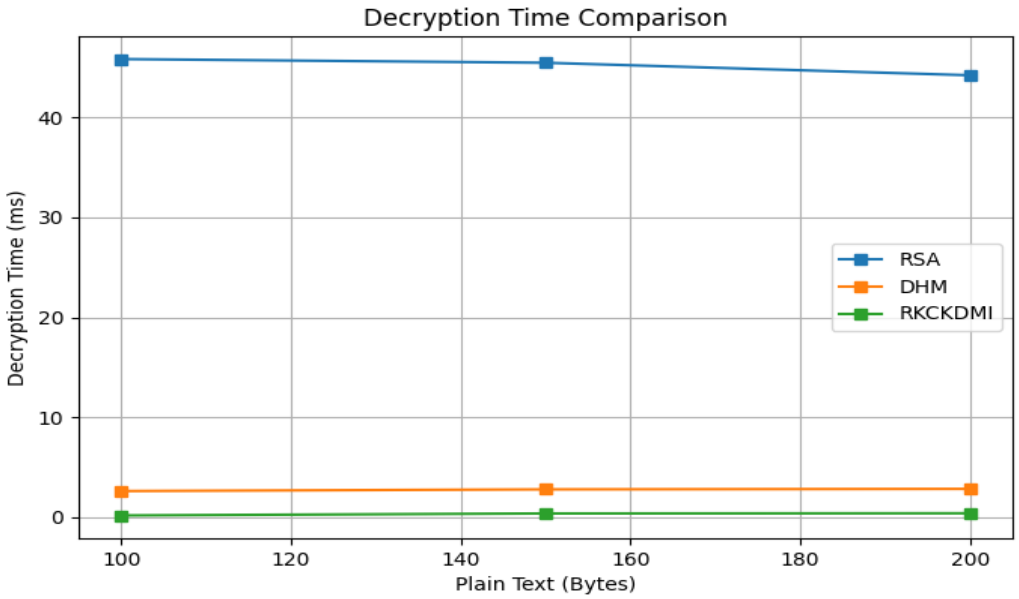

Figure 11: Decryption Time Comparison of RSA, DHM and RKCKDMI

The comparison of decryption times of RSA, DHM, and RKCKDMI algorithms for varying plaintext sizes is given in the graph. RSA has the maximum decryption time, always above 40 ms. DHM takes moderately with approximately 2–3 ms, whereas RKCKDMI always has the minimum decryption time, below 1 ms, indicating better efficiency and scalability as mentioned in figure 11.

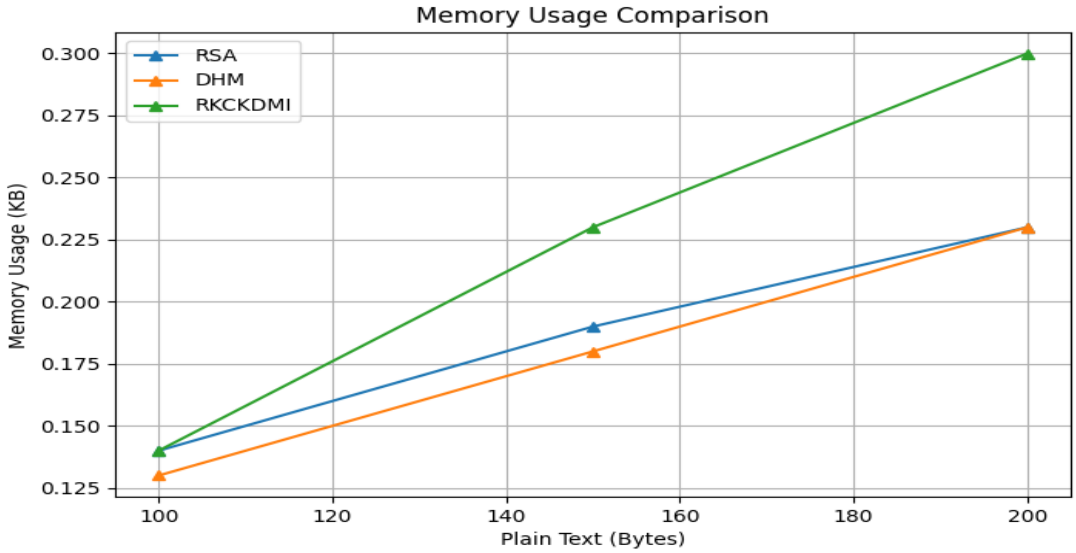

Figure 12: Memory Usage Comparison of RSA, DHM and RKCKDMI

The figure 12 indicates memory usage comparison of RSA, DHM, and RKCKDMI for varying plaintext sizes. RSA and DHM increase steadily, with DHM dipping slightly in usage. RKCKDMI, though good in decryption time, uses the most memory, up to 0.3 KB at 200 bytes, showing a compromise between performance and usage.

Threat Model

The framework is evaluated under a conventional adversary model. The adversary is able to tap the communication channels, carry out a chosen plaintext attack, try to replay the messages, and also try a man-in-the-middle attack. The adversary is restricted in terms of computational power and cannot solve the discrete logarithm problem or the integer factorization problem in polynomial time. This is the threat model in which the framework's claims are evaluated.

3.4 Security Analysis

The security analysis emphasizes robust protection by means of 256-bit XOR encryption, which is immune to brute-force attack. A combination of RSA and Diffie-Hellman makes defense stronger by making key-breaking more difficult. As public values are shared, the risk of interception is low. In total, key-breaking time remains unfeasible, maintaining highly robust and secure communication against possible attacks.

Strength of Keys

The power of XOR encryption will rely on the computed 256-bit key, which is impossible to be brute-forced. RSA encryption gives another level of security by requiring breaking Diffie-Hellman and RSA together to penetrate, which vastly increases security.

Resistance to Key-Exchange Interception

With Diffie-Hellman, what is transmitted is not the key itself, but just public values. This makes it very difficult for intermediaries to mount man-in-the-middle attacks on the process of key exchange.

Key-Breaking Time

This estimate for the time it will take to break the key is extremely robust; it makes the key effectively impractical even for people who could somehow utilize very powerful resources.

The amount of time needed to brute-force a key can be estimated by the formula:

$$T = \frac{2^k}{R} \quad \text{---------- (3)}$$

Where, T is the time taken to compromise the key, k is the size of the key in bits and R is the number of attempts made in a second. The value of R varies based on the attacker's hardware and available resources. Example Calculation for a 256-bit key and assuming:

$$T = \frac{2^{256}}{10^{12}} \; T \approx 3.67\times10^{61}\text{years} \quad \text{---------- (4)}$$

This demonstrates that brute-forcing such a key is computationally infeasible.

Table 8: Comparative Analysis of key breaking (DHM and RKCKDMI)

Algorithm	Key Size (bits)	Key Strength
DHM	1024	1.8×10308 years
RKCKDMI	1024	5.7×10303 year

The table 8 reveals that for 1024-bit keys, DHM registers an estimated brute-force resistance of around 1.8×10^{308} years, whereas RKCKDMI provides 5.7×10^{303} years. Though DHM seems more secure in absolute time, both schemes offer effectively unbreakable security and provide for immunity against brute-force attacks with existing and future computational powers.

Table 9: Comparative Analysis of key breaking (DES and RKCKDMI)

Algorithm	Key Size (bits)	Key Strength
DES	64	41.67 Days
RKCKDMI	64	5.8 years

The table 9 shows the vulnerability of DES with 64-bit keys, providing no more than 41.67 days of brute-force security, which is extremely insecure. It contrasts with RKCKDMI with the same key length providing 5.8 years, demonstrating much greater security. This is indicative of RKCKDMI having superior resistance when compared to standard DES encryption.

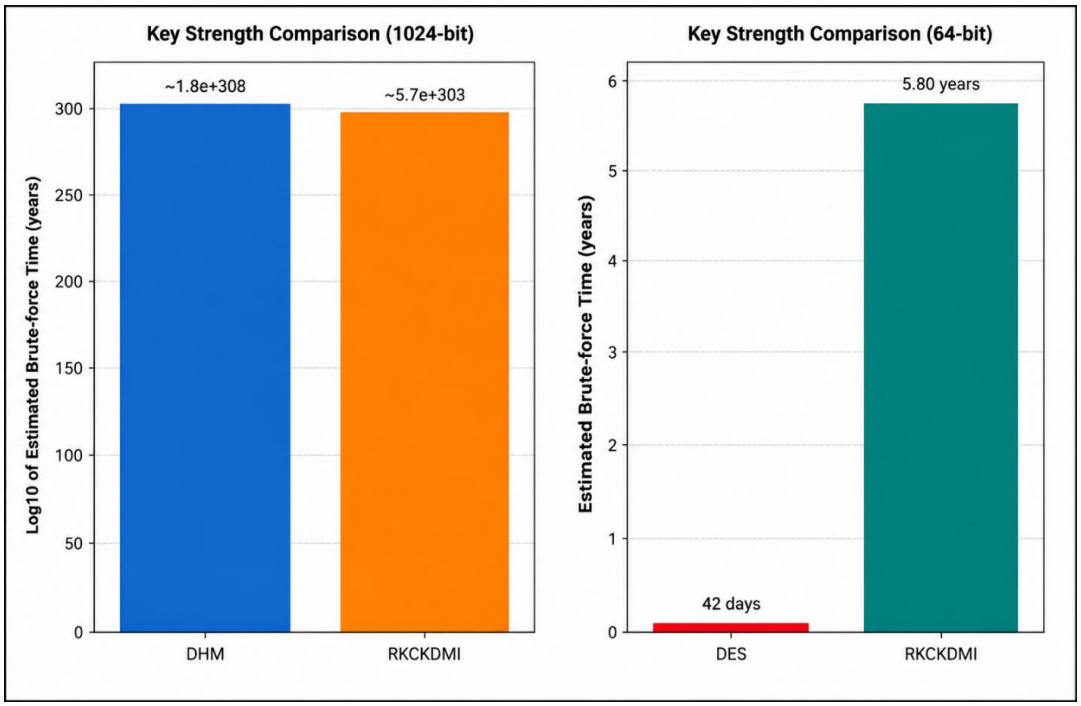

Figure 13: Comparison of Key Strength (DHM and RKCKDMI - DES and RKCKDMI)

Figure 13 compare essential strength in terms of brute-force resistance. For 1024-bit strength, DHM and RKCKDMI have a very high level of security of over 10^303 years. However, at 64-bit strength, DES is vulnerable to being broken in 42 days, whereas RKCKDMI shows much greater resistance, taking approximately 5.8 years, showing greater cryptographic dependability.

Table 10: Comparative Analysis of key breaking (AES, ECC and RKCKDMI)

Algorithm	Key Size (bits)	Key Strength
AES	256	3.67×1061years.
ECC	256	3.67×1077 years
RKCKDMI	256	3.67×1077 years

The AES comparison of 256-bit algorithms yields an estimated resistance against brute-force of 3.67×10^{61} years, while ECC and RKCKDMI yield much better security with 3.67×10^{77} years each. This places ECC and RKCKDMI as much stronger, providing good protection, while AES continues to offer extremely high but relatively lower resistance as mentioned in table 10.

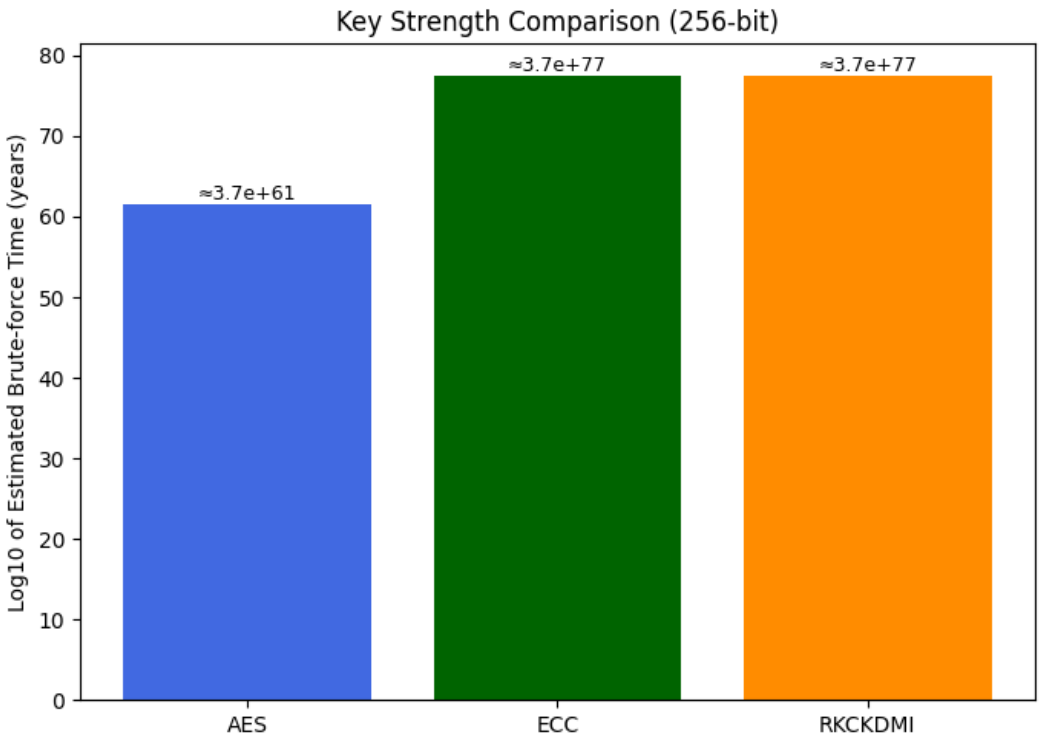

Figure 14 Comparison of Key Strength (AES, ECC and RKCKDMI)

Figure 14 provides a comparative strength of 256-bit encryption. AES shows high resistance with a brute-force time estimate of 10^61 years. However, ECC and RKCKDMI are far better than AES, both taking approximately 10^77 years, which shows their higher level of strength against brute-force attacks and provides highly secure cryptographic protection.

Table 11: Comparative Analysis of key breaking with different key sizes (RSA and RKCKDMI)

Algorithm	**Key Size (bits)**	**Key Strength**
RSA	1024	5.7×10^{310}years.
RKCKDMI		5.7×10^{294}years.
RSA	2048	1.02×10^{613}years.
RKCKDMI		1.02×10^{603}years.
RSA	4096	3.29×10^{1236}years.
RKCKDMI		3.29×10^{1220}years.

The calculation indicates RSA getting greater brute-force resistance at larger key sizes, with 1024-bit being 5.7×10^{310} years, 2048-bit being 1.02×10^{613} years, and 4096-bit being 3.29×10^{1236} years. RKCKDMI indicates slightly lower for all sizes but is still highly secure, providing good resistance against all practical brute-force attacks as mentioned in table 11.

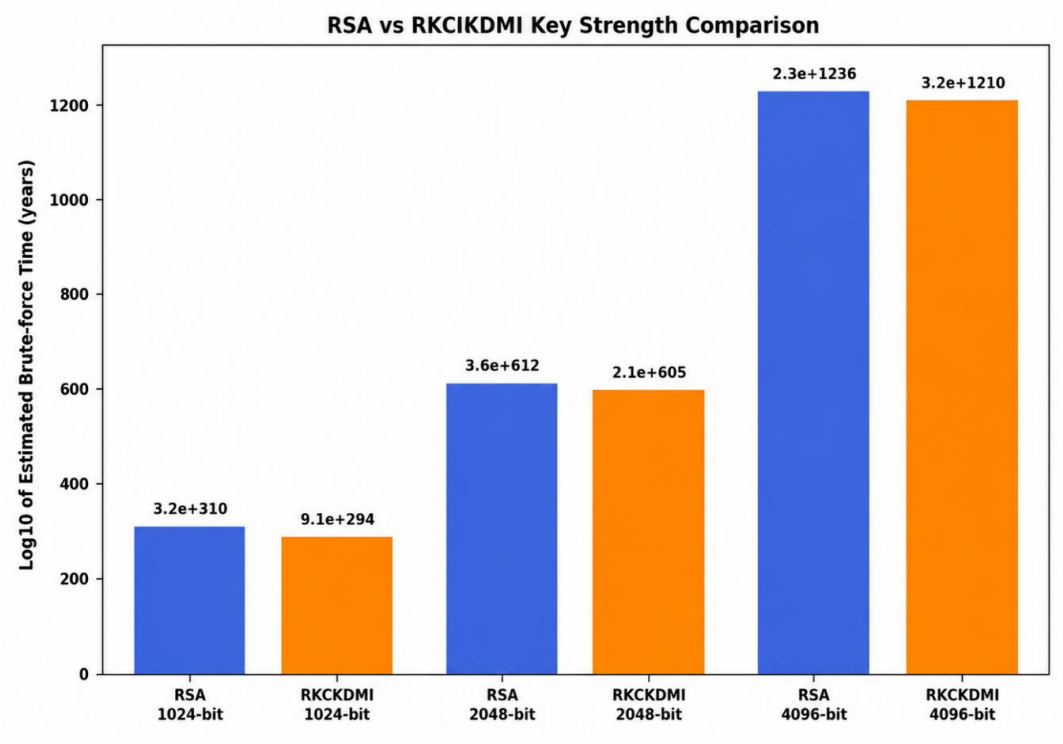

Figure 15: Comparative Analysis of key breaking with different key sizes (RSA and RKCKDMI)

The comparison graph of RSA and RKCKDMI key strengths between 1024, 2048, and 4096-bit keys is provided below. Both methods exhibit exponential brute-force resistance with key size increase. RSA has consistently higher values, but RKCKDMI has similar protection, making it an excellent alternative to RSA for contemporary cryptographic use with scalable security as mentioned in figure 15.

4. Conclusion

The paper presents a hybrid framework in cryptography using the Diffie-Hellman key exchange, RSA-based key encapsulation, and a keystream masking scheme based on the SHA-256 hash function. The paper demonstrates the efficiency of the framework in terms of computational efficiency and the confidentiality of data under the hardness assumptions. The framework is presented as an alternative to the standardized authenticated encryption schemes in a modular and lightweight form. Further research is also required in the direction of modeling the framework.

Conflict of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgement

The authors sincerely thank Sacred Heart College (Autonomous), Tirupattur, Tamil Nadu, India, for providing the infrastructure and support necessary to carry out this research. We also extend our gratitude to the faculty and peers from the Department of Computer Science for their valuable feedback and encouragement throughout this project.

References

1. Flores-Carapia, R., Silva-García, V. M., & Cardona-López, M. A. (2023). A dynamic hybrid cryptosystem using chaos and diffie–hellman protocol: An image encryption application. Applied Sciences, 13(12), 7168. https://doi.org/10.3390/app13127168
2. Yousif, S. F. (2021, February). Secure voice cryptography based on Diffie-Hellman algorithm. In IOP Conference Series: Materials Science and Engineering (Vol. 1076, No. 1, p. 012057). IOP Publishing. doi:10.1088/1757-899X/1076/1/012057
3. Alomari, G., & Aljarah, A. (2021). Efficiency of using the Diffie-Hellman key in cryptography for internet security. Turkish Journal of Computer and Mathematics Education, 12(6), 2039-2044.

4. Sagala, S. (2024). Application of Diffie-Hellman Algorithm Method (Key-Exchange) in Cryptography. Login: Jurnal Teknologi Komputer, 18(02), 114-119. https://doi.org/10.58471/login.v18i02.114
5. Standard, E. (2021). Enhanced Health-Care Protection Using Advanced Encryption Standard and Diffie Hellman Key Exchange Algorithm. International Journal, 10(2). https://doi.org/10.30534/ijatcse/2021/331022021
6. Sahoo, A., Mohanty, P., & Sethi, P. C. (2022). Image encryption using RSA algorithm. In Intelligent Systems: Proceedings of ICMIB 2021 (pp. 641-652). Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-19-0901-6_56
7. Adeniyi, E. A., Falola, P. B., Maashi, M. S., Aljebreen, M., & Bharany, S. (2022). Secure sensitive data sharing using RSA and ElGamal cryptographic algorithms with hash functions. Information, 13(10), 442. https://doi.org/10.3390/info13100442
8. Hoobi, M. M., Sulaiman, S. S., & AbdulMunem, I. A. (2020, November). Enhanced multistage RSA encryption model. In IOP Conference Series: Materials Science and Engineering (Vol. 928, No. 3, p. 032068). IOP Publishing. DOI: 10.1088/1757-899X/928/3/032068
9. Obaid, T. S. (2020). Study a public key in RSA algorithm. European Journal of Engineering and Technology Research, 5(4), 395-398. http://dx.doi.org/10.24018/ejers.2020.5.4.1843
10. Abouelkheir, E., & El-Sherbiny, S. (2022). Enhancement of speech encryption/decryption process using RSA algorithm variants. Human-centric Computing and Information Sciences, 12. https://doi.org/10.22967/HCIS.2022.12.006
11. Golovko, G., Matiashenko, A., & Solopihin, N. (2021). Data encryption using xor cipher. Системи управління, навігації та зв'язку. Збірник наукових праць, 1(63), 81-83. DOI: 10.26906/SUNZ.2021.1.081
12. Madhu, D., Vasuhi, S., & Samydurai, A. (2024). Dynamic 8-bit XOR algorithm with AES crypto algorithm for image steganography. Signal, Image and Video Processing, 18(Suppl 1), 429-445. https://doi.org/10.21203/rs.3.rs-3980991/v1
13. Masood, F., Driss, M., Boulila, W., Ahmad, J., Rehman, S. U., Jan, S. U., ... & Buchanan, W. J. (2022). A lightweight chaos-based medical image encryption scheme using random shuffling and XOR operations. Wireless personal communications, 127(2), 1405-1432. https://doi.org/10.1007/s11277-021-08584-z
14. Baalaji, K. D. P. D. K. (2025). Image Encryption and Decryption Algorithm using XOR Operator. DOI: 10.15680/IJIRSET.2025.1404438
15. Saeed, V. A., & Sadiq, B. H. (2023). Image encryption based on AES algorithm and XOR operation. Academic Journal of Nawroz University, 12(3), 533-539. https://doi.org/10.25007/ajnu.v12n3a1643
16. Salah, O., El-Sawy, A., & Taha, M. (2023). A Hybrid Algorithm For Enhancement of the Data Security during Network Transmission Based on RSA and DH. International Journal of Intelligent Engineering & Systems, 16(3).
17. Alatawi, M. N. (2023). A hybrid cryptographic cipher solution for secure communication in smart cities. Int. J. Comput. Netw. Appl, 10, 776-791.
18. Silva-García, V. M., Flores-Carapia, R., & Cardona-López, M. A. (2024). A Hybrid cryptosystem incorporating a new algorithm for improved entropy. Entropy, 26(2), 154. https://doi.org/10.3390/e26020154
19. Kumar, P., Saxena, V., & Singh, K. V. (2023). Analysis of Hybrid Cryptography for Secure Exchange of Information. International Journal of Computer Applications, 185, 37-42.
20. Deokar, R. B. (2023). File Transfer on Cloud using Diffie-Hellman Key Exchange in Conjunction with AES Encryption (Doctoral dissertation, Dublin, National College of Ireland).
21. Almalawi, A., Hassan, S., Fahad, A., & Khan, A. I. (2024). A hybrid cryptographic mechanism for secure data transmission in edge AI networks. International Journal of Computational Intelligence Systems, 17(1), 24. https://doi.org/10.1007/s44196-024-00417-8
22. Popoola, O., Rodrigues, M., Marchang, J., Shenfield, A., & Popoola, J. (2024). Hybrid encryption for smart home healthcare: ensuring data confidentiality and security. https://doi.org/10.1016/j.iot.2024.101314
23. Nadaf, R., & Bhairannawar, S. (2023). Design of Hybrid Encryption Algorithm using AES and RSA for key exchange. Degrés, 8(1).
24. Raziq, A., Qureshi, K. N., Yar, A., Zrar Ghafoor, K., & Jeon, G. (2024). Lightweight hybrid cryptography algorithm for wireless body area sensor networks using cipher technique.
25. Krishnadoss, P., Krishnan, P. T., Paramasivam, N., Kesavan, D. S., & Raagav, A. T. (2024). Dynamic Approach for Time Reduction in RSA Algorithm through Adaptive Data Encryption and Decryption. International Journal of Intelligent Engineering & Systems, 17(4).
26. Singamaneni, K. K. (2025). A novel lightweight hybrid cryptographic framework for secure smart card operations. EURASIP Journal on Information Security, 2025(1), 19. https://doi.org/10.1186/s13635-025-00204-8
27. Ghadi, D. M. (2023). A STUDY ON MODIFIED RSA CRYPTOSYSTEM. International Journal of Applied Sciences and Technology, 5.
28. Iliyasu, U., Nuhu, A. A., & Yakubu, M. M. (2023). Enhanced data security for file sharing: A hybrid approach combining RSA and Modified DES ciphers. Dutse Journal of Pure and Applied Sciences, 9(3a), 96-106. 10.4314/dujopas.v9i3a.11
29. Muhammed, R. K., Rashid, Z. N., & Saydah, S. J. (2025). A Hybrid Approach to Cloud Data Security Using ChaCha20 and ECDH for Secure Encryption and Key Exchange. Kurdistan Journal of Applied Research, 10(1), 66-82. https://doi.org/10.24017/science.2025.1.5