



International Journal of Electronic Security and Digital Forensics

ISSN online: 1751-9128 - ISSN print: 1751-911X
<https://www.inderscience.com/ijesdf>

Secure system to secure crime data using hybrid: RSA-AES and hybrid: Blowfish-Triple DES

Raghav Modi, Ananjay Singh Jammoria, Akshat Pattiwar, Arnav Agrawal, S.P. Raja

DOI: [10.1504/IJESDF.2025.10059534](https://doi.org/10.1504/IJESDF.2025.10059534)

Article History:

Received:	23 April 2023
Last revised:	11 July 2023
Accepted:	13 July 2023
Published online:	23 December 2024

Secure system to secure crime data using hybrid: RSA-AES and hybrid: Blowfish-Triple DES

Raghav Modi*, Ananjay Singh Jammoria,
Akshat Pattiwar, Arnav Agrawal and
S.P. Raja

School of Computer Science and Engineering,
Vellore Institute of Technology,
Vellore, Tamilnadu, India
Email: raghav04m@gmail.com
Email: ananjaysingh2016@gmail.com
Email: akshatpattiwar512@gmail.com
Email: arnavsagrawal@gmail.com
Email: avemariaraja@gmail.com

*Corresponding author

Abstract: Data security is the project's primary goal. We suggest hybrid cryptography as a technique to keep the data secure. When the sender tries to email the recipient the criminal data, it will be encrypted with a symmetric key utilising symmetric encryption. In most cases, the recipient receives a symmetric key that he can use to decrypt the data. However, we employ hybrid cryptography to further increase security by encrypting the symmetric key using asymmetric encryption and transmitting both the encrypted symmetric key and the encrypted crime data to the recipient. The encrypted symmetric key is decrypted by the recipient using their private key, and the encrypted criminal data is then safely delivered to the recipient using the decrypted symmetric key. In the police department data about offenders, including their background, record, and information about their prison records and officers who treated them is maintained in the cloud. Yet, that information is moved from one department to another for departmental purposes. The hackers may hack this process and contaminate the data. We recommend hybrid cryptography as a strategy to keep this data secure.

Keywords: Advanced Encryption Standard; AES; Rivest-Shamir-Adleman; RSA; Triple Data Encryption; 3DES; Blowfish.

Reference to this paper should be made as follows: Modi, R., Jammoria, A.S., Pattiwar, A., Agrawal, A. and Raja, S.P. (2025) 'Secure system to secure crime data using hybrid: RSA-AES and hybrid: Blowfish-Triple DES', *Int. J. Electronic Security and Digital Forensics*, Vol. 17, Nos. 1/2, pp.194–232.

Biographical notes: Raghav Modi is a BTech student and researcher at Vellore Institute of Technology, He has published papers on artificial intelligence in healthcare, and stock market prediction using machine learning techniques, and he is also a reviewer in easy-chair publications. His works include machine learning, deep learning, artificial intelligence, and cybersecurity.

Ananjay Singh Jammoria is currently pushing for a Bachelor of Technology degree in Computer Science at the esteemed Vellore Institute of Technology. In addition, he has a strong interest in machine learning and problem-solving and is constantly keen to explore new ideas and techniques in this fascinating area.

Akshat Pattiwar is a passion-driven computer science student at VIT Vellore. He is currently working in the field of artificial intelligence and cyber security. He works with ML/DL-based research projects.

Arnav Agrawal is a BTech student and researcher at Vellore Institute of Technology. He loves applying technology to real-world challenges, especially in sustainable development. He thinks technology may help solve climate change and poverty. He wants to use IoT to build smart cities and improve urban living. He is driven, passionate, and talented, making him a promising computer science and IoT professional.

S.P. Raja completed his BTech in Information Technology in 2007 from Dr. Sivanthi Aditanar College of Engineering, Tiruchendur. He completed his ME in Computer Science and Engineering in 2010 from Manonmaniam Sundaranar University, Tirunelveli. He completed his PhD 2016 from Manonmaniam Sundaranar University, Tirunelveli. His area of interest is image processing and cryptography. He is having more than 14 years of teaching experience. He has published seven papers in international journals, 25 in international conferences and 12 in national conferences.

1 Introduction

The world of technology is rapidly developing, and new advancements and innovations are making life easier than ever for everyone. As a result of these advancements, new challenges also appear, and one of the largest issues that most individuals run into is the protection of their online data. Consumers still experience this issue when they save their data online on cloud servers. The issue can be resolved with the use of cryptography. Cryptography is the process of converting clear text into incomprehensible text and vice versa. The word comes from the Greek word *kryptos*, which means hidden. The information or data employing these cryptographic techniques can only be accessed by authorised users using the keys from the cloud servers. Using cryptography serves the dual purposes of securing data from hackers and other outside parties while maintaining its secrecy and authenticity.

Without sufficient security, the data stored online, which can be very private and sensitive, could be abused or hacked. Thus, the data must be protected from all hostile attempts. Also, due to the size of data centres and the significance of the data stored there, it is important for strong security with swift processing and resource-efficient algorithms in addition to high security in general. Both of these issues may have a solution in hybrid cryptography.

The first hybrid cryptography algorithm put forth a strategy for protecting the data's assets employing the duo of Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA). Data is encrypted and decrypted using a single key by the symmetric cryptographic technique known as the AES. AES is used to encrypt data that has been divided up into blocks. Each block has a length of 128 bits (16 bytes). After that, each block is used to construct an 8×8 vector. Several rounds of the algorithm were run on each block. Each round consists of four steps: substitution, row and column shifting, column mixing, and round key insertion using a set of public and private keys, the asymmetric algorithm RSA encrypts and decrypts data. Only after utilising the public

key to encrypt the data, can it be decrypted using the private key counterpart. On the other side, only the public key can be used to decrypt data if it has been encrypted with the private key.

In order to address security issues, the second hybrid cryptography algorithm suggested a way for safe file storage on the cloud. In the system being supplied, multiple symmetric cryptographic techniques are being used. The data is divided into three pieces for security and encrypted and decoded using the Blowfish and Triple Data Encryption (3DES) methods.

2 Literature survey

Salman and Alomary (2022) presented the idea of combining cryptographic algorithms like Elliptic Curve Cryptography and Fully Homomorphic Encryption with the concept of unique values leading to the decrease in time of encryption and decryption. This paper discusses the ECC and FHE algos as it is a lightweight algo and provides high security. Unique values can be used for decreasing the encryption/decryption time. Experiments concluded that the encryption time reduced by 43% and decryption by 56%. Zhang and Hong (2022) studied characteristics of DES algorithm and optimisation is done. Key is increased from 64 to 128 bits. By increasing the bits of keys, the security is increased and the possibility of attack is decreased. It also proposes Enhanced Feistel Structure which is the extended feistel structure which can be applied to various encryption algorithms with feistel structure. Aminuddin (2020) studies a technique for using the RSA and AES cryptographic algorithms to protect the assets of the Android application from theft. Sharma et al. (2021) suggested that the main concern that each person must address is security. The intended use of the proposed system is to securely store and retrieve data from the cloud under the management and control of the data owner. We use hybrid cryptography in the proposed paradigm, combining different techniques for encryption and decryption, to produce a truly secure system. Cryptography and steganography techniques are used to address security-related difficulties with cloud data storage. In order to secure data, 3DES and Blowfish are used. They began by outlining the security services, such as secrecy and authentication. Lu et al. (2022) has proposed an architecture that revolves around LAN-to-LAN tunnel sessions, and Virtual Private Networks. Now in both of these cases, the main constraint or the area of concern is the end-to-end security usually in the messaging scenarios. Most of the internet protocols provide session keys for different communications sessions to each user which sounds great and very secure but what if the key gets stolen? Here, the attacker can easily make changes. When it comes to encryption in messaging, RSA is always there, but it has got some serious problems that have to be fixed. So, for this, they have proposed a more efficient and rather lightweight (SHRSA) authentication stack containing four layers that makes it even more secure. Some of the major issues with the RSA that have been fixed here are the partial key exposure, long encryption time, and the low complexity of the prime numbers. Anand and Singh (2023) propose a novel hybrid optimisation-based approach for secure medical data hiding in the industrial internet of things (IIoT). Their scheme enhances data security by concealing sensitive medical information, utilising optimisation techniques to safeguard IIoT systems. Barbosa et al. (2017) has started with the common problem statement saying that with the increasing number of devices and data stored in them, these things' impact on the life of people has been increasing by a lot and so to

ensure its safety it is important to come up with some good encryption algorithms. Elamir et al. (2021) have basically focused on multimedia files and their transfer securely with the help of an encryption algorithm. Also, the audio files and the video files both have been encrypted using different encryption algorithms. Later, from the cryptogram, the metadata was extracted. Data mining techniques play a huge role here as they provide algorithm identification. The first step includes the multi-media files getting encrypted. Here they have used four algorithms that are Blowish, DES, RC4 and RSA. Later, the files after being encrypted are sent to the algorithms of data mining. Data mining algorithms used here are J48, complement naïve Bayes, FT, and PART.

2.1 Organisation of the paper

The paper is discussed as follows. Motivation behind this research paper is summarised in Section 3. Cryptographic algorithms are discussed in Section 5. Management aspects of key is discussed in Section 6. The security analysis of all the algorithms is discussed in Section 7. Practical implementation consideration of hybrid cryptography is discussed in Section 8. Threat models and comparison with different algorithms is discussed in Section 9. Experiment results and performance evaluation of different metrics is discussed in Section 10. Finally, conclusion is discussed in Section 11.

3 Motivation

Cryptography has been the most important mechanism behind achieving present security services such as data integrity, confidentiality, and authentication. With the current world scenarios, where the world is moving towards an era of IoT and cloud technologies, providing the best security services is an important and tedious task to do. As it also opens a lot of vulnerabilities for adversaries to exploit and harm the organisations or individuals. This gives cryptosystems a huge opportunity to shine, including public key cryptography. This gives the task to the security service providers to come up with much more efficient hybrid cryptosystems better than the previous ones in almost every parameter. Also, another thing to be considered is that the variety of data that is being transmitted over the internet is also way too wide. It varies from simple text messages to multimedia files such as audio, and video files, or documents of various file types or extensions. This makes it a lot of vast processes to come up with not only efficient new hybrid cryptosystems but also different ones for different data types. For example, the ones used for encrypting text data might be different from how encryption is commonly done for multimedia files.

Also, the attackers have been coming up with some hard-to-tackle attacks. Some of them are rainbow tables, offline hash cracking, collisions, and birthday attacks. Mostly they are categorised into various categories like Brute force attacks, ciphertext-only attacks, chosen plaintext attacks (CPA), chosen ciphertext attacks (CCA), known plaintext attacks, and key and algorithm attacks. Considering the rapid advancement in the attackers and their attacking methods, it again tells how much it is important to keep the security up to date and security services or cryptosystems should also be updated and improved with time. Companies have been doing most of their work online and considering the amount of time and money people are devoting to the internet and the amount of data that is traveling through it tells how important network security is.

4 Selection of particular cryptographic algorithms

- RSA is widely used asymmetric encryption algorithm which uses mathematical complexity of large numbers to provide secure key exchange and also used in digital signatures as it is very difficult to break. It is efficient on small data for encryption and decryption so it used on for secure transfer of symmetric keys.
- Triple DES (3DES) is symmetric key encryption algorithm and is more secure than DES and also provides higher level of security. It is also compatible with legacy systems or can be used when legacy compatibility is required.
- Blowfish is another symmetric key encryption algorithm that is used because of its simplicity and efficiency. It supports a wide range of key sizes making it suitable and adaptable for different and varied security requirements. It is considered secure for most the applications.
- AES is a symmetric key algorithm that is highly used and widely adopted. Efficiency and performance for encryption and decryption of large amount of data are main factors for its widespread usage.

5 Cryptographic algorithms

5.1 RSA

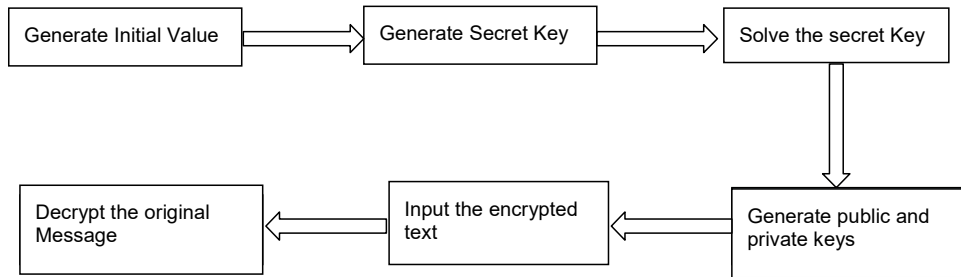
The RSA algorithm execution flow chart is shown in Figure 1. RSA is a public-key cryptosystem that is widely used for secure communication over the internet. The research paper introduces XRSA, an improved RSA-based public key encryption scheme (Imam et al., 2022). XRSA demonstrates enhanced efficiency and security for public key encryption, potentially benefiting various applications and industries that rely on secure data transmission and communication. We have used it for referring the basic RSA algorithm. The algorithm is named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman. The research proposes a QoS aware cluster head selection and hybrid cryptography routing protocol to enhance efficiency and security in VANETs. In Jubair et al. (2022), the protocol aims to improve data transmission and security in vehicular ad hoc networks, benefiting transportation systems and road safety. This paper has also used some AES and displayed its performance so we took this as a reference. The RSA algorithm is based on the mathematical concepts of modular arithmetic, prime factorisation, and number theory. To withstand modern attacks, it is recommended to use RSA key lengths of at least 2,048 bits for general-purpose applications and for applications requiring long-term security, which require extended data retention periods, key lengths of 3,072 bits or higher are recommended.

- *Hardware configurations:* RSA encryption and decryption operations involve large integer arithmetic, which can be computationally intensive. Hardware configurations with faster processors, larger cache sizes, and optimised integer arithmetic capabilities can significantly improve RSA performance.
- *Software optimisations:* implementations can utilise techniques like the Chinese Remainder Theorem (CRT) for private key operations, which can speed up RSA

decryption. Additionally, using optimised libraries and algorithms for modular exponentiation and modular arithmetic can improve performance.

- *Algorithm parameters:* the security of RSA relies on the size of the key used. Larger key sizes provide better security but can be slower to process. Choosing an appropriate key size based on the desired level of security and performance requirements is crucial.

Figure 1 RSA algorithm



Security strengths

- **Key exchange and encryption:** RSA provides a robust method for secure key exchange and encryption. It allows two parties who have exchanged public keys to securely communicate and encrypt messages that only the intended recipient can decrypt using their private key.
- **Strong mathematical foundation:** RSA is based on the mathematical difficulty of factoring large composite numbers. The security of RSA relies on the practical impossibility of factoring the product of two large prime numbers, ensuring that the private key remains secure.

Security weaknesses

- **Key size:** the security of RSA is dependent on the size of the key used. As computing power increases over time, larger key sizes are required to maintain the same level of security. Smaller key sizes can be vulnerable to brute-force attacks or advances in factoring algorithms.
- **Padding and randomness:** the use of proper padding schemes and sufficient randomness in RSA operations is crucial to prevent attacks such as CCA, padding oracle attacks, or timing attacks. Poor implementation or weak randomness generation can compromise the security of RSA.

Comparison with baseline (AES algorithm)

- **Performance:** RSA is typically slower compared to AES due to its computationally intensive operations involving large integer arithmetic. The research paper conducts a comparative analysis of AES and RSA algorithms regarding data security in cloud computing. It explores their respective strengths and weaknesses, focusing on

encryption and decryption processes. The findings are relevant to cloud computing practitioners and researchers, helping them make informed decisions about the best encryption algorithms for protecting sensitive data in cloud environments (Fatima et al., 2022). RSA encryption and decryption operations require more computational steps and thus are slower compared to symmetric algorithms like AES.

- **Energy efficiency:** RSA generally consumes more energy per operation compared to AES due to its higher computational requirements. The complex arithmetic operations involved in RSA encryption and decryption contribute to higher energy consumption. Fatima et al. (2022) conduct a comparative analysis of AES and RSA algorithms regarding data security in cloud computing. It explores their respective strengths and weaknesses, focusing on encryption and decryption processes. The findings are relevant to cloud computing practitioners and researchers, helping them make informed decisions about the best encryption algorithms for protecting sensitive data in cloud environments.

Here are the key mathematical components of the RSA algorithm:

- *Key generation:* the RSA algorithm requires the generation of a public and a private key. To generate the keys, two large prime numbers p and q are chosen randomly. The product of these two primes, $n = pq$, is used as the modulus for both keys. The totient of n , $\phi(n) = (p - 1)(q - 1)$, is also calculated.
- *Public key encryption:* to encrypt a message M using the RSA algorithm, the message is first converted to a number m , where $m < n$. The receiver's public key, denoted by (e, n) , is then used to compute the ciphertext $c = m^e \bmod n$. The sender then sends the ciphertext c to the receiver.
- *Private key decryption:* to decrypt the ciphertext c using the RSA algorithm, the receiver uses their private key, denoted by (d, n) , to compute the plaintext message $m = c^d \bmod n$. The receiver can then convert the number m back into the original message M .
- *Security:* the security of the RSA algorithm relies on the difficulty of factoring large integers. It is believed to be secure because it is computationally infeasible to factor the product of two large prime numbers. Therefore, an attacker would not be able to compute the private key from the public key alone.

Key generation

Choose two large prime numbers p and q . Calculate $n = p * q$ which is used as modulus for process of encryption and decryption and next compute the totient of n i.e.,

$$n = (p - 1) * (q - 1)$$

Choose an integer e such that $1 < e < \text{totient}(n)$. Calculate modular multiplicative inverse of $e \bmod \text{totient}(n)$

$$d * e \equiv 1 \pmod{\text{totient}(n)} \quad (1)$$

So the public key is (e, n) and private key is (d, n)

Encryption

Plain text converted to numerical representation and the resultant number is encrypted as follows: Public key: (e, n)

$$\text{CipherTextC} = M^e \pmod{n} \quad (2)$$

Decryption

During decryption, transform the cipher text to plain text:

$$\text{PlainTextM} = C^d \pmod{n} \quad (3)$$

Example

Choose two large prime numbers $p = 17$ and $q = 11$. Compute $n = p * q = 17 * 11 = 187$. Compute the totient of n : $\phi(n) = (p - 1) * (q - 1) = 16 * 10 = 160$. Choose a public exponent e such that $1 < e < \phi(n)$ and e is co-prime to $\phi(n)$, i.e., $\gcd(e, \phi(n)) = 1$. Let $e = 7$.

Compute the private exponent d such that $d \equiv e^{-1} \pmod{\phi(n)}$, i.e., d is the modular inverse of e modulo $\phi(n)$. In this case, $d = 23$, since $7 * 23 \equiv 1 \pmod{160}$.

The public key is $(e, n) = (7, 187)$ and the private key is $(d, n) = (23, 187)$. Suppose we want to encrypt the message 'HELLO'. We convert each letter to its corresponding ASCII code, which gives us:

$$H = 72$$

$$E = 69$$

$$L = 76$$

$$L = 76$$

$$O = 79$$

To encrypt each of these values, we use the public key (n, e) as follows:

$$C1 = 72^7 \pmod{187} = 66$$

$$C2 = 69^7 \pmod{187} = 93$$

$$C3 = 76^7 \pmod{187} = 61$$

$$C4 = 76^7 \pmod{187} = 61$$

$$C5 = 79^7 \pmod{187} = 46$$

So the encrypted message is the sequence of numbers 66 93 61 61 46.

To decrypt the message, we use the private key d as follows:

$$M1 = 66^{23} \pmod{187} = 72$$

$$M2 = 93^{23} \pmod{187} = 69$$

$$M3 = 61^{23} \pmod{187} = 76$$

$$M4 = 61^{23} \pmod{187} = 76$$

$$M5 = 46^{23} \pmod{187} = 79$$

Converting each number back to its corresponding ASCII code, we get the original message 'HELLO'.

5.2 Triple DES

A symmetric-key encryption method called Triple DES, also referred to as 3DES or TDES, is frequently used to protect data transfer over insecure networks. The algorithm is a development of the original DES algorithm, which was deemed insecure because of developments in cryptanalysis and had a key size of 56 bits. This paper focuses on the image encryption using chaos method and gives us the insight regarding the non-suitability of AES and DES for image encryption due to repetitive information pixels depict in similar images. In our paper, we have elaborately mentioned about the standardized algorithms such as AES, DES and many others (Zia et al., 2022). Triple DES operates on a key size of 168 bits, consisting of three 56-bit keys used successively, and employs a block cipher to encrypt data in 64-bit blocks.

To ensure sufficient security, it is recommended to use key lengths of at least 168 bits (equivalent to using three 56-bit keys). However, it is advisable to use more advanced algorithms such as AES to mitigate against future attacks.

- *Hardware configurations:* Triple DES performs multiple encryption/decryption rounds using the DES algorithm. Hardware configurations with efficient DES implementations or dedicated hardware accelerators can improve the performance of Triple DES.
- *Software optimisations:* software optimisations for Triple DES can include loop unrolling, efficient key scheduling, and optimised substitution box (S-box) implementations. Utilising parallelism, such as SIMD instructions, can also improve performance.
- *Algorithm parameters:* Triple DES operates on a fixed block size of 64 bits and supports different keying options (e.g., 2-key and 3-key). Using a longer key (3-key) provides stronger security but increases computational overhead.

Security strengths

- **Wide adoption:** Triple DES has been extensively used and studied for many years, which has helped identify and address various security issues. Its widespread implementation and analysis have increased confidence in its security.
- **Key length:** Triple DES uses a key length of 168 bits, providing a significantly larger keyspace compared to the original DES algorithm (56 bits). The increased key length enhances resistance against brute-force attacks.

Security weaknesses

- **Key size:** while Triple DES improves the key length compared to DES, a 168-bit key size is relatively small by modern cryptographic standards. The availability of more powerful computational resources and advancements in attacks make longer key sizes desirable for stronger security.

- Performance: Triple DES can be computationally intensive, especially when performing three iterations of the DES algorithm. This can result in slower encryption and decryption speeds, which may impact practical usage in high-performance or time-sensitive applications.

Comparison with baseline (AES algorithm)

- Performance: Triple DES is slower compared to AES due to its multiple encryption/decryption rounds using the DES algorithm. The additional rounds of encryption/decryption operations increase computational overhead and result in slower performance compared to AES.
- Energy efficiency: Triple DES consumes more energy per operation compared to AES due to its higher computational requirements and the nature of the DES algorithm. Although less energy-efficient than AES, Triple DES is still more efficient compared to RSA for similar operations.

Each block of plaintext or ciphertext must be subjected to the DES algorithm three times with three different keys in order to derive Triple DES mathematically. It is outside the purview of this answer to give a complete mathematical explanation of the DES algorithm, which is a complex combination of permutation and substitution operations. However, the general principles behind the algorithm can be summarised as follows.

Initial permutation

A fixed permutation table is used to perform an initial permutation on the 64-bit block of plaintext. In a predetermined manner, this rearranges the block's component parts. For instance the 1st bit of the plain text is replaced by the 58th bit of the plain text, the 2nd bit is replaced by the 50th bit of the plain text and so on. Table 1 represents the juggling of the bit positions of the original plain-text block.

Key generation

In this step, the already converted 64-bit key to 56-bit key by discarding every 8th bit of the initial key split into two halves of 28-bit each. Then the two parts are circularly shifted depending upon the round number. After this happens the keys are joined together and only 48-bit is selected or rather we can observe that only 48-bits out of the 56-bit key is left.

Table 1 Initial permutation table

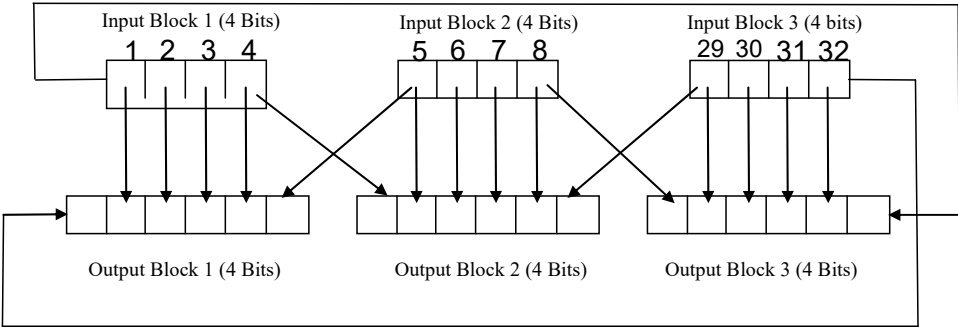
58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Expansion permutation

In the expansion permutation, the RPT is expanded from 32-bits to 48-bits. This is done by first dividing the RPT or LPT into 8 blocks of 4 bits each. Then the 4-bit block is

expanded into a 6-bit block by adding two bits (one at the beginning and one at the ending of the block). Figure 2 depicts the expansion clearly.

Figure 2 Permutation expansion



After the expansion, the 48-bit key is XORed with 48-bit RPT and resulting output is given to the substitution.

S box substitution

It accepts the 48-bit input which is the result of XOR operation between the key and the RPT and produces a 32-bit output using the substitution technique. The technique is performed by eight substitution boxes. The 48-bit input is split into eight sub blocks, each block is given to an individual S box. The S box transforms 6 bits to 4 bits. Hence, the output of a 48-bit input block is a 32-bit output block.

P box permutation

The output of the S-box is 32 bits. These 32-bits are given as an input to the P-box where the straightforward permutation mechanism is used and the input goes through simple permutation.

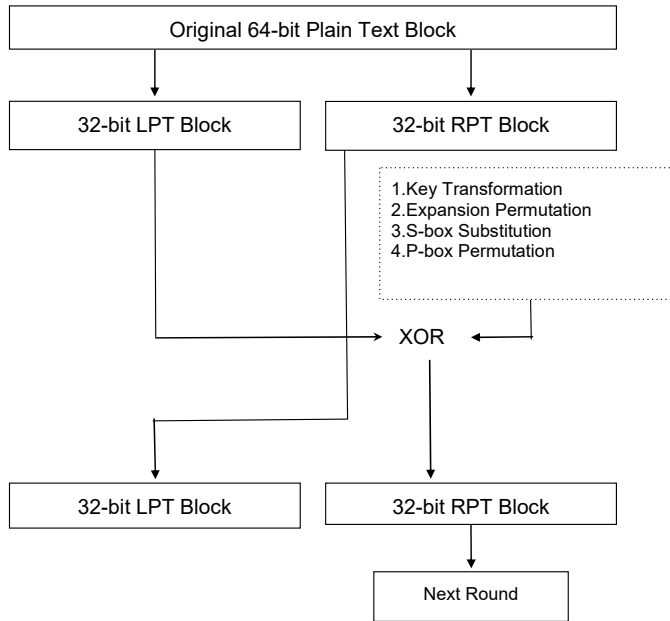
XOR and swap

Till now we were performing operations only on the RPT and the LPT was unchanged. So, in this step the LPT is XORed with the output given by the P-box Permutation. The result of this becomes the new RPT and the old right half (RPT) becomes the new left half, in a process of swapping.

Final permutation

At the end of the last round (i.e., 16 rounds) the final permutation is performed. This is a simple transposition. For instance, the 40th input bit takes the position of the 1st output bit and so on. The output of the final permutation is the 64-bit encrypted block.

Figure 3 shows the Triple DES algorithm flow chart.

Figure 3 Triple DES algorithm

Encryption/decryption

Combinations of permutation, substitution, and bitwise XOR operations are used during the encryption and decryption processes. There are four primary operations in each round:

- expansion: using a fixed expansion table, the 32-bit block of data is enlarged to 48 bits
- substitution: a fixed S-box is used to substitute each of the 8 6-bit blocks that make up the 48-bit block
- permutation: a fixed permutation table is applied to the 32-bit block
- XOR: the 48-bit subkey produced by the key schedule is XORed with the 32-bit block.

Example

As an example of how Triple DES is used in practice, let's consider the encryption of a message using Triple DES with 3 keys.

Suppose we have the following 64-bit message that we want to encrypt:

01101110 11011111 00010111 01110011 10101001 00011101 10010010 11110100

To encrypt this message using Triple DES, we first need to generate the three 64-bit keys, K1, K2, and K3. For this example, let's assume that we have the following three keys:

K1 = 1010111011011010 11100101 01011001 10011111 00101101 01110111 11010010
 K2 = 0110101110110010 11010110 10100111 10001110 10111111 01001001 11110010
 K3 = 1110100100101111 01110100 11010011 01001001 00111010 10010111 11011101

Next, we encrypt the message using the following steps.

Encryption with K1

Using a fixed permutation table, we apply an initial permutation to the message. A permuted message block is the end outcome. The permuted message block is then split into two 32-bit portions, L0 and R0. Using the K1-generated subkeys and a combination of permutation, substitution, and XOR operations, we encrypt data in 16 rounds.

After the 16 rounds, we obtain the intermediate ciphertext block C1:

10001110 10100101 11111011 01011100 10011010 01101100 01000101 11010010

Decryption with K2

The initial permutation we used for the message is also used for C1. A permuted ciphertext block is the end result. The permuted ciphertext block is then split into two 32-bit portions, L0 and R0. Using the K2 subkeys and a combination of permutation, substitution, and XOR operations, we carry out 16 rounds of decryption.

After the 16 rounds, we obtain the intermediate plaintext block P2:

11110110 11001011 10111100 11011010 01110110 11011100 10010010 11001111

Encryption with K3

The initial permutation we used for the message is also used for P2. A permuted plaintext block is the end result. The permuted plaintext block is then split into two 32-bit portions, L0 and R0. Using the K3-generated subkeys and a combination of permutation, substitution, and XOR operations, we encrypt data in 16 rounds.

After the 16 rounds, we obtain the final ciphertext block C3:

01011100 11000001 00100010 01110111 01001101 11011111 00101111 11000010

Therefore, the encrypted message is:

01011100 11000001 00100010 01110111 01001101 11011111 00101111 11000010

By using K3 to decode the C3 ciphertext block, K2 to encrypt the subsequent plaintext block, and K1 to decrypt the intermediate C3 ciphertext block, we may reverse the encryption process and decrypt the message.

Example with two keys

Here's a mathematical example of Triple DES encryption using two keys (i.e., 112-bit key length):

Let's say we have the following 64-bit input in hexadecimal format: 0x0123456789abcdef.

The first step is to generate two 56-bit keys, K1 and K2, from the provided 112-bit key. For simplicity, let's assume the keys are:

K1 = 0x133457799BBCDFF1

K2 = 0xAABB09182736CCDD

Next, we will perform the following operations:

Encryption with Key 1: We first encrypt the input using the first 56-bit key (K1) with the DES algorithm. The output of this operation is a 64-bit ciphertext.

Input: 0x0123456789abcdef

Key: K1 = 0x133457799BBCDFF1

Using the DES algorithm, we get the following output:

Output: 0x85e813540f0ab405

Decryption with Key 2: We have to decrypt the output of the previous step using the second 56-bit key (K2) and the DES algorithm. The result of this step is another 64-bit ciphertext.

Input: 0x85e813540f0ab405

Key: K2 = 0xAABB09182736CCDD

Using the DES algorithm, we get the following output:

Output: 0x5d9d4eeffc45a3b7

Encryption with Key 1: Finally, we encrypt the output of the previous step again using the first 56-bit key (K1) and the DES algorithm. The output of this step is the final 64-bit ciphertext.

Input: 0x5d9d4eeffc45a3b7

Key: K1 = 0x133457799BBCDFF1

Using the DES algorithm, we get the following output:

Output: 0xa9c75f8efc6cbd7f

So the final encrypted output using Triple DES with two keys is: 0xa9c75f8efc6cbd7f

To decrypt the message, we would reverse the steps:

Decryption with Key 1: Decrypt the input using Key 1 with the DES algorithm.

Encryption with Key 2: Using Des Algorithm we have to encrypt step 1's output using Key 2.

Decryption with Key 1: Using Des Algorithm we have to decrypt step 2's output using Key 1.

The final output should be the original plaintext input: 0x0123456789abcdef.

5.3 Blowfish

Blowfish is a very commonly used encryption methodology. It was designed as a substitute to another famous cipher, the DES in 1993. It has proved to be much quicker than the DES algorithm with a faster overall rate of encryption. It is a block cipher that works on a block. It is not subject to any patent meaning it can be freely used by anyone.

It has 64 bit blocks, 18 subkeys, 32 bit keys, 4 substitution boxes (512 entries of 32 bits in each), and has a total of 16 rounds.

For most of the applications, key lengths of 128 bits are considered secure. Key lengths of 256 bits is recommended for applications with higher security requirements to ensure compatibility and security and also to withstand potential future attacks.

- *Hardware configurations:* Blowfish is a symmetric block cipher that does not have specific hardware acceleration instructions. However, efficient implementations can still benefit from faster processors, larger caches, and optimised memory access.

- *Software optimisations:* Blowfish implementations can be optimised by using lookup tables for the S-boxes, parallelism (e.g., SIMD instructions), loop unrolling, and efficient key scheduling. Utilising hardware features like AES-NI for certain operations (e.g., key expansion) might also enhance performance.
- *Algorithm parameters:* Blowfish supports variable key sizes (32 to 448 bits) and operates on 64-bit blocks. The choice of key size affects both security and performance, with larger key sizes generally providing stronger security at the cost of increased computation time.

Security strengths

- **High speed:** Blowfish is designed to be computationally efficient, making it suitable for applications that require fast encryption and decryption. Its algorithmic simplicity contributes to its high speed.
- **Variable key length:** Blowfish supports key lengths from 32 bits to 448 bits, making it adaptable to different security requirements. Longer key lengths provide stronger security against brute-force attacks.

Security weaknesses

- **Vulnerability to certain attacks:** Blowfish has some known vulnerabilities that can impact its practical usage in certain scenarios. These vulnerabilities include:
- **Weak keys:** certain key values can lead to weak encryption that is susceptible to attacks.
- **Birthday attacks:** Blowfish is vulnerable to birthday attacks when used to encrypt a large amount of data with the same key. This can potentially compromise its security in certain situations.
- **Lack of formal verification:** Blowfish has not undergone the same level of formal analysis and scrutiny as some other widely adopted encryption algorithms. This can introduce uncertainty regarding its security and resilience against unknown attacks.

Comparison with baseline (AES algorithm)

- **Performance:** Blowfish is generally slower compared to AES due to its more complex key scheduling and substitution box (S-box) operations. The additional computational steps in Blowfish contribute to slower performance compared to AES.
- **Energy efficiency:** Blowfish consumes more energy per operation compared to AES due to its more complex operations. However, it is generally more energy-efficient compared to RSA for similar operations.

It has a few steps given as follows.

Sub key generation

- As already stated, it uses 18 subkeys, both for the encryption as well as decryption process and both use the same keys.

- It generates a P-Array containing 32-bit elements and the 18 subkeys are stored in it which is later initialised with the P_i 's digits.
- It generates 18 32-bit keys from $P[0]$ to $P[17]$ with hexadecimal representation.

Example:

$P[5]$: 567856A8

- After this, all the input keys are used to modify the subkeys,

$$P[m] = P[m] \oplus (m+1)^{\text{th}} \text{ 32 bits of input key}$$
- This step creates a new modified P-Array.

S-box initialisation

- In Blowfish, for both encryption and decryption 4 S-Boxes are used ($S[0]$ - $S[3]$). There are 256 32-bit entries in the S-Box.
- After initiating our P-Box, our array's supposed to be initiated using the P_i 's digits.

Encryption

This step is further divided into two parts:

- a Rounds:
 - This step includes 16 rounds in total in which the input of plain text is taken in each round from the last round and the corresponding subkey.
 - 64-bit input is divided into left and right halves and left half bits are XOR to the entries in P-Array and are then passed through the function F.
 - Later it is XOR to the right-half.
 - In output, these forms left half bits, and the left half from the previous round forms the right half bits.
 - It also includes the use of a function F.
- b Post-processing:
 - The output coming from the previous 16 rounds is then processed in this step.
 - Firstly, the 64-bit output from the previous step is divided into 32-bit left and right halves.
 - Then they are XOR with the entries of the P-Array.
 - After running this step, the cipher text is generated.

Decryption

- It is kind of similar to the encryption process.
- The only change is that 18 subkeys are used in reverse order.

Example

Let's understand it in a more elaborate way with the help of an example. So for this we are going to see the process of Blowfish with the help of an example.

Plain text: '78797abcabc78797'

Key: 'ccdd67388376ccdd'

Giving this input, the 18 subkeys would be generated using the steps given above. In this example, the 18 subkeys would be:

Generated sub-key 1: e8e20db0

Generated sub-key 2: 06d5c40e

Generated sub-key 3: dfc4ed16

Generated sub-key 4: 8006bf99

Generated sub-key 5: 68d45f1a

Generated sub-key 6: aae9fd0d

Generated sub-key 7: c4f39da0

Generated sub-key 8: 6f38a054

Generated sub-key 9: 89f546de

Generated sub-key 10: bba6dfaa

Generated sub-key 11: 728901f7

Generated sub-key 12: b79fc0b1

Generated sub-key 13: 0c714e8f

Generated sub-key 14: 4a0a9c00

Generated sub-key 15: f359b28d

Generated sub-key 16: 3631c5ca

Generated sub-key 17: 5ecbb2e1

Generated sub-key 18: 0a0f37c6

After this the 16-step process of encryption would take place that would give values as follows:

round 0: 213a3cfe909b770c

round 1: 8169604527eff8f0

round 2: 5ba807db5ead8d53

round 3: 5c3a6995dbaeb842

round 4: eba79a6234ee368f

round 5: 29e7f0aa414e676f

round 6: 30894d46ed146d0a

round 7: daf434f55fb1ed12

round 8: 157178275301722b

round 9: 5752010caed7a78d

round 10: 48246d1625db00fb

round 11: 6781c2feffbada7

round 12: 1b01f5ec6bf08c71

round 13: a31af721510b69ec

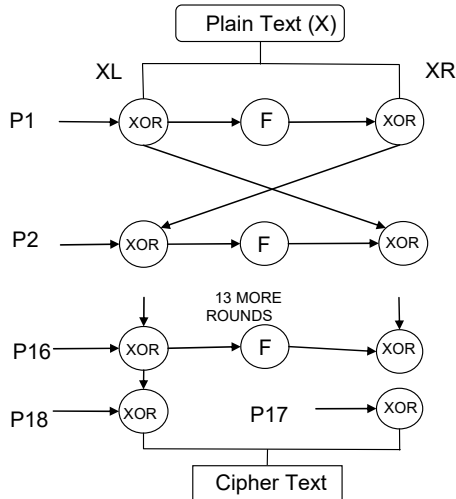
round 14: 4e0acb78504345ac

round 15: 8bb96371783b0eb2

After this, and the pre-processing, the output would be given in the form of the cipher text.

In this case, ciphertext: 72343974d572d190.

Figure 4 Blowfish encryption



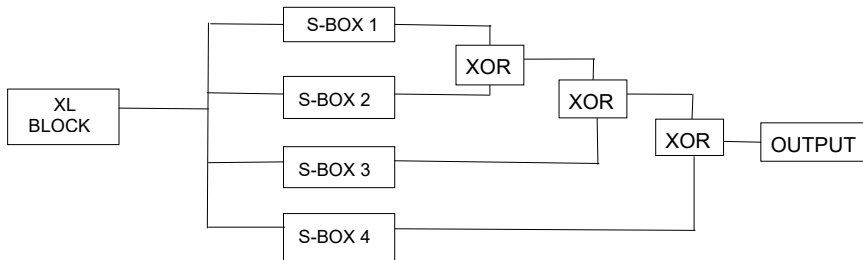
Execution of Blowfish algorithm is represented in Figure 4.

Another example to understand the basics of this algorithm is:

P: 'Hello, World!'

K: 'mysecretkey'

Figure 5 F function



F function which is utilised in Blowfish algorithm is represented in Figure 5.

The message can be written in binary as:

```
01001000 01100101 01101100 01101100 01101111 00101100 00100000
01010111 01101111 01110010 01101100 01100100 00100001
```

After dividing the message in 64-bit blocks, let's take the first block for understanding the algorithm in detail,

- First block:

```
01001000 01100101 01101100 01101100 01101111 00101100
00100000 0101 0110
```

- 64-bit encrypted output:

```
11001000 11110010 00101111 11111000 01110011 11011111
00000000 000110 10
```

After doing it for the whole message the output would be:

```
11001000 11110010 00101111 11111000 01110011 11011111 00000000
000110 10 01111110 01110110 11110001 1011100000001110 11111010
00100111 100101 01 0000111011011000 11110010 00010011 10111101
11011000 11100100 110110 11 1111011101111100 10100011 11010100
10001101 11011110 00001111 101101 01 0011000101110111 10110110
00110000 11101101 11100010 01100000 000111 10
```

i.e., C8922FF873DF0062 7E76F1B8 0EFA2795 0E5C58D8 E73EDCDB F7FCA3D4 843DCB5C 8DDE0F9B 3177B6E0 ED7B60E in HEXEADCEIMAL.

5.4 AES

AES is a symmetric encryption algorithm that operates on fixed-size blocks of data, with a block size of 128 bits and key sizes of 128, 192, or 256 bits. The AES encryption process involves several steps, including key expansion and a series of rounds that include substitution, transposition, and mixing operations. This paper deals with the encryption on the data which is stored by the user on the database using AES, elliptic curve encryption and signature. Here, they have compared all the above-mentioned algorithms for providing more secure database for end users (Zaw et al., 2019). AES is widely used for securing electronic data, including sensitive information transmitted over networks, stored on devices, or processed by applications. The security of AES is based on the difficulty of breaking its encryption key, which requires an impractical amount of time and resources using current technology.

AES key length of 128 bits is recommended to ensure adequate protection. For application requiring higher security, 256 bits of key length are recommended. This paper showcases white-box applications of AES and DES encryption algorithm. It also elucidates on white box cryptography and innovative AES operations (Jubair et al., 2022).

- *Hardware configurations:* AES is a symmetric encryption algorithm that relies on block operations. Modern processors often have hardware acceleration instructions (such as AES-NI) that can significantly speed up AES encryption and decryption.
- *Software optimisations:* utilising optimised AES implementations that take advantage of hardware acceleration, parallelism, and efficient memory access patterns can greatly enhance AES performance. Additionally, implementing countermeasures against side-channel attacks (e.g., cache timing attacks) is important for securing AES implementations.

- *Algorithm parameters:* AES supports different key sizes (128, 192, and 256 bits). Larger key sizes generally provide stronger security but may result in slower performance. The choice of key size should consider the desired level of security and the performance trade-off.

Security strengths

- **Strong security:** AES is considered highly secure and has been extensively analysed by cryptographers worldwide. It has withstood rigorous scrutiny and has not been broken or found to have any practical vulnerabilities when used properly.
- **Key length and variability:** AES supports key lengths of 128, 192, and 256 bits, providing a wide range of options for different security requirements. Longer key lengths offer stronger security against brute-force attacks.

Security weaknesses

- **Implementation vulnerabilities:** while the AES algorithm itself is secure, vulnerabilities can arise from flawed implementations, side-channel attacks, or poor key management practices. Careful attention must be paid to proper implementation and adherence to security guidelines to avoid such vulnerabilities.
- **Quantum computing:** like other symmetric encryption algorithms, AES is vulnerable to attacks by large-scale quantum computers, which can potentially break the encryption by leveraging Shor's algorithm or other quantum algorithms. Post-quantum cryptographic algorithms are being developed as alternatives to mitigate this threat.

Here are the steps involved in the AES algorithm:

- *Key expansion:* AES uses a fixed block size of 128 bits, but the key can be 128, 192, or 256 bits long. The key expansion algorithm takes the original key and creates a series of round keys, one for each round of the encryption process.

Let's assume we have a 128-bit key:

key = k0 k1 k2 k3 k4 k5 k6 k7 k8 k9 ka kb kc kd ke kf

We create an array of words by dividing the key into 32-bit words:

$$w[0] = k0\ k1\ k2\ k3 \quad (4)$$

$$w[1] = k4\ k5\ k6\ k7 \quad (5)$$

$$w[2] = k8\ k9\ ka\ kb \quad (6)$$

$$w[3] = kc\ kd\ ke\ kf \quad (7)$$

Next, we expand the key by generating additional words. This is done using a key schedule that involves a combination of XOR and substitution operations.

For $i = 4$ to 43 (total of 44 words), we perform the following steps:

a We assign $w[i]$ to $w[i-1]$.

- b If i is a multiple of 4, we perform the following operations on the current word:

We rotate the word by 8 bits to the left.

We apply the S-box substitution to each of the 4 bytes in the word.

We XOR the word with a round constant (which is a value derived from a mathematical formula).

- c We XOR the current word with the word at index $(i - 4)$.

Here's the expanded key in mathematical notation:

$$w[i] = w[i - 4] \text{ XOR } \text{SubWord}(\text{RotWord}(w[i - 1])) \text{ XOR } \text{Rcon}(i/Nk)$$

where

- i is the current index ($4 \leq i \leq 43$)
 - Nk is the number of 32-bit words in the original key (4 for a 128-bit key)
 - $\text{RotWord}()$ is a function that rotates the bytes of a word by 8 bits to the left
 - $\text{SubWord}()$ is a function that applies the S-box substitution to each byte in a word
 - $\text{Rcon}()$ is a function that generates the round constant for a given index i (based on a mathematical formula)
- *Initial round:* the plaintext is divided into 16-byte blocks, and the first round of the algorithm begins. Each byte in the block is XORed with a corresponding byte in the round key.

Assuming we have a 128-bit plaintext block P and a 128-bit key K .

We perform a bitwise XOR operation between the plaintext block P and the first 128 bits of the key schedule W , which is generated during the key expansion process. This operation is called *AddRoundKey* and can be expressed as:

$$P' = P \text{ XOR } W[0]$$

where P' is the resulting state after the *AddRoundKey* operation.

- *Rounds:* the AES algorithm consists of multiple rounds, the number of which depends on the length of the key. Each round involves a series of four operations: *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. These operations are performed on each 16-byte block of the plaintext. *Sub Bytes:* Each byte in the block is replaced with a corresponding byte from a fixed substitution table (S-box). After the *AddRoundKey* operation, we apply a byte-wise substitution using the S-box lookup table S . This operation is called *SubBytes* and can be expressed as:

$$P''[i, j] = S(P'[i, j]) \quad (8)$$

where P'' is the resulting state after the *SubBytes* operation, and i and j denote the row and column indices of the state matrix. *ShiftRows:* The rows of the block are shifted by a certain number of bytes.

After the *SubBytes* operation, we perform a row-wise circular shift on the state matrix according to a fixed pattern. The first row is left unchanged, the second row is

shifted one byte to the left, the third row is shifted two bytes to the left, and the fourth row is shifted three bytes to the left. This operation is called ShiftRows and can be expressed as:

$$P'''[i, j] = P''[i, (j+i) \bmod 4] \quad (9)$$

where P''' is the resulting state after the ShiftRows operation.

MixColumns: each column of the block is multiplied by a fixed matrix. After the Shift Rows operation, we perform a column-wise transformation on the state matrix using a fixed matrix multiplication with a 4×4 matrix M . This operation is called MixColumns and can be expressed as:

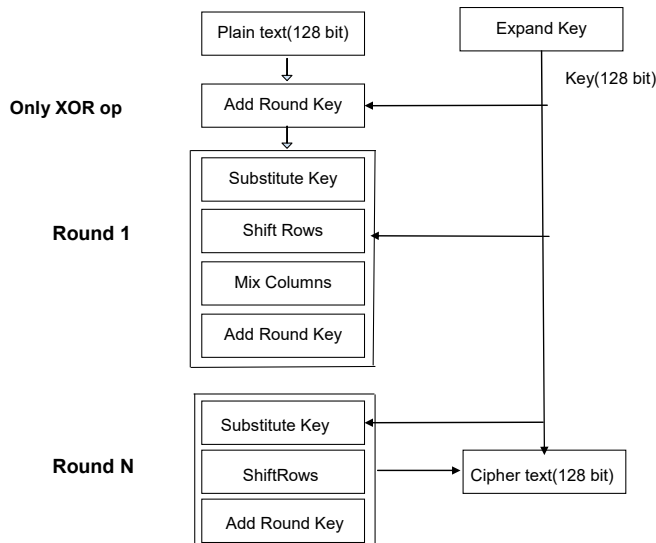
$$\begin{aligned} P'''[i, j] = & (M[i, 0] * P''[0, j]) \text{ XOR } (M[i, 1] * P''[1, j]) \\ & \text{XOR } (M[i, 2] * P''[2, j]) \text{ XOR } (M[i, 3] * P''[3, j]) \end{aligned} \quad (10)$$

where P''' is the resulting state after the MixColumns operation.

AddRoundKey: Each byte in the block is XORed with a corresponding byte in the round key.

- *Final round*: the final round of the algorithm is similar to the previous rounds, but it does not include the MixColumns operation.
- *Output*: the resulting ciphertext is the encrypted form of the original plaintext.
- *Decryption*: to decrypt the ciphertext, the process is reversed. The ciphertext is divided into 16-byte blocks, and each block undergoes the inverse of each operation used in encryption. The round keys are used in reverse order.

Figure 6 AES algorithm



The AES algorithm is widely used in various applications that require secure communication and data storage, including e-commerce, online banking, and secure communications.

Steps involved in AES algorithm is represented in Figure 6 which provides the overall flow of the algorithm.

Example:

Here's an example of AES encryption using a 128-bit key.

- Plaintext: 'This is a secret message'
- Key: '2b7e151628aed2a6abf7158809cf4f3c'

Key expansion

The 128-bit key '2b7e151628aed2a6abf7158809cf4f3c' is expanded into a key schedule of 11 round keys, each of 128 bits.

Padding

Since the plaintext is not a multiple of the block size, it needs to be padded to a multiple of 128 bits. In this case, we'll use PKCS#7 padding, which adds 4 bytes of padding with the value 0x04 to the end of the plaintext:

Plaintext: 54 68 69 73 20 69 73 20 61 20 73 65 63 72 65 74
20 6D 65 73 73 61 67 65 04 04 04 04 04

Encryption

The plaintext is divided into 128-bit blocks:

- Block 1: 54 68 69 73 20 69 73 20 61 20 73 65 63 72 65 74
- Block 2: 20 6D 65 73 73 61 67 65 04 04 04 04 04 04 04

For each block, the following steps are performed:

- Perform the AES encryption on the block using the current round key:
 - Block 1: 54 68 69 73 20 69 73 20 61 20 73 65 63 72 65 74
AES_Encrypt = 29 C3 50 5F 57 14 20 F6 40 22 99 B3 1A 02 D7 3A
 - Block 2: 20 6D 65 73 73 61 67 65 04 04 04 04 04 04 04
AES_Encrypt = 06 43 1F E3 D9 46 7B 3B E9 EF F8 8D 43 FD 43 5A
- The resulting ciphertext block becomes the encrypted block for the next block.

Ciphertext

The resulting ciphertext is the concatenation of all encrypted blocks:

29 C3 50 5F 57 14 20 F6 40 22 99 B3 1A 02 D7 3A
06 43 1F E3 D9 46 7B 3B E9 EF F8 8D 43 FD 43 5A

This ciphertext can be sent over the network to the receiver, who can decrypt it using the same key and mode in the reverse order of operations.

5.5 Hybrid: RSA and AES

Two well-known encryption methods, RSA and AES, are frequently combined to form a hybrid encryption scheme. With this method, a session key is safely exchanged using RSA and then AES is utilised to encrypt data.

Two keys are utilised in RSA encryption: a public key for data encryption and a private key for data decryption. The following equations can be used to explain the RSA encryption procedure:

$$\text{Cryptography: } C = M^e \bmod n \quad (11)$$

$$M = C^d \bmod n \text{ for decryption} \quad (12)$$

where the initial message is M . The coded message is C .

e is the modulus, which is the product of the two huge prime numbers p and q , and d is the private key exponent.

Here's an example of RSA-AES hybrid encryption:

Suppose Alice wants to send a message m to Bob using hybrid RSA-AES encryption. Here are the steps involved:

- Generate a random AES key K , which is a binary string of length n .
- Encrypt the message m using AES with the key K to obtain the ciphertext C , which is also a binary string of length n .
- Choose a recipient, Bob, who has a public-private RSA key pair (e, d) and a public key (N, e) , where N is the RSA modulus and e is the encryption exponent.
- Encrypt the AES key K using RSA with Bob's public key (N, e) to obtain the ciphertext C' , which is a binary string of length m , where m is the bit length of N .
- Send the pair (C, C') to Bob.

To decrypt the message, Bob uses the following steps:

- Decrypt the RSA ciphertext C' using his private key d to obtain the AES key K .
- Decrypt the AES ciphertext C using the key K to obtain the plaintext message m .

Here is an example with small numbers for illustration purposes:

Suppose Alice wants to send the message $m = 100$ to Bob using hybrid RSA-AES encryption. Let $n = 8$ and $m = 16$ for simplicity.

- Generate a random AES key $K = 01011010$.
- Encrypt the message $m = 100$ using AES with the key K to obtain the ciphertext $C = 11011011$.
- Bob chooses a public-private RSA key pair (N, e, d) such that $N = 77$, $e = 7$, and $d = 43$. His public key is $(N, e) = (77, 7)$.

- Encrypt the AES key $K = 01011010$ using RSA with Bob's public key $(N, e) = (77, 7)$ to obtain the ciphertext $C' = 28$.
- Send the pair $(C, C') = (11011011, 28)$ to Bob.

To decrypt the message, Bob uses the following steps:

- Decrypt the RSA ciphertext $C' = 28$ using his private key $d = 43$ to obtain the AES key $K = 01011010$.
- Decrypt the AES ciphertext $C = 11011011$ using the key $K = 01011010$ to obtain the plaintext message $m = 100$.

Error handling and robustness

- Impact of data errors: in a hybrid setup, RSA is typically used for key exchange and AES for actual data encryption. Data errors or corruption during the encryption or decryption process primarily affect the AES portion. The impact can vary depending on the extent of data corruption and the error detection/correction mechanisms employed.
- Execution time and memory usage: error handling and the detection of corrupted data can introduce additional computational overhead during the AES encryption or decryption process. This may result in slightly increased execution time and memory usage compared to error-free scenarios.
- Robustness and error recovery: AES itself does not provide built-in error recovery mechanisms. However, it is possible to use error detection codes (such as checksums) or error correction codes (such as Reed-Solomon codes) to enhance the robustness of the encrypted data. The ability to recover from errors depends on the error detection/correction techniques employed in the specific implementation.

Comparison with baseline (AES algorithm)

- Performance: the performance of a hybrid RSA and AES approach depends on the workload distribution between the two algorithms. RSA is typically used for key exchange or digital signatures, while AES is used for bulk encryption. The performance will vary based on the proportion of operations performed by each algorithm.
- Energy efficiency: the energy efficiency of a hybrid RSA and AES approach depends on the workload distribution between the two algorithms. As RSA is typically less energy-efficient compared to AES, the energy efficiency will be influenced by the proportion of operations performed by each algorithm.

5.6 Hybrid: Blowfish and Triple DES

Blowfish and 3DES are both symmetric key encryption algorithms, which means that they use the same key for encryption and decryption. To create a hybrid of Blowfish and 3DES, we can use one algorithm to encrypt the data and the other algorithm to encrypt the key used by the first algorithm.

Here is an example of how we can use Blowfish and 3DES together to encrypt a message:

- Generate a random key K1 for Blowfish encryption, and a random key K2 for 3DES encryption.
- Encrypt the message using Blowfish with the key K1. The output of this encryption is the encrypted message M1.
- Encrypt the key K1 using 3DES with the key K2. The output of this encryption is the encrypted key K1.
- Combine the encrypted key K1 and the encrypted message M1 into a single message M2.
- The final output is the encrypted message M2, which contains both the encrypted key K1 and the encrypted message M1.

To decrypt the message, we follow the reverse process:

- Decrypt the key K1 using 3DES with the key K2.
- Decrypt the message M1 using Blowfish with the key K1.

The final output is the decrypted message M1.

As for the mathematical calculations involved in the encryption and decryption process of both Blowfish and 3DES, they are beyond the scope of this response, as they are quite complex and involve various mathematical operations, such as bitwise XOR, modular arithmetic, and substitution permutations.

Error handling and robustness

- Impact of data errors: similar to the RSA-AES hybrid, data errors or corruption primarily affect the Blowfish encryption or decryption process in this hybrid setup. The impact would depend on the extent of data corruption and the error detection/correction mechanisms employed.
- Execution time and memory usage: error handling and the detection of corrupted data can introduce additional computational overhead during the Blowfish encryption or decryption process. This may result in slightly increased execution time and memory usage compared to error-free scenarios.
- Robustness and error recovery: Blowfish, like AES, does not inherently provide error recovery mechanisms. However, similar techniques such as error detection or error correction codes can be used to enhance the robustness of the encrypted data. The specific implementation's error recovery capabilities would determine the algorithm's ability to recover from errors.

Comparison with baseline (AES algorithm)

- Performance: similar to the hybrid RSA and AES approach, the performance of a hybrid Blowfish and Triple DES approach depends on the workload distribution

between the two algorithms. The performance will vary based on the proportion of operations performed by each algorithm.

- Energy efficiency: the energy efficiency of a hybrid Blowfish and Triple DES approach depends on the workload distribution between the two algorithms. As both Blowfish and Triple DES are generally less energy-efficient compared to AES, the energy efficiency will be influenced by the proportion of operations performed by each algorithm.

Implementation of hybrid: Blowfish and Triple DES

We first split the key into two parts: the first 16 bytes are used for Blowfish, and the last 8 bytes are used for DES. We then initialise the two ciphers with their respective keys and encrypt the plaintext using Blowfish first, and then DES. To decrypt the ciphertext, we simply reverse the process by first decrypting with DES and then with Blowfish.

We have a long key that we split into two parts for Blowfish and DES. We then have a plaintext message that we want to encrypt using both algorithms. The ciphertext is printed in hexadecimal format, and we also decrypt the ciphertext to verify that we can recover the original plaintext.

6 Key management aspects

- 1 *Key generation and selection:* generation and selection of appropriate keys for each algorithm.
 - AES: 128,192 or 256 bits of key length which should be generated using Cryptographically Secure Pseudo_random Number Generator(CSPRNG).
 - 3DES: keys of 168 bits derived from three individual 56-bit keys.
 - Blowfish: supports key lengths in range of 32 to 448 bits. Using CSPRNG with sufficient entropy
 - RSA: key pairs consists of public key and private key. Generation of keys involves selection of appropriate prime numbers. Key lengths of 2,048 bits or higher is recommended
- 2 *Key storage and distribution:*
 - AES: keys should be stored securely which includes practices such as Hardware Security Modules(HSMs), encrypted key files etc. These keys be distributed using secure key protocols such as Diffie-Hellman key exchange. Asymmetric encryption can be used to securely transmit keys.
 - 3DES: store 3DES keys using methods such as encryption or key wrapping with strong access controls. Secure key distribution methods like asymmetric encryption or secure channels be used to authorise parties.
 - Blowfish: securely store keys by using encryption/ key wrapping algorithms or techniques. Store keys using strong access controls. Distribute the keys using secure channels or asymmetric encryption methods to ensure that confidentiality is maintained during transmission.

- RSA: keys should be safeguarded with strong encryption and store the keys securely using mechanisms such as HSMs and public keys can be freely distributed. Private keys must be stored confidentially

3 Key rotation and revocation:

Regularly rotating the keys is recommended so that the impact of compromised key can be limited. Procedures for revoking of compromised or obsolete AES keys.

4 Key lifecycle management:

Key lifecycle management process should be established which includes rotation, usage, key generation, destruction, distribution. Roles and responsibilities must be clearly defined at each stage so that accountability can be ensured.

5 Key backup and recovery:

Regular backup of encryption keys so that data loss can be prevented in case of system failures or key corruption. Secure backup and recovery procedures should be implemented to protect key materials and ensure the availability when needed.

7 Comprehensive security analysis

7.1 Chosen ciphertext attacks

- AES: AES is resistant to CCA attacks when secure mode such as CTR or GCM operations. These mode operations provides prevention of adversary from information gain about the plaintext, semantic security or encryption key.
- 3DES: it is vulnerable to CCA because of its block-based structures. Using modes such as cipher block chaining (CBC) with secure padding scheme can help to mitigate vulnerability.
- RSA: resistant to CCA due to difficulty of factoring large numbers. Security due to strength of private key and use padding schemes optimal asymmetric encryption padding (OAEP).
- Blowfish: doesn't have inherent protection against CCA and therefore crucial to use secure modes such as CBC to mitigate vulnerability.

7.2 Chosen plaintext attacks

- AES: Resistant to CPA when used along with secure modes of operation such as CTR or GCM. Using these ensures that each ciphertext blocks depends on both encryption key and plaintext which prevents the attacker from getting visibility to the plaintext.
- 3DES: vulnerable to CPA due to block based nature. Use of CBC with secure padding provides protection.

- RSA: resistant to CPA due to infeasibility of computations of calculating private key from public key.
- Blowfish: doesn't inherent protection against plaintext attacks. Choice of secure mode such as CBC and appropriate padding scheme for mitigation of vulnerability.

8 Practical implementation considerations in hybrid cryptography

8.1 Platform compatibility

- AES: compatible and widely supported with platforms which include Windows, Linux, macOS and mobile OS such as android and iOS.
- Triple DES: this is also compatible with various platforms similar to AES which makes it portable.
- RSA: RSA is also supported and can be implemented on various platform and mobile operating systems.
- Blowfish: platform independent and supported by different platforms.

8.2 Programming languages and available libraries

- AES: can be implemented using programming languages like Python, Java, cpp, etc. and some of the libraries that provides AES implementations are Crypto++, OpenSSL, JCE.
- Triple DES: can be implemented using programming languages and libraries used in AES.
- RSA: CPP, python are some of languages used for implementation. OpenSSL, cryptography.io are some of the libraries that provides RSA function.
- Blowfish: it can be implemented using CPP, Python, etc. and OpenSSL, Crpto++ libraries can be used.

8.3 Integration and interoperability considerations

- While integrating AES, Triple DES, RSA and Blowfish in hybrid cryptography, factors such as key exchange formats, compatibility with cryptographic protocols and stands like PKCS#7 or X.509 certificates, interoperability between programming languages and platforms must be considered.
- Standardised formats like PEM for storage and exchange of keys and compatibility with protocols such as SSH or TLS should be used.

8.4 Parallelisability

- Refers to ability of concurrent execution of algorithm across multiple threads or processing units which improves overall performance and speed of encryption and decryption.
- AES: highly parallelisable. Encryption and decryption operations divided in blocks which makes it suitable for parallel processing.
- 3DES: unlike original DES algorithm, 3DES has some level of parallelisability.
- RSA: unlike symmetric algorithm, RSA is not easily parallelisable due to modular exponentiation and calculations.
- Blowfish: parallelisable across multiple threads or cores.

8.5 Hardware accelerations

- Graphical processing units (GPU): can be leveraged and used for performing cryptographic operations. Algorithms such as Blowfish and AES can be accelerated by making use of GPUs.
- Field-programmable gate array (FPGA): re-programmable hardware which can be optimised for cryptographic operations, encryption algorithms can be accelerated.
- Dedicated hardware modules: these modules are specifically designed for handling of encryption and decryption operations and provides faster processing time as compared to general purpose processors.

9 Threat models and comparison with different algorithms

9.1 Threat models

When considering the threat models for RSA, Triple DES, AES, and Blowfish algorithms, it is important to define the assumptions about the capabilities and resources available to potential attackers. The threat models help set the context for the performance analysis and provide insights into the relevance of the results in various security scenarios. Here are some common threat models to consider:

- Brute-force attacks: the threat model assumes that attackers have the computational resources to exhaustively search the entire keyspace in an attempt to break the encryption. The analysis would consider the time required for an attacker to perform a brute-force attack against the given algorithms with different key sizes.
- CPA: the threat model assumes that an attacker can choose specific plaintext inputs and observe the corresponding ciphertext outputs. The analysis would evaluate the security of the algorithms against these attacks and measure the impact on their performance.

- Side-channel attacks: the threat model incorporates attacks that exploit information leaked during the computation of the encryption algorithms, such as timing information, power consumption, or electromagnetic emissions. The analysis would explore the resistance of the algorithms to these attacks and their performance implications.
- Known plaintext or ciphertext attacks: the threat model assumes that an attacker has access to a set of known plaintext-ciphertext pairs and aims to deduce the encryption key or recover additional plaintexts. The analysis would assess the vulnerability of the algorithms to these attacks and consider the impact on performance.
- Quantum attacks: the threat model considers the future availability of large-scale quantum computers that can efficiently break the underlying cryptographic assumptions of the algorithms. The analysis would examine the potential impact of quantum attacks on the security of the algorithms and their performance.

9.2 *Comparison of algorithms against different types of attacks*

Brute-force attacks

- RSA: the security of RSA relies on the computational difficulty of factoring large prime numbers. As the key size increases, the time required to perform a brute-force attack grows exponentially, making it infeasible to break with current computational resources.
- Triple DES: the key size of Triple DES is 168 bits, but due to the meet-in-the-middle attack, the effective security strength is reduced to 112 bits. Nonetheless, a brute-force attack against Triple DES is still computationally infeasible.
- AES: AES is a symmetric encryption algorithm with key sizes of 128, 192, and 256 bits. Even with the smallest key size of 128 bits, AES is resistant to brute-force attacks within practical timeframes.
- Blowfish: Blowfish supports key sizes from 32 to 448 bits. Larger key sizes provide increased resistance against brute-force attacks. It remains secure against brute-force attacks with current computational resources.

Chosen plaintext attacks

- RSA: RSA is vulnerable to CPA if the same message is encrypted multiple times with different public keys. Proper padding schemes, such as OAEP, should be used to mitigate this vulnerability.
- Triple DES: Triple DES does not have inherent vulnerabilities against CPA. However, if an attacker can observe a sufficient number of plaintext-ciphertext pairs, they may be able to perform a statistical analysis to deduce information about the key.

- AES: AES is secure against CPA. It is resistant to known-plaintext attacks as well.
- Blowfish: Blowfish is also secure against CPA, as long as the key remains secret.

Side-channel attacks

- RSA: RSA is susceptible to side-channel attacks, particularly timing attacks and power analysis attacks. Implementations need to employ countermeasures, such as constant-time algorithms and masking techniques, to mitigate these vulnerabilities.
- Triple DES: Triple DES is vulnerable to side-channel attacks, including timing and power analysis attacks. Countermeasures, such as randomising the encryption process and using constant-time algorithms, can help enhance its resistance.
- AES: AES is susceptible to side-channel attacks, such as timing attacks and power analysis attacks. Countermeasures like constant-time implementations and protecting against cache-based attacks are essential to mitigate these threats.
- Blowfish: Blowfish is also susceptible to side-channel attacks. Implementations should consider countermeasures like constant-time algorithms and protecting against cache-based timing attacks.

Known plaintext or ciphertext attacks

- RSA: RSA is not vulnerable to known plaintext or ciphertext attacks when implemented correctly.
- Triple DES: Triple DES can resist known plaintext or ciphertext attacks, provided the keys are used properly and not reused.
- AES: AES is secure against known plaintext or ciphertext attacks.
- Blowfish: Blowfish is also secure against known plaintext or ciphertext attacks.

Quantum attacks

- RSA: RSA is vulnerable to attacks by large-scale quantum computers through Shor's algorithm, which can efficiently factor large numbers. Post-quantum cryptographic algorithms, such as those based on lattice problems or multivariate polynomials, are being studied as potential replacements for RSA.
- Triple DES: Triple DES is not currently known to be vulnerable to quantum attacks.
- AES: AES is also not known to be susceptible to quantum attacks.
- Blowfish: Blowfish is resistant to quantum attacks with the currently known algorithms.
- Regarding the hybrid encryption schemes:

- Hybrid-RSA and AES: the RSA algorithm is typically used for key exchange or key wrapping, while AES is used for the actual encryption. The security analysis for RSA and AES, as discussed above, applies to their respective roles in the hybrid scheme.
- Hybrid- Triple DES and Blowfish: similar to the hybrid scheme with RSA and AES, the security analysis for Triple DES and Blowfish in their respective roles follows the evaluations mentioned earlier.

10 Experimental results and performance evaluation

10.1 Dataset

- The National Incident-Based Reporting System (NIBRS) – this database maintained by the FBI contains detailed information about criminal incidents reported by law enforcement agencies across the United States. The database includes information about the nature of the offense, the victim and offender demographics, and other details about the incident.
- The Uniform Crime Reporting (UCR) Program – this database maintained by the FBI contains data on crime reported by law enforcement agencies across the United States. The database includes information about the number of reported crimes, the location of the crime, and other details about the incident.
- The Crime Open Database (CODE) – this is an open-source database of crime statistics from around the world. The database includes information about the type of crime, the location of the crime, and other details about the incident.
- The International Crime Victimization Survey (ICVS) – this is a cross-national study of crime victimisation and fear of crime. The dataset includes information about the types of crimes experienced by respondents, the location of the crime, and other details about the incident.

10.2 Impact of key size

In the field of hybrid cryptographic algorithms, the key size plays a crucial role in determining the performance and security of the encryption scheme. The key size refers to the length of the cryptographic keys used in the algorithm. In this response, we will discuss the impact of key size on the performance of algorithms, explaining how different key sizes affect execution time and memory usage, as well as any trade-offs between key size and security.

- Execution time: the key size directly influences the execution time of cryptographic algorithms. Generally, larger key sizes require more computational resources and longer processing times. As the key size increases, the number of possible keys also increases exponentially. Consequently, encryption and decryption operations become more computationally intensive and time-consuming. Therefore, larger key sizes can result in slower execution times compared to smaller key sizes.

- **Memory usage:** the key size also affects the memory usage of cryptographic algorithms. Larger key sizes typically require more memory to store the keys during encryption and decryption processes. The additional memory requirements can impact the overall performance of the system, especially in resource-constrained environments. Algorithms with larger key sizes may require more memory resources, which could limit their applicability in devices with limited memory capacity.
- **Trade-offs between key size and security:** the relationship between key size and security involves a trade-off. On one hand, larger key sizes offer increased resistance against cryptographic attacks, making it more difficult for attackers to guess or compute the encryption key. This enhances the security of the algorithm, as larger key sizes typically require significantly more computational power and time to break through exhaustive search attacks.

10.3 Experimental results

Various inputs were taken with varied sizes of text. Cryptographic algorithms and hybrid algorithms explained in Section 4 were applied on these input texts. Plain texts were converted to cipher texts using different types of keys and encryption/decryption techniques.

Table 2 Time complexity (in ms)

<i>Input texts/ cryptography algorithms</i>	<i>AES</i>	<i>3DES</i>	<i>RSA</i>	<i>Blowfish</i>	<i>Hybrid: RSA and AES</i>	<i>Hybrid: Blowfish and 3DES</i>
Input text 1	1.32	1.927	1.33	1.2	1.64	1.1
Input text 2	1.45	1.642	1.30	1.3	1.64	1.4
Input text 3	1.13	1.017	1.06	1.2	1.72	1.3
Input text 4	1.35	1.128	1.35	1.23	1.00	1.34
Input text 5	1.54	1.005	1.54	1.16	1.04	1.1
Input text 6	1.57	1.009	1.08	1.22	1.05	1.2
Input text 7	1.40	1.574	1.38	1.24	1.54	1.09
Input text 8	1.38	1.335	1.42	1.27	1.04	1.7
Input text 9	1.71	1.083	0.82	1.2	1.23	1.11
Input text 10	1.57	1.650	1.51	1.2	1.54	1.11
Input text 11	3.11	2.92	2.87	2.88	2.14	2.09
Mean	1.611	1.460	1.4236	1.36	1.38	1.28
Standard-deviation	0.578	0.668	0.5434	0.45	0.36	0.37

10.4 Performance evaluation

Performance of cryptography algorithms on texts are analysed on the basis of time required and memory usage for different sizes of inputs.

Table 3 Key generation time (in ms)

<i>Input texts/ cryptography algorithms</i>	<i>AES</i>	<i>3DES</i>	<i>RSA</i>	<i>Blowfish</i>	<i>Hybrid: RSA and AES</i>	<i>Hybrid: Blowfish and 3DES</i>
Input text 1	0.02	0.07	0.03	0.02	0.64	0.81
Input text 2	0.02	0.02	0.03	0.03	0.64	0.84
Input text 3	0.03	0.07	0.06	0.02	0.72	0.83
Input text 4	0.02	0.08	0.05	0.03	0.09	0.34
Input text 5	0.04	0.05	0.04	0.06	0.04	0.011
Input text 6	0.07	0.09	0.08	0.02	0.05	0.92
Input text 7	0.04	0.04	0.08	0.04	0.54	0.19
Input text 8	0.08	0.05	0.02	0.07	0.04	0.67
Input text 9	0.01	0.03	0.02	0.02	0.23	0.11
Input text 10	0.07	0.06	0.01	0.02	0.54	0.31
Input text 11	0.25	0.22	0.23	0.22	0.18	0.18
Mean	0.064	0.068	0.06	0.04	0.358	0.451
Standard-deviation	0.071	0.043	0.065	0.057	0.245	0.318

Table 4 Encryption time (in ms)

<i>Input texts/ cryptography algorithms</i>	<i>AES</i>	<i>3DES</i>	<i>RSA</i>	<i>Blowfish</i>	<i>Hybrid: RSA and AES</i>	<i>Hybrid: Blowfish and 3DES</i>
Input text 1	1.3	1.920	1.22	1.15	1.60	1.005
Input text 2	1.43	1.637	1.25	1.24	1.63	1.060
Input text 3	1.10	1.009	1.01	1.18	1.70	1.053
Input text 4	1.31	1.118	1.23	1.18	1.03	1.034
Input text 5	1.49	1.996	1.35	1.08	1.00	1.051
Input text 6	1.48	1.005	1.01	1.18	1.02	1.092
Input text 7	1.39	1.570	1.26	1.16	1.50	1.009
Input text 8	1.35	1.330	1.29	1.19	1.01	1.007
Input text 9	1.67	1.077	0.77	1.18	1.20	1.011
Input text 10	1.47	1.644	1.48	1.18	1.51	1.011
Input text 11	2.11	2.13	2.65	2.31	1.78	1.56
Mean	1.415	1.412	1.286	1.216	1.36	1.049
Standard-deviation	0.311	0.465	0.542	0.379	0.365	0.028

Several performance evaluation metrics such as time complexity (Table 2), key generation time (Table 3), encryption time (Table 4), decryption time (Table 5), memory usage (Table 6), energy consumption (Table 7) are compared between different cryptographic algorithm.

Table 5 Decryption time (in ms)

<i>Input texts/ cryptography algorithms</i>	<i>AES</i>	<i>3DES</i>	<i>RSA</i>	<i>Blowfish</i>	<i>Hybrid: RSA and AES</i>	<i>Hybrid: Blowfish and 3DES</i>
Input text 1	1.37	1.983	1.322	1.215	1.760	1.105
Input text 2	1.48	1.787	1.525	1.324	1.863	1.260
Input text 3	1.29	1.119	1.401	1.418	1.870	1.753
Input text 4	1.56	1.328	1.423	1.218	1.203	1.234
Input text 5	1.67	2.596	1.435	1.108	1.300	1.651
Input text 6	1.78	1.525	1.201	1.218	1.202	1.392
Input text 7	1.67	1.875	1.426	1.216	1.750	1.709
Input text 8	1.23	1.635	1.329	1.319	1.101	1.807
Input text 9	1.4	1.271	0.277	1.218	1.320	1.511
Input text 10	1.67	1.844	1.548	1.218	1.75	1.811
Input text 11	2.09	2.56	2.54	1.98	1.99	2.12
Mean	1.54	1.717	1.362	1.278	1.589	1.511
Standard-deviation	0.33	0.502	0.497	0.335	0.394	0.377

Table 6 Memory usage (in bytes)

<i>Input texts/ cryptography algorithms</i>	<i>AES</i>	<i>3DES</i>	<i>RSA</i>	<i>Blowfish</i>	<i>Hybrid: RSA and AES</i>	<i>Hybrid: Blowfish and 3DES</i>
Input text 1	16,897	11,226	9,460	11,344	12,282	11,480
Input text 2	16,812	11,650	10,096	11,360	12,332	12,010
Input text 3	16,231	11,626	9,792	11,336	12,657	11,952
Input text 4	16,648	11,642	9,892	11,352	12,255	12,002
Input text 5	16,629	11,666	9,946	11,376	14,670	11,992
Input text 6	16,457	11,674	9,938	11,384	12,216	12,000
Input text 7	16,796	11,634	9,975	11,344	12,282	11,960
Input text 8	16,413	11,666	9,973	11,376	12,314	11,992
Input text 9	16,212	11,650	9,964	11,360	12,255	11,976
Input text 10	16,212	11,650	9,964	11,360	12,255	11,976
Input text 11	40,123	39,989	35,879	33,560	28,709	29,143
Mean	18,157.27	13,780.9	10,885.2	12,070.4	12,494.7	12,026.3
Standard-deviation	6,508.06	12,389.6	8,919.7	9,129.1	2,470.5	572.8

Table 7 Energy consumption

<i>AES</i>	<i>3DES</i>	<i>RSA</i>	<i>Blowfish</i>	<i>Hybrid: RSA and AES</i>	<i>Hybrid: Blowfish and 3DES</i>
AES encryption and decryption operations involve simpler arithmetic and fewer computational steps compared to RSA. AES is known for its efficiency and optimised implementations, making it relatively more energy-efficient compared to RSA. The energy consumption will depend on the size of the input text and the specific implementation details.	Triple-DES encryption and decryption operations require more computational steps compared to AES. The energy consumption will be influenced by the size of the input text and the specific implementation details. While Triple-DES is generally less energy-efficient than AES, it is still more efficient compared to RSA for similar operations.	RSA encryption and decryption operations involve complex arithmetic operations and are generally considered less energy-efficient compared to symmetric encryption algorithms like AES. The larger the input text, the more computational operations are required, resulting in higher energy consumption.	Blowfish encryption and decryption operations involve more complex key scheduling and substitution box (S-box) operations compared to AES. The energy consumption will depend on the size of the input text and the specific implementation details. Blowfish is generally less energy-efficient compared to AES but more efficient compared to RSA for similar operations.	The energy efficiency of a hybrid RSA and AES approach will depend on the workload distribution between the two algorithms and the size of the input text. RSA operations typically consume more energy compared to AES operations. The overall energy efficiency will be influenced by the proportion of operations performed by each algorithm.	The energy efficiency of a hybrid Blowfish-Triple-DES approach will depend on the workload distribution between the two algorithms and the size of the input text. Triple-DES operations generally consume more energy compared to Blowfish operations. The overall energy efficiency will be influenced by the proportion of operations performed by each algorithm.

11 Conclusions

In this paper hybrid cryptographic algorithms like AES, RSA, 3DES and Blowfish have been implemented. Through the run tests on various metrics such as key generation time, encryption-decryption times, memory usage, energy consumption, time complexity etc we found out that the hybrid encryption algorithms are providing far better performance in almost every single criteria than the traditional methods. The above techniques can be used to securely store and transmit crime data keeping into consideration various factors like computation power, time required for encryption and decryption and also the efficiency along with the security of the algorithms.

References

- Abid, R., Iwendi, C., Javed, A.R. et al. (2023) 'An optimised homomorphic CRT-RSA algorithm for secure and efficient communication', *Pers. Ubiquit. Comput.*, Vol. 27, pp.1405–1418, <https://doi.org/10.1007/s00779-021-01607-3>.
- Aminuddin, A. (2020) 'Android assets protection using RSA and AES cryptography to prevent app piracy', *3rd International Conference on Information and Communications Technology (ICOIACT)*, Yogyakarta, Indonesia, pp.461–465, doi: 10.1109/ICOIACT50329.2020.9331988.
- Anand, A. and Singh, A.K. (2023) 'A hybrid optimization-based medical data hiding scheme for industrial internet of things security', *IEEE Transactions on Industrial Informatics*, Vol. 19, No. 1, pp.1051–1058, doi:10.1109/TII.2022.3164732.
- Atikah, N., Ashila, M.R., Rachmawanto, E.H. and Sari, C.A. (2019) 'AES-RC4 encryption technique to improve file security', in *Book AES-RC4 Encryption Technique to Improve File Security*, IEEE, pp.1–5.
- Barbosa, F.M., Vidal, A.R.S.F., Almeida, H.L.S. and de Mello, F.L. (2017) 'Machine learning applied to the recognition of cryptographic algorithms used for multimedia encryption', *IEEE Latin America Transactions*, Vol. 15, No. 7, pp.1301–1305, doi: 10.1109/TLA.2017.7959350.
- Bhattacharjya, A., Zhong, X. and Li, X. (2019) 'A lightweight and efficient secure hybrid RSA (SHRSA) messaging scheme with four-layered authentication stack', *IEEE Access*, Vol. 7, pp.30487–30506, DOI: 10.1109/ACCESS.2019.2900300.
- Cilardo, A., Coppolino, L., Mazzocca, N. and Romano, L. (2006) 'Elliptic curve cryptography engineering', *Proceedings of the IEEE*, February, Vol. 94, No. 2, pp.395–406, DOI: 10.1109/JPROC.2005.862438.
- Elamir, M.M., Mabrouk, M.S. and Al-Atabany, W.I. (2021) 'An e-health system for encrypting biosignals using Triple DES and hash function', *3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, Giza, Egypt, pp.11–14, doi: 10.1109/NILES53778.2021.9600097.
- Fatima, S., Rehman, T., Fatima, M., Khan, S. and Ali, M.A. (2022) 'Comparative analysis of AES and RSA algorithms for data security in cloud computing', *Engineering Proceedings*, Vol. 20, No. 1, p.14.
- Hagui, I., Msolli, A., Helali, A. and Hassen, F. (2021) 'Based blockchain-lightweight cryptography techniques for security information: a verification secure system for user authentication, pp.1–5, DOI: 10.1109/ICCAD52417.2021.9638751.
- Huang, X. and Wang, W. (2015) 'A novel and efficient design for an RSA cryptosystem with a very large key size', in *IEEE Transactions on Circuits and Systems II: Express Briefs*, October, Vol. 62, No. 10, pp.972–976, DOI: 10.1109/TCSII.2015.2458033.
- Imam, R., Anwer, F. and Nadeem, M. (2022) 'An effective and enhanced RSA based Public key encryption scheme (XRSA)', *Int. J. Inf. Technol.*, Vol. 14, pp.2645–2656.

- Jubair, M.A. et al. (2022) 'A QoS aware cluster head selection and hybrid cryptography routing protocol for enhancing efficiency and security of VANETs', *IEEE Access*, Vol. 10, pp.124792–124804, doi: 10.1109/ACCESS.2022.3224466.
- Lu, Y., Zhang, W. and Cao, L. (2022) 'Data security encryption method based on improved AES algorithm', *Global Reliability and Prognostics and Health Management (PHM-Yantai)*, Yantai, China, pp.1–6, doi: 10.1109/PHM-Yantai55411.2022.9942058.
- Rachmawanto, E.H., Budi Handoko, L., Umam, C., Jatmoko, C. and Ali, R.R. (2022) 'Triple DES cryptography based on hash function and DSA for digital certificate authentication', *2022 International Seminar on Application for Technology of Information and Communication (iSemantic)*, Semarang, Indonesia, pp.71–76, DOI: 10.1109/iSemantic55962.2022.9920438.
- Reddy, A., Das, A.K., Yoon, E.-J. and Yoo, K.-Y. (2016) 'A secure anonymous authentication protocol for mobile services on elliptic curve cryptography', *IEEE Access*, Vol. 4, pp.1–21, DOI: 10.1109/ACCESS.2016.2596292.
- Salman, Z. and Alomary, A. (2022) 'An efficient approach to reduce the encryption and decryption time based on the concept of unique values', *2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, Sakheer, Bahrain, pp.535–540, DOI: 10.1109/3ICT56508.2022.9990852.
- Sharma, V., Chauhan, A., Saxena, H., Mishra, S. and Bansal, S. (2021) 'Secure file storage on cloud using hybrid cryptography', *5th International Conference on Information Systems and Computer Networks (ISCON)*, Mathura, India, pp.1–6, doi: 10.1109/ISCON52037.2021.9702323.
- Shukla, P.K., Aljaedi, A., Pareek, P.K., Alharbi, A.R. and Jamal, S.S. (2022) 'AES based white box cryptography in digital signature verification', *Sensors*, Vol. 22, No. 23, p.9444.
- Sun, H.-M., Wu, M.-E., Ting, W.-C. and Hinek, M.J. (2007) 'Dual RSA and its security analysis', *IEEE Transactions on Information Theory*, Vol. 53, pp.2922–2933, DOI: 10.1109/TIT.2007.901248.
- Wu, Y. and Dai, X. (2020) *Encryption of Accounting Data Using DES Algorithm in Computing Environment*, 1 January, pp.5085–5095.
- Ye, G., Jiao, K. and Huang, X. (2021) 'Quantum logistic image encryption algorithm based on SHA-3 and RSA', *Nonlinear Dynamics*, Vol. 104, DOI: 10.1007/s11071-021-06422-2.
- Zaw, T.M., Thant, M. and Bezzateev, S.V. (2019) 'Database security with AES encryption, elliptic curve encryption and signature', *2019 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*, St. Petersburg, Russia, pp.1–6, doi: 10.1109/WECONF.2019.8840125.
- Zhang, X. and Hong, J. (2022) 'Research on improvement of DES encryption algorithm', *7th International Conference on Signal and Image Processing (ICSIP)*, Suzhou, China, pp.362–366, doi: 10.1109/ICSIP55141.2022.9886612.
- Zia, U., McCartney, M., Scotney, B. et al. (2022) 'Survey on image encryption techniques using chaotic maps in spatial, transform and spatiotemporal domains', *Int. J. Inf. Secur.*, Vol. 21, pp.917–935.