20BCE1724

Rahul Sivaselvam

AES-256 bit encryption

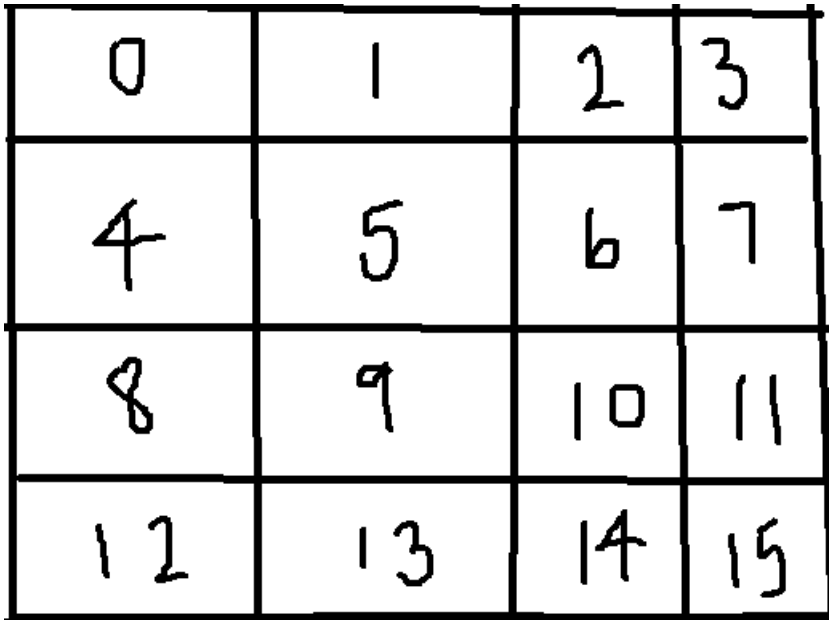
The Advanced Encryption Standard (AES) (also known as the Rijndael algorithm) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. It supersedes the Data Encryption Standard (DES) which was published in 1977. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

AES 256 is a virtually impenetrable symmetric encryption algorithm that uses a 256-bit key to convert plain text or data into a cipher.

Advantages

- Unbreakable by Brute Force: AES 256 encryption is highly secure and resistant to brute force attacks. While it's theoretically possible to crack AES encryption with immense computational power and time, it would take an infeasible amount of resources and a number of years (10 – 18 years) to do so. This level of security makes AES 256 a robust choice for protecting sensitive data.
- Symmetric Key Encryption: AES 256 uses symmetric key encryption, where the same key is used for both encryption and decryption. This simplifies the encryption process and allows for faster encryption speeds. Symmetric encryption is particularly suitable for internal or organizational data protection, as well as encrypting large volumes of data efficiently.
- Enhanced Security against Data Breaches: In the event of a security breach, AES 256 encryption helps contain the breach and prevent unauthorized access to the encrypted data. Even if hackers gain access to the systems, the encrypted data remains unreadable without the encryption key. This reduces the likelihood of data theft, compliance issues, and ransomware attacks, providing an additional layer of protection for sensitive information.
- Strongest AES Encryption Layer: AES 256 is the most secure encryption layer in the AES protocol, surpassing AES 128 and AES 192. While AES 128 and AES 192 are also robust encryption methods, AES 256 offers an even higher level of security. As technology advances, especially with the emergence of quantum computers, AES 256 is considered the most future-proof option for secure file transfers and data protection.

- By leveraging AES 256 encryption, you can ensure the confidentiality and integrity of your data, protect against unauthorized access, and mitigate the risks associated with data breaches. It offers a powerful and reliable security solution for safeguarding sensitive information in various applications and industries.



A single block is 16 bytes, a 4x4 matrix holds the data in a single block, with each cell holding a single byte of information.

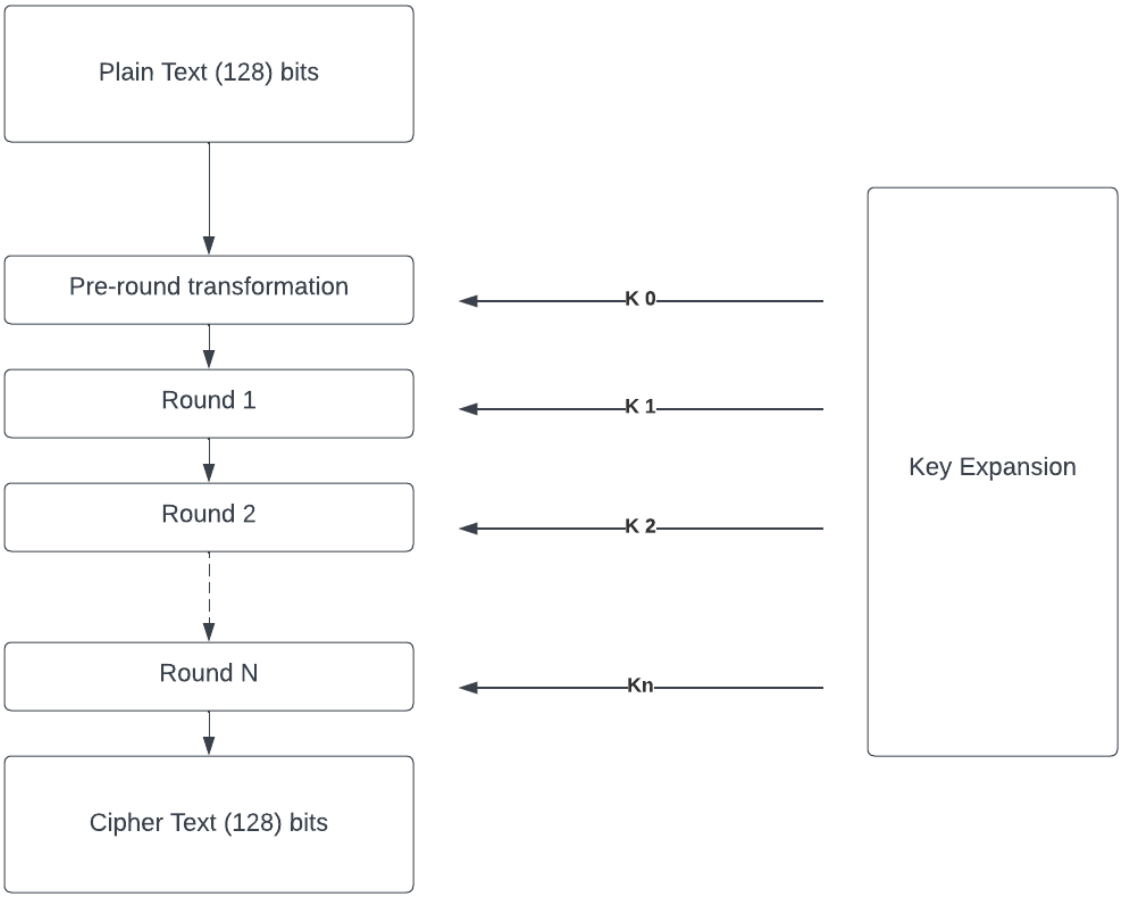
AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits (or 16 bytes) of the input data at a time.

The number of rounds depends on the key length as follows :

- 128 bit key – 10 rounds
- 192 bit key – 12 rounds
- 256 bit key – 14 rounds

A Key Schedule algorithm is used to calculate all the round keys from the key. So, the initial key is used to create many different round keys which will be used in the corresponding round of the encryption.



- The first step of AES 256 encryption is dividing the information into blocks of 4x4 columns of 16 bytes, as AES has a 128 bit block size.
- **Bite Substitution -** In this step, each byte of data is substituted with another byte of data.
- **Shifting Rows -** The AES algorithm then proceeds to shift rows of the 4x4 arrays. Bytes on the 2nd row are shifted one space to the left, those on the third are shifted two spaces, and fourth row is shifted thrice to the left.
- **Mixing Columns -** The AES algorithm uses a pre-established matrix to mix the 4x4 columns of the data array. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result. This step is skipped in the last round.
- **Add Round Key -** Now the resultant output of the previous stage is XOR-ed with the corresponding round key.

RSA Cipher

AES as discussed previously is based on substitution and permutation.

Public-key cryptography are based on mathematical functions rather than on substitution and permutation. Public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

There is a misconception that public-key encryption is better than symmetric encryption, cryptanalytically speaking, the security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher. There is nothing in principle about either symmetric or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis.

Applications of public-key cryptography systems.

Encryption/decryption: The sender encrypts a message with the recipient's public key.
Digital signature: The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
Key exchange: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

The **RSA** scheme is a cipher in which the plaintext and ciphertext are integers between 0 and n-1 for some n. A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 2^{1024}.

RSA makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is, the block size must be less than or equal to $\log_2(n) + 1$, in practice , the block size is i bits, where $2^i < n <= 2^{i+1}$.

Encryption and decryption are of the following forms, for some plaintext block M and ciphertext block C.

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus, this is a public-key encryption algorithm with a public key of PU = {e, n} and a private key of PR = {d, n}. For this algorithm to be satisfactory for public-key encryption, the following requirements must be met.

1. It is possible to find values of e, d, and n such that Med mod n = M for all M 6 n.

2. It is relatively easy to calculate Me mod n and Cd mod n for all values of M 6 n.

3. It is infeasible to determine d given e and n

p, q, two prime numbers (private, chosen)

n = pq (public, calculated)

e, with gcd(f(n), e) = 1; 1 6 e 6 f(n) (public, chosen)

d K e-1 (mod f(n)) (private, calculated

f(n) is the Euler totient function

Key Generation by Alice

Select p, q	p and q both prime, p != q
Calculate n = p x q	
Calculate Φ (n) = (p - 1)(q - 1)	
Select integer e	gcd (Φ (n), e) = 1; 1 < e < Φ (n)
Calculate d	d K e^{-1} (mod Φ (n))
Public key	PU = {e, n}
Private key	PR = {d, n}

Encryption by Bob with Alice's Public Key

Plaintext:	M < n
Ciphertext:	C = M^e mod n

Decryption by Alice with Alice's Public Key

Ciphertext:	C
Plaintext:	M = C^d mod n

A **hash function** H accepts a variable-length block of data M as input and produces a fixed-size hash value. A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either (a) a data object that maps to a pre-specified hash result (the one-way property) or (b) two data objects that map to the same hash result (the collision-free property). Because of these characteristics, hash functions are often used to determine whether or not data has changed.

Applications of Cryptographic Hash Functions

Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent (i.e., contain no modification, insertion, deletion, or replay). In many cases, there is a requirement that the authentication mechanism assures that purported identity of the sender is valid. When a hash function is used to provide message authentication, the hash function value is often referred to as a message digest.

The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message. The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value. If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered.

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as

$$Ci = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

where
C_i = ith bit of the hash code, 1 ... i ... n
m = number of n-bit blocks in the input
b_{ij} = ith bit in jth block
$\oplus$ = XOR operation

This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^{-n}. With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So, if a 128-bit hash value is used, instead of an effectiveness of 2^{-128}, the hash function on this type of data has an effectiveness of 2^{-112}.

Scenario/Problem: We will implement the RSA algorithm to securely transmit a message between two parties. The sender will encrypt the message using the recipient's public key, and the recipient will decrypt the message using their private key.

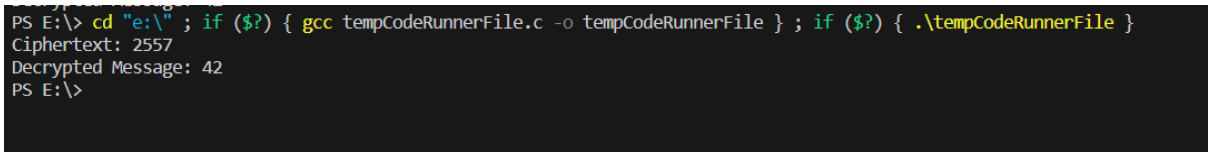
Step-by-Step Implementation:

1. Generate Large Prime Numbers:
    - We need two large prime numbers, p and q, to generate the RSA keys.
    - Choose two distinct prime numbers, e.g., p = 61 and q = 53.
    - These numbers are small for demonstration purposes, but in practice, much larger prime numbers are used to ensure security.
2. Calculate n and φ(n):
    - Calculate n by multiplying p and q: n = p * q.
    - Calculate φ(n) (Euler's totient function) as: φ(n) = (p - 1) * (q - 1).
3. Select the Public Key:
    - Choose a number e such that 1 < e < φ(n) and gcd(e, φ(n)) = 1.
    - Typically, e is a small prime number or a power of 2, e.g., e = 17.
4. Calculate the Private Key:
    - Calculate d such that (d * e) % φ(n) = 1.
    - In other words, find the modular multiplicative inverse of e modulo φ(n).
    - We can use the Extended Euclidean Algorithm to compute d.
    - For simplicity, we'll assume d = 413.
5. Encryption:
    - Convert the message to its numerical representation, such as ASCII values.
    - Use the public key (e, n) to encrypt the message using modular exponentiation.
    - The encryption formula is: ciphertext = (message^e) % n.
6. Decryption:
    - Use the private key (d, n) to decrypt the ciphertext using modular exponentiation.
    - The decryption formula is: message = (ciphertext^d) % n.

Code Implementation in C:

```
#include <stdio.h>

// Function to perform modular exponentiation
long long modularExponentiation(long long base, long long exponent, long long modulus) {
    long long result = 1;
    base = base % modulus;

    while (exponent > 0) {
        if (exponent % 2 == 1)
            result = (result * base) % modulus;
        exponent = exponent >> 1;
        base = (base * base) % modulus;
    }

    return result;
}

int main() {
    // Step 1: Generate Large Prime Numbers
    long long p = 61;
    long long q = 53;

    // Step 2: Calculate n and φ(n)
    long long n = p * q;
    long long phiN = (p - 1) * (q - 1);
```

```
    // Step 3: Select the Public Key
    long long e = 17;

    // Step 4: Calculate the Private Key
    long long d = 413;

    // Step 5: Encryption
    long long message = 42; // Plaintext
    long long ciphertext = modularExponentiation(message, e, n);

    // Step 6: Decryption
    long long decryptedMessage = modularExponentiation(ciphertext, d, n);

    printf("Ciphertext: %lld\n", ciphertext);
    printf("Decrypted Message: %lld\n", decryptedMessage);

    return 0;
}
```



Security Analysis:

Potential Threats/Vulnerabilities:

1. Brute Force Attack: An attacker could try to factorize the public modulus n to determine the private key d. This becomes computationally infeasible for large prime numbers.
2. Timing Attacks: An attacker might analyze the time taken during the modular exponentiation operation to gain information about the private key d.

Countermeasures/Best Practices:

1. Choose Large Prime Numbers: Use sufficiently large prime numbers to make factorization difficult.
2. Secure Key Generation: Implement a secure random number generator for selecting prime numbers and the public/private keys.
3. Padding: Apply padding schemes like OAEP (Optimal Asymmetric Encryption Padding) to avoid potential vulnerabilities associated with encrypting small messages directly.

Limitations/Trade-offs:

1. Computational Overhead: The RSA algorithm involves modular exponentiation, which can be computationally expensive for large numbers. This can impact performance in resource-constrained environments.
2. Key Management: Proper key management is crucial for the security of the RSA algorithm. Safeguarding private keys and ensuring secure key exchange between parties is essential.

Conclusion:

The RSA algorithm is a widely used asymmetric encryption algorithm that provides secure communication between parties. By implementing and analyzing the RSA algorithm, we've seen its practical application in encrypting and decrypting messages. However, it is important to consider potential threats, follow best practices, and be aware of the limitations and trade-offs involved in the implementation. Cryptography, including algorithms like RSA, plays a vital role in cybersecurity and ethical hacking by ensuring confidentiality, integrity, and authenticity of data in various applications.