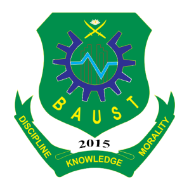
**Bangladesh Army University of Science and Technology (BAUST), Saidpur**

**Department of Computer Science and Engineering (CSE)**

****

**LAB REPORT**

**Course Title:** Computer Security Sessional

**Course Code:** CSE 4102

**Report No:** 01

**Report Tittle:** Encryption Algorithms – Vigenère, Vernam, One-Time Pad, DES, AES, and RSA.

**Comment:**

**Submitted To:**

**Teacher Name:** Ashrafun Zannat

**Designation:** Assistant Professor

**Teacher Name:** Alifa Shanzidah Manarat

**Designation:** Lecturer

**Signature & Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Submitted By:**

**Name:** A. K. M. Masudur Rahman

**ID:**

**Level: 4 Term: I**

**Submission Date:** 13.11.2025

**Lab Report:** Encryption Algorithms – Vigenère, Vernam, One-Time Pad, DES, AES, and RSA

**Objective**

1. To study and understand the working principles of the Vigenère Cipher, Vernam Cipher, One-Time Pad (OTP), DES, AES, and RSA encryption algorithms.
2. To implement these algorithms conceptually using the C programming language.
3. To analyze the mathematical operations involved in each encryption method.
4. To compare performance, security strength, and practical use of classical and modern cryptography.
5. To visualize the working structure of symmetric and asymmetric encryption using diagrams and examples.

**Introduction**

Cryptography is the science and art of securing communication by converting readable information (plaintext) into an unreadable form (ciphertext) using mathematical algorithms and secret keys. It ensures data confidentiality, integrity, and authenticity in both personal and industrial digital communications. The process of encryption prevents unauthorized access to information, while decryption restores the original data using the same or related key.

Over centuries, cryptographic techniques have evolved from simple manual ciphers to complex mathematical systems that power modern cybersecurity. This report focuses on six important encryption algorithms that represent distinct eras of cryptography — Vigenère Cipher, Vernam Cipher, One-Time Pad (OTP), Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Rivest–Shamir–Adleman (RSA) algorithm. Together, they illustrate the evolution of cryptographic strength and design philosophy — from substitution ciphers to key-based symmetric and asymmetric encryption.

**1. Vigenère Cipher**

The Vigenère Cipher is one of the earliest polyalphabetic substitution ciphers, introduced in the 16th century by Blaise de Vigenère. It was considered “unbreakable” for several centuries until modern frequency analysis techniques were developed.

Unlike the Caesar Cipher, which shifts every letter of plaintext by a fixed number, the Vigenère Cipher uses a repeating keyword that determines how much each letter is shifted individually. Each letter of the key represents a shift value (A = 0, B = 1, …, Z = 25), and encryption is done using modular arithmetic over 26 alphabets.

This cipher provides better security than monoalphabetic systems because it distributes letter frequencies, making cryptanalysis harder. However, if the key is short or reused, it can be attacked using methods like the Kasiski examination or Friedman test.

**Advantages:**

* Simple to implement and understand.
* More secure than monoalphabetic ciphers (e.g., Caesar Cipher).
* Useful for demonstrating basic cryptographic principles.

**Disadvantages:**

* Key repetition leads to pattern leakage.
* Vulnerable to frequency analysis and Kasiski attacks.
* Not suitable for modern secure communications.

**2. Vernam Cipher**

The Vernam Cipher, introduced by Gilbert S. Vernam in 1917, marked a turning point in the history of encryption. It was one of the first ciphers designed for use with telegraph systems and operated on binary data rather than letters.

It is a symmetric cipher that uses the XOR (exclusive OR) logical operation between each bit of the plaintext and the corresponding bit of the key. The key must be of the same length as the plaintext, ensuring that each bit is uniquely masked.

When the key is truly random and used only once, the Vernam Cipher becomes the One-Time Pad (OTP) — the only known cipher proven to provide perfect secrecy, as shown by Claude Shannon’s information theory. However, in practical applications, Vernam Cipher’s security depends heavily on key randomness and secure key exchange.

**Advantages:**

* Simple mathematical operation (XOR).
* Resistant to frequency analysis if the key is random.
* Works efficiently in hardware and software applications.

**Disadvantages:**

* Key management is difficult if messages are long.
* Reusing keys compromises security.
* Slightly more complex than classical substitution ciphers.

**3. One-Time Pad (OTP)**

The One-Time Pad (OTP) is a special case of the Vernam Cipher that achieves absolute cryptographic security. It uses a completely random key that is as long as the plaintext and is used only once.

The encryption and decryption operations are both performed using bitwise XOR:

Here, , , and represent the plaintext, ciphertext, and key bits respectively.

OTP is theoretically unbreakable because every possible plaintext of a given length is equally likely for any ciphertext if the key is random. However, its practical use is limited because of the difficulty in generating, distributing, and storing large random keys securely. Despite these limitations, OTP remains a benchmark of ideal cryptographic security.

**Advantages:**

* Provides absolute security (Shannon’s perfect secrecy).
* Immune to cryptanalysis if implemented correctly.
* Conceptually simple but mathematically powerful.

**Disadvantages:**

* Key must be truly random and as long as the message.
* It is difficult to distribute and manage large keys securely.
* Impractical for general-purpose communication.

**4. Data Encryption Standard (DES)**

The Data Encryption Standard (DES), developed by IBM in the early 1970s and adopted as a U.S. federal standard in 1977, represents the beginning of modern computer-based cryptography.

DES is a symmetric block cipher that encrypts data in 64-bit blocks using a 56-bit key. It is based on a structure called a Feistel network, which divides the block into two halves and performs a series of 16 rounds of transformations involving permutation and substitution.

Each round uses a unique subkey derived from the main key through a process of shifting and permutation. The Feistel structure allows the same algorithm to be used for both encryption and decryption.

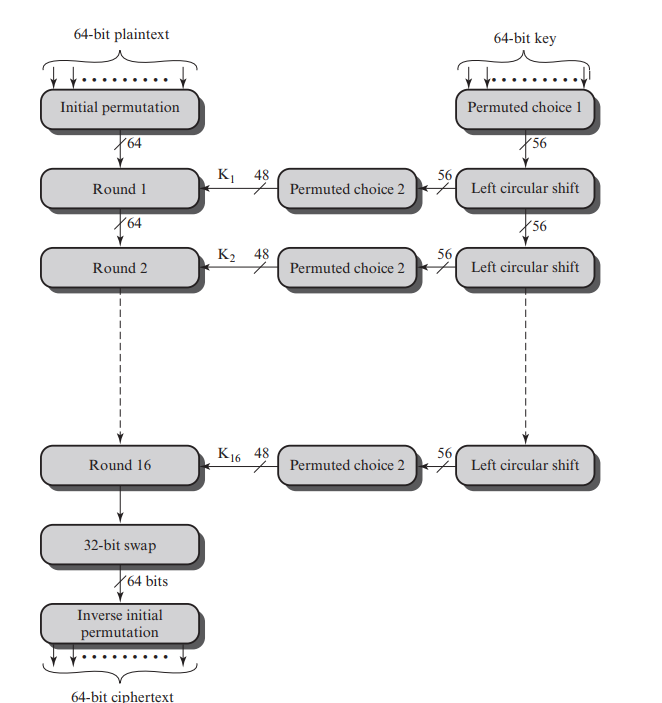
Although DES was once widely used in banking and government systems, advances in computing made its 56-bit key length vulnerable to brute-force attacks. It was later replaced by Triple DES (3DES) and eventually by the Advanced Encryption Standard (AES).

**Advantages:**

* Simple Feistel structure; easy to understand.
* Efficient in hardware.
* Well-studied and historically important.

**Disadvantages:**

* Short 56-bit key; vulnerable to brute-force.
* Small 64-bit block size.
* Obsolete for modern security needs.



**Figure 1:** General Structure of DES Encryption Algorithm.

**5. Advanced Encryption Standard (AES)**

The Advanced Encryption Standard (AES), introduced by the National Institute of Standards and Technology (NIST) in 2001, replaced DES as the modern global standard for symmetric encryption.

AES is based on the Rijndael algorithm, designed by Joan Daemen and Vincent Rijmen. Unlike DES’s Feistel structure, AES uses a Substitution–Permutation Network (SPN) that processes data in 128-bit blocks with key sizes of 128, 192, or 256 bits.

AES performs multiple transformation rounds (10, 12, or 14 depending on key size):

1. **SubBytes:** Non-linear byte substitution using an S-box.
2. **ShiftRows:** Row-wise shifting of bytes to ensure diffusion.
3. **MixColumns:** Mixing of bytes within each column to further diffuse data.
4. **AddRoundKey:** XORing with a subkey derived from the main key.

AES is highly secure, efficient, and resistant to modern cryptanalytic attacks. It is widely used in wireless communication, VPNs, SSL/TLS, banking systems, and data encryption tools.

A diagram of a computer program

AI-generated content may be incorrect.

**Figure 2:** General Structure of AES Encryption Algorithm

**Advantages:**

* Strong security with 128/192/256-bit keys.
* Fast and efficient in hardware/software.
* Worldwide standard for modern encryption.

**Disadvantages:**

* More complex to implement than DES.
* Vulnerable to side-channel attacks if poorly implemented.
* Requires secure key management.

**6. Rivest–Shamir–Adleman (RSA) Algorithm**

The RSA algorithm, developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman, is the most widely used asymmetric (public-key) encryption system.

Unlike symmetric ciphers (which use one key for both encryption and decryption), RSA employs two mathematically related keys:

* A **public key** for encryption, shared openly.
* A **private key** for decryption, kept secret.

RSA’s security is based on the mathematical difficulty of factoring large prime numbers. The algorithm involves:

1. Selecting two large primes and .
2. Computing and .
3. Choosing an encryption key such that and .
4. Computing the decryption key such that .
5. Encryption and decryption follow:

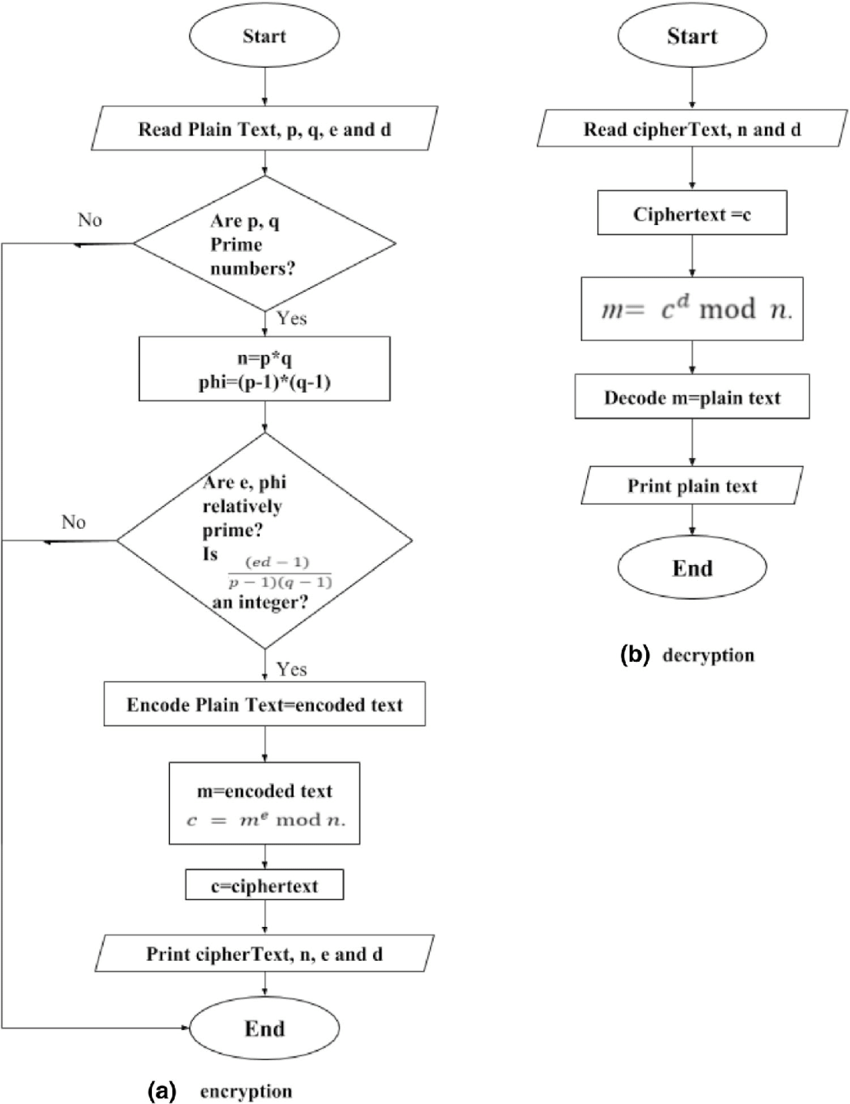
RSA is used for digital signatures, secure key exchange, and authentication in protocols like HTTPS, PGP, and blockchain systems. However, it is computationally slower than symmetric algorithms and vulnerable if small key sizes are used.

**Advantages:**

* Enables secure communication without sharing a secret key (asymmetric).
* Supports digital signatures and authentication.
* Security is based on the difficulty of factoring large prime numbers.

**Disadvantages:**

* Computationally slower than symmetric algorithms (DES/AES).
* Requires large key sizes (1024–4096 bits) for strong security.
* Vulnerable to implementation flaws or poor key management.



**Figure 3:** Flow Chart of RSA Asymmetric Key Encryption and Decryption.

**Comparison Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | Type | Key Length | Block Size | Structure | Security Level | Key Usage | Typical Application |
| Vigenère | Symmetric | Variable (keyword) | Character-based | Substitution | Low | Repeated | Educational use |
| Vernam | Symmetric | Equal to plaintext | Bitwise | XOR | High (if random) | One-time | Stream encryption concept |
| One-Time Pad | Symmetric | Equal to plaintext | Bitwise | XOR | Perfect secrecy | One-time | Military / secure messaging |
| DES | Symmetric | 56 bits | 64 bits | Feistel | Moderate (obsolete) | Reusable | Legacy systems |
| AES | Symmetric | 128/192/256 bits | 128 bits | SPN | Very high | Reusable | Modern encryption (Wi-Fi, SSL) |
| RSA | Asymmetric | 1024–4096 bits | N/A | Modular arithmetic | Very high | Key pair | Digital signatures, secure key exchange |

**Table 1:** Comparison Table of Vigenère, Vernam, One-Time Pad (OTP), DES, AES and RSA.

**Algorithms:**

**1. Vigenère Cipher**

1. Choose a keyword.
2. Repeat the keyword to match the length of the plaintext.
3. Convert letters to numeric values (A=0, B=1, ..., Z=25).
4. Encrypt using:
5. Decrypt using:

**2. Vernam Cipher**

1. Select a key with the same length as the plaintext.
2. Convert both plaintext and key to binary form.
3. Apply XOR bit by bit:
4. For decryption:

**3. One-Time Pad (OTP)**

1. Generate a **random key** equal to plaintext length.
2. Perform bitwise XOR between plaintext and key.
3. Ciphertext is obtained as:
4. Decrypt using the same XOR process.
5. Destroy the key after use to maintain secrecy.

**4. Data Encryption Standard (DES)**

1. Divide plaintext into 64-bit blocks.
2. Apply Initial Permutation (IP).
3. Split data into Left (L) and Right (R) halves.
4. Perform 16 Feistel rounds:
   * Expand R to 48 bits using Expansion (E) function.
   * XOR with round key .
   * Apply Substitution (S-box).
   * Apply Permutation (P-box).
   * Swap halves.
5. Combine L and R, apply Final Permutation (FP).
6. Decrypt by applying keys in reverse order.

**5. Advanced Encryption Standard (AES)**

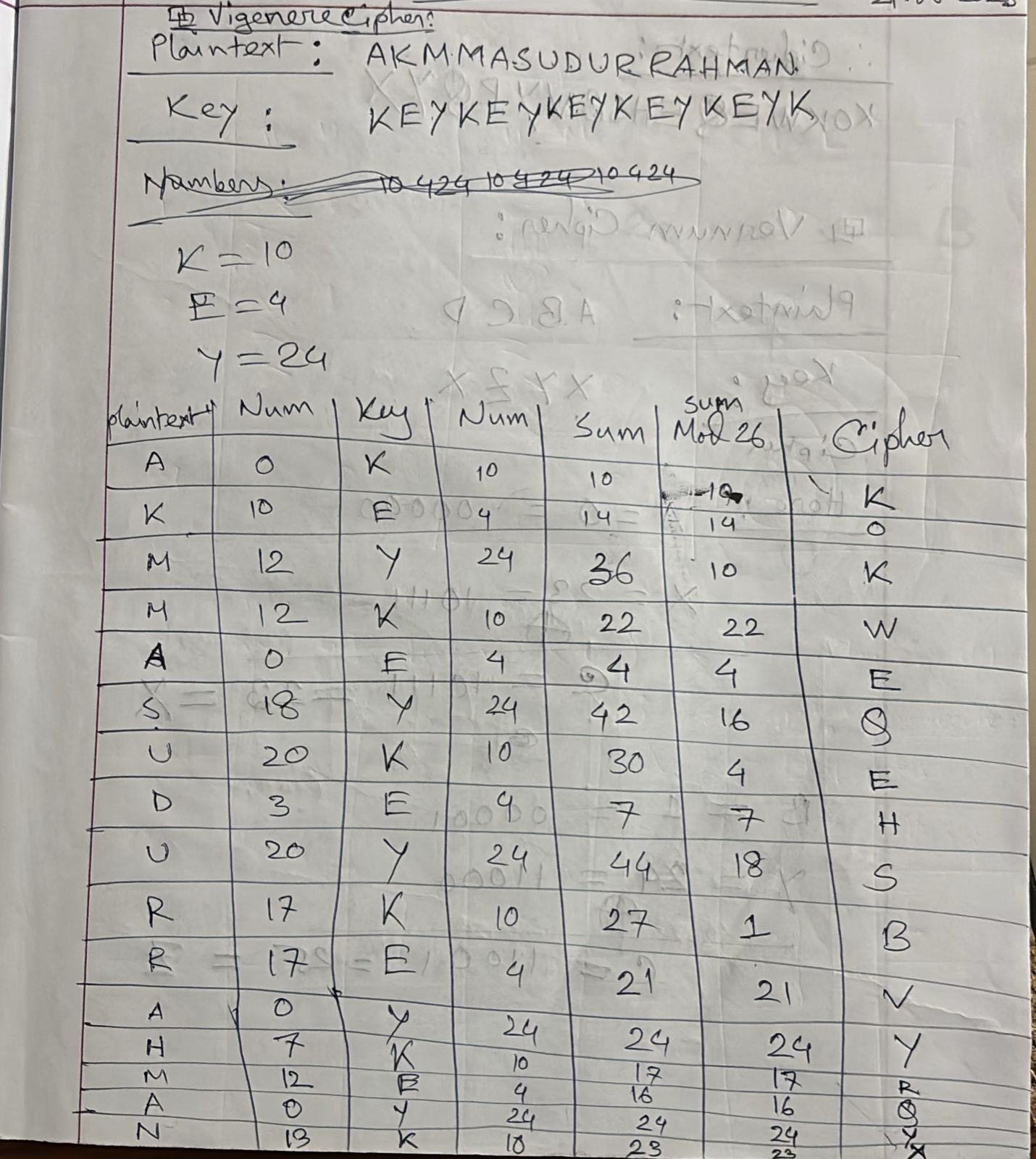
1. Divide data into 128-bit blocks.
2. Perform **AddRoundKey** using the initial key.
3. For 10, 12, or 14 rounds (depending on key size):
   * **SubBytes:** Non-linear byte substitution using S-box.
   * **ShiftRows:** Circular left shift of matrix rows.
   * **MixColumns:** Mix each column via matrix multiplication.
   * **AddRoundKey:** XOR with round key.
4. The final round omits MixColumns.
5. Combine the bytes to form ciphertext.

**6. RSA Algorithm**

1. Choose two primes and .
2. Compute .
3. Compute .
4. Choose such that and gcd(e, φ(n)) = 1.
5. Compute such that .
6. Public key = (e, n), Private key = (d, n).
7. Encryption:
8. Decryption:

**Mathematical Examples**

**1. Vigenère Cipher**



A white board with black writing

AI-generated content may be incorrect.

**2. Vernam Cipher**

A piece of paper with writing on it

AI-generated content may be incorrect.

A piece of paper with writing on it

AI-generated content may be incorrect.

**3. One-Time Pad**

Just Like Vernam Cipher but the key doesn’t repeat and of the same size as the plaintext.

**4. DES Example**

A piece of paper with writing on it

AI-generated content may be incorrect.

A piece of paper with writing on it

AI-generated content may be incorrect.

A piece of paper with writing on it

AI-generated content may be incorrect.

A paper with numbers and symbols

AI-generated content may be incorrect.

**5. AES Example**

A piece of paper with writing on it

AI-generated content may be incorrect.

A paper with writing on it

AI-generated content may be incorrect.

A piece of paper with writing on it

AI-generated content may be incorrect.

A close-up of a paper

AI-generated content may be incorrect.

A piece of paper with writing on it

AI-generated content may be incorrect.

A close-up of a notebook

AI-generated content may be incorrect.

A piece of paper with writing on it

AI-generated content may be incorrect.

**6. RSA Example**

A white paper with writing on it

AI-generated content may be incorrect.

A paper with math equations and numbers

AI-generated content may be incorrect.

A white paper with writing on it

AI-generated content may be incorrect.

**Codes:**

**1. Vigenère Cipher**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int charToNum(char c) {

c = toupper(c);

switch(c) {

case 'A': return 0; case 'B': return 1; case 'C': return 2;

case 'D': return 3; case 'E': return 4; case 'F': return 5;

case 'G': return 6; case 'H': return 7; case 'I': return 8;

case 'J': return 9; case 'K': return 10; case 'L': return 11;

case 'M': return 12; case 'N': return 13; case 'O': return 14;

case 'P': return 15; case 'Q': return 16; case 'R': return 17;

case 'S': return 18; case 'T': return 19; case 'U': return 20;

case 'V': return 21; case 'W': return 22; case 'X': return 23;

case 'Y': return 24; case 'Z': return 25;

default: return -1;

}

}

char numToChar(int n) {

switch(n) {

case 0: return 'A'; case 1: return 'B'; case 2: return 'C';

case 3: return 'D'; case 4: return 'E'; case 5: return 'F';

case 6: return 'G'; case 7: return 'H'; case 8: return 'I';

case 9: return 'J'; case 10: return 'K'; case 11: return 'L';

case 12: return 'M'; case 13: return 'N'; case 14: return 'O';

case 15: return 'P'; case 16: return 'Q'; case 17: return 'R';

case 18: return 'S'; case 19: return 'T'; case 20: return 'U';

case 21: return 'V'; case 22: return 'W'; case 23: return 'X';

case 24: return 'Y'; case 25: return 'Z';

default: return '?';

}

}

void vigenereEncrypt(char plaintext[], char key[], char ciphertext[]) {

int textLen = strlen(plaintext);

int keyLen = strlen(key);

for (int i = 0; i < textLen; i++) {

int p = charToNum(plaintext[i]);

int k = charToNum(key[i % keyLen]);

int c = (p + k) % 26;

ciphertext[i] = numToChar(c);

}

ciphertext[textLen] = '\0';

}

void vigenereDecrypt(char ciphertext[], char key[], char plaintext[]) {

int textLen = strlen(ciphertext);

int keyLen = strlen(key);

for (int i = 0; i < textLen; i++) {

int c = charToNum(ciphertext[i]);

int k = charToNum(key[i % keyLen]);

int p = (c - k + 26) % 26;

plaintext[i] = numToChar(p);

}

plaintext[textLen] = '\0';

}

int main() {

char plaintext[100], key[100], ciphertext[100], decrypted[100];

printf("Enter plaintext (A-Z only): ");

scanf("%s", plaintext);

printf("Enter key (A-Z only): ");

scanf("%s", key);

vigenereEncrypt(plaintext, key, ciphertext);

printf("Encrypted text: %s\n", ciphertext);

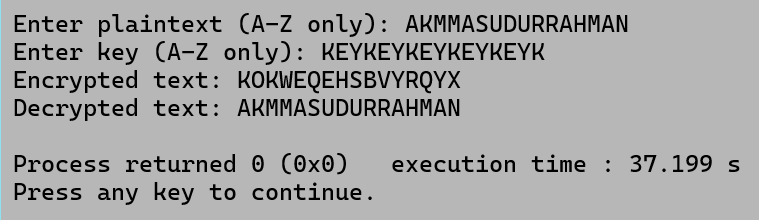
vigenereDecrypt(ciphertext, key, decrypted);

printf("Decrypted text: %s\n", decrypted);

return 0;

}

**Output:**

****

**2. Vernam Cipher**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int charToNum(char c)

{

c = toupper(c);

switch(c)

{

case 'A':

return 0;

case 'B':

return 1;

case 'C':

return 2;

case 'D':

return 3;

case 'E':

return 4;

case 'F':

return 5;

case 'G':

return 6;

case 'H':

return 7;

case 'I':

return 8;

case 'J':

return 9;

case 'K':

return 10;

case 'L':

return 11;

case 'M':

return 12;

case 'N':

return 13;

case 'O':

return 14;

case 'P':

return 15;

case 'Q':

return 16;

case 'R':

return 17;

case 'S':

return 18;

case 'T':

return 19;

case 'U':

return 20;

case 'V':

return 21;

case 'W':

return 22;

case 'X':

return 23;

case 'Y':

return 24;

case 'Z':

return 25;

default:

return -1;

}

}

char numToChar(int n)

{

switch(n)

{

case 0:

return 'A';

case 1:

return 'B';

case 2:

return 'C';

case 3:

return 'D';

case 4:

return 'E';

case 5:

return 'F';

case 6:

return 'G';

case 7:

return 'H';

case 8:

return 'I';

case 9:

return 'J';

case 10:

return 'K';

case 11:

return 'L';

case 12:

return 'M';

case 13:

return 'N';

case 14:

return 'O';

case 15:

return 'P';

case 16:

return 'Q';

case 17:

return 'R';

case 18:

return 'S';

case 19:

return 'T';

case 20:

return 'U';

case 21:

return 'V';

case 22:

return 'W';

case 23:

return 'X';

case 24:

return 'Y';

case 25:

return 'Z';

default:

return '?';

}

}

void vernamCipher(char text[], char key[], char result[])

{

int textLen = strlen(text);

int keyLen = strlen(key);

for (int i = 0; i < textLen; i++)

{

int p = charToNum(text[i]);

int k = charToNum(key[i % keyLen]);

int c = (p ^ k) % 26;

result[i] = numToChar(c);

}

result[textLen] = '\0';

}

int main()

{

char plaintext[100], key[100], ciphertext[100];

printf("Enter plaintext: ");

scanf("%s", plaintext);

printf("Enter key: ");

scanf("%s", key);

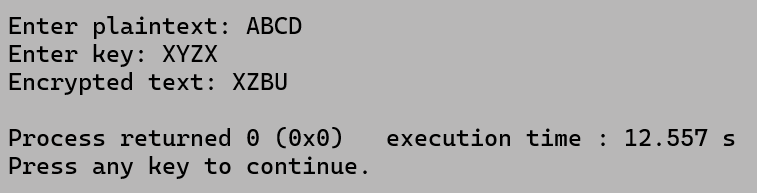
vernamCipher(plaintext, key, ciphertext);

printf("Encrypted text: %s\n", ciphertext);

return 0;

}

**Output:**

****

**3. One-Time Pad**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int charToNum(char c)

{

c = toupper(c);

switch(c)

{

case 'A':

return 0;

case 'B':

return 1;

case 'C':

return 2;

case 'D':

return 3;

case 'E':

return 4;

case 'F':

return 5;

case 'G':

return 6;

case 'H':

return 7;

case 'I':

return 8;

case 'J':

return 9;

case 'K':

return 10;

case 'L':

return 11;

case 'M':

return 12;

case 'N':

return 13;

case 'O':

return 14;

case 'P':

return 15;

case 'Q':

return 16;

case 'R':

return 17;

case 'S':

return 18;

case 'T':

return 19;

case 'U':

return 20;

case 'V':

return 21;

case 'W':

return 22;

case 'X':

return 23;

case 'Y':

return 24;

case 'Z':

return 25;

default:

return -1;

}

}

char numToChar(int n)

{

switch(n)

{

case 0:

return 'A';

case 1:

return 'B';

case 2:

return 'C';

case 3:

return 'D';

case 4:

return 'E';

case 5:

return 'F';

case 6:

return 'G';

case 7:

return 'H';

case 8:

return 'I';

case 9:

return 'J';

case 10:

return 'K';

case 11:

return 'L';

case 12:

return 'M';

case 13:

return 'N';

case 14:

return 'O';

case 15:

return 'P';

case 16:

return 'Q';

case 17:

return 'R';

case 18:

return 'S';

case 19:

return 'T';

case 20:

return 'U';

case 21:

return 'V';

case 22:

return 'W';

case 23:

return 'X';

case 24:

return 'Y';

case 25:

return 'Z';

default:

return '?';

}

}

void vernamCipher(char text[], char key[], char result[])

{

int textLen = strlen(text);

int keyLen = strlen(key);

if(textLen != keyLen) {

printf("Error! Key Must be Equal to the Plaintext!\n");

result[0] = '\0';

return;

}

for (int i = 0; i < textLen; i++)

{

int p = charToNum(text[i]);

int k = charToNum(key[i % keyLen]);

int c = (p ^ k) % 26;

result[i] = numToChar(c);

}

result[textLen] = '\0';

}

int main()

{

char plaintext[100], key[100], ciphertext[100];

printf("Enter plaintext: ");

scanf("%s", plaintext);

printf("Enter key: ");

scanf("%s", key);

vernamCipher(plaintext, key, ciphertext);

printf("Encrypted text: %s\n", ciphertext);

return 0;

}

**Output:**

**A grey background with black text

AI-generated content may be incorrect.**

**4. DES Example**

#include <stdio.h>

#include <string.h>

int PC1[56] = {57,49,41,33,25,17,9,1,58,50,42,34,26,18,

10,2,59,51,43,35,27,19,11,3,60,52,44,36,

63,55,47,39,31,23,15,7,62,54,46,38,30,22,

14,6,61,53,45,37,29,21,13,5,28,20,12,4

};

int PC2[48] = {14,17,11,24,1,5,3,28,15,6,21,10,

23,19,12,4,26,8,16,7,27,20,13,2,

41,52,31,37,47,55,30,40,51,45,33,48,

44,49,39,56,34,53,46,42,50,36,29,32

};

int SHIFTS[16] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

int E\_TABLE[48] = {32,1,2,3,4,5,4,5,6,7,8,9,

8,9,10,11,12,13,12,13,14,15,16,17,

16,17,18,19,20,21,20,21,22,23,24,25,

24,25,26,27,28,29,28,29,30,31,32,1

};

void permute(char \*in, char \*out, int \*table, int n)

{

for(int i=0; i<n; i++) out[i] = in[table[i]-1];

}

void leftShift28(char \*in, char \*out, int shift)

{

for(int i=0; i<28; i++) out[i] = in[(i+shift)%28];

}

void xorBits(char \*a, char \*b, char \*out, int n)

{

for(int i=0; i<n; i++) out[i] = (a[i]==b[i])?'0':'1';

}

void expansionE(char \*R, char \*ER)

{

for(int i=0; i<48; i++) ER[i] = R[E\_TABLE[i]-1];

}

int main()

{

char plaintext[65] = "0000000100100011010001010110011110001001101010111100110111101111";

char key[65] = "0001001100110100010101110111100110011011101111001101111101000001";

char PC1\_out[57], C[4][29], D[4][29], CD[57], K[4][49];

permute(key, PC1\_out, PC1, 56);

strncpy(C[0], PC1\_out,28);

C[0][28]='\0';

strncpy(D[0], PC1\_out+28,28);

D[0][28]='\0';

printf("C0=%s\nD0=%s\n\n",C[0],D[0]);

for(int i=1; i<=3; i++)

{

leftShift28(C[i-1], C[i], SHIFTS[i-1]);

leftShift28(D[i-1], D[i], SHIFTS[i-1]);

strncpy(CD, C[i],28);

strncpy(CD+28, D[i],28);

CD[56]='\0';

permute(CD, K[i], PC2,48);

K[i][48]='\0';

printf("Round %d key K%d = %s\n\n", i,i,K[i]);

}

char L[4][33], R[4][33];

strncpy(L[0], plaintext,32);

L[0][32]='\0';

strncpy(R[0], plaintext+32,32);

R[0][32]='\0';

for(int i=1; i<=3; i++)

{

char ER[49], xorRes[49];

expansionE(R[i-1], ER);

ER[48]='\0';

xorBits(ER,K[i],xorRes,48);

xorRes[48]='\0';

strncpy(L[i], R[i-1],32);

L[i][32]='\0';

for(int j=0; j<32; j++)

R[i][j] = L[i-1][j]^xorRes[j]?'1':'0';

R[i][32]='\0';

printf("Round %d L%d=%s\n", i,i,L[i]);

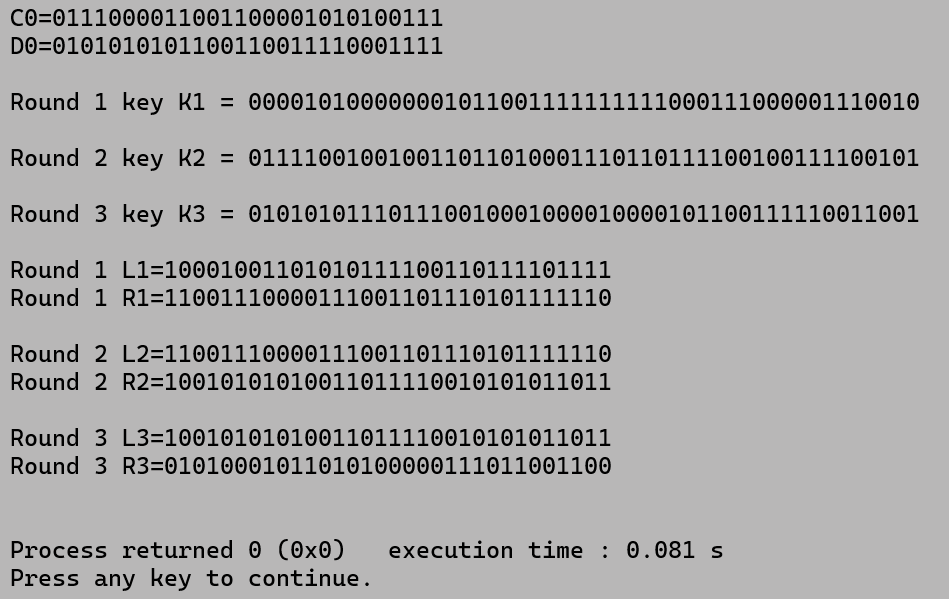
printf("Round %d R%d=%s\n\n", i,i,R[i]);

}

return 0;

}

**Output:**

****

**5. AES Example**

#include <stdio.h>

#include <stdint.h>

uint8\_t sbox[256] =

{

0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,

0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,

0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,

0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,

0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,

0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,

0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,

0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,

0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,

0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,

0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,

0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,

0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,

0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,

0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,

0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16

};

uint8\_t Rcon[11] = {0x00,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36};

void RotWord(uint8\_t\* w) { uint8\_t t=w[0]; w[0]=w[1]; w[1]=w[2]; w[2]=w[3]; w[3]=t; }

void SubWord(uint8\_t\* w) { for(int i=0;i<4;i++) w[i]=sbox[w[i]]; }

void KeyExpansion(uint8\_t key[16], uint8\_t roundKey[16])

{

uint8\_t w0[4],w1[4],w2[4],w3[4],w4[4],w5[4],w6[4],w7[4],temp[4];

for(int i=0;i<4;i++) { w0[i]=key[i]; w1[i]=key[i+4]; w2[i]=key[i+8]; w3[i]=key[i+12]; }

for(int i=0;i<4;i++) temp[i]=w3[i];

RotWord(temp); SubWord(temp); temp[0]^=Rcon[1];

for(int i=0;i<4;i++) w4[i]=w0[i]^temp[i];

for(int i=0;i<4;i++) w5[i]=w1[i]^w4[i];

for(int i=0;i<4;i++) w6[i]=w2[i]^w5[i];

for(int i=0;i<4;i++) w7[i]=w3[i]^w6[i];

for(int i=0;i<4;i++)

{

roundKey[i]=w4[i]; roundKey[i+4]=w5[i]; roundKey[i+8]=w6[i]; roundKey[i+12]=w7[i];

}

}

void AddRoundKey(uint8\_t state[16], uint8\_t key[16]) { for(int i=0;i<16;i++) state[i]^=key[i]; }

void SubBytes(uint8\_t state[16]) { for(int i=0;i<16;i++) state[i]=sbox[state[i]]; }

void ShiftRows(uint8\_t state[16])

{

uint8\_t t;

t=state[1]; state[1]=state[5]; state[5]=state[9]; state[9]=state[13]; state[13]=t;

t=state[2]; state[2]=state[10]; state[10]=t;

t=state[6]; state[6]=state[14]; state[14]=t;

t=state[3]; state[3]=state[15]; state[15]=state[11]; state[11]=state[7]; state[7]=t;

}

uint8\_t xtime(uint8\_t x) { return ((x<<1)^((x&0x80)?0x1b:0x00)); }

void MixColumns(uint8\_t state[16])

{

uint8\_t t[16];

for(int c=0;c<4;c++)

{

int i=c\*4;

t[i] = xtime(state[i])^(xtime(state[i+1])^state[i+1])^state[i+2]^state[i+3];

t[i+1] = state[i]^xtime(state[i+1])^(xtime(state[i+2])^state[i+2])^state[i+3];

t[i+2] = state[i]^state[i+1]^xtime(state[i+2])^(xtime(state[i+3])^state[i+3]);

t[i+3] = (xtime(state[i])^state[i])^state[i+1]^state[i+2]^xtime(state[i+3]);

}

for(int i=0;i<16;i++) state[i]=t[i];

}

void PrintState(uint8\_t state[16])

{

for(int i=0;i<16;i++){ printf("%02x ", state[i]); if((i+1)%4==0) printf("\n"); }

printf("\n");

}

void PrintKey(uint8\_t key[16], const char\* name)

{

printf("%s:\n", name);

for(int i=0;i<16;i++){ printf("%02x ", key[i]); if((i+1)%4==0) printf("\n"); }

printf("\n");

}

int main()

{

uint8\_t key[16]= {0x2b,0x7e,0x15,0x16,0x28,0xae,0xd2,0xa6,0xab,0xf7,0x15,0x88,0x09,0xcf,0x4f,0x3c};

uint8\_t plaintext[16]= {0x32,0x43,0xf6,0xa8,0x88,0x5a,0x30,0x8d,0x31,0x31,0x98,0xa2,0xe0,0x37,0x07,0x34};

uint8\_t roundKey[16], roundKey2[16];

KeyExpansion(key, roundKey);

PrintKey(key, "Initial Key (used in AddRoundKey 0)");

printf("After AddRoundKey(0):\n");

AddRoundKey(plaintext,key);

PrintState(plaintext);

SubBytes(plaintext);

printf("After SubBytes:\n");

PrintState(plaintext);

ShiftRows(plaintext);

printf("After ShiftRows:\n");

PrintState(plaintext);

MixColumns(plaintext);

printf("After MixColumns:\n");

PrintState(plaintext);

PrintKey(roundKey, "Round 1 Key (used after MixColumns)");

AddRoundKey(plaintext,roundKey);

printf("After Round 1:\n");

PrintState(plaintext);

KeyExpansion(roundKey, roundKey2);

SubBytes(plaintext);

printf("Round 2 - After SubBytes:\n");

PrintState(plaintext);

ShiftRows(plaintext);

printf("Round 2 - After ShiftRows:\n");

PrintState(plaintext);

PrintKey(roundKey2, "Round 2 Key (used in AddRoundKey 2)");

AddRoundKey(plaintext, roundKey2);

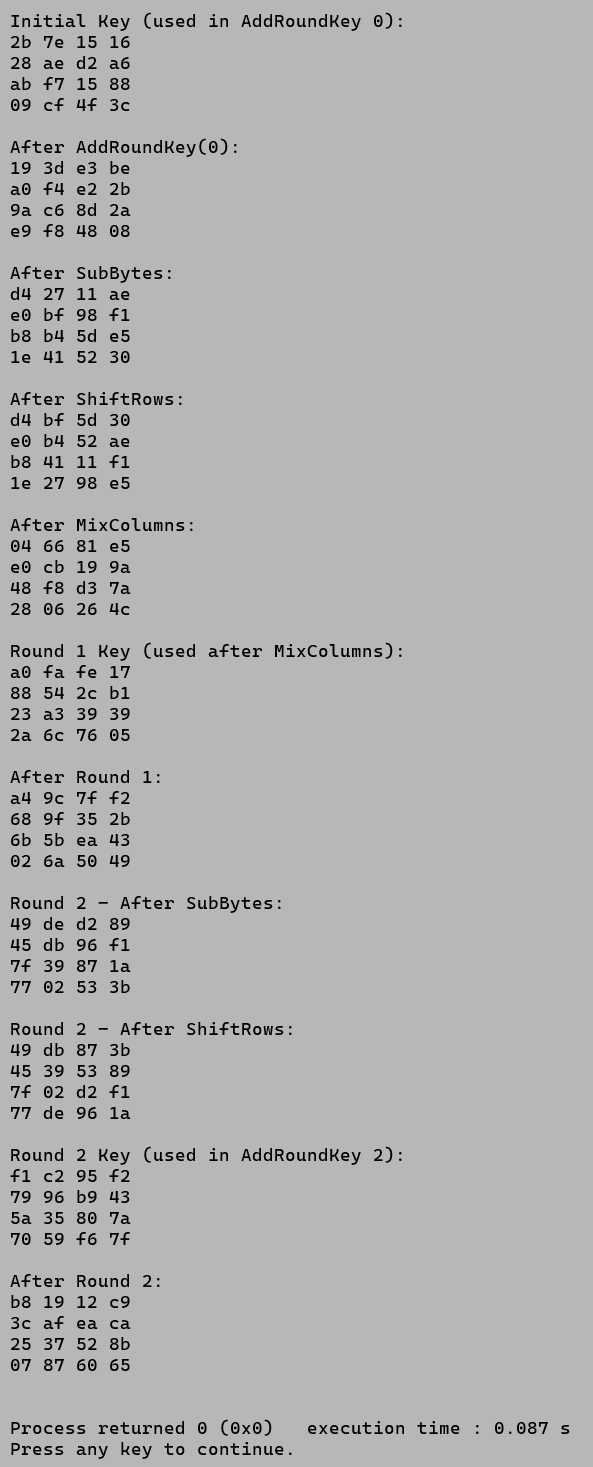
printf("After Round 2:\n");

PrintState(plaintext);

return 0;

}

**Output:**

****

**6. RSA Example**

#include <stdio.h>

int gcd(int a, int b) {

if (b == 0)

return a;

return gcd(b, a % b);

}

int modInverse(int e, int phi) {

int d;

for (d = 1; d < phi; d++) {

if (((d \* e) % phi) == 1)

return d;

}

return -1;

}

int main() {

int p, q;

printf("Enter the first Prime Number, p = ");

scanf("%d", &p);

printf("Enter the second Prime Number, q = ");

scanf("%d", &q);

printf("\n");

int n = p \* q;

int phi = (p - 1) \* (q - 1);

int e;

for (e = 2; e < phi; e++) {

if (gcd(e, phi) == 1)

break;

}

int d = modInverse(e, phi);

printf("\n========== RSA KEY GENERATION ==========\n");

printf("Step 1: Selected primes: p = %d, q = %d\n", p, q);

printf("Step 2: Compute n = p \* q = %d \* %d = %d\n", p, q, n);

printf("Step 3: Compute phi(n) = (p-1)\*(q-1) = %d\n", phi);

printf("Step 4: Select e such that 1 < e < phi(n) and gcd(e, phi(n)) = 1\n");

printf("Let, e = %d\n", e);

printf("Step 5: Compute d such that (d \* e) mod phi(n) = 1\n");

printf("Calculated d = %d\n", d);

printf("\n");

printf("\n==========================================\n");

printf("Public Key (e, n) = (%d, %d)\n", e, n);

printf("Private Key (d, n) = (%d, %d)\n", d, n);

printf("\n========================================\n\n");

int msg;

printf("Enter a message (integer < %d) to encrypt: ", n);

scanf("%d", &msg);

long long int C = 1;

for (int i = 0; i < e; i++)

C = (C \* msg) % n;

long long int M = 1;

for (int i = 0; i < d; i++)

M = (M \* C) % n;

printf("\n========== RSA ENCRYPTION ==========\n");

printf("Plain message (M) = %d\n", msg);

printf("Ciphertext (C) = (M^e) mod n = (%d^%d) mod %d = %lld\n", msg, e, n, C);

printf("====================================\n");

printf("\n========== RSA DECRYPTION ==========\n");

printf("Ciphertext (C) = %lld\n", C);

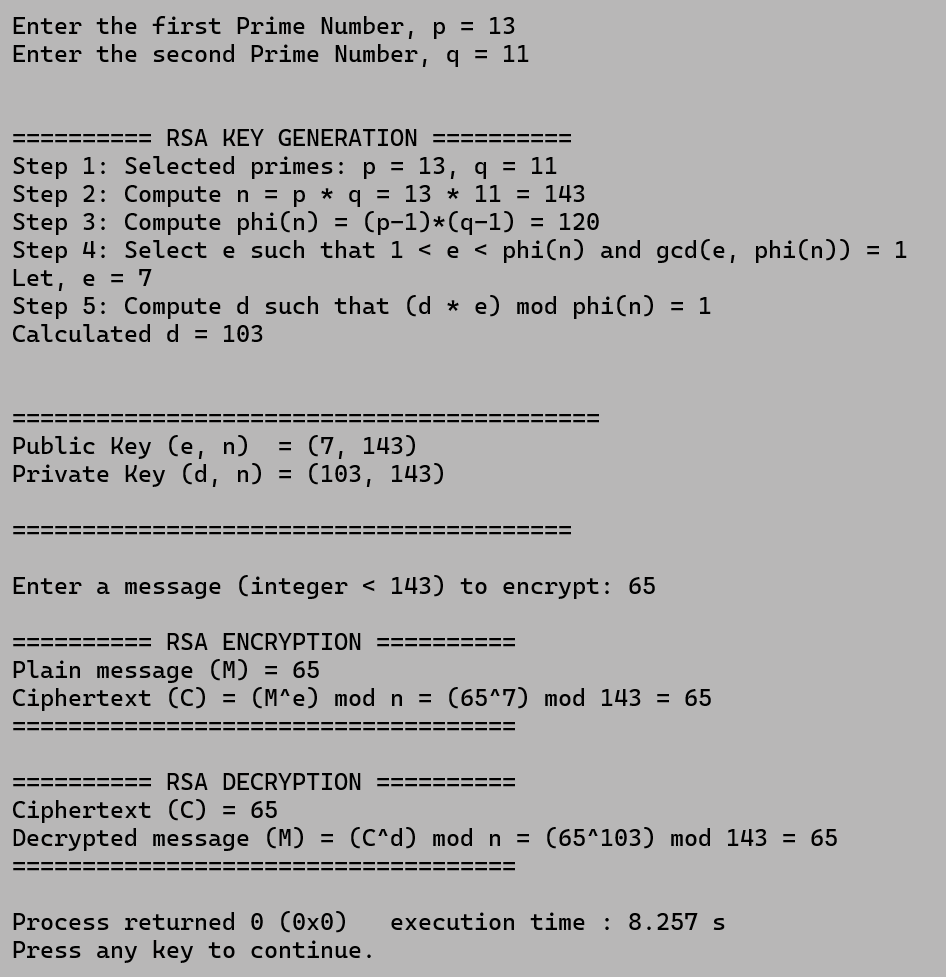
printf("Decrypted message (M) = (C^d) mod n = (%lld^%d) mod %d = %lld\n", C, d, n, M);

printf("====================================\n");

return 0;

}

**Output:**

****

**Conclusion**

This experiment illustrates the evolution of cryptography from simple substitution systems to advanced mathematical algorithms. The Vigenère Cipher demonstrates classical encryption concepts, while the Vernam and One-Time Pad introduce the notion of perfect secrecy. DES established the foundation for modern block ciphers but was eventually replaced by AES, which remains a global standard for symmetric encryption. RSA, as an asymmetric system, enabled secure digital communication through public-key cryptography. Each algorithm differs in speed, complexity, and purpose, but together they showcase the technological advancement in securing information — from manual encoding to mathematically proven modern encryption.