University of Engineering and Technology, Lahore

Department of Computer Science

Project Report

Team Members

Name	Roll Number	Email
Duaa Zehra	2025-CS-134	tcs179669@gmail.com
Alishba Rafi	2025-CS-162	-

Session: Fall'25 Morning **Section:** C **Semester:** 1st

Course: CSC101 - Discrete Mathematics **Teacher:** Mr. Waqas Ali

January 4th, 2026

Table of Contents

Contents

Project Report

Submission Deadline: January 6, 2026

1. BASIC CRYPTOGRAPHIC-SUITE

2. CHOSEN TRACK

- **Track 5:** Number Theory Applications

3. Summary

This project implements a **Basic Cryptographic Suite** that demonstrates how fundamental concepts from **number theory and discrete mathematics** are applied in classical and modern cryptography. The problem addressed by this project is understanding how mathematical theory—particularly modular arithmetic, greatest common divisors, and prime-based computations — forms secure communication systems.

The project includes implementations of **Caesar Cipher**, **Affine Cipher**, and **RSA Cryptosystem**, allowing users to encrypt and decrypt text through a menu-driven command-line interface. Each algorithm was implemented from first principles rather than relying on external cryptographic libraries, ensuring a strong emphasis on mathematical correctness and learning.

The methodology involved identifying the mathematical foundations behind each cipher, translating those principles into efficient algorithms, and validating correctness through input validation and modular arithmetic techniques. Functions such as Euclid's Algorithm, Extended Euclidean Algorithm, and fast modular exponentiation were implemented to support encryption and decryption processes.

Key achievements include:

- A fully functional CLI-based cryptographic toolkit
- Correct handling of modular inverses and coprime constraints
- RSA encryption with block-based message encoding
- Robust input validation to prevent invalid operations

This project is significant in the context of discrete mathematics because it demonstrates how abstract mathematical concepts directly enable real-world security mechanisms. By bridging theory and implementation, the project reinforces the practical importance of number theory in computer science and cybersecurity.

4. Technology Stack & Libraries Utilized

- **Programming Languages:** C++
  Chosen for its high performance and strong support for low-level arithmetic operations essential in cryptography.

- **Libraries:**

  `<iostream>` – Used for input/output operations.

`<string>` – Used for text handling and manipulation.

These standard libraries were selected due to their reliability, portability, and availability without external dependencies.

- **Development Environment:**

  IDE: Visual Studio Code

  Compiler: GCC / MinGW (C++17 standard)

  Version Control: Git (GitHub)

5. List of Features

1. **Caesar Cipher Encryption**
   Shifts letters forward by a fixed key value.
   [Core Feature]
2. **Caesar Cipher Decryption**
   Reverses Caesar encryption using the same key.
   [Core Feature]
3. **Affine Cipher Encryption**
   Encrypts text using the function $C = (aP + b) \bmod 26$.
   [Core Feature]
4. **Affine Cipher Decryption**
   Uses modular inverse of a to recover plaintext.
   [Core Feature]
5. **RSA Encryption**
   Encrypts blocks of plaintext using public-key cryptography.
   [Core
6. **RSA Decryption**
   Uses private key generation via Extended Euclidean Algorithm.
   [Core Feature]
7. **Input Validation System**
   Prevents invalid menu selections and non-numeric input.
   [Core Feature]

6. Mathematical Foundations

Relevant Concepts

- **Modular Arithmetic**
- **Greatest Common Divisor (GCD)**
- **Coprime Integers**
- **Prime Numbers**
- **Modular Inverse**
- **Exponentiation by Squaring**

Formal Definitions

- **GCD:**
  $\gcd(a, b)$is the largest integer dividing both $a$and b.

- **Modular Inverse:**
  An integer a^{-1}such that

$$a \cdot a^{-1} \equiv 1 \pmod{m}$$

- **RSA Encryption:**

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

- **RSA Decryption:**

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

Proofs & Correctness

- **Correctness of GCD:**
  Euclid's Algorithm terminates because the remainder strictly decreases.

- **Correctness of RSA:**
  Based on Euler's Theorem:

$$M^{\phi(n)} \equiv 1 \bmod n$$

- **Time Complexity:**
  - GCD: $O(\log n)$
  - Modular Exponentiation: $O(\log e)$

Illustrative Example

Affine Encryption Example:

- Plaintext: A → 0
- $a = 5, b = 8$
- $C = (5 \cdot 0 + 8) \bmod 26 = I$

7. Examination of Key Functions/Methods

Function 1: `gcd(int a, int b)`

Purpose: Checks coprimality for Affine and RSA keys
Complexity: $O(\log n)$

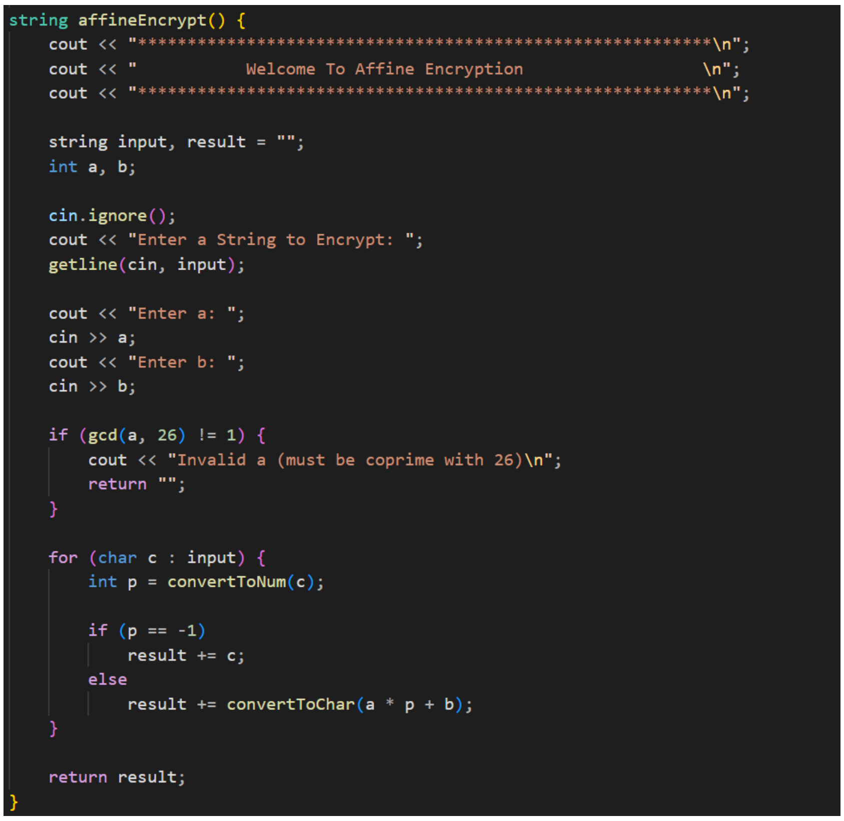
```
int gcd(int a, int b) {
    while (b != 0) {
        int t = b;
        b = a % b;
        a = t;
    }
    return a;
}
```

Function 2: `extendedGCD(int a, int b, int &x, int &y)`

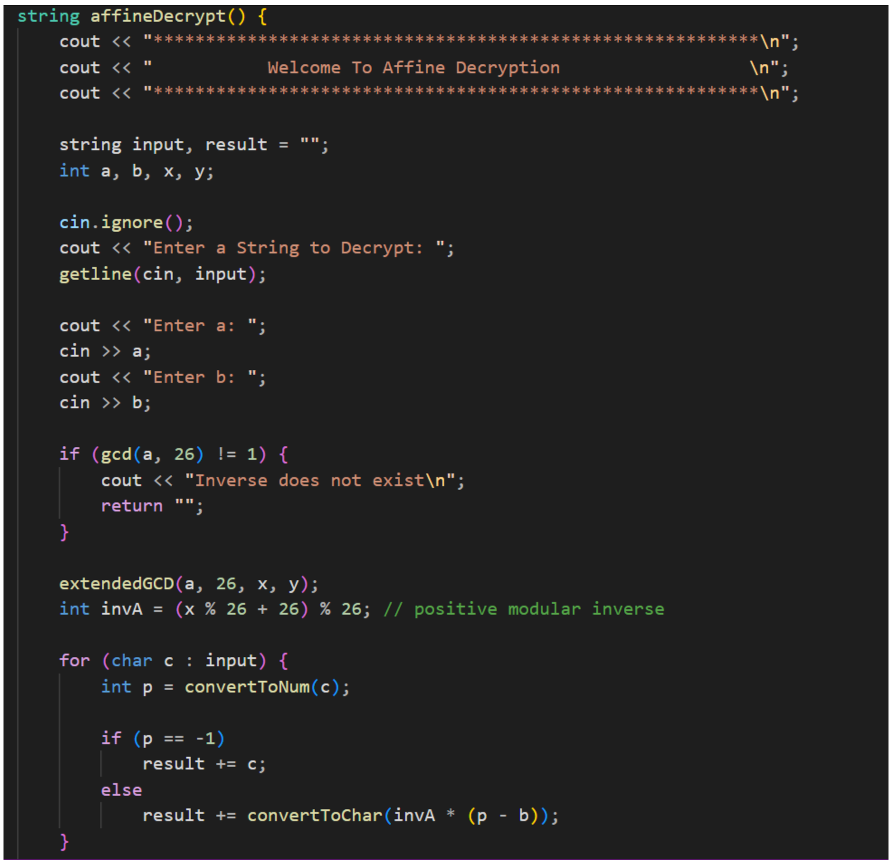
Purpose: Computes modular inverse
Importance: Essential for RSA and Affine decryption
Complexity: $O(\log n)$

Function 3: `modPow(int base, int exp, int mod)`

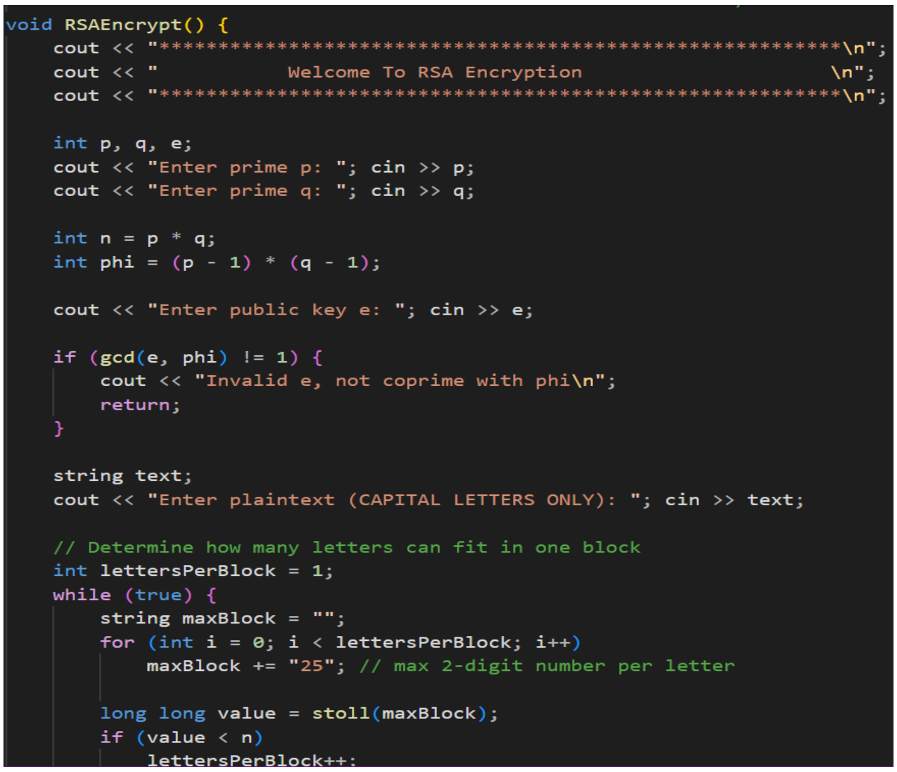
Purpose: Efficient modular exponentiation
Optimization: Exponentiation by squaring
Complexity: $O(\log exp)$



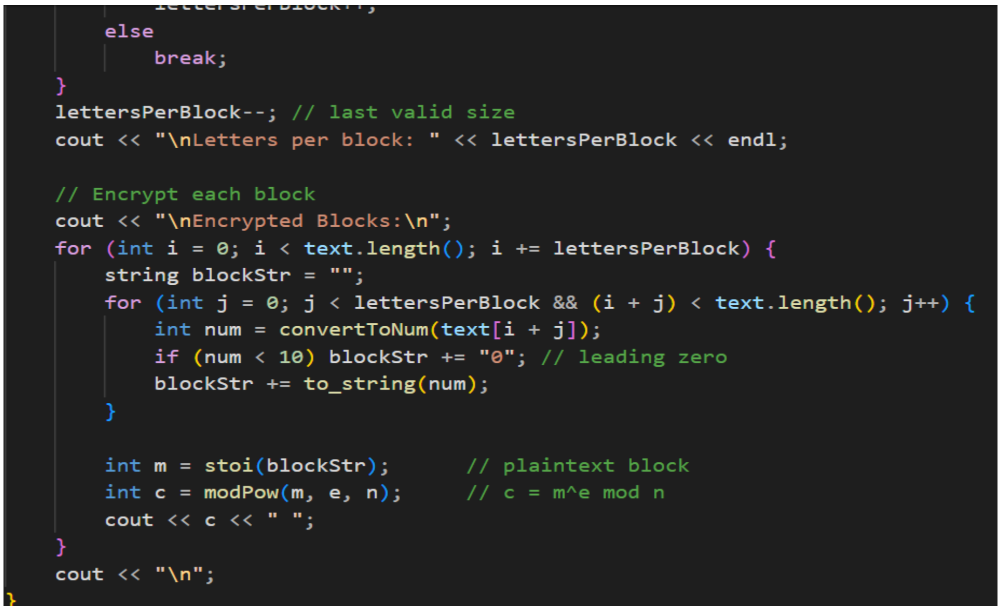
```
string affineEncrypt() {
    cout << "***********************************************************\n";
    cout << "           Welcome To Affine Encryption                  \n";
    cout << "***********************************************************\n";

    string input, result = "";
    int a, b;

    cin.ignore();
    cout << "Enter a String to Encrypt: ";
    getline(cin, input);

    cout << "Enter a: ";
    cin >> a;
    cout << "Enter b: ";
    cin >> b;

    if (gcd(a, 26) != 1) {
        cout << "Invalid a (must be coprime with 26)\n";
        return "";
    }

    for (char c : input) {
        int p = convertToNum(c);

        if (p == -1)
            result += c;
        else
            result += convertToChar(a * p + b);
    }

    return result;
}
```



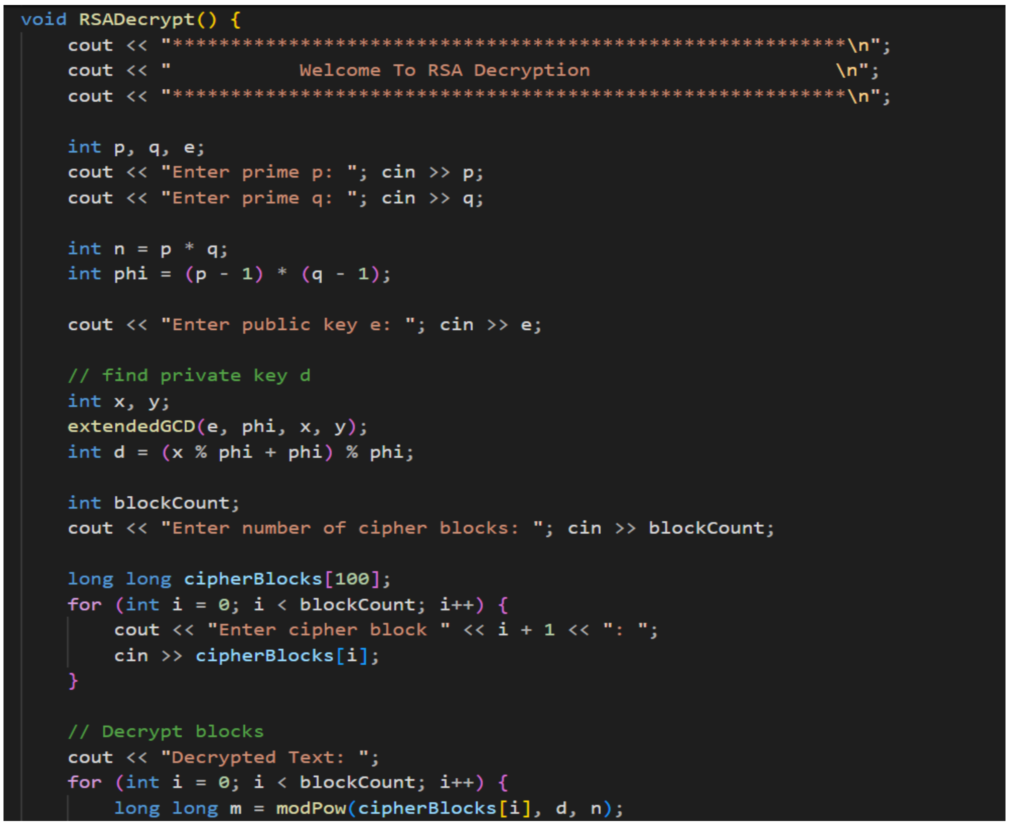
```cpp
string affineDecrypt() {
    cout << "***********************************************************\n";
    cout << "           Welcome To Affine Decryption                  \n";
    cout << "***********************************************************\n";

    string input, result = "";
    int a, b, x, y;

    cin.ignore();
    cout << "Enter a String to Decrypt: ";
    getline(cin, input);

    cout << "Enter a: ";
    cin >> a;
    cout << "Enter b: ";
    cin >> b;

    if (gcd(a, 26) != 1) {
        cout << "Inverse does not exist\n";
        return "";
    }

    extendedGCD(a, 26, x, y);
    int invA = (x % 26 + 26) % 26; // positive modular inverse

    for (char c : input) {
        int p = convertToNum(c);

        if (p == -1)
            result += c;
        else
            result += convertToChar(invA * (p - b));
    }
```



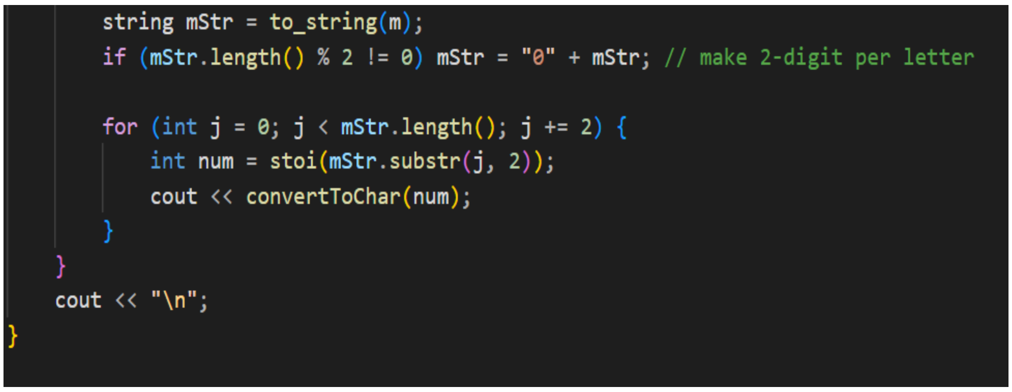
```cpp
void RSAEncrypt() {
    cout << "***********************************************************\n";
    cout << "           Welcome To RSA Encryption                     \n";
    cout << "***********************************************************\n";

    int p, q, e;
    cout << "Enter prime p: "; cin >> p;
    cout << "Enter prime q: "; cin >> q;

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

    cout << "Enter public key e: "; cin >> e;

    if (gcd(e, phi) != 1) {
        cout << "Invalid e, not coprime with phi\n";
        return;
    }

    string text;
    cout << "Enter plaintext (CAPITAL LETTERS ONLY): "; cin >> text;

    // Determine how many letters can fit in one block
    int lettersPerBlock = 1;
    while (true) {
        string maxBlock = "";
        for (int i = 0; i < lettersPerBlock; i++)
            maxBlock += "25"; // max 2-digit number per letter

        long long value = stoll(maxBlock);
        if (value < n)
            lettersPerBlock++;
```



```
            lettersPerBlock++;
        else
            break;
    }
    lettersPerBlock--; // last valid size
    cout << "\nLetters per block: " << lettersPerBlock << endl;

    // Encrypt each block
    cout << "\nEncrypted Blocks:\n";
    for (int i = 0; i < text.length(); i += lettersPerBlock) {
        string blockStr = "";
        for (int j = 0; j < lettersPerBlock && (i + j) < text.length(); j++) {
            int num = convertToNum(text[i + j]);
            if (num < 10) blockStr += "0"; // leading zero
            blockStr += to_string(num);
        }

        int m = stoi(blockStr);      // plaintext block
        int c = modPow(m, e, n);     // c = m^e mod n
        cout << c << " ";
    }
    cout << "\n";
}
```



```
void RSADecrypt() {
    cout << "**********************************************************\n";
    cout << "            Welcome To RSA Decryption                     \n";
    cout << "**********************************************************\n";

    int p, q, e;
    cout << "Enter prime p: "; cin >> p;
    cout << "Enter prime q: "; cin >> q;

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

    cout << "Enter public key e: "; cin >> e;

    // find private key d
    int x, y;
    extendedGCD(e, phi, x, y);
    int d = (x % phi + phi) % phi;

    int blockCount;
    cout << "Enter number of cipher blocks: "; cin >> blockCount;

    long long cipherBlocks[100];
    for (int i = 0; i < blockCount; i++) {
        cout << "Enter cipher block " << i + 1 << ": ";
        cin >> cipherBlocks[i];
    }

    // Decrypt blocks
    cout << "Decrypted Text: ";
    for (int i = 0; i < blockCount; i++) {
        long long m = modPow(cipherBlocks[i], d, n);
```



```
        string mStr = to_string(m);
        if (mStr.length() % 2 != 0) mStr = "0" + mStr; // make 2-digit per letter

        for (int j = 0; j < mStr.length(); j += 2) {
            int num = stoi(mStr.substr(j, 2));
            cout << convertToChar(num);
        }
    }
    cout << "\n";
}
```

8. Limitations

- Scope: Not suitable for real-world secure communication
- RSA Constraints: Small primes only
- Performance: No big integer support
- Input Assumptions: RSA plaintext assumes uppercase letters
- Security: No padding or hashing

10. User Interface

Interface Overview

The project uses a **menu-driven command-line interface** focused on simplicity and clarity.

Screen Documentation

Main Menu

- Options 1–9 for cipher selection
- Numeric input only

Cipher Screens

- Text input
- Key input
- Encrypted/Decrypted output displayed immediately

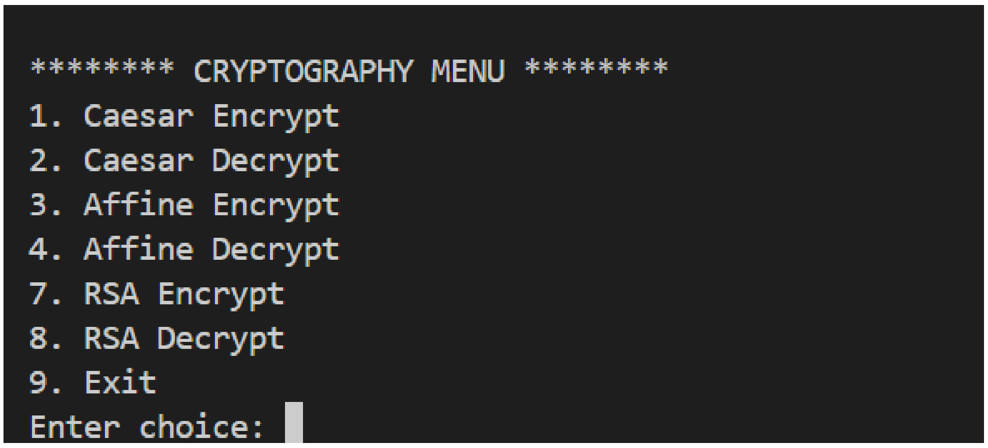
Sample Test Cases

Test Case 1: Caesar Encrypt

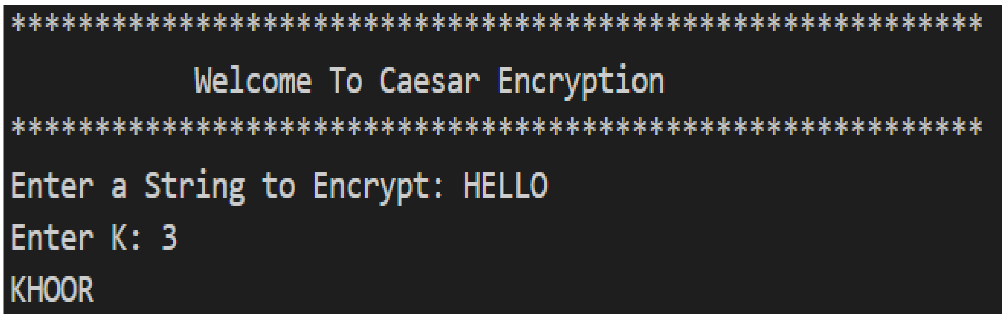
- Input: `HELLO`, K = 3
- Output: `KHOOR`
- Status: Pass

Test Case 2: Affine Decrypt

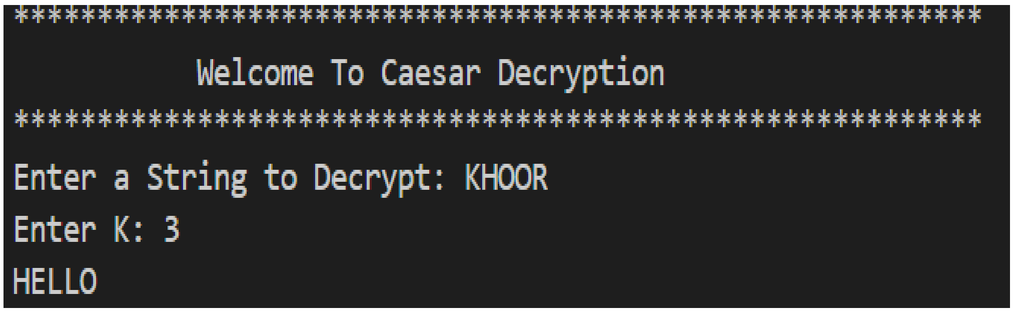
- Input: `IHHWVCSWFRCP`, a=5, b=8
- Output: `AFFINECIPHER`
- Status: Pass



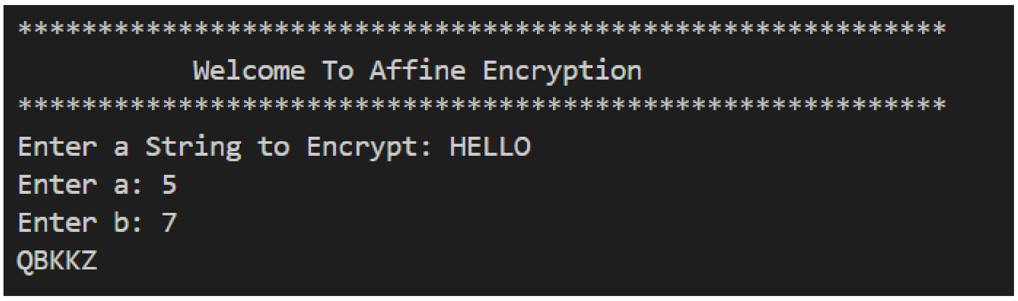
```
******** CRYPTOGRAPHY MENU ********
1. Caesar Encrypt
2. Caesar Decrypt
3. Affine Encrypt
4. Affine Decrypt
7. RSA Encrypt
8. RSA Decrypt
9. Exit
Enter choice:
```



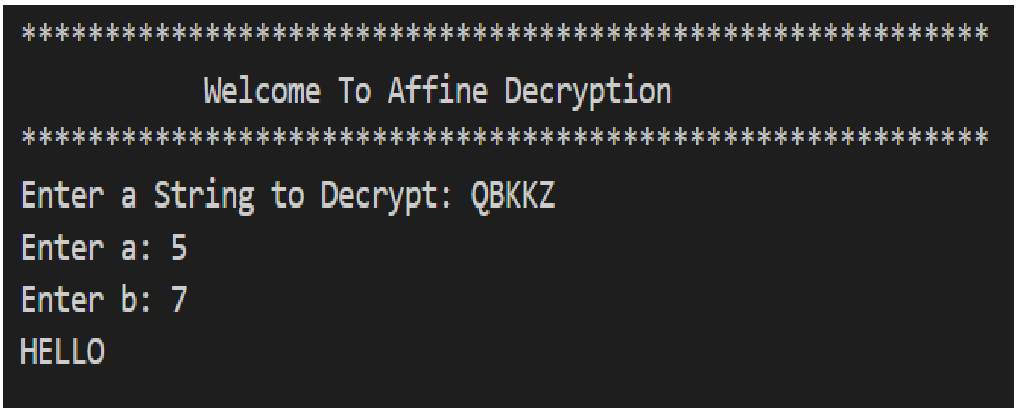
```
****************************************************
          Welcome To Caesar Encryption
****************************************************
Enter a String to Encrypt: HELLO
Enter K: 3
KHOOR
```



```
****************************************************
          Welcome To Caesar Decryption
****************************************************
Enter a String to Decrypt: KHOOR
Enter K: 3
HELLO
```



```
****************************************************
          Welcome To Affine Encryption
****************************************************
Enter a String to Encrypt: HELLO
Enter a: 5
Enter b: 7
QBKKZ
```



```
****************************************************
          Welcome To Affine Decryption
****************************************************
Enter a String to Decrypt: QBKKZ
Enter a: 5
Enter b: 7
HELLO
```

9. Future Improvements

- Add AES or DES implementations
- Support lowercase RSA plaintext
- Use big integer libraries
- Add file-based encryption
- Improve UI with GUI
- Integrate hashing algorithms

10. Conclusion

This project successfully demonstrates the practical application of **number theory in cryptography**. By implementing classical and modern encryption algorithms from scratch, the project strengthened understanding of modular arithmetic, coprimality, and algorithmic efficiency. Challenges such as modular inverses and RSA block handling were overcome through mathematical reasoning and careful implementation. Overall, this project highlights the strong relationship between discrete mathematics theory and secure computing systems

13. References & Learning Sources

- **GitHub:** **https://github.com/tcs179669-sys/DM---Project**
- https://github.com/Alishba-rafi/Discrete-Mathematics.git
- **Video Demo:** **https://www.linkedin.com/posts/alishba-rafi-237909336_sharing-our-cryptography-project-demo-i-activity-7414374106768445441-nopP?utm_source=social_share_send&utm_medium=android_app&rcm=ACoAAFR_CgsBQU4AkUwQTGTw5zDP2CzBvt5CDMo&utm_campaign=whatsapp**
- **Textbooks:** Course textbook and any additional books consulted (e.g., Rosen, *Discrete Mathematics and Its Applications*, 8th ed.)
- **C++ Reference Documentation** (cppreference.com)
- **GeeksforGeeks** – RSA Algorithm
- **AI Assistance:** ChatGPT used for explanation refinement and report structuring