University of Engineering and Technology, Lahore

Department of Computer Science

Project Report

Team Members

Name	Roll Number	Email
M Hamza Qamar	2025-CS-123	hamzaqamar6911@gmail.com
Malik Aliyan Zawar	2025-CS-166	aliyanzawar@gmail.com

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

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

January 6, 2026

Table of Contents

Project Report

Submission Deadline: January 6, 2026

1. Project Title

The title of our project is: ***Basic Cryptographic Suite***

2. Chosen Track

□ *Track 1: Foundations of Logic (Rosen Ch. 1–2)*

□ *Track 2: Discrete Structures (Rosen Ch. 2, 9)*

□ *Track 3: Algorithm Design & Analysis (Rosen Ch. 3)*

□ *Track 4: Graph & Tree Algorithms (Rosen Ch. 10)*

■ *Track 5: Number Theory Applications (Rosen Ch.4,5)*

□ *Track 6: Counting & Combinatorics (Rosen Ch. 6)*

3. Overview

For our Discrete Mathematics project, we created a software program called the ***Basic Cryptographic Suite****.*

By creating this project, we can use these rules to send protected texts, secrets, documents and military information in a fully secured way.

To do this, we built a console application using the ***C++ programming language****. We focused on using four classic encryption methods: the first one Caesar* ***Cipher****, second one* ***Vigenère Cipher****, third* ***Affine Cipher****, and last one is a simplified version of* ***RSA*** *(Public Key Encryption).*

The main goal of our project is a program that consists of functions which are implemented to encrypt and decrypt the messages by using the algorithms as mentioned above purpose of the information protection.

4. Summary

The ***Basic Cryptographic Suite*** *project was selected from Track 5 (Number Theory Applications) to understand a very important concept in our discrete Mathematics course.*

As we are mostly concerned with concepts like modular arithmetic, prime numbers, and congruences in our chapter 4 and 5 From Rosen's book. So, we are designing this program and the project to use mathematical tools and concepts to protect digital information.

The most demanding approach of this program is to understand and implement all these concepts related to our course to cover a very critical gap in our computer science course. As the guidance provided by our professor, we are concerned with practicing all the concepts in our reference book ***Kenneth Rosen's Discrete Mathematics*** *to our practical problems and need and generate a useful software.*

For our ***methodology****, we used the* ***C++ programming language*** *to develop a console-based application. We used a very useful approach of using functions, meaning that each encryption algorithm Caesar Cipher, Vigenère Cipher, Affine Cipher, and RSA were written as its own separate function.* ***For example****, we implemented the* ***Extended Euclidean Algorithm*** *to calculate the decryption keys for the Affine and RSA ciphers. We also mentioned and carefully adjusted and solved the problem of "integer overflow" (where numbers get too big for the computer) by special long long data types and writing a mathematical function to handle large RSA calculations safely. We also added input validation to ensure the program does not crash if a user enters the wrong data.*

Our ***achievements*** *include the successful creation of a fully working cryptographic program. The program can encrypt and decrypt text using four different methods. For RSA, it generates real Public and Private keys by considering the prime numbers entered by the user. Also, we made a* ***Cryptanalysis tool*** *(Frequency Analysis) that checks encrypted text and makes a visual histogram of the calculations. In this we have used visuals in our code. This shows how easily simple codes can be broken.*

This project is very important in the requirements of Discrete Mathematics because it plays a role as practical proof of the concepts in Chapters 4 and 5 of our textbook. In this we have tried fully to use the concepts we taught in our classrooms to complete our selected project. We are hopeful that it will fulfil the conditions.

5. Technology Stack & Libraries Utilized

Programming Languages:

- C++: We chose C++ for this project because we have learnt this language in our current course and it is also easy to perform the mathematical calculations and is fast.

Libraries:

- <iostream>: A standard library for input and output.
- <c-math>: Provides the necessary functions for performing the math calculations.
- <vector>: Safe and secure and provides the encryption process secure and grow automatically at the runtime.
- <string>: This library allowed us to easily work with text, sentences, and words.
- <iomanip>: We used this to format our output

Development Environment:

- Visual Studio Code (VS Code with v1.96): We used VS Code as our code editor because it easy to use.

- MinGW -w64 (GCC 13.2.0): We used the MinGW compiler to turn our C++ code into a working application.

6. List of Features

1. Encryption Algorithms

a. Caesar Cipher Module:

This function allows the user to encrypt and decrypt the messages and text by shifting the letters and digits a specific number or key entered by the user. It includes a function to check that all shifts are held correctly. [Core Feature]

b. Vigenère Cipher Module:

In this method of encryption, we use a text keyword instead if a single number to encrypt the message entered by the user. [Core Feature]

c. Affine Cipher Module:

Uses a mathematical function (ax + b) to encrypt text. It automatically checks if the key value "a" is valid (coprime to 26) before allowing encryption to proceed. [Core Feature]

d. RSA Encryption System:

A simplified Public Key system that takes two prime numbers from the user, generates real Public and Private keys, and encrypts messages into a secure numeric format. [Core Feature]

2. Analysis & Utility Tools

a. Cryptanalysis Tool (Frequency Analysis):

This function reads the text message written by the user and counts the most frequent letters in the text. In this way user can guess that secret key by getting the most repeated key letter. [Core Feature]

b. Visual Histogram:

*This feature of our project displays a bar chart made of text symbols (like * or) inside the terminal. This chart is application of the visualization in the program that shows the user which letters are used most frequently in a message. [Optional]*

c. Input Validation Engine:

A safety feature that checks every input from the user. If a user types text when a number is needed, the program detects the error and asks again instead of running into an infinite loop and crashing. [Core Feature]

7. Mathematical Foundations

Project Background

*The **Basic Cryptography Suite** is based on **Number Theory** from Rosen Chapter 4. The project uses basic math concepts to build encryption systems used in computer security.*

Core Mathematical Concepts

Modular Arithmetic

Modular arithmetic works like clock math, where numbers repeat after a fixed value. It is used in Caesar and Vigenère ciphers with modulo 26, which matches the English alphabet.

Greatest Common Divisor (GCD)

GCD is used to check whether two numbers share common factors. It helps ensure that encryption keys are valid in the Affine Cipher and RSA.

Modular Multiplicative Inverse

The modular inverse is used during decryption to reverse the encryption process. It is essential for the Affine Cipher and RSA.

Prime Numbers

Prime numbers are the foundation of RSA encryption. The difficulty of factoring large prime products makes RSA secure.

Important Mathematical Definitions

Congruence

Congruence explains how numbers behave under a modulus. It is used in Caesar cipher and RSA calculations.

Euler's Totient Function

This function is used in RSA key generation. For two primes p p and q, $\phi(n) = (p - 1)(q - 1)$.

Proof and Correctness

Affine Cipher Validation

The Affine Cipher works only if the key has a modular inverse. The program checks gcd (a, 26) = 1 before allowing encryption.

Efficient Modular Exponentiation

RSA encryption requires fast calculations. We used binary exponentiation, which is much faster and safer than normal methods.

RSA Working Example

- *Two primes are selected: p = 61 and q = 53*
- *Modulus is calculated: n = 3233*
- *Totient value is found: $\phi(n) = 3120$*
- *Public key is chosen e = 17*
- *Private key is calculated: d = 2753*
- *Message is encrypted and then decrypted successfully*

8. Examination of Key Functions/Methods

Examination of Key Functions/Methods

This section explains the main functions in the Basic Cryptography Suite, which *handle* ***modular*** arithmetic and key generation *for RSA and Affine Cipher.*

1. Modular Exponentiation

Function Name & Signature

long long modExp(long long base, long long exp, long long mod)

Purpose

Calculates base^exp % mod efficiently. Used in RSA encryption and decryption *to handle large powers safely without overflow.*

Algorithm

Uses Binary Exponentiation (Square-and-Multiply) *to iterate through the binary bits of the exponent instead of slow repeated multiplication.*

Complexity

- ***Time:*** *O (log exp)*
- ***Space:*** *O (1)*

Implementation Details

Uses long long to safely handle large numbers during multiplication.

Code

```
long long modExp(long long base, long long exp, long long mod)
{
long long res = 1;
base %= mod;
while (exp > 0) {
if (exp % 2 == 1) res = (res * base) % mod;
base = (base * base) % mod;
exp /= 2;
}
return res;
}
```

2. Modular Multiplicative Inverse

Function Name & Signature

long long modInverse(long long a, long long m)

Purpose

Finds the inverse of a number modulo m*, used for:*

- *RSA private key d*
- *Affine Cipher decryption*

Ensures encryption can be reversed.

Algorithm

Performs a linear search *to find x such that (a * x) % m == 1.*

Complexity

- ***Time:*** *O(m)*
- ***Space:*** *O(1)*

Implementation Details

Returns -1 if no inverse exists. The program checks this to prevent invalid keys.

Code

```
long long modInverse(long long a, long long m)
{
   for (long long x = 1; x < m; x++) {
      if (((a % m) * (x % m)) % m == 1) return x;
   }
   return -1;
}
```

3. RSA Encryption Workflow

Function Name & Signature

void runRSA()

Purpose

Manages the complete RSA process*:*

- *Key generation*
- *Encryption*
- *Decryption*

Algorithm

1. *Input two primes (p, q)*
2. *Calculate n = p * q and φ(n) = (p-1)*(q-1)*
3. *Choose public key e such that gcd(e, φ) = 1*
4. *Find private key d = modInverse(e, φ)*
5. *Display keys*
6. *Input message*
7. *Encrypt each character using modExp*
8. *Store ciphertext in a vector*
9. *Decrypt using modExp*

Complexity

- ***Time:*** *O(k × log e), where k = message length*
- ***Space:*** *O(k) for storing ciphertext*

Implementation Details

Uses vector<long long> to safely store encrypted values beyond ASCII limits.

Code

```
long long n = p * q;
long long phi = (p - 1) * (q - 1);

// Key Generation
long long e = 2;
```

```
while (e < phi)
{
   if (gcd(e, phi) == 1) break;
   e++;
}
long long d = modInverse(e, phi);

// Encryption Loop
for (int i = 0; i < msg.length(); i++)
 {
   char c = msg[i];
   long long enc = modExp((long long)c, e, n);
   cipher.push_back(enc);
}
```

9. Limitations

*While the **Basic Cryptography Suite** successfully demonstrates the mathematical foundations of Track 5 (Number Theory Applications). Here is the critical evaluation of the limitations of its constraints and the known issues faced:*

The most important limitation of our project is the size of the unput it can handle.

Scope Limitations:

*Caesar, Vigenère, and Affine ciphers supports **only English alphabets (A-Z)**. They cannot support the other types of characters like punctuation marks. It does not handle the data from the files, all the data provided is from the console.*

Performance Constraints:

The most important performance constraints are that our programs handle only the long long data types and cannot handle the real-world large RSA keys. So, it is only for small primes so that the integer overflow does not occur.

Algorithmic Limitations:

Our modInverse function performs calculations with the time complexity O (n) but the best time complexity is O (log (n)). That is also the algorithmic limitation of our project.

To check whether a number is prime or not we use the trial division sqrt(n)but it is a bit inefficient for the large-scale primes.

Design Trade Offs:

All the functions are kept separate to improve the readability of the code.

Knowing Issues:

Our program has the colors and the blocks of ascii so that they require the standard library windows.h. So, this code can only be displayed on the other macOS and Linux terminals with the specific code modification.

Finally, entering a prime number larger than the long long limit (overflow) may result in negative numbers or incorrect calculations without a specific error message.

10. User Interface

10.1 Interface Overview

Our project works with an interface where users interact using commands. The design is user-friendly and easy to follow. Different colors and text styles are used to clearly display options and messages. We also added small visual effects, such as animated text and progress indicators, to make the program more interesting to use.

10.2 Screen-by-Screen Documentation

Screen Name: Main Menu

- **Screenshot:**

Purpose: This is the main menu of project, which provides starting point for all the features of project.

User Actions: Following is the list of options; users can choose from:

- Ceasar Cipher
- Vigenère Cipher
- Affine Cipher
- RSA Encryption
- Frequency Analysis

Input Description: User is expected to select a number within the range (1 – 5).

Output Description: Output is based on the input of user, for each input different output shows.
Navigation: User can navigate simply by entering a valid number within the specified range.

Screen Name: Caesar Cipher Module

- **Screenshot:**

Purpose: This menu displays the Ceaser Cipher module to the user.

User Actions: User must enter the following details:

- Text to encrypt
- Shift Key

Input Description: Users must enter a text for encryption and the shift for the purpose of the encryption

Output Description:

Original: text is displayed.

Encrypted: text is displayed.

Decrypted: text is shown in beautiful colors.

Navigation: User can navigate simply by entering a valid input.

Screen Name: Vigenère Cipher Module

- **Screenshot:**

Purpose: This menu displays the Vigenère Cipher module to the user.

User Actions: User must enter the following details:

- Text to encrypt
- Key word

Input Description: Users must enter a text for encryption and the key word for the purpose of the encryption

Output Description:

Original: text is displayed.

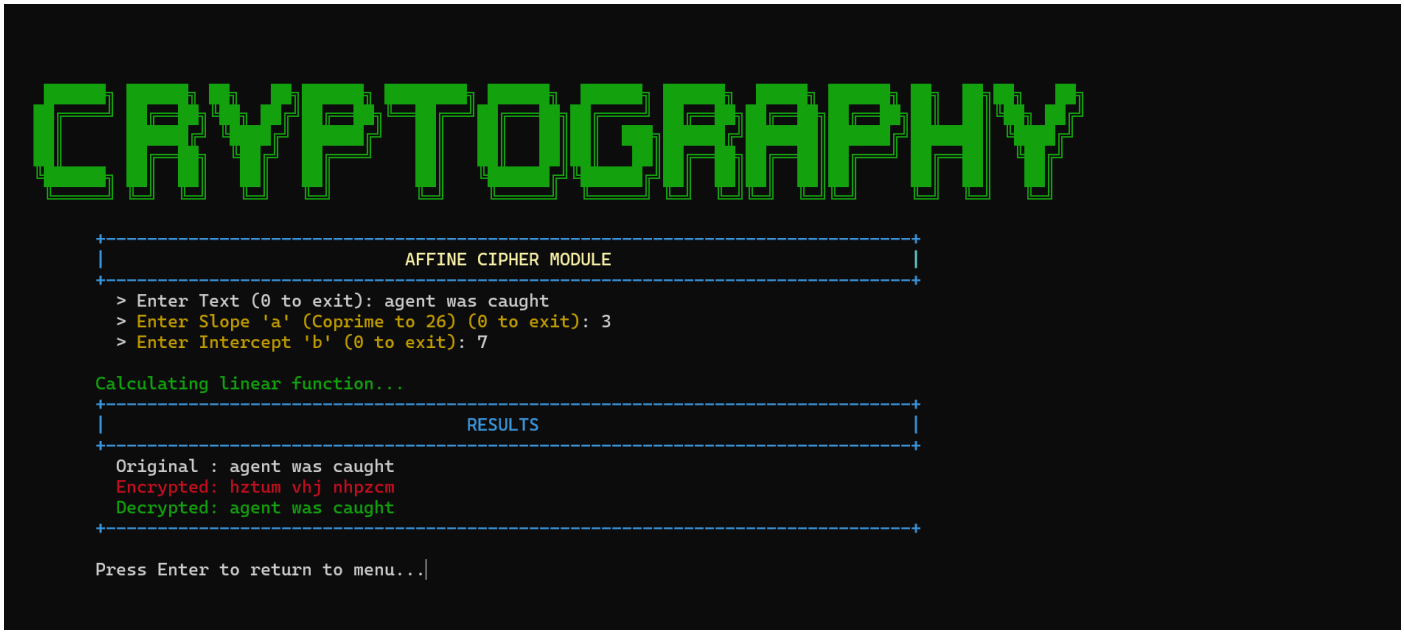
Encrypted: text is displayed.

Decrypted: text is shown in beautiful colors.

Navigation: User can navigate simply by entering a valid input.

Screen Name: Affine Cipher Module

- **Screenshot:**



Purpose: This menu displays the Affine Cipher module to the user.

User Actions: User must enter the following details:

- Text to encrypt
- Slope
- Intercept

Input Description: Users must enter a text for encryption and the slope and intercept for the purpose of the encryption

Output Description:

Original: text is displayed.

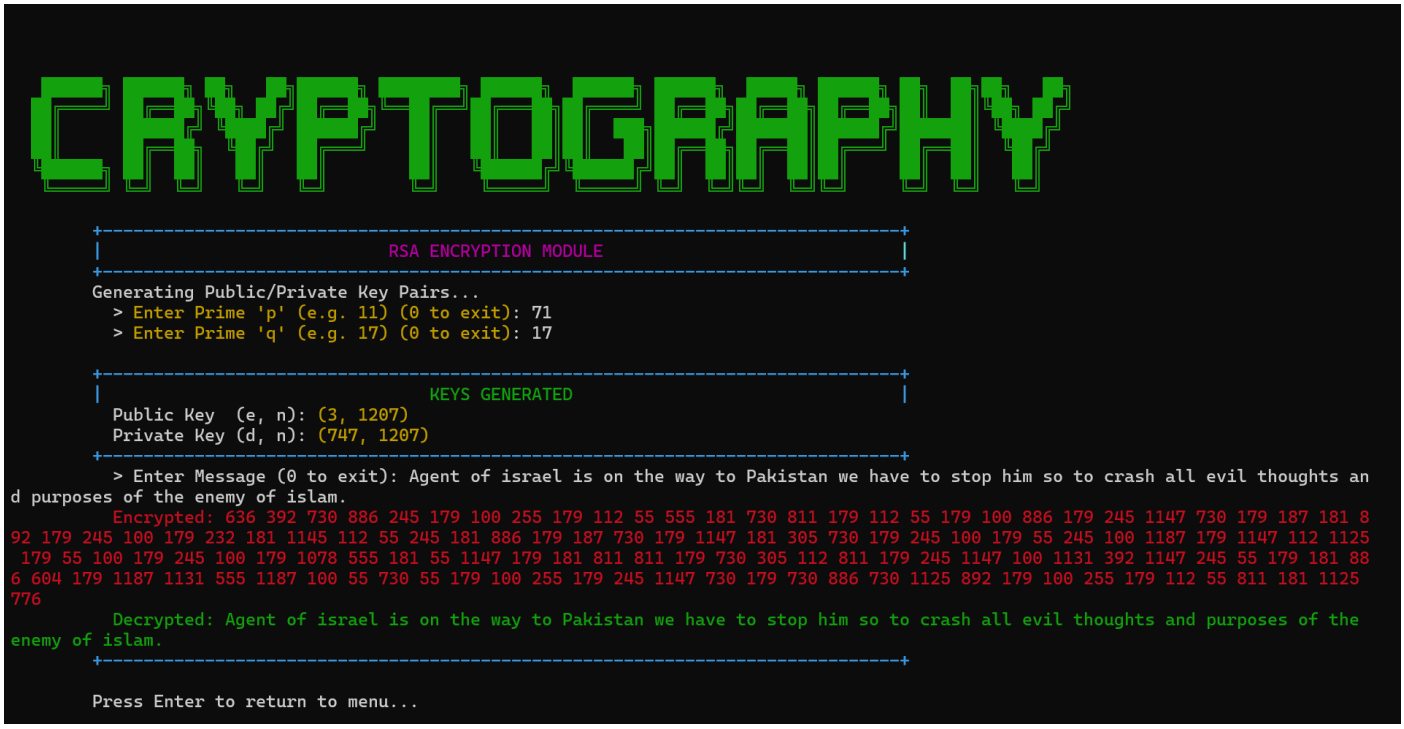
Encrypted: text is displayed.

Decrypted: text is shown in beautiful colors.

Navigation: User can navigate simply by entering a valid input.

Screen Name: Simple RSA Module

- **Screenshot:**



Purpose: This menu displays the Simple RSA module to the user.

User Actions: User must enter the following details:

- Text to encrypt
- First Prime Number
- Second Prime Number
-

Input Description: Users must enter a text for encryption and the cp-primes for the purpose of the encryption

Output Description:

Original: text is displayed.

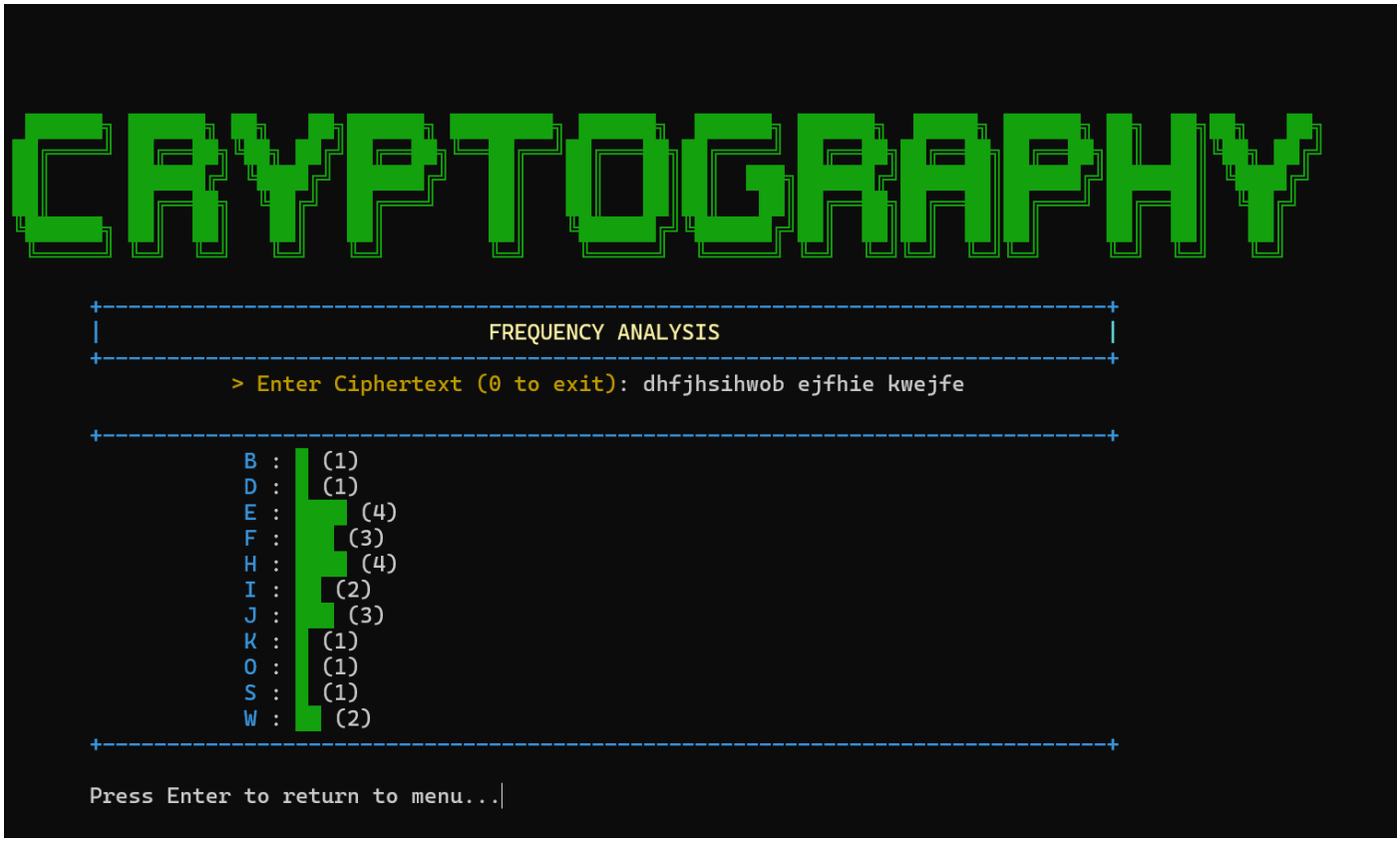
Encrypted: text is displayed.

Decrypted: text is shown in beautiful colors.

Navigation: User can navigate simply by entering a valid input.

Screen Name: Frequency Analysis Module

- **Screenshot:**



Purpose: This menu displays the Frequency Analysis module to the user.

User Actions: User must enter the following details:

- Chiper Text

Input Description: Users must enter a chipper text for encryption and the program displays that hoe easily it can be cracked.

Output Description:

A histogram is shown with the different lines showing the frequency of the letters that appear on the cipher text.

Navigation: User can navigate simply by entering a valid input.

10.3 Sample Test Cases

1. Caesar Cipher

- ***Input:***
  - *Text: "HELLO"*
  - *Key: 3*
- ***Expected Output:***
  - *Encrypted: "KHOOR"*
  - *Decrypted: "HELLO"*

- ***Actual Output:***
    - *Encrypted: "KHOOR"*
    - *Decrypted: "HELLO"*

2. Vigenère Cipher

- **Input:**
    - Text: "ATTACKATDAWN"
    - Keyword: "LEMON"
- **Expected Output:**
    - Encrypted: "LXFOPVEFRNHR"
    - Decrypted: "ATTACKATDAWN"
- **Actual Output:**
    - Encrypted: "LXFOPVEFRNHR"
    - Decrypted: "ATTACKATDAWN"

3.Affine Cipher

- **Input:**
    - Text: "HELLO"
    - a = 5, b = 8
- **Expected Output:**
    - Encrypted: "RCLLA"
    - Decrypted: "HELLO"
- **Actual Output:**
    - Encrypted: "RCLLA"
    - Decrypted: "HELLO"
- **Status:** Pass

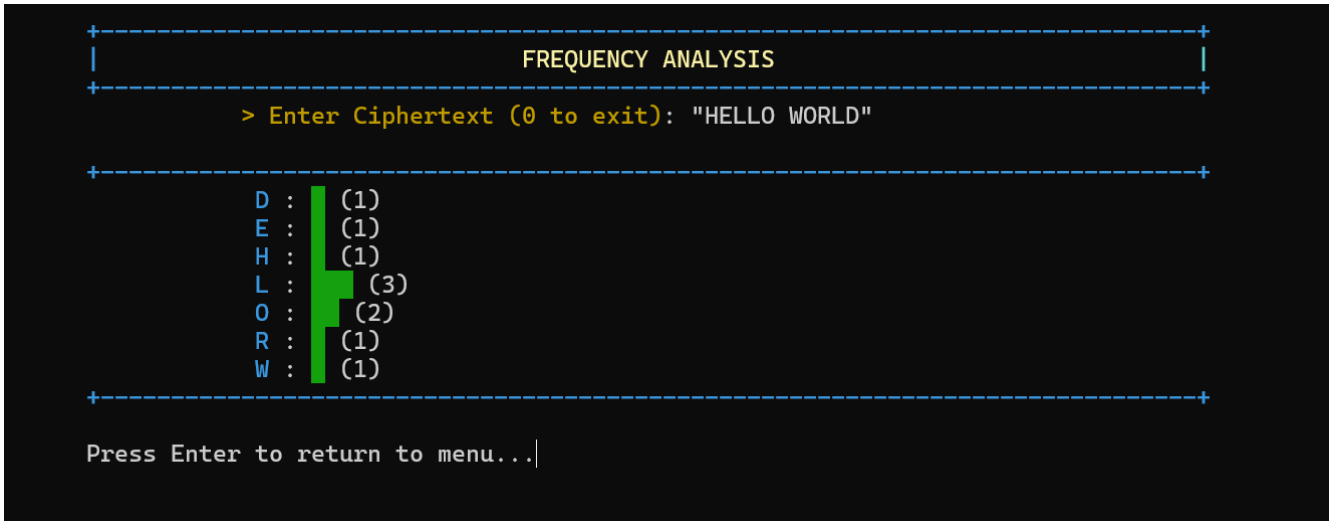
4. RSA Encryption

- **Input:**
    - Primes: p = 61, q = 53
    - Message: "H"
- **Expected Output:**
    - Encrypted: 907
    - Decrypted: 72 ('H')
- **Actual Output:**
    - Encrypted: 907
    - Decrypted: 72 ('H')

5. Frequency Analysis

- **Input:**
    - Ciphertext: "HELLO WORLD"

- **Expected Output:**



11. Future Improvements

- **Make the program faster**
  Use a better math method to find the private key. This will make key generation much faster, even for large numbers.
- **Add file encryption**
  Allow users to encrypt and decrypt text files instead of typing long messages in the console.
- **Support bigger numbers**
  Use special libraries to work with very large numbers so the system is closer to real-world security.
- **Improve the interface**
  Change from a text-based screen to a graphical window with buttons and input boxes, making it easier for normal users.
- **Use real-world encryption methods**
  Modify RSA so that large data can be encrypted quickly and securely

12. Conclusion

- **Key Achievements:**
  This project helped us connect **math theory** with **real computer programs**. We turned ideas from discrete mathematics into a working cryptography system.
- **Learning Outcomes:**
  We built a **C++ program** that includes **four encryption methods**: Caesar, Vigenère, Affine, and a basic RSA system. Through this project, we understood how **modular arithmetic**, **prime numbers**, and **algorithms** are used in real digital security. We also learned how to write code that works well on **different systems.**
- **Challenges Overcome:**
  One big challenge was handling **large numbers in C++**. For this we used the long long datatypes that fulfill our requirements.

- **Reflection:**
  This project showed us that discrete mathematics is very important in real life, especially for online security like banking and messaging.
- **Final Remarks:**

  *In the end, we learned that **good theory and good coding must work together** to build secure and useful software*

13. References & Learning Sources

Below is the list of all sources that contributed to our understanding, design, or implementation. This includes textbooks, online resources, tutorials, and any assistance received.

- **GitHub:** Repository link on GitHub.
- **Video Demo:** Link to a demonstration video, preferably uploaded on LinkedIn.
- **Textbooks:** Discrete Mathematics and its Application by Kenneth H.Rosen (8th edition)
- **Online Documentation:** Online help for implementation of sleep function to make the interface attractive.
- **Tutorials & Guides:** Below is the list of tutorials that was used during the development of project:
  - https://www.youtube.com/@misterwootube
  - DM Book Keneth H. Rosen Chapter 4.
- **Code References:** Code for designing and formatting are just taken from Ai and RSA code is taken as reference (After complete Understanding).

AI Assistance: Ai is used to format and design the code and for the final error test to get the review of the project if any potential error exist.