7/12/2025

Computer Organization and Assembly Language Project Report

CLO: Design and develop solutions for challenges related to computer organization and assembly language Programming

Submitted To:

Mam Maryam Mehmood Malik

Submitted By:

Arshman Abbas 241567

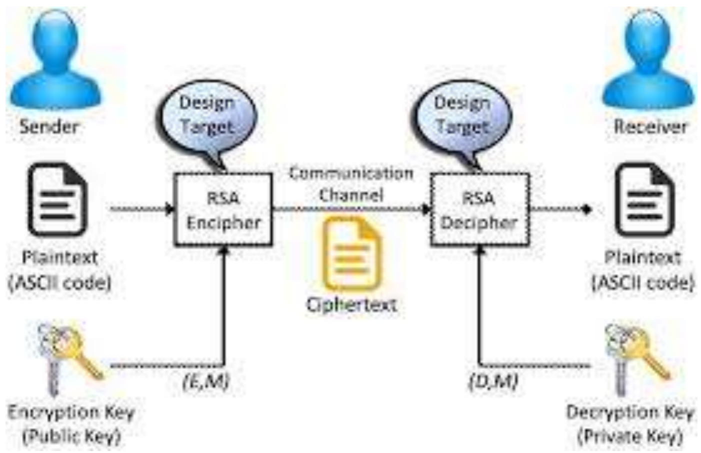
National Cyber Security Academy
Department of Cyber Security
Air University, Islamabad

Title:
RSA File Encryption & Decryption System in Assembly Language

Table of Contents

1. Introduction

RSA is a widely used public-key encryption algorithm that helps secure digital data and communication. It uses two keys: a **public key** for encrypting messages and a **private key** for decrypting them. The algorithm is named after its creators, **Rivest, Shamir, and Adleman**, and its security is based on the difficulty of factoring the product of two large prime numbers.



2. How RSA Works

1. Key Generation

Two large prime numbers are chosen and multiplied to create a **modulus**. Using this modulus and another number *e*, a **public key** is created. The corresponding **private key** is calculated so that only it can decrypt messages encrypted with the public key.
In this project:

- **n = 3233**
- **e = 17**
- **d = 2753**

These are mathematically correct and commonly used textbook RSA demo keys.

2. Encryption

Anyone can use the **public key** to encrypt a message. The process converts the message into numbers and applies mathematical operations to secure the data
.

3. Decryption

Only the **private key** can decrypt the encrypted message, ensuring that the content remains confidential and can only be read by the intended recipient.

Applications of RSA

- **Secure Communications:** Encrypts emails and sensitive information transmitted over networks.
- **Digital Certificates:** Protects digital signatures and verifies the identity of the issuer.

3. Objectives

- Understand how RSA encryption and decryption work.
- Write the RSA algorithm using 32-bit MASM assembly.
- Generate two prime numbers manually in assembly.
- Create public and private RSA keys.
- Perform modular arithmetic (GCD, inverse, exponent).
- Encrypt a user-entered message.
- Decrypt the ciphertext back to the original message.
- Practice low-level concepts like registers, loops, and procedures.
- Build a simple and clear menu-based interface

4. Background

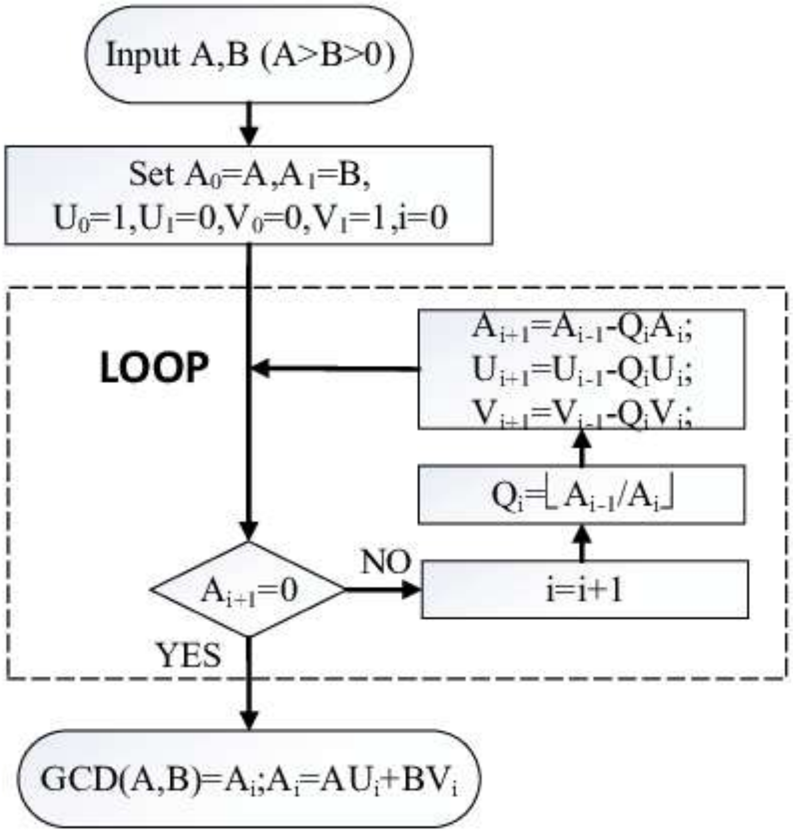
RSA **a popular public-key encryption** method that is used to protect data during communication. It works by using two keys: a public key for encrypting the message and a private key for decrypting it. The security of RSA comes from selecting two prime numbers and performing mathematical operations that are easy to do but very hard to reverse without the private key.

This project focuses on understanding how RSA works by implementing it in **32-bit MASM assembly language**. Instead of relying on built-in functions, all steps like prime number checking, modular arithmetic, GCD, and exponentiation are written manually. Working in assembly helps us see how the CPU performs these operations internally and gives a deeper understanding of both cryptography and low-level programming.

Extended Euclidean algorithm

The Extended Euclidean Algorithm (EEA) is an important tool in cryptography. It builds on the standard Euclidean Algorithm, which finds the greatest common divisor (GCD) of two numbers. The EEA goes further to find numbers that can help express the GCD as a combination of the two original numbers. If the two numbers are coprime, it can be used to find the modular inverse. In RSA, this modular inverse is used to calculate the private key efficiently.

$$a \cdot x + b \cdot y = \gcd(a, b)$$


Input A,B (A>B>0)
Set A_0=A,A_1=B, U_0=1,U_1=0,V_0=0,V_1=1,i=0
LOOP
$A_{i+1}=A_{i-1}-Q_iA_i$; $U_{i+1}=U_{i-1}-Q_iU_i$; $V_{i+1}=V_{i-1}-Q_iV_i$;
$Q_i=\lfloor A_{i-1}/A_i \rfloor$
$A_{i+1}=0$
NO
i=i+1
YES
GCD(A,B)=A_i;A_i=AU_i+BV_i

5. System Design

5.1 Input Handling

- The user launches the program through the terminal:

program.exe input.txt

- **GetCommandTail** extracts the filename from the command line.
- Extra spaces are skipped manually.
- The file is opened using OpenInputFile.

5.2 Reading the File

- Data is loaded into a buffer (inBuffer) of size **5000 bytes**
- The number of bytes read is stored in bytesRead

5.3 Mode Selection

The user chooses:
1 → Encrypt
2 → Decrypt
Console input is taken using ReadChar

5.4 Encryption Logic (Byte → Word)

- Each byte from the file is loaded into EAX.
- Modular exponentiation is applied:

C = M^e mod n

- Output is stored as a **16-bit word**, doubling file size.

5.5 Decryption Logic (Word → Byte)

- Every 2-byte encrypted integer is read.
- Decryption computes:

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

- Output is stored back as a single byte.

5.6 Writing Output File

Results are saved into **output.txt** using:

- CreateOutputFile
- WriteToFile
- CloseFile

6. Implementation

6.1 Tools Used

- MASM / ML.exe
- Irvine32 Library
- Visual Studio 2022
- Win32 Console

6.2 Command-Line Parsing

The system reads filenames without user input:

- Gets the command line tail
- Removes spaces
- Copies filename into buffer
- Opens file

This makes the program feel like a *real* command-line tool.

6.3 Modular Exponentiation (ModExp)

This is your star procedure.
It performs modular exponentiation using:

- repeated squaring
- bit checking (test ebx,1)
- modular reduction (div ecx)

This is an efficient O(log e) algorithm.

6.4 Encryption & Decryption Buffers

The program performs:

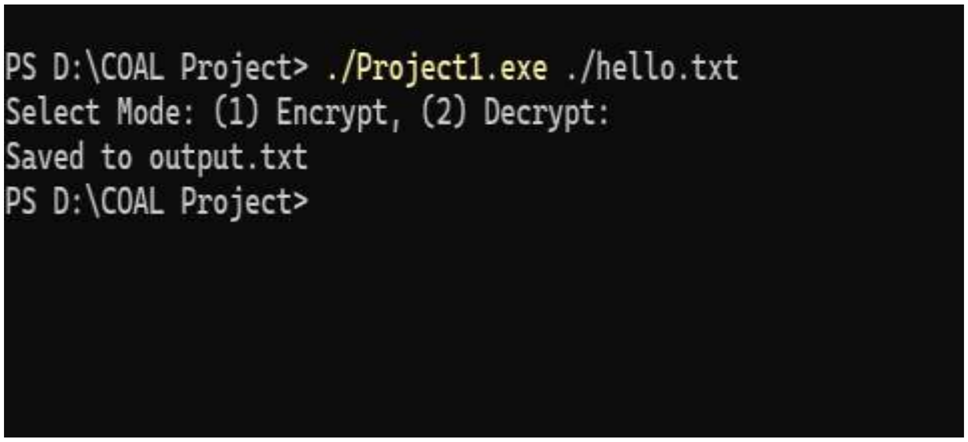
- **byte → word** conversion
- **word → byte** conversion

7. Testing & Outputs

Test Case 1: Encrypt File

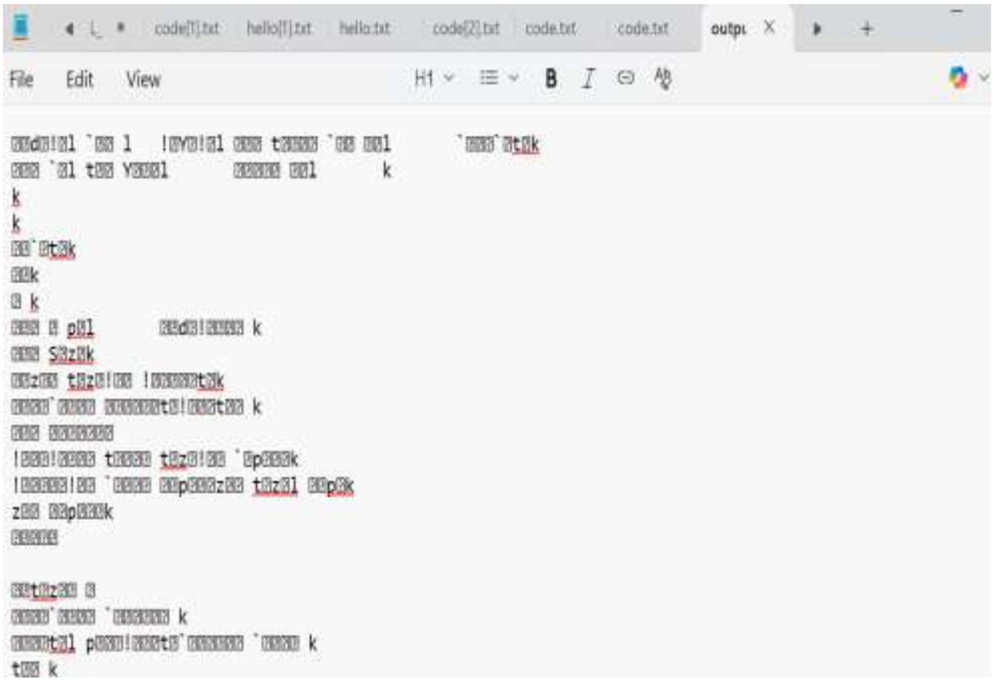
Input: hello.txt
Program actions:

- Reads file
- Encrypts each byte
- Produces *double sized* encrypted file
- Saves to output.txt



```
PS D:\COAL Project> ./Project1.exe ./hello.txt
Select Mode: (1) Encrypt, (2) Decrypt:
Saved to output.txt
PS D:\COAL Project>
```

This screenshot shows file is **encrypted** and saved in output.txt



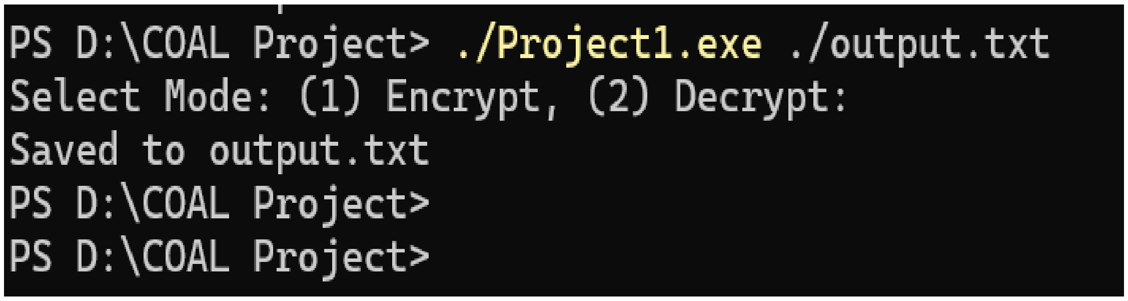
```
[illegible]
```

The encrypted file will look like this ,the RSA Encryption.

Test Case 2: Decrypt File

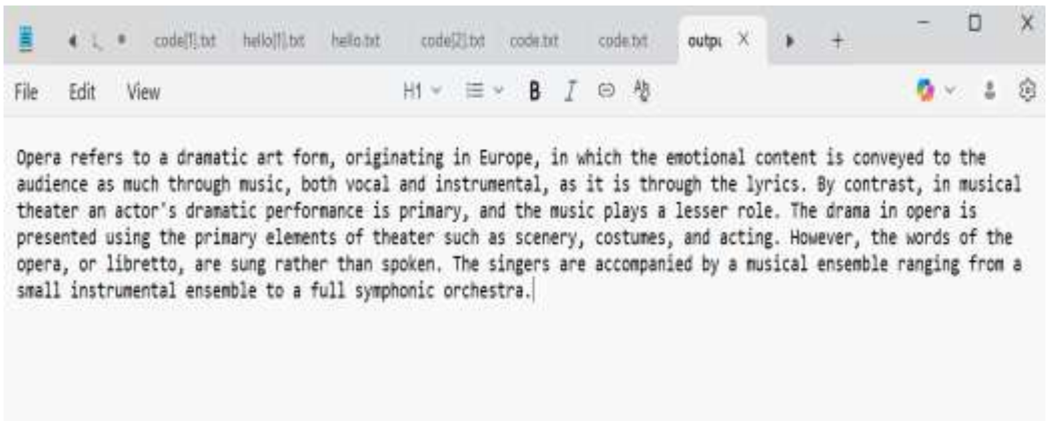
Input: encrypted hello.txt
The program:

- Reads 2-byte chunks
- Applies **C^d mod n**
- Restores original ASCII byte
- Saves output



```
PS D:\COAL Project> ./Project1.exe ./output.txt
Select Mode: (1) Encrypt, (2) Decrypt:
Saved to output.txt
PS D:\COAL Project>
PS D:\COAL Project>
```

This screenshot shows file is **decrypted** and saved in output.txt



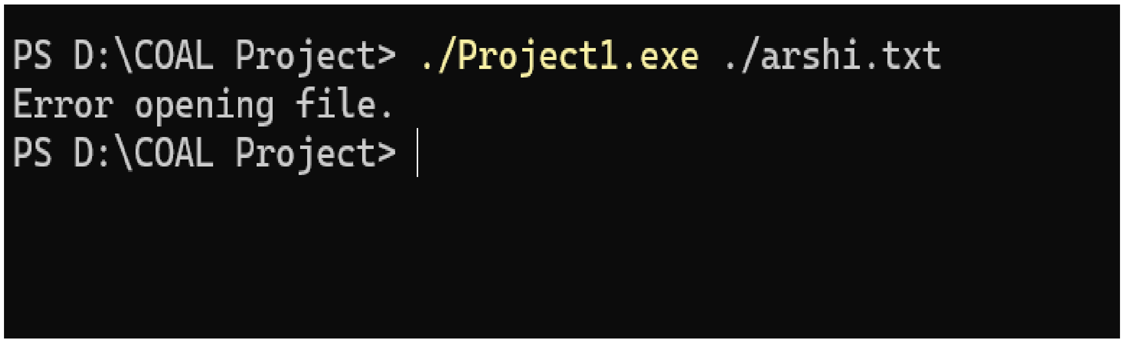
```
Opera refers to a dramatic art form, originating in Europe, in which the emotional content is conveyed to the
audience as much through music, both vocal and instrumental, as it is through the lyrics. By contrast, in musical
theater an actor's dramatic performance is primary, and the music plays a lesser role. The drama in opera is
presented using the primary elements of theater such as scenery, costumes, and acting. However, the words of the
opera, or libretto, are sung rather than spoken. The singers are accompanied by a musical ensemble ranging from a
small instrumental ensemble to a full symphonic orchestra.
```

This above screenshot shows **decrypted text** which is our original text

Test Case 3: Invalid File

If file doesn't exist:

- Shows message **"Error opening file"**



```
PS D:\COAL Project> ./Project1.exe ./arshi.txt
Error opening file.
PS D:\COAL Project>
```

8. Challenges Faced

- **Limited 32-bit Assembly Environment**

32-bit Assembly is a limited environment for performing RSA in MASM because there was no support for high level arithmetic, and so all large integers, register management, and overflow handling was done manually.

- **Generating Prime Numbers**

It was difficult and tedious to generate valid 32-bit prime numbers by using divisibility tests combined with my own square root algorithm to optimize speed, so it took some creativity to achieve.

- **Modular Exponentiation**

It was not straightforward to implement M^e mod N and C^d mod N efficiently in assembly. In order to maintain accuracy and avoid stack overflow errors, I spent a considerable amount of time designing the necessary loops carefully.

- **Computing d (the Private Key) via the Extended Euclidean Algorithm (EEA)**

The process of computing d (the Private Key) was complex, as I had to handle some signs in the Extended Euclidean Algorithm, and the final output required validating that it fell within the correct range.

- **Converting User Input to Integer (and vice versa)**

The process of converting user input into integers to perform encryption (and vice versa) required many loops to ensure that every conversion is carefully validated to avoid runtime crashes.

- **Debugging MASM Code**

Debugging MASM code was time consuming and difficult because there was so much leeway in the way that encryption and decryption were performed. Many instructions needed to be traced at the instruction level to find the source of mismatches.

- **Formatting the Output**

Displaying encrypted hexadecimal values in a more readable fashion while keeping all text easily readable in the terminal environment required very specific manipulations of strings and precise allocation of memory.

Conclusion

This RSA Encryption and Decryption project using MASM demonstrated quite well how such core cryptographic operations are effectuated at an assembly level. Manual handling of prime generation, modular exponentiation, and key computation showed both the mathematical rigor of RSA itself and the precision to be used in such low-level programming.

The technical challenges notwithstanding, the final system succeeded in performing key generation, encryption, and decryption, thus validating the logic of the algorithm and the accuracy of the program. Generally, this project improved the understanding of the fundamental principles in cryptography and further strengthened the practical skills in assembly language; hence, it turned out to be both valuable and professionally relevant for learning.

9. References

- Rivest, R., Shamir, A., & Adleman, L. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2), 120–126, 1978.
- Irvine, K. R. *Assembly Language for x86 Processors*, 7th Edition, Pearson, 2010.
- Stallings, W. *Cryptography and Network Security: Principles and Practice*, 7th Edition, Pearson, 2017.
- Detmer, D. *Programming Windows with MASM 6.1*, Microsoft Press, 2002

THE END