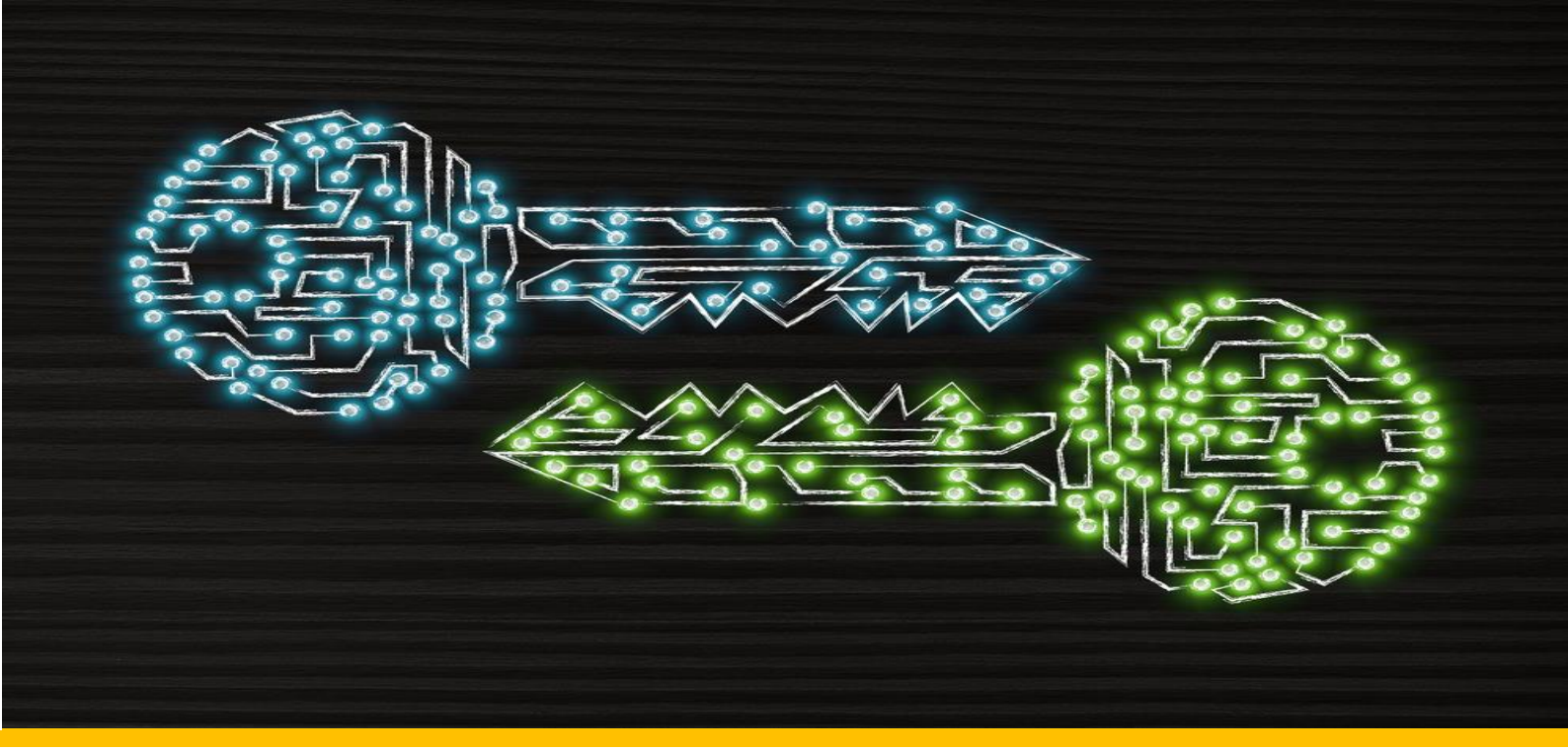


Report CSL2 RSA

BTS1230-23/24 – Paul Holderbaum & Yassin Ait

Version 1.0/22/12/2023

Bern University
of Applied Sciences

TI

Contents

1 Introduction

Since the advent of the pandemic in 2022 Cyber attacks have increased by over 50%. Companies and individuals are constantly getting hacked. It's now essential more than ever to educate yourself in cybersecurity. The Computer Science Lab 2 is therefore a great opportunity to learn about cryptography by Implementing an RSA crypto system. RSA short for Rivest, Shamir and Adleman is an algorithm used to encrypt and decrypt messages using private and public key cryptography, and the goal of this laboratory was to implement a simple demonstrator program of RSA key generation, encryption and decryption. The implementation of this lab had to be implemented in groups of 2. This report aims to explain how the project was implemented, why certain choices were made over others, what were the problems encountered and what conclusion can be made from this.

2 Plannification

2.1 Understanding the problem

The first thing that was done was to get acquainted with the subject. Resources like youtube, books, courses, websites were used to understand how the RSA encryption works under the hood.

2.2 Division of labor

To succeed in the implementation of this Lab excellent group work was mandatory. That is why this group decided to split the implementation of the program between backend and front-end, like most software companies do.

2.3 Merging source code

When working in a team of developers merging source code can cause a lot of issues and bugs. For this reason this group opted to use the versioning system git and have a single remote repository on github. Additionally all commits to the remote repo were made from the same computer to avoid dealing with merge conflicts. The modules and the corresponding functions were all placed in the first implementation of the program so every member would modify the function in the right place.

3 Procedure

3.1 API Implementation

The API has been separated in different modules ; io, keygen, encrypt and decrypt.

IO
This module deals with user input and program output. It also ensures that the program doesn't crash by using good input validation.

- **int read_number()**
  This function asks the user to enter a positive number. If the user enters an invalid value, the function asks again. After 3 tries, it terminates and returns −1.
- **int display_cryp(long cryp[], size_t elements)**
  This function prints the encrypted message to stdout, and returns the number of values written or -1 in case of an error.
- **int read_cryp(long cryp[], size_t elements)**
  This function reads an encrypted message from stdin, stores it in cryp and returns the number of values read or -1 in case of an error.
- **int display_msg(char msg[], size_t elements)**
  writes the decrypted message to stdout and returns the number of characters written or -1 in case of an error.

Problems	Solutions	Possible Improvements
Handling input with spaces using scanf()	Clearing the input buffer and using regular expressions.	Taking the input as a string and parsing it using strtok()

Keygen
This module is used to generate public and private keys.

- **int is_prime(int number)**
  Checks if a number is prime and returns 1 if it is prime, 0 if the number is not prime, -1 if there is an error. The mathematical definition of a prime number was used to determine if a number is prime. A primes number is only divisible by one and itself, so it must have only 2 factors.
- **long gcd(long a, long b)**
  Returns the gcd of a and b, or -1 if there is an error. We used the Euclidean algorithm to decrease the runtime to < O(n), where n is the number of digits smaller than the number b.

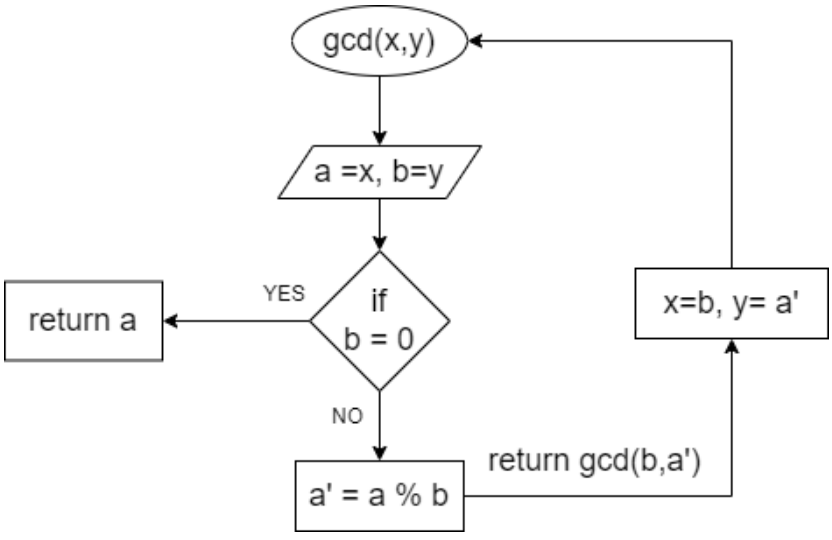


Figure 1 : gcd

- **long modulus(int p, int q)**
  Returns the RSA modulus (n) of 2 numbers p and q, or -1 in case of an error. $n = pq$
- **long phi(int p, int q)**
  Returns the Euler's totient phi of p and q, or -1 in case of an error. $\varphi = (p-1)(q-1)$
- **long public_exponent(long phi)**
  This function first performs input validation on the parameter phi. If the input parameter is not valid the function returns -1. It then loops through every odd number in the range [3, phi[. If the gcd between that number and phi is 1, then it is added to an array and a counter variable is incremented. A random index is generated and used to select a random public exponent.
- **long private_exponent(long e, long phi)**
  This function generates the private key d and uses a helper function gcdExtended from the user module which has a runtime complexity of O(log(n)). This function first performs input validation on the parameters, then calculates the gcd (g) of phi and e using the extended Euclidean algorithm. If g is not equal to 1 it returns -1. If d < 0 then it increments d with phi and returns d.

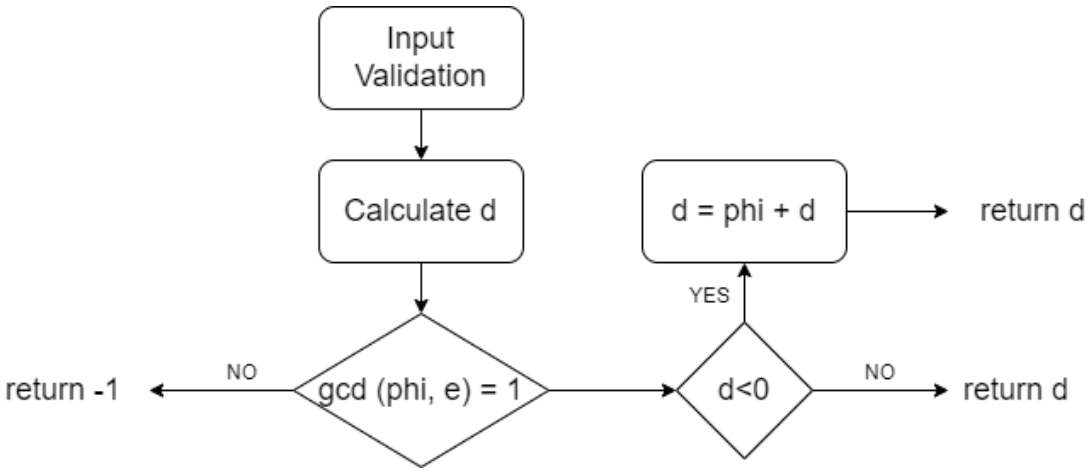
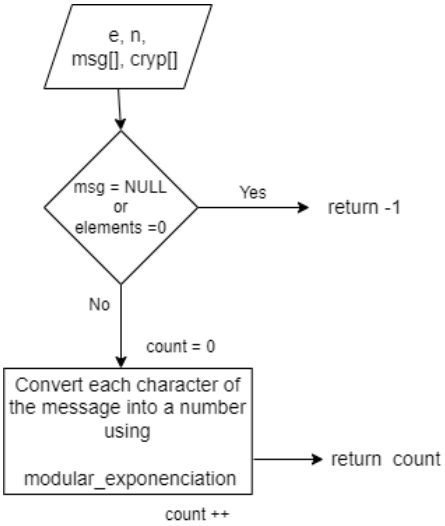


Figure 2 : private_exponent

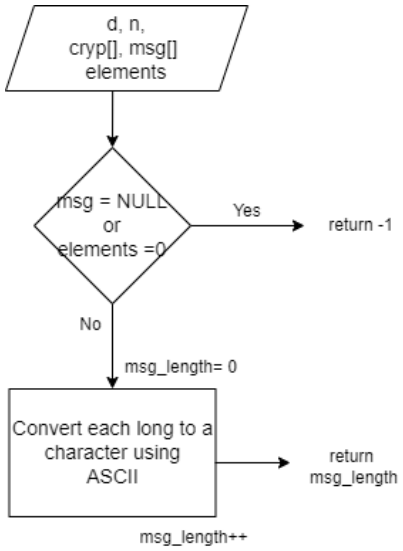
Problems	Solutions	Possible Improvements
Stack overflows, (core dumped).	Used more efficient algorithms like the Euclidean algorithm	
Stack overflow in public_exponent() when trying to store many different values of e	Ignore all even numbers in the loop to reduce the number of instructions	
		In the function is_prime() loop only till $\sqrt{n}$ and the runtime efficiency will be O($\sqrt{n}$) rather than O(n)
Implementation of The extended Euclidean algorithm for the private exponent	Used code from geekforgeeks for the implementation	

Encrypt
This module is used to encrypt messages. It uses a helper function from the user module to perform modular exponenciation.

The function *int encrypt(long e, long n, char msg[], long cryp[], size_t elements)* first performs input validation on all of its parameters that is check for null pointers, check if msg is null terminated. If the input is not valid, the function returns -1. It then loops through every character in msg, convert it into a long using typecasting and ASCII table and then apply modular exponencitation to the result. A counter variable is incrememted at every successful pass of the loop and returned at the end.



Decrypt
This module is used to decrypt messages.
The function *int decrypt(long d, long n, long cryp[], char msg[], size_t elements)* first performs input validation on every parameter. It then loops through every long from 0 to elements, until the number 0 is reached. At every pass, each number converted to a character with corresponding ASCII value by using a type cast. If there is no corresponding ASCII value the character '?' is placed. A counter variable msg_length is incremented at each successful pass and is returned at the end of the function if no error was found.



Problems	Solutions	Possible Improvements
Converting between different data types reliably	Used type casting with ASCII tables and excluded all non ASCII values	Use a standtand of conversion that includes more characters than ASCII

User
This module contains all user defined functions namely

- **long modular_exponentiation(long base, long exp, long mod)**
  This function implements the "square and multiply" method for modular exponenciation which is more efficient that just using a normal for loop. The runtime complexity is O(log n) rather than O(n).
- **long gcdExtended(long phi, long e, long* x, long* y)**
  This function implements the extendent Euclidean algorithm to find the gcd of

3.2 Demonstrator app Implementation

3.2.1 Architecture

The name of the program is enigma which contains functionality for key generation, encryption, decryption as well as a Riddle game.

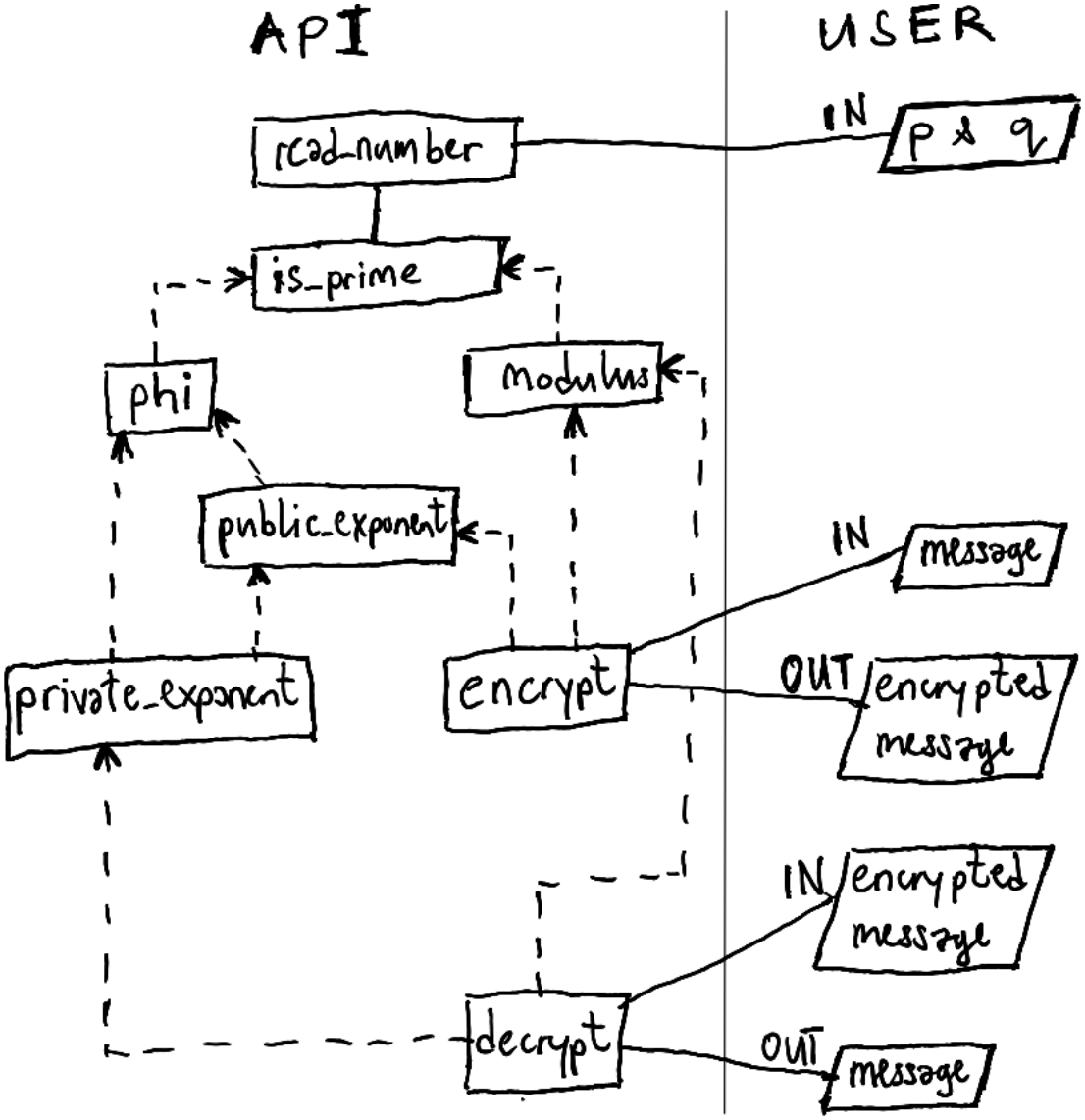


Figure 3: Architecture

3.2.2 Functionality

The Menu of the Enigma CLI app allows the user to choose from a series of actions namely;

- Generate keys: which asks the user for 2 prime numbers p & q and generates the private and public key.
- Encrypt a message:
- Decrpyt a message:
- Riddle Game: The program asks a question to the user and gives the user the encrypted answer. The user then has to guess the answer and decrypt the message to verify.

4 Conclusion

This project enabled us to put into practice all what we learned in class, especially modular programming. It also enabled us to understand how to work in a group of developers with one central source code. We encountered many problems and had to think and research a lot to solve them. We also implemented the most efficient algorithms and tried to have a runtime under O(n). There are still a lot of improvements that can be made. For example the code in the main function can be refactored into functions that would be in the user module. We learned a lot about how C works and the strengths and weaknesses of the language and, we are proud of the work we did for this lab.

5 List of Figures

6 Sources

Lab description
P. Mainini, S. Büttler, rsa-lab-description.pdf, BFH, 2023

Stack overflow
https://stackoverflow.com/

Geek for Geeks extended Euclidean algorithm
https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/