EE212-Microprocessors Project 1 Assignment

Yiğit Yıldırım & Ali Necat Karakuloğlu

Fall 2023

1 Introduction

In this project, you will implement a simple numeric encryption algorithm called RSA. The idea behind RSA is that given two prime numbers p, q; $n = pq$ is extremely hard to factorize. You will calculate $m = (p-1)(q-1)$ and find an integer e such that $gcd(m, e) = 1$, i.e. m and e are relatively prime. Then, you will find another **positive integer** d such that $ed\ (mod\ m) = 1$. You may want to use Bezout's Identity with m and e to find such an integer d. Thus, you will calculate various integer values and store them in Freedom Board. The system will consists of a keypad and 16x2 LCD module, thus enable these modules before starting to implement the encryption/decryption system. Following functionalities of your implementation will be the basis for grading.

2 Implementation

2.1 Required Functionalities

- Having two modes for encryption and decryption.
- Waiting until user inputs a mode selection.
- After mode is selected, ask the user to input a message consisting of 8 digits.
- Takes the input and separates it into two blocks, each containing a 4 digit **integer**.
- After the operation is done, displays the corresponding output in a proper format as a 8 digit number.
- Print indicating texts on LCD to guide users. For example, you can guide an user while mode selection by printing **"Press 'A' for encryption mode, 'D' for decryption mode"**, or ask the user to input his/her message with **"Enter your message:"**. Note that user interface and experience **(UI/UX)** are a grading criterion. For unreadable messages and faulty input entering, your grade will be deducted.
- Regardless of the mode chosen, integers n, m, e, d must be calculated within the code.

2.2 Implementation Steps

1. Make sure that tutorial code works properly on your setup. This project is based on what you have done on tutorial.

2. Follow the algorithm steps and examples in this **link** if you wish. Furthermore, you may use the pseudocode given for calculating e, d. See the next section for a theoretical discussion and Pseudocode section for an implementation. Note that different approaches are always welcome.

3 Assumptions

- Choose $p = 59$ and $q = 61$.
- The user enters 8 digits regardless of the mode, no more or no less.
- While decrypting or encrypting, it might be the case that M^e or C^d might be too big to evaluate directly. You must find an efficient way to calculate $b^e \ (mod \ n)$, without having overflow. **HINT:** Search for *Modular Exponentiation.*
- **Bezout's Identity** Given two integers m and e, there always exists two other, possibly non-unique, integers x, y such that

$$gcd(m, e) = xm + ye \ , \tag{1}$$

  and x, y are called **Bezout Coefficients** of m and e.
  (1) implies that if m and e are relatively prime, then

$$1 - xm = ye \rightarrow 1 = ye \ (mod \ m) \ . \tag{2}$$

  For an implementation of calculating GCD of the numbers along with their Bezout coefficients, see the pseudocode section.

- Integer d must always be positive for RSA to function. Observe that multiplicative inverse under modulo is not unique.
- Example Display for Receiving Encryption/Decryption Input:
  LCD ROW1: Enter your message:
  LCD ROW2: 21901761
- Example Display for Encryption Output:
  LCD ROW1: Encrypted Message:
  LCD ROW2: 8472285

4 Grading

- UI/UX (*20 points*)
- Correct Extended Euclidean for calculating d, e (*50 point*)
- Correct Modular Exponentiation for encryption and decryption (*30 points*)

5 Pseudocode

```
/* Main Function Structure*/

    Get a mode choice
    if mode is 'A'
        Get block 1 as a number from keypad
        Get block 2 as a number from keypad
        print Encrypt(block1, block2)
    else if mode is 'D'
        Get block 1 as a number from keypad
        Get block 2 as a number from keypad
        print Decrypt(block1, block2)

    else
        print "Wrong Key"



extendedEuclidean(int a, int b){
    /*  Given two integers a,b; calculate their Bezout coefficients and
    BCD*/
    s <- 0, old_s <- 1, t <- 1, old_t <- 0, r <- b, old_r <- a;


    while r != 0
        //Quotient of old_r divided by r
        quotient <- old_r / r
        Store r in a temporary variable called temp
        //Remainder of old_r divided by r
        r <- old_r - r * quotient
        old_r <- temp
        Store s in a temporary variable called temp
        //Substract quotient times s from old_s
        s <- olds - s * quotient
        old_s <- temp
        Store t in a temporary variable called temp
        //Substract quotient times t from old_t
        t <- old_t - t * quotient
        old_t <- temp

    gcd(a,b) = old_r = old_s * a + old_t * b

}
```