Course Project: ARM Programming

RSA Algorithm

EN.605.204 Computer Organization

Introduction

One of the most widely used encryption algorithms in modern day computing is RSA (Rivest-Shamir-Adleman). The famous algorithm is named after its creators, Ron Rivest, Adi Shamir and Leonard Adleman. RSA is an asymmetric cryptographic algorithm; meaning there are two different keys (public and private) used to encrypt and decrypt messages. This is commonly known as public-key cryptography.

The RSA algorithm security relies heavily on the difficulty of factoring large integers. The public key consists of the product of two large prime numbers. The private key is derived from the same two prime numbers used to obtain the public key.

Please make sure you read the following to become familiar with the RSA algorithm.

https://brilliant.org/wiki/rsa-encryption/

https://simple.wikipedia.org/wiki/RSA_algorithm

https://www.di-mgt.com.au/rsa_alg.html#note2

Assignment

This programming assignment is an exercise in assembly coding and team collaboration. You will be randomly assigned to a team. Team sizes will be approximately 3-4 students. Each team will have a private team channel created within MS Teams. This channel will be used to collaborate and work through design and construction of your program. Each team channel will be monitored for collaboration activity. Students are expected to participate and contribute. Additionally, each team member will be expected to submit the team evaluation after completing the assignment.

First step: As a team, create a software design document that maps out your team solution. I would suggest reading the following article on “How to write a good software design doc” https://www.freecodecamp.org/news/how-to-write-a-good-software-design-document-66fcf019569c/. There is a lot of freedom in this step. The intent is to have the team think about the requirements and deliverables before diving into coding.

Second step: As a team, please write an ARM Assembly program that implements the RSA algorithm. Your development of the code should follow what was created in your software design document. The expectation is to develop a library of ARM Assembly functions (RSALib) that will be called from a main ARM Assembly program. For our purpose, we will keep our inputs as small as possible to ease calculations. In reality, much larger numbers would be used to ensure security. ***However, if your team***

is feeling adventurous and looking for an additional 10 bonus points, implement a function capable of handling very large numbers that can be used for the necessary calculations. A little research may be required.

Using what you have learned about the RSA algorithm, ensure your team's program follows the high-level steps noted below. You have freedom in how functions may be implemented. All team members are to be contributors to the code. I suggest splitting up functions between team members. All should contribute to the construction of the main program.

Be creative. Have fun with this really cool algorithm!

<u>Minimal requirements for the assignment:</u>

1. Display prompts for user actions:
    a. Generate Private and Public Keys
    b. Encrypt a Message
    c. Decrypt a Message

2. Generate Private and Public Keys
    a. Display prompts to input two positive integers **p** and **q**. For simplicity, let's keep p < 50 and q < 50. Can be larger is implementing the function for large values.
    b. Check if both integers are prime
    c. Calculate the modulus **n** for the public and private keys: **n = p * q**
    d. Calculate the totient: **Φ(n) = (p – 1) (q – 1)**
    e. Display a prompt and input a small public key exponent value **e**
    f. Implement the following parameters for the public key exponent **e**
        i. **e** must be a positive integer
        ii. 1 < e < Φ(n)
        iii. **e** is co-prime to Φ(n). This means that e and Φ(n) share no common factors other than 1. In other words, **gcd(e, Φ(n)) = 1** gcd : greatest common divisor.

    g. Write an ARM function, **cpubexp** for calculating the public key exponent. Additionally, write an ARM **gcd** function to find the greatest common divisor used in calculating the public key exponent.
    h. Calculate the private key exponent **d** such that
    de ≡ 1 (mod Φ(n)) → d = (1 + x * Φ(n)) / e for some integer **x**

Course Project: ARM Programming

RSA Algorithm

EN.605.204 Computer Organization

Write an ARM function, **cprivexp** for calculating the private key exponent.

3. Encrypt a Message
   a. Prompt to enter a message for encryption
   b. Example: "**Hello from TEAM x"** (where x is your team number) or some similar short message
   c. Using the message you have input, determine the ascii equivalent for each character. The numeric ascii value will be used for encryption/decryption purposes. For simplicity, you can encrypt each character separately. I also recommend spaces between each output value so reading will be easier. Your encrypted message should be written to a file called "encrypted.txt". Make sure you open and close the file properly.
   d. We will use the following equation to encrypt the characters of our message. Using our public key values (n, e) for the equation **c =** $\boldsymbol{m^e}$ **mod n** we have the following:
      - **c** is our cipher text (encrypted text) value
      - **m** is the individual plaintext character of our message "H","e","l","l","o"," ","f","r","o","m"," ", "T","E","A","M"," ","x" or whatever small message you chose.
      - **e** is our public key exponent from step 2
      - **n** is the calculated modulus from step 2 for our public and private keys.

4. Decrypt a Message
   a. Open and read the file "encrypted.txt" that contains encrypted text.
   b. We will use the following equation to decrypt the characters of our message. Using our private key values (n, d) for the equation **m =** $\boldsymbol{c^d}$ **mod n** we have the following:
      - **m** is the decrypted individual plaintext character of our message – should be "H","e","l","l","o"," ","f","r","o","m"," ", "T","E","A","M"," ","x" or whatever small message you chose.
      - **c** is our cipher text (encrypted text) value
      - **d** is our private key exponent from step 2
      - **n** is the calculated modulus from step 2 for our public and private keys.

c. Write your decrypted message to a file called “plaintext.txt”

Bonus (10 points)

Since the RSA algorithm is intended for data exchange between parties using private and public keys, try exchanging a message with another team. Find another team that is willing to try the exchange, generate the necessary keys, exchange the appropriate keys, exchange the encrypted file, decrypt the file to obtain the plaintext message.

Provide evidence demonstrating the exchange worked. (Information about the team you exchanged with, screenshots, files, etc…)

Have fun!

Deliverables

Part I

- Create and submit a software design document that maps out your team solution.

Please submit your team RSA design document in a DOC or PDF file named <Your JHEDID>_team#_RSADesignDoc. Each member of a team should submit the same design document.

Part II

- A code library containing **at a minimum** the following functions:
  1. ***gcd*** *– function to find the greatest common divisor*
  2. ***pow*** *– function to perform exponentiation*
  3. ***modulo*** *– function to perform modulo operation*
  4. ***cpubexp*** *– function for all calculations related to the public key exponent*
  5. ***cprivexp*** *– function for all calculations related to the private key exponent*
  6. ***encrypt*** *– function to perform encryption*
  7. ***decrypt*** *– function to perform decryption*
- A main program used to pull everything together as a functioning application.
- Screenshots, files, etc… that demonstrate your program functioning properly as defined by the requirements.

Your code should be well documented. Specific contributor(s) to a piece of code should be noted in the documentation.

Please submit your well documented source code for your team RSA library in a file named <Your JHEDID>_team#_RSALib.s. Additionally include the main program in a file named named <Your JHEDID>_team#_RSA.s along with a PDF file called <Your JHEDID>_team#_rsa.pdf containing screenshots for the test runs noted above. Include your encrypted.txt and plaintext.txt files. Each member of a team should submit the same program, screenshots, etc…