Ira A. Fulton Schools of Engineering
Arizona State University

CSE 539: Applied Cryptography

RSA Project
(50 points)

Purpose

In this project, you will gain first-hand experience implementing the RSA algorithm. You will be given prime numbers greater than 200 bits, testing your ability to work with very large prime numbers. You will learn how to implement the Extended Euclidean algorithm and how to encrypt and decrypt using RSA.

Objectives

Students will be able to

- Implement the Extended Euclidean algorithm (https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm)
- Test ways to work with very large prime numbers.
- Implement the RSA algorithm.

Technology Requirements

It is recommended to implement programming assignments using .NET Core 6.0. However, you are free to use any other modern general-purpose programming language of your choice. It is your responsibility to investigate the availability of existing libraries to successfully complete the project if you decide to go with other languages.

Information on how to install the .NET Core SDK can be found here: Install .NET Core SDK.
You can create a new project using this command: `dotnet new console --output project_name`.
You can also run your project by going into your project directory and using: `dotnet run`.

Here are some helpful links for this particular project:

- BigInteger Class
- BigInteger.Pow Method
- BigInteger.ModPow Method

Directions

You will be given large numbers greater than 200 bits in the form of (e, c). To calculate the value of the number, compute 2^e - c. For instance the number x = 1048575 can be represented as (20, 1) as $x=1048575 = 2^{20} - 1$.

To implement the RSA algorithm, you will need several values such as the prime numbers p and q, e coprime to phi(n), and d such that e*d mod phi(n) = 1. To encrypt with RSA, you need a message m and the values e and n. To decrypt, you need the encrypted message and the values d and n. In this assignment, you are required to perform both encryption and decryption. You will be provided with the following arguments:

1. p_e in base 10
2. p_c in base 10
3. q_e in base 10
4. q_c in base 10
5. e_e in base 10
6. e_c in base 10
7. Ciphertext in base 10
8. Plaintext message in base 10

As mentioned before, you can calculate big numbers like e by calculating $e = 2^{e_e} - e_c$. In order to calculate the value of d, you must employ the Extended Euclidean algorithm. While you may consult other resources on how the Extended Euclidean algorithm works, you must not copy the code you find online. It is expected that you will produce your own implementation.

After calculating the value of d, you can verify that it is correct using the equation e*d mod phi(n) = 1. If this equation does not evaluate correctly, your value for d is incorrect. Once you have calculated all of these values, decrypt the ciphertext given to you and encrypt the given plaintext. Your program should output these two values as a comma-separated pair, with the decrypted plaintext first.

An example execution for the C# program is provided below. In this example, the same message was used for the plaintext and the ciphertext (after encrypting it):

Command:

dotnet run 254 1223 251 1339 17 65535
665360471203741455389167879818680042064385392489107347134952768837246935744345821049009780797011745391671027067254225827884817276195462354405082146945791756026041

Expected output:

1756026041,665360471203741455389167879818680042064385392489107347134952768837246935744345821049009780797011745391671027067254225827884817276195462354405082146945791756026041

Note: We've highlighted to show the input and expected outputs:

Inputs: p_e, p_c, q_e, q_c, e_e, e_c, encrypted message C, plaintext P

Outputs: decrypted text, encrypted cipher

Code Structure

In summary, your task is to implement three essential functions: ***encrypt***, ***decrypt***, and ***generatePrivateKey***. Below are the required function definitions in C#. While you have the flexibility to choose any programming language, it's essential that your code contains function definitions provided below.

```
public BigInteger generatePrivateKey(BigInteger e, BigInteger p, BigInteger q){
    // Implement the method
    // return d;
}

public string decrypt(string ciphertext, BigInteger key, BigInteger N){
    // Implement the method
    // return plaintext;
}

public string encrypt(string plaintext, BigInteger key, BigInteger N){
    // Implement the method
    // return ciphertext;
}
```

Submission Directions for Project Deliverables

- Your code
- Screenshots of running the example provided above along with inputs and outputs.
- A README file for setup and steps necessary to run your program.
- Compress your project folder along with the screenshots into a .zip archive. and submit that file. Please name your submission Firstname_Lastname.zip