CSE13s Winter 2022
Assignment 5: Public Key Cryptography
Theodore Ikehara, tikehara@ucsc.edu
Due: February at 11:59 pm
Design Doc

1. Introduction:
    a. About this Lab:
    Cryptography is one of the most essential components to network and data security. Without this passwords would not mean a thing to people that know how to monitor a network. Anybody could access any data that they wanted to without anything telling them that they can't. The world would be in chaos and personal data would not exist. People would still be relying on pen and paper to keep data secure and use physical locks and security personnel to make sure their data does not get breached. However, this has all changed due to the discovery of encryption. The process of sending and receiving data through the network with the confidence that the information is not compromised. In this lab our goal is to recreate and gain a further understanding of how cryptography works and the natural phenomena brought through mathematical discoveries, and how these discoveries are so significant. As for something we use so often and most of the time take for granted, the least we can do is to understand how these certain components work in practice. In this lab we will be creating the key generator, the encryptor, and the decryptor.
    b. The RSA Algorithm:
    There are many different algorithms and methods that produce similar outcomes, as in sending and receiving data securely, but in this lab we will mainly be focusing on the RSA algorithm. This algorithm that we will be studying and producing is based on a mathematical phenomena that two large prime numbers are extremely hard for computer hardware in the modern day to factor. In fact this process is so difficult to accomplish the entire security of the network and many multi million dollar industries rely on this method to keep their data secure. This algorithm requires two keys in order to send and receive

data. The public key and private key. The public key is used to only encrypt data and thus anybody can see this key without the fear of your encryption being cracked. The private key is only kept by the receiving computer and cannot be shared with anyone, as anyone with this key can decrypt anything that is encrypted using the public key and thus the encryption is compromised.

2. The implementation:
    a. Keygen:
    In this module we will be producing a public and private key pair for encrypting and decrypting our data. This will be done by producing a large prime number pair to base the keys off of. In the key generator the program should start by parsing the command line options and deal with the options accordingly. Then we should open the public and private key files and set the permission to allow for only users to read and write, anyone else will be declined. This is so that anyone without the correct permission cannot read or edit the file. We will set a random seed so our results can be made consistent with the files given by the professor. Then generate a public and private key with the given functions and sign them using user inputted names. After all is complete make sure to close the files and end the program successfully.
    b. Encryptor:
    In this module we will be encrypting the data using our key that was generated using our keygen module. This module should accept given command line options and start with parsing these options. Then we should open the public key and read this public key from the open file. Then we will use the rsa encrypt function to encrypt the data and write this to a file then we will exit the program successfully after closing the file successfully.
    c. Decryptor: In this module we will be decrypting the data that was encrypted using our encryption module. This module should start by parsing the command line options and react as given in the design doc. Then we should open the private key and use this to decrypt the file that was outputted by the encryptor. This should then close the files and return successfully. With the decrypted file as an output.

3. Code layout and operation description and pseudo code:
    a. Keygen:

        void rsa_make_pub():
        // (this is the main function for making a public key)
        make_prime(p)
        make_prime(q)

        void rsa_write_pub():
        // (this function is used by the main public key function to write to the public key file)
        void rsa_read_pub():
        // (this function is used by the main public key function to read the public key file)

        void rsa_make_priv():
        // (this is the main function for making private keys)
        void rsa_write_priv():
        // (this function is used by the main private key function to write to the private key file)
        void rsa_read_priv():
        // (this function is used by the main private key generation function to read the private key file)

        void rsa_encrypt():
        // (this function uses the rsa key to encrypt data)
        void rsa_encrypt_file():
        // (this function uses the encryption function to write the encryption to a file)

        void rsa_decrypt():
        // (this function uses the rsa key to decrypt data)
        void rsa_decrypt_file():
        // (this function uses the decrypt function and write it to a file)

void rsa_sign():
// (this function performs rsa signing with mod n)
bool rsa_verify():
// (this function performs rsa verification and returns a bool value)

b. Encryptor:
   In this file we will be able to produce an encrypted file using the other libraries.
c. Decryptor:
   In this program we will be able to produce a decrypted file in this program.

The RSA library description and pseudo code:

Void rsa_make_pub:
    Lower = bits/4
    Upper = (3*bits)/4
    Get random range(lower, upper)
    Generate p, q prime numbers splitting bits
    lcm(p-1, q-1)
    Loop till coprime is found
    This coprime will be public exponent e

Rsa_write_pub
    Use gmp print function
    Print to input file
Read_pub
    Same thing but with fscanf
Rsa_make_priv
    Calculate lambda
    Mod inverse with e
Rsa_write_priv
    fprintf(pvfile)

```
Rsa_read_priv
        fscanf(pvfile)
Rsa_encrypt
        pow_mod()
Rsa_encrypt file
        Take block
        K = floor(log(n - 1)/8)
        Create first block 0xFF
        Loop till end of file
        Encrypt block write to file
Rsa_decrypt
        Pow_mod
Rsa_decypt file:
        Invert encrypt file
Rsa_sign
        Pow_mod
Rsa_verify
        If compare auth with m ==0
        Return true;
```

Numtheory pseudo code:

```
gcd(a, b){
        while(b!=0){
        B = a % b
        }
}
mod_inverse(a,n){
        R,r` = n,a
        T,t` = 0,1
        While r` != 0{
        Q = r // r`
        r,r`=r`,r-q*r`
        t,t`=t`,t-q*t`
```

```
        If r>1
        Return no inverse
        If t<0
        t=t+n
        Return t

        }
}
pow_mod(a, d, n){
        v=1
        p=a
        While d>0
        If odd(d)
        V = v*x modn
        P = p*p modn
        D = d//2
}
miller_rabin(n, iters){
        For i 1-k
        Choose random from 2, n-2
        Y = pow_mod(a,r,n)
        If y ==1
        Return false
        If y!=n-1
        Return false
        Else return true
}
make_prime(){
        gen_random()
        Loop till prime tests positive
}
```

Keygen pseudo code:

1. Getopt
2. Fopen

3. Fchmod, fileno
4. Randstate_init
5. Rsa_make_pub
6. Rsa_make_priv
7. Sign and convert string to int
8. Write the files
9. Print if verbose enabled

Encryptor

1. Getopt
2. Fopen pub key
3. Read the pub key
4. Convert username
5. Encrypt file using rsa encrypt

Decrypt

1. Getopt
2. Fopen private key
3. Verify private key
4. Decrypt using rsa_decrypt file