Contents

Assignment 6: Public Key Cryptography

Nelly Sin

November 2021

Introduction

To me, cryptography sounded very intimidating when I first heard about it. I knew it was something that have to do with keys, encryption and decryption of files, but I genuinely did not understand why we use it, why is it so important to the internet, and how exactly does all of these connect together and work.
It turns out, there are multiple types of cryptography. The one we are focusing today is asymmetric cryptography or public-key cryptography. This type of cryptography system uses two pairs of keys. One is the public key, that can be shared to others, and the other is private keys, that is only owned by the owner. Both keys depend on another. While the public key encrypts the messages to the intended user the intended user must decrypt it by using their private key to unlock what the secret message is. Allowing one another to communicate without any disruption and securely.
A public key in the cryptography must match to only one private key. Together, they are used to encrypt and decrypt messages. If you encode a message using a person's public key, they can only decode it using their matching private key.

1 Task

Task one that generates keys (private and public) encrypt files decrypt files
GMP library: We must install the package and read the manual for the functions we are using in this assignment.
Global random state variable called state, generating large random numbers. Using the random seed for the state. We also must clear it.

2 Randstate

This initialize the global random state named "state" and the random seed. We will be using this throughout the assignment.

3 Numtheory

Modular exponentiation using the binary squaring method. We must translate it to gmp: mpzt

3.1 Power-Mod(a,d,n)

This is a fast modular exponentiation, computing base raised to the "exponent" power modulo, "modulus" and storing the computed result in out.
v = 1
p = a
while d is greater than 0
check if d is odd
if d is odd, assign v to (v x p)
p will equal to (p x p) mod n
d will equal to (d/2)

3.2 Primality testing:

Because these are very large numbers we must use a randomized number to tell us if something is prime. We will using Miller-Rabin Primality testing Given integer "n" we choose a positive integer a ¡ n. Letting

$$2^s d = n - 1 \tag{1}$$

create a loop such that d must be odd
for range 1 to k
choose random positive integer with of an array withn - 2
y = POWER-MOD(a,d,n)
if y is not 1 or y is not n - 1
j = 1
while j ¡ s - 1 and y != n -1
y = POWER-MOD(y, 2, n)
if y == 1
return false
j = j + 1
y != n -1
return false
return true

3.2.1 Is Prime

This is part of the Miller-Rabin primality test to indicate whether n is prime or not using the "iters" of the Miller-Rabin iteration. Whe we create the two large prime numbers of p and q it verifies whether or not the integer is prime.

3.2.2 Make Prime

This function generates a new prime number stored in p. The prime that is generated must be tested in isprime using "iters".

3.3 Modular Inverses

We will be using Euclid's algorithm, which is an efficient method to compute the greatest common divisor of two integers, the largest number that can divide them both with a remainder of 0. We must implement a function to compute the GCD, greatest common divider.

3.4 Greatest Common Divider

Greatest common divider will be used
GCD(a,b)
while b != 0
t = b
b = a mod b
a = t
return a

3.4.1 Mod-Inverse

When we use mod-inverse, this will be an essential part of decrypting of the keys.
(r, r') = (n, a)
(t, t') = (0,1)
while r' != 0
q = floor(r / r')
rtemp = r
r = r'
r' = rtemp - q x r'
ttemp = t
t = t'
t' = ttemp - q x t'
if r > 1
return no inverse
if r < 0
t = t + n
return t

4 RSA Library

Making of the private and public keys: Making of the public and private keys was very confusing to me at first. I think the idea of how it's a way of two

parties communicate took a longer time for me to understand. However, the assignment documentation and additional videos that describe how this works clear things.
A great example that is often used, is thinking about a mailbox. Everybody can put mail in the mailbox but the owner of the mailbox is the only one that can open the mailbox.
Doubling down, we can observe the communication between Alice and Bob. Alice can encrypt her message using Bob's public key; however, Bob is the only person who can decrypt using his private key. This means that any sender can use the receiver's public key, but the receiver can be the only one who can access these messages using their private key. Hence, why the key is called private.
We can see public and private keys in encryption and decryption everyday in our lives when we use the internet. Such as going on a website and noticing the "lock" indicating the security of the website.
In these RSA functions are also using the Number Theory, to make our keys. Because the RSA file will be mostly consist of mathematical concepts that must be used.

4.1 RSA: Making keys

While we are making keys, we will first be finding prime numbers by using our makeprime function for p and q. But, we will also need to know the number of bits for each prime number is less than $\log_2(n)$.
The random number in the range *[nbits/4, (3 x nbits)/4]*.
Where is is [lower, upper] bounds. But the remaining bits will be stored to q.
The number of Miller-Rabin iterations is specified in iters and it is predefined (default) setting as 50.
We will then compute the totient using equation:
totient(n) = (p1)(q1)
Before we do this, we will need to loop until it reaches to the number of bits we want. This will mean we need to generate a prime number for right number of bits for *n*. This involves making the public and private key to a very large prime number (this will be our p and q) and with the correct number of bits.
The public key should be the co-prime of 1. Thus, the private key should be the modulo of the public key, in order to encrypt and decrypt.
A special thing to note is that we are factoring large prime numbers.
So, when the user calls keygen, it will create two new files for the private key and public key. The public key file should consist the public key and the user's name. While the private key should just include the private key and nothing else.

5 Key Generation

When we generate our keys, our public and private keys must be stored in two seperate files – both can be renamed by the user, but the default will be rsa.pub and rsa.priv. When we write our public key in the public key file we are writing the generated key using rsamakepub and writing the user's name. When writing to our private key file, we will be using the modulusinverse to make our private key. The private key file should only consist of the private key.
We will also be taking in the command lines where:

- -b: specifies the minimum bits needed for the public modulus n.
- -i: specifies the number of Miller-Rabin iterations for testing primes (default: 50).
- -n pbfile: specifies the public key file (default: rsa.pub).
- -d pvfile: specifies the private key file (default: rsa.priv).
- -s: specifies the random seed for the random state initialization (default: the seconds since the UN I X epoch, given by time(NULL)).
- -v: enables verbose output.
- -h: displays program synopsis and usage.

6 Encryption

When we encrypt we will be reading the file with the text we would want to encrypt. However, in order to actually encrypt, we will need to use the public key. By doing so, we will need to read the file with the public key along with the user name after generating our keys.
While we are encrypting the content of the infile, we are encrypting the blocks.

- -i: specifies the input file to encrypt (default: stdin).
- -o: specifies the output file to encrypt (default: stdout).
- -n: specifies the file containing the public key (default: rsa.pub).
- -v: enables verbose output.
- -h: displays program synopsis and usage.

7 Decryption

Similar to what we have written and the idea for encryption, we are doing the same with decryption. However, instead of using the public key, we will be reading the encrypted file and read the private key to decode the message. Similar to the encrypting the file. We are also decrypting the contents of the infile, and writing to the outfile. We are decrypting the blocks of the infile. Verbose printing is printing out how many bits for the fragments in the times we are calling them out while using keygen, encrypt, and decrypt. (Explaing verbose for keygen, encryption, and decryption)

- -i: specifies the input file to decrypt (default: stdin).
- -o: specifies the output file to decrypt (default: stdout).
- -n: specifies the file containing the private key (default: rsa.priv).
- -v: enables verbose output.
- -h: displays program synopsis and usage.

8 Citation

Special acknowledgements to the TA's, tutors, and Professor Long for helping me throughout the concepts and providing some advice for my pseudo code and publishing some pseudo code.
The code I have provided in my work are not all written by me, but by tutors and TA's that provided further help for me. And credit to the professor for providing additional help from his public repository that he sent to the class.

9 Graphics

There are some additional graphics I have added to understand this assignment and cryptography more.

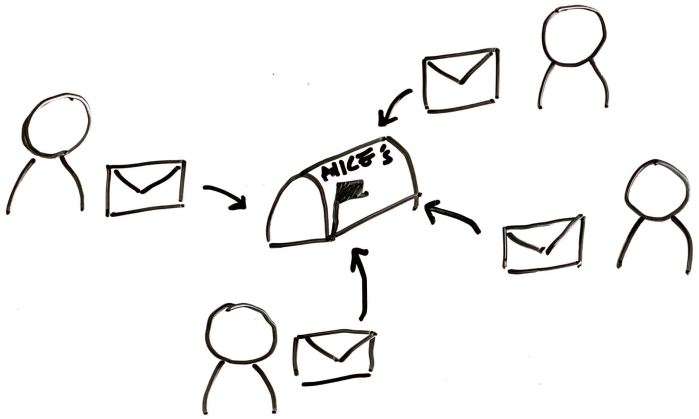

Figure 1: Everyone can access Alice's mailbox.

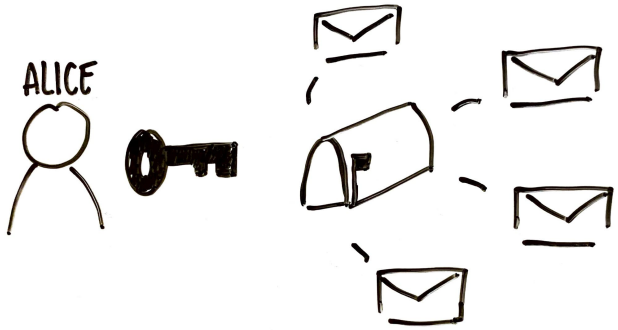

Figure 2: But only Alice can unlock it.

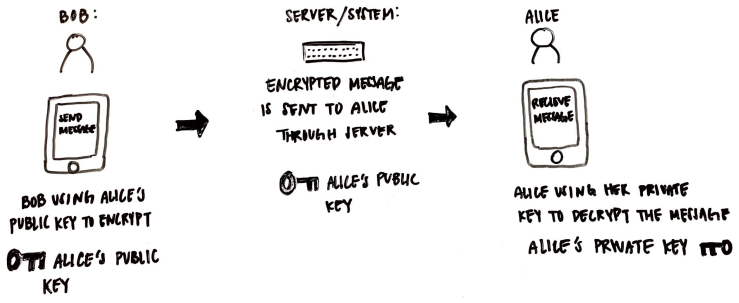

Figure 3: A more detailed version of what will be going on when they are messaging through a server.