Lab Assignment 27/03/2020:

Firstly this is directory structure:

CN_lab_2017A7PS0930G_RSA (folder)

- publicClient (folder)
  - power.h
  - encryptWithRsa.h
  - client.c
  - public_key.txt
  - Makefile
- privateServer (folder)
  - file.h
  - power.h
  - millerRabin.h
  - gcd_extended.h
  - generateVals.h
  - server.c
  - Makefile
  - .HIDDEN_FOLDER (folder)
    - private_key.txt

It is assumed that the users of publicClient folder have no idea of the privateServer folder. Also they can send encrypted messages to the server once a public_key.txt file is made available to them (indicating that the server has started a session). Users of privateServer folder are aware of the publicClient folder and can add files to it.

The respective files and folders are now explained:

publicClient files:

power.h

This header file has a function that calculates power using fast modular exponentiation.

```
#include<stdio.h>
#include<stdlib.h>

int power(int a, int b,  int n)
{
	if(b == 0) return 1;
	if(a == 1 || b == 1) return a;

	int ans = power(a, b/2, n);

	if(b % 2 == 0)
	{
		return ( (ans % n) * (ans % n) ) % n;
	}
```

```
        else
        {
                return ( ( ( (ans % n) * (ans % n) ) % n) * (a % n) ) % n;
        }
}
```

---

encryptWithRsa.h

This is a header file which makes an encryption function available to the C file that includes it. It uses the public keys n and e to encrypt the message.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include"power.h"

int encrypt(int plaintext, int e, int n)
{
        int ciphertext;

        ciphertext = power(plaintext, e, n);

        return ciphertext;
}
```

---

client.c

This C file acts as a client that connect to the server at the mentioned port number on a PC with the respective IP address. On running this, the user is prompted to enter a message as a that is encrypted and encoded character by character and sent to the server. The values of n and e (public keys) are obtained from the file public_key.txt made available by the server. This client can send only one message at a time; if we want to send multiple messages at a time, we can do so by including the code in a forever running while loop. Moreover, the encoding standard is assumed to be the same all over the two domains (i.e: publicClient and privateServer folder users would use same technique for encoding or decoding). First, each character of the string is encrypted by taking its integer equivalent and then concatenating this value to a string with a space as is done by the encode() function.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <limits.h>
```

```
#include <errno.h>
#include"encryptWithRsa.h"

int digits(int n)
{
	int counter = 0;

	while( n != 0 )
	{
		counter++;
		n /= 10;
	}

	return counter;
}

char* encode(char* str, int e, int n)
{
  int i = 0;

  char* final;

  final = (char*) malloc( strlen(str) * ( digits(n) + 1 ) );

  final[0] = '\0';

  while(str[i] != '\0')
  {
	int ch = (int)str[i], ch2;
    ch2 = encrypt(ch , e, n);

    char* buf;
    sprintf(buf, "%d ", ch2);

    strcat(final, buf);

    i++;
  }

  return final;
}

void error(char *msg)
{
   perror(msg);
   exit(0);
}

int main(int argc, char *argv[])
{
   int E, N;
```

```
    FILE* f;

    f = fopen("public_key.txt", "r"); // server has told everyone that the file pubic_key.txt that he will generate will have n and e in that same respective order with a space seperating them.

    char str1[10], str2[10];

    fscanf(f, "%s %s", str1, str2);

    N = atoi(str1);
    E = atoi(str2);

    fclose(f);

    printf("Public keys n = %d and e = %d\n", N, E);
    int sockfd, portno, n;

    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    printf("Please enter the message : ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);

    printf("\n");

    char* ciphertext = encode(buffer, E, N);

            // printf("FinalText after encryption: %s\n\n", ciphertext);
```

```
    n = write(sockfd,ciphertext,strlen(ciphertext));
    if (n < 0)
        error("ERROR writing to socket");
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
        error("ERROR reading from socket");
    printf("%s\n",buffer);
    return 0;
}
```

---

public_key.txt

This has the n and e valus stored in the same respective order with a space between them.
It is made available when the server starts its first session until which it doesn't exist. With subsequent sessions it stores the updated values of n and e.

Makefile

Used to compile the client.c file. It produces an executable named client which can removed by typing rm in the terminal.

privateServer files:

file.h

This header file stores the name and location of the file that stores the public keys and the file that stores the private key.

```
#define PRIVATE "./.HIDDEN_FOLDER/private_key.txt"
#define PUBLIC "../publicClient/public_key.txt"
```

Only the user of privateServer folder knows these files and their location.

power.h

Works similar to that of the publicClient folder's power.h file.

millerRabin.h

This is a header file that uses miller rabin's technique to predict whether an input number is prime or not with high degree of accuracy.

```
#include<stdio.h>
#include<stdlib.h>
#include"power.h"

int isPrime(int n) // uses miller rabin
{
	int i = 0;

	if( n == 2 || n == 3 ) return 1;
	if( n == 1 || n % 2 == 0 ) return 0;

	// printf("****************\n");

	int a = (rand() % (n));

	while(i < 64)
	{
		int r = n - 1;

		int x = power(a, r, n);

		while( r % 2 == 0 )
		{
			if( x == 1 || x == n - 1 )
			{
				r = r / 2;
				x = power(a, r, n);
			}
			else
			{
				return 0;
			}
		}

		a++;
		i++;
	}

	//printf("****************\n");

	return 1;
}
```

---

gcd_extended.h

This header file has a function that computes gcd and the respective coefficients in the diophantine equation using the extended gcd algorithm for the two integers.

```
#include<stdio.h>
```

```
#include<stdlib.h>


int gcd_extended(int a,          int b,    int *x,  int *y)
{
        if( a == 0 )
        {
                *x = 0, *y = 1;

                return b;
        }


        int x1, y1;

        int gcd = gcd_extended(b % a, a, &x1, &y1);

        *x = y1 - (b / a) * x1;
        *y = x1;

        return gcd;
}
```

---

generateVals.h

This header file generates the required parameters for an RSA encryption session using the power function, gcd_extended, millerRabin functions defined in the header files mentioned before. MAX determines max values of the primes p (= 2 * MAX) and q (= MAX).

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#include"gcd_extended.h"
#include"millerRabin.h"
#include"file.h"

#define MAX 100

int n, e; // public key

void generateVals()
{
        int p, q, phi;
        /* step 1 choose two large random num and check whether they are prime, then get n = pq*/

        srand(time(NULL));

        p =  (MAX + rand() % (MAX + 1)) ;
```

```
while( !isPrime(p) )
{
        p = (MAX + rand() % (MAX + 1)) ;
}

q = (rand() % (MAX + 1)) ;

while( !isPrime(q) )
{
        q = (rand() % (MAX + 1)) ;
}

n = p * q;

/*step 2: get phi(n) */

phi = (p - 1) * (q - 1);

/*step 3: get public key exponent*/

l:

e = rand() % phi;

while( !isPrime(e) )
{
        e = ( rand() % phi ) ;
}

if( phi % e == 0 ) goto l;

/*step 4: get private key */

int d = 0, y = 0, gcd;

gcd = gcd_extended(e, phi, &d, &y);

if( d < 0 )
{
        int v1 = -1 * ( (-1 * d) % phi );

        d = (phi + v1) % phi;
}

FILE* f;

f = fopen(PRIVATE, "w"); // writing d and phi to the private file
```

// we assume the sender does not know the location of this file or anybody ensuring privacy of private key d.

```
        char str[30];

        sprintf(str, "%d %d (d and phi respectively)", d, phi);

        for (int i = 0; str[i] != '\0'; i++)
        {
      /* write to file using fputc() function */
      fputc(str[i], f);
   }

   fclose(f);

   f = fopen(PUBLIC, "w");

        sprintf(str, "%d %d (n and e respectively)", n, e);

        for (int i = 0; str[i] != '\0'; i++)
        {
      /* write to file using fputc() function */
      fputc(str[i], f);
   }

   fclose(f);


/*
        printf("p: %d \n", p);
        printf("q: %d \n", q);
        printf("n: %d \n", n);
        printf("phi: %d \n", phi);
        printf("e: %d \n", e);
        printf("gcd : %d\n", gcd);
        printf("d: %d, y: %d \n\n", d, y);f
*/
        /* now ready to encrypt */

        return;
}
```

Here we see that this function generateVals() generates the public key and private key and stores them in the files public_key.txt and private_key.txt respectively. public_key.txt is made public by storing it in the publicClient folder whereas private_key,txt is only known to the user of the privateServer folder and is stored in a hidden folder .HIDDEN_FOLDER.

server.c

This is the server that first gernerates the required parameters for an RSA encryption session by invoking generateVals() and the functions as a normal server. Everytime the server is stopped and

rerun the RSA parameters are regenerated and stored in the respective files. The server accepts a message sent by the client, decodes and decrypts it character by character and displays the message. The decoding standard is know to be the same throughout the two domains (like in encoding) and is the exact opposite algorithm as of encoding as is shown by the decode() function.

```
/* A simple server in the internet domain using TCP
   The port number is passed as an argument
   This version runs forever, forking off a separate
   process for each connection
   gcc server2.c -lsocket
*/
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <limits.h>
#include <errno.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include"generateVals.h"
#include"file.h"

int d, phi;

int decrypt(int ciphertext)
{
  int finaltext = power(ciphertext, d, n);
  return finaltext;
}

char* decode(char* val)
{
   int i = 0, integer = 0, j = 0;
   char* str;

   str = (char*) malloc( 5000 * sizeof(char) ); // assuming sender won't send text larger than 5000 characters

   while (val[i] != '\0')
   {
      if(val[i] != ' ')
      {
        integer = integer * 10 + (int) val[i] - 48;
      }
      else
      {
        // printf("hi: %d\n", integer);
        integer = decrypt(integer);
        str[j++] = (char) integer;
        integer = 0;
```

```
      }

      i++;
    }
   str[j] ='\0';
   return (str);
}

void dostuff(int); /* function prototype */

void error(char *msg)
{
    perror(msg);
    exit(1);
}

void generateParametersForRsa()
{
    generateVals();

    FILE* f;

    f = fopen(PRIVATE, "r");

    char str1[10], str2[10];

    fscanf(f, "%s %s", str1, str2);

    d = atoi(str1);
    phi = atoi(str2);

    fclose(f);
}

int main(int argc, char *argv[])
{
     generateParametersForRsa();

     int sockfd, newsockfd, portno, clilen, pid;
     struct sockaddr_in serv_addr, cli_addr;

     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0)
        error("ERROR opening socket");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
```

```
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
              error("ERROR on binding");
     listen(sockfd,5);
     clilen = sizeof(cli_addr);
     while (1) {
        newsockfd = accept(sockfd,
              (struct sockaddr *) &cli_addr, &clilen);
        if (newsockfd < 0)
           error("ERROR on accept");
        pid = fork();
        if (pid < 0)
           error("ERROR on fork");
        if (pid == 0)  {
           close(sockfd);
           dostuff(newsockfd);
           exit(0);
        }
        else close(newsockfd);
     } /* end of while */
     return 0; /* we never get here */
}

/******** DOSTUFF() *********************
 There is a separate instance of this function
 for each connection.  It handles all communication
 once a connnection has been established.
 *****************************************/
void dostuff (int sock)
{
   int n;
   char buffer[256];

   bzero(buffer,256);
   n = read(sock,buffer,255);
   if (n < 0) error("ERROR reading from socket");

   printf("Here is the message received: %s\n", buffer);

   char* plaintext;

   plaintext = decode(buffer);

   printf("FinalText after decryption: %s\n\n", plaintext);

   n = write(sock,"I got your message",18);
   if (n < 0) error("ERROR writing to socket");
}
```

Makefile

Used to compile the server.c file. It produces an executable named server which can removed by typing rm in the terminal.

.HIDDEN_FOLDER (folder):

private_key.txt

This file has d and phi values stored with a space in between each other respectively.
This file is being protected by being stored in a hidden folder which only users of privateServer folder know. If any intruder arrives, he is assumed to believe there is no trace of the d and phi values (private keys).

COMPILING THE CODE:

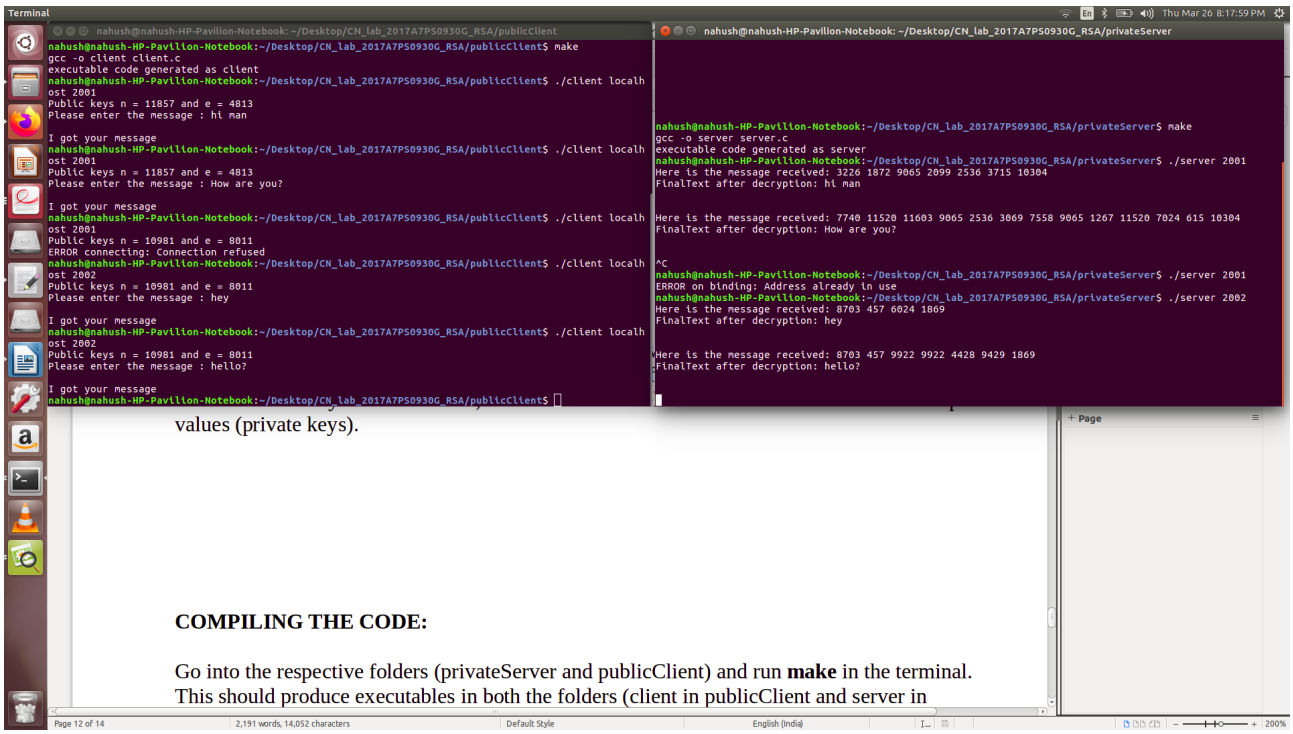
Go into the respective folders (privateServer and publicClient) and run **make** in the terminal.
This should produce executables in both the folders (client in publicClient and server in privateServer) by running their respective makefiles. Run **rm** to remove the executables from either folders if needed.

To run the executable server, we go into the folder privateServer and run the command in terminal:
./server [port number of your choice]
This should start a server session at th chosen port number on the local system.

To run the executable client, we go into the folder publicClient and run the command in terminal:
./client [IP address of server] [port number of server]
If it is the same system then IP address is **localhost.**

Ouput Produced and their pictures:

Zoom into the picture if it is not clear.



Note: The server doesn't show the d and phi values to its users (like client which shows n and e to its users), as these users are authorized and know the location of the file private_key.txt .