Research Project
Quantum Cryptography

Table of contents

Abstract

Quantum cryptography is the modern way to secure communication. Through Quantum cryptography it is going to be impossible to attack and decrypt the message. The idea behind Quantum Cryptography is to use Quantum Mechanics which implies the existence of multiple worlds with tiny differences. Using quantum physics we can make private keys so that hackers could not secretly copy the key completely. Before scientists believed the nucleus to be the powerhouse of the cell and neutrons, electrons, and protons to be fundamental parts of matter, later quantum became the fundamental particle. Quantum Cryptography is more reliable because it depends on the laws of physics rather than mathematics. By natural law, Quantum Cryptography based encrypted messages cannot be hacked, because if someone is trying to eavesdrop, the information will be changed and hence will leave traces. The idea behind Quantum Cryptography is to encode the key as a sequence of photons and then exchange between the two parties via optical fibers.

Problem Statement

Before Quantum Cryptography the preferred ways to communicate between two parties include using RSA Encryption. In this, the message is encrypted and converted into unreadable form (Cipher Text). For a hacker to figure the key, has to factorize a huge number into a product of primes. This factorization involves a lot of computational time, making it impossible. But mathematicians have been trying to find algorithms that make factorization easy. As a result of Moore's Law, it is computationally possible to up the power of computers. Hence Scientists had to come up with a more secure algorithm. The current algorithms which we are using depends on integer factorization and discrete logarithm problem and elliptic curve discrete logarithm problem. These can be problems can be solved using the powerful quantum computing. The algorithm used to crack these is called as "Shor's Algorithm".

Introduction

In 1935, Einstein along with Boris Podolsky and Nathan Rosan summarized the work on the fundamental theory of nature. In summary, it meant that if we somehow learned the physical property of a system in some way, then the property must be physically real. That property shall have a determinate value. The theory of EPR (Einstein, Podolsky, and Rosan) was further developed by David Bohm. In his theory, he suggested each particle has two sets of spins. When we look at classical mechanics, the normal cryptography data is divided into bits i.e. 0 and 1. In quantum mechanics we have qubits. Instead of being either 0 or 1, these qubits are combinations of 0 and 1. It has something from 0 and something from 1. One qubit can be 75 % ‘0’ and 25 % ‘1’. The identity is on a spectrum. Quantum Mechanics also has entanglement. Consider in classical mechanics to have two bits hence the possible outcomes are 4 (00, 01, 10, and 11). But in quantum mechanics, the two-bit system will be itself in a combination of all four possibilities. This allows the system to not just stick to only 4 outcomes, but have many open possibilities like 10% of ‘00’, 23% of ’01’ , 69 % of ‘10’, and 1 % of ‘11’. This is known as entanglement and it is powerful. Each bit has a spin, and there are two spins possible. Different directions of spins depend on Uncertainty relations. Two spins may be upwards or downwards. This is with respect to one chosen direction. That state is spherically symmetric and hence the direction doesn’t play a major role. If sender send the direction with up or down and the receiver studies it left or right then the direction for left or right is 50 – 50 percent. These two particles are emitted from a source keeping in mind that the total sum of the two particles has to be zero. While sending the two particles, they are separated such that they don’t mix with each other. When these qubits interact with each other we can develop algorithms. We can assume the x component of one particle if we know the x component of the other particle, because the spin components have to be opposite values. For the first particle if the measure is positive, then for the second particle, it has to be negative. If we try to measure the x component of one particle it has no effect on the other particle. The spin components have predetermined values of +1 or -1.

After we put the spins on the particles, next these particles are emitted along the z-axis of a channel selected by Alice and Bob. After the transmission is complete, Alice and Bob will announce in public which analyzer orientations they used for each measurement and divide the measurements into two groups: a first group for which they used different analyzer orientations and a second group for which they used the same analyzer orientation. In the process sometimes they have rejected some of the particles which they failed to register.

They give the results out and from the second group of measurements we get the resultant secret string of the bits which is the key. On the other hand we have Eve trying to snoop on the information being transmitted. Any eavesdropper cannot snoop on the information from the particles while they are travelling because the information is not encoded here.

Quantum Computers are made using the unusual behavior of subatomic parts of electrons. The way the electrons spin form the base for this new type of computing. The method to control the electrons is tough. A team of physicists lead by Jason Petta developed a method to build a quantum computer which works using millions of quantum bits. A stream of microwave photons were used to analyze pairs of electrons which were trapped in a tiny place called as quantum dot. The spin state of the electrons gives us information about the qubit. To make quantum dots, the team isolated a pair of electrons on a small section of material called a semiconductor nanowire. The wire is made so thin such that it can hold electrons. Then the electrons were held at a place the spin of the electrons were read.

Key Distribution

In quantum public key distribution we cannot send plain text in the channel directly. Instead a string of random bits are sent. At first the two users share no secret information. They just check if the channel or the transmission medium have high probability for eaves dropping. If they observe that the channel is safe and there is no sign of eavesdropping they start sending the secret bits with one-time padding to conceal the meaning of plain text. If disturbance is observed during the transmission they discard the bits.

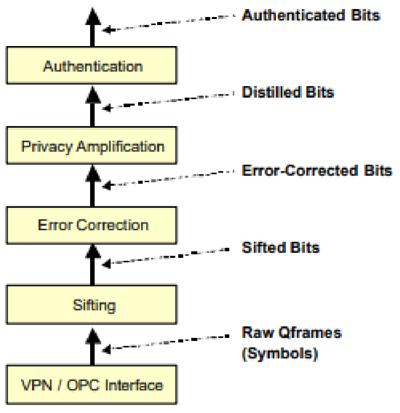
Alice chooses a random bit string and a random sequence of polarization bases followed by a beam of photons. Each of that photon has one bit from the string. Whether the bit represents 0 or 1 is determined from the way photon is standing. The receiver gets the useful information from only the half of the photons which are detected on the right polarization bases. Two photos in an EPR pair are anti correlated. They will always have opposite polarizations, when they are on the same basis. Some of the photons are likely to be lost during the transmission of the message or due to inability of an efficient detector. Since there is a mix of rectilinear and diagonal photons during the transmission if there is a hacker it will alter the transmission and it will produce a disagreement between the sender and the receiver. It can be shown that no measurement on a photon in transit by a hacker who is aware of the photon's original basis after he has performed his measurement can yield more than half expected bits of information about the main bit encoded by that photon and that any measurement yielding b bits of expected information must cause a disagreement. Sender and receiver can thus test for a snooper by comparing some of the bits on which they think they should agree. They check for some bits by randomly selecting some of the bit positions of the correctly received bits. If all the comparisons agree then the communication has been done without intrusion.

When we want to prove security of a cryptosystem we have a computational system and a reduction form. If we break the security of a cryptosystem that means we have broken the computational assumption behind the system. Post Quantum Cryptosystem they were developed keeping in mind to be secure from quantum attacks.

QKD has the main advantage of providing confidentiality of keys. Public keys are experiencing uncertainty because decryption is mathematically intraceable. Once if the classic methods are broken it can also lead to giving out information about communication done in the past. QKD does provide authentication by itself but as of now the systems include prepositioning of secret keys at two devices being used in the hash – based authentication system or QKD – public key techniques. Trying to implement QKD- public key technique can be computationally and logically challenging. This approach can be more vulnerable to DOS attacks. Key delivery systems must distribute keys quickly enough to prevent encryption devices from running out of key bits. There is a race between the rate at which content is installed and the rate at which it is used for encryption and decryption. As of now the QKD systems are running at 1000 bits per second which is slow. If the keying material is used as input for less reliable algorithms like the Advanced Encryption Standard, it may be suitable. The keying material is important for a secure connection. The system has a primary goal of transferring the keying material without getting disrupted. Mostly the system is fragile service because the communication is point to point. If an intruder was trying to snoop into the channel or there is a trouble with optical fiber then the keying material will be disrupted and stopped. Attackers might be able to conduct traffic analysis on a key delivery system.

QKD Protocol Stack

There are five stages in protocol stack of Quantum Key Distribution.



Sifting is the process where Sender and Receiver discard all the failed qubits from a pulses. This might be because of fault in sender's transmitter, or Receiver's detector fault or the photons got lost in between transmission. After this round of protocol is done, after a sift and sift transaction, both sender and the receiver discards all the useless symbols from their storage. Only those whose basis matched will be kept. Sifting dramatically prunes the number of symbols.

Error Correction allows Alice and Bob to determine all the "error bits" among their shared, sifted bits and correct them such that the sender and the receiver share the same sequence of error free bits. Error bits are ones that sender transmitted as 0 but the receiver received them as 1 or vice versa. This can be caused by noise or attackers. Error correction in quantum Cryptography has a unusual constraint namely, evidence revealed in error detection and correction must be assumed to be known to attacker and thus to reduce entropy available for key material. Error correction is done using Cascade protocol and algorithm. It will accurately detect and correct many errors.

Privacy Amplification is the process where sender and receiver reduce Attacker's knowledge about the shared bits to an acceptable level. A linear function is chosen by either of sender or receiver where the security has to be increased. The Hash function is chosen over a Galois Field GF[2^n] where n is the number of bits as input, rounded to multiple of 32.

Authentication allows both the sender and receiver to protect the system against Man in the Middle attacks. Authentication must be performed on an ongoing basis for all management traffic since attacker may flood into the communication. A small secret key is used to select a hash function from the family to generate authentication hash of the public correspondence between them. This secret key cannot be used again on different data without compromising the security. The VPN data traffic is also applied with some of these techniques.

Limitations

Since it is early stages of development, there are limitations with Quantum Cryptography too when we attempt to use it practically. Sometimes the hacker can destroy the detector with an intense, powerful pulse, making the receiver unable to process the photons. Sometimes, it can happen so that the transmitter might generate a second photon with the same information. If the hacker gets to know about this then the hacker can simply snoop on the second photon and no one would suspect the lost photon. Sometimes the qubits which we transfer might have anomalies. The existing experimental process for quantum cryptography is very delicate. Attacks of low intensity can disturb the security easily. In a quantum system where we have a four state scheme the sender has two conjugate bases for the polarization of single photons. In basis they have two orthogonal states.

Conclusion

Quantum Computing is here to stay. Problems that are hard for classical mechanics which require computational time equal to a lifetime of Universe can be solved by using Quantum Computing easily and faster. Using Quantum teleportation we can send the data from one place to another without actually sending data physically. Quantum Cryptography was discovered in the United States by Stephen Wiesner at the Columbia University in New York. He introduced the idea of quantum conjugate coding. Quantum Computing opens up new ways for key exchanging. He used it for unforgeable banknotes. Since then many scientists have worked on Quantum Cryptography. In 1984, the main breakthrough occurred when Bennett and Brassard used for secure communication. They called it BB84, the first quantum cryptography protocol. Over the years, Quantum cryptography has been seeing drastic improvement in its technology because we can add more security to our communication over large distance using entanglement swapping. Companies such as Magi Q Technologies, Toshiba, Quintessence Labs manufacture different quantum cryptography systems. Samsung has recently announced their new phone named Quantum 2 which has built – in quantum cryptography technology for more security. The chip inside is made by a company called ID Quantique. It works by capturing noise with an LED and a CMOS image sensor.

Citations

- C. H. Bennett, G. Brassard, and A. K. Ekert. Quantum Cryptography. *Scientific American*, 267(4):50–57, October 1992.
- C. Zimmer. Perfect Gibberish. *Discover*, 13(12):92–99, December 1992
- Chip Eliott, Dr. David Pearson, Dr. Gregory Troxel. Quantum Cryptography in Practice
- Quantum cryptography. How quantum technologies enable… | by Alex Schuckert | Many Body Physics | Medium
- Understanding Quantum Cryptography | Hacker Noon
- Quantum Communications and Cryptography. Alexander V. Sergienko
- Quantum cryptography - Wikipedia

Report

Program 1 – RSA Decryption

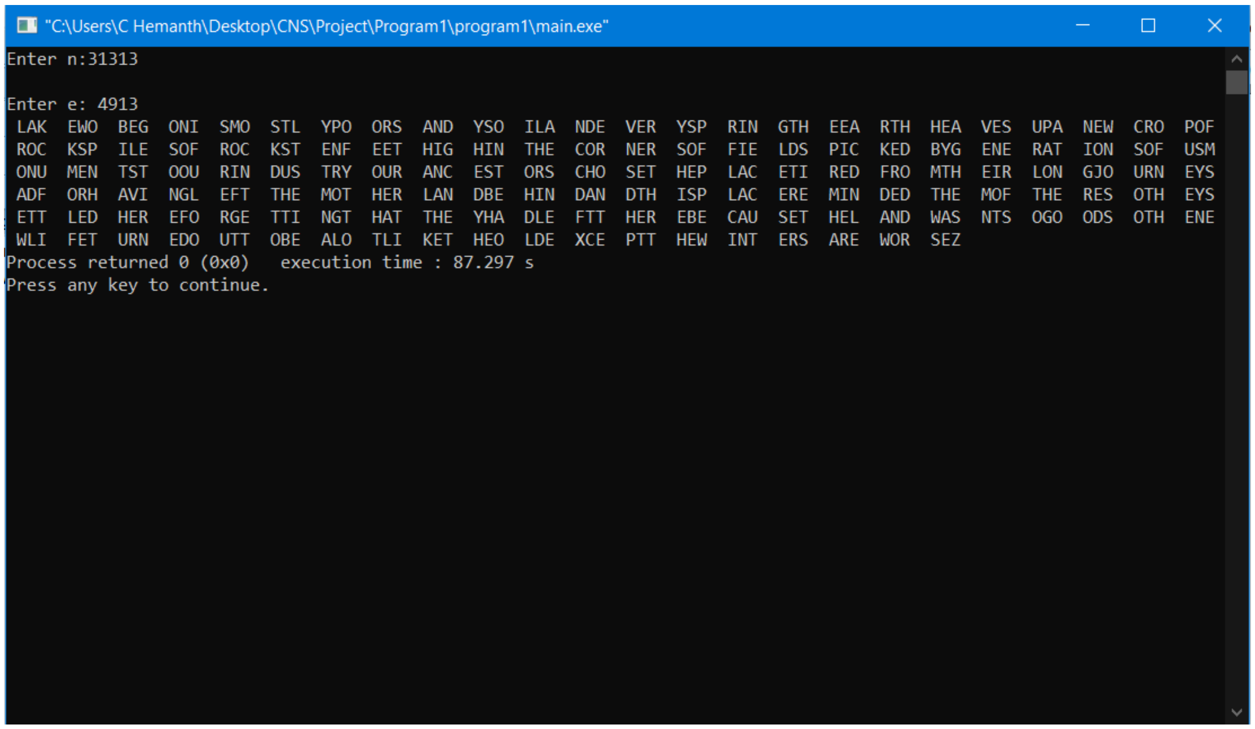
This program has been written in C language. The Cipher Text is stored in array of datatype 'long' . The length of the Cipher text array has been found out using the function CalcLenCip(). The Length was found out to be 139. The values of n and e are taken from the user. The value of n is sent to the function CalcPhi() function. In this function we find the values of p and q. We know from rules of RSA that p and q are going to be prime numbers. Hence we can skip all the even numbers. We have a nested for loop, with two loop variables. When we get two different values of two variables, it implies we have the values of p and q. The stopping condition for our nested for loop is when the product of our loop variables is equal to the value of n.

Now that we have values of p and q, we can find value of phi(n) which is equal to (p-1)*(q-1). Next we calculate the value of e^{-1} using the calcEInv() function. In the calcEInv() function we run a for loop starting from 2 and forcefully end it when loop variable times e mod phi(n) becomes 1. When this happens, we break the for loop and save value of loop variable, which becomes our value for e^{-1}. Then we send each of our Cipher word to decrypt() function.

In the decrypt() function, we first send each value to Dec() function. The Dec() function returns a long value. In Dec() function we use the square and multiply method. We send it to the fast() method with three parameters, cipher text, d,n. We do this because plain text = c^{d}mod n, from formula. The returned value from fast exponentiation is then run through three nested for loops. Since we know that the plain text is in the form of (a*26*26 + b*26 + c) we have three nested loop variables running from 0 until 26(0 being A and 25 being Z). when the three loop variables satisfy the condition we forcefully exit the nested for loop before saving the values of the three loop variables.

The Three loop variables are our required three alphabets decrypted. We can add 65 (ASCII value of A) to get the Alphabetical decrypted values. We repeat the process by sending all the Cipher text into decrypt() function all of them. The decrypted text in Alphabetical form is

LAK EWO BEG ONI SMO STL YPO ORS AND YSO ILA NDE VER YSP RIN GTH EEA RTH HEA VES UPA NEW CRO POF ROC KSP ILE SOF ROC KST ENF EET HIG HIN THE COR NER SOF FIE LDS PIC KED BYG ENE RAT ION SOF USM ONU MEN TST OOU RIN DUS TRY OUR ANC EST ORS CHO SET HEP LAC ETI RED FRO MTH EIR LON GJO URN EYS ADF ORH AVI NGL EFT THE MOT HER LAN DBE HIN DAN DTH ISP LAC ERE MIN DED THE MOF THE RES OTH EYS ETT LED HER EFO RGE TTI NGT HAT THE YHA DLE FTT HER EBE CAU SET HEL AND WAS NTS OGO ODS OTH ENE WLI FET URN EDO UTT OBE ALO TLI KET HEO LDE XCE PTT HEW INT ERS ARE WOR SEZ


```
"C:\Users\C Hemanth\Desktop\CNS\Project\Program1\program1\main.exe"
Enter n:31313

Enter e: 4913
 LAK  EWO  BEG  ONI  SMO  STL  YPO  ORS  AND  YSO  ILA  NDE  VER  YSP  RIN  GTH  EEA  RTH  HEA  VES  UPA  NEW  CRO  POF
 ROC  KSP  ILE  SOF  ROC  KST  ENF  EET  HIG  HIN  THE  COR  NER  SOF  FIE  LDS  PIC  KED  BYG  ENE  RAT  ION  SOF  USM
 ONU  MEN  TST  OOU  RIN  DUS  TRY  OUR  ANC  EST  ORS  CHO  SET  HEP  LAC  ETI  RED  FRO  MTH  EIR  LON  GJO  URN  EYS
 ADF  ORH  AVI  NGL  EFT  THE  MOT  HER  LAN  DBE  HIN  DAN  DTH  ISP  LAC  ERE  MIN  DED  THE  MOF  THE  RES  OTH  EYS
 ETT  LED  HER  EFO  RGE  TTI  NGT  HAT  THE  YHA  DLE  FTT  HER  EBE  CAU  SET  HEL  AND  WAS  NTS  OGO  ODS  OTH  ENE
 WLI  FET  URN  EDO  UTT  OBE  ALO  TLI  KET  HEO  LDE  XCE  PTT  HEW  INT  ERS  ARE  WOR  SEZ
Process returned 0 (0x0)   execution time : 87.297 s
Press any key to continue.
```

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

long Cipher [] = {6340, 8309, 14010, 8936, 27358, 25023, 16481, 25809, 23614, 7135,
                  24996, 30590, 27570, 26486, 30388, 9395, 27584, 14999, 4517, 12146, 29421, 26439,
                  1606, 17881, 25774, 7647, 23901, 7372, 25774, 18436, 12056, 13547, 7908, 8635, 2149,
                  1908, 22076, 7372, 8686, 1304, 4082, 11803, 5314, 107, 7359, 22470, 7372, 22827, 15698,
                  30317, 4685, 14696, 30388, 8671, 29956, 15705, 1417, 26905, 25809, 28347, 26277, 7897, 20240,
                  21519, 12437, 1108, 27106, 18743, 24144, 10685, 25234, 30155, 23005, 8267, 9917, 7994, 9694,
                  2149, 10042, 27705, 15930, 29748, 8635, 23645, 11738, 24591, 20240, 27212, 27486, 9741, 2149,
                  29329, 2149, 5501, 14015, 30155, 18154, 22319, 27705, 20321, 23254, 13624, 3249, 5443, 2149,
                  16975, 16087, 14600, 27705, 19386, 7325, 26277, 19554, 23614, 7553, 4734, 8091, 23973, 14015,
                  107, 3183, 17347, 25234, 4595, 21498, 6360, 19837, 8463, 6000, 31280, 29413, 2066, 369, 23204,
8425,
                  7792, 25973, 4477, 30989
                 };


//139 words 0 to 138


long Ciplen, n, e, phi,p,q, phiN,d, eInv;

void CalcPhi(long n)
{
    long i, num = n;
    for (i = 1; i <= num; i+=2)
    {
        for( int j = 1; j <= num; j+=2)
            if ((j* i == num )&& (i != 1))
            {
                //    printf("In CalcPhi %ld  %ld\n", i, j);
                if( (i!=1)&& (j!=1) )
                {
                    //       printf("\nIn if i = %ld, j = %ld", i, j);
                    p = j;
                    q = i;
                }
                break;
            }
    }
    phiN = (p-1)*(q-1);
}
void calcLenCip()
{
    int i;
    for( i = 0; ; i++)
    {
        //  printf(" %ld = %ld \n ", Ciplen, Cipher[i]);
        Ciplen++;
        if(Cipher[i] == 30989) break;
    }

}
long fast( long x, long H, long n)
{
  //  printf("\n In fast, einv = %ld", H);
    long h;
    unsigned long long r;
    int bin[32];
    int i;

    r = x;
    i = 0;

    /* Converts H in Binary */
```

```c
    while( H > 0 )
    {

        if (H % 2 == 0)
        {
            bin[i] = 0;
        }
        else
        {
            bin[i] = 1;
        }

        H = H/2;
        i++;

    }

    i--; //t-1

    while(i>0)
    {

        r = (r * r) % n;

        if( bin[--i] == 1 )
        {
            r = (r * x) % n;
        }

    }
  //  printf("\n test = %ld eINv = %d n = %ld r  = %ld\n",x,H, n,r );
    return r;

}
void calcEInv( )
{

   // printf("\n in e inverse");
    for( long i = 2; ; i++)
    {
        if(((i*e)%phiN == 1) )
        {
            eInv= i;
            break;

        }
    }
//    printf("eInv = %ld ", eInv );
    d = eInv;

}

long Dec( int here)
{
  //  printf("\nIn Dec , %ld", here);


     long decr = fast ( here, d, n);
    // printf("\n here%ld,decr =  %ld\n",here, decr );
    return decr;

}
void decrypt(int here)
{
    long testing = Dec(here);
    //printf("\n test = %ld\n", testing);
    int i,ii, j,jj,k,kk, flag = 0;
    for( i = 0; i<26; i++)
    {
        for(j = 0; j<26; j++ )
        {
            for( k = 0; k < 26; k++)
            {
                if( ((i*26*26)+(j*26)+k) == testing)
                {
                    printf(" %c%c%c ", i+65, j+65, k+65);
                    ii = i;
                    jj = j;
                    kk = k;
                    flag = 1;
                    break;

                }
            }
        }
    }


}
int main()
{
    long f;

    printf("Enter n:");
    scanf("%ld", &n);
    printf("\nEnter e: ");

    scanf("%ld", &e);
    calcLenCip();// Calculates the length of Cipher text array
    CalcPhi(n);// Calculates Phi of n
    calcEInv();// Calculates e inverse
   // printf("p = %ld, q = %d", p, q);
   // printf("\n phin = %ld", phiN);

    for( f = 0; f < Ciplen; f++)
    {
     //   printf("%ld = ", f );
        decrypt((Cipher[f]));
    }


}
```

Report

Program 2 – El Gamal Decryption

This program has been written in C language. The Cipher Text is stored in array of datatype 'long' . The length of the Cipher text array has been found out using the function CalcCiphLen(). The Length was found out to be 204 . This 204 elements are divided into 102 pairs, each pair consists of c1, c2. To decrypt in ELGamal system the formula is [c2*($c1^{d}$ $)^{-1}$]mod p. We have c1 and c2, to find value of d, CalcD() function has been used. We get value of d using the formula e2 = $e1^{d}$mod p. Since we know the value of e2, e1 and p, to find out d, for loop has been used. Our For loop varies values of d from 2, until p-1 values. Our for loop forcefully exits when e2 value equals $e1^{d}$ mod p.

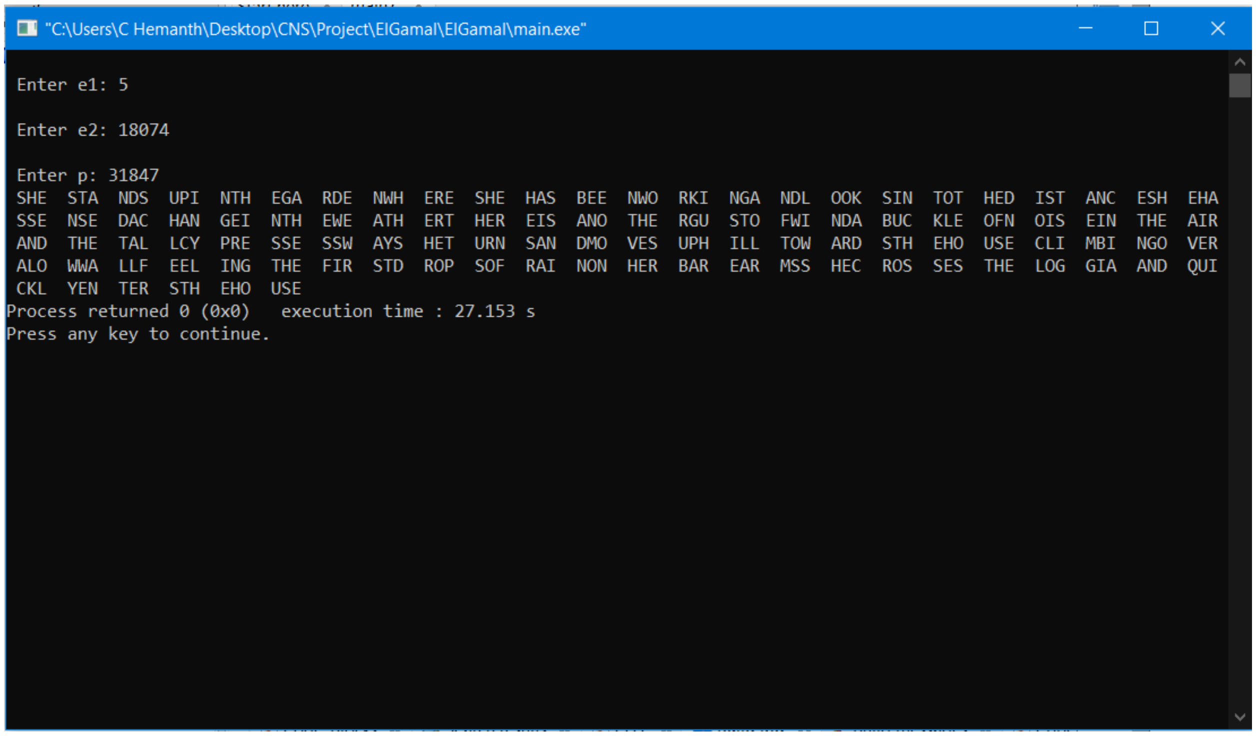
Now that we have values of c2, c1, d and p, we can use the formula P = [c2 * $(c1^{d})^{-1}$] mod p. We decrypt in Decrypt() function. Using Fermat's Little Theorem we can calculate $(c1^{d})^{-1}$as $c1^{p-1-d}$ .Since the value of $c1^{p-1-d}$ is very bigger to calculate, we use square and multiply method. The square and multiply method is written in fast(). The fast() takes three inputs which are c1, p-1-d and p. The returned value from fast() is multiplied with c2 and then mod with p, to get the decrypted integer value. One after other all the 102 decrypted values are added to the array Dec[].

This decrypted values present in the Dec array still have to bring back to the alphabetical form. Each value from the array is then sent to Inv() function. Since we know each that each value is in the form of

(a*26*26 + b*26 + c), three nested for loops have been used. Three for loops start from 0 until 26, because a, b, c can only take values 0-25 (0 being A and 25 being Z). When the three loop variables satisfy the condition ie (a*26*26 + b*26 + c) == test, the nested for loop is forcefully exited, saving the values of three loop variables. These three loop variables when added with 65 (ASCII value of A), we get the plain text and hence this process is repeated for all the values of Dec[] array.

The plain text after running the program is:

SHE STA NDS UPI NTH EGA RDE NWH ERE SHE HAS BEE NWO RKI NGA NDL OOK SIN TOT HED IST ANC ESH EHA SSE NSE DAC HAN GEI NTH EWE ATH ERT HER EIS ANO THE RGU STO FWI NDA BUC KLE OFN OIS EIN THE AIR AND THE TAL LCY PRE SSE SSW AYS HET URN SAN DMO VES UPH ILL TOW ARD STH EHO USE CLI MBI NGO VER ALO WWA LLF EEL ING THE FIR STD ROP SOF RAI NON HER BAR EAR MSS HEC ROS SES THE LOG GIA AND QUI CKL YEN TER STH EHO USE



"C:\Users\C Hemanth\Desktop\CNS\Project\ElGamal\ElGamal\main.exe"

```
 Enter e1: 5

 Enter e2: 18074

 Enter p: 31847
 SHE  STA  NDS  UPI  NTH  EGA  RDE  NWH  ERE  SHE  HAS  BEE  NWO  RKI  NGA  NDL  OOK  SIN  TOT  HED  IST  ANC  ESH  EHA
 SSE  NSE  DAC  HAN  GEI  NTH  EWE  ATH  ERT  HER  EIS  ANO  THE  RGU  STO  FWI  NDA  BUC  KLE  OFN  OIS  EIN  THE  AIR
 AND  THE  TAL  LCY  PRE  SSE  SSW  AYS  HET  URN  SAN  DMO  VES  UPH  ILL  TOW  ARD  STH  EHO  USE  CLI  MBI  NGO  VER
 ALO  WWA  LLF  EEL  ING  THE  FIR  STD  ROP  SOF  RAI  NON  HER  BAR  EAR  MSS  HEC  ROS  SES  THE  LOG  GIA  AND  QUI
 CKL  YEN  TER  STH  EHO  USE
Process returned 0 (0x0)   execution time : 27.153 s
Press any key to continue.
```

```c
#include <stdio.h>
#include <stdlib.h>
long cipLen, e1, e2,p, d,Dec[102], DecLen  ;//8
long cip[] = {3781, 14409, 31552,3930,27214,15442, 5809, 30274,
              5400, 31486, 19936, 721, 27765, 29284, 29820, 7710,
              31590, 26470, 3781, 14409, 15898, 30844, 19048, 12914,
              16160, 3129, 301, 17252, 24689, 7776, 28856, 15720,
              30555, 24611, 20501, 2922, 13659, 5015, 5740, 31233,
              1616, 14170, 4294, 2307, 2320, 29174, 3036, 20132,
              14130, 22010, 25910, 19663, 19557, 10145, 18899, 27609,
              26004, 25056, 5400, 31486,9526, 3019,12962, 15189,
              29538, 5408, 3149, 7400, 9396, 3058,27149, 20535,
              1777, 8737, 26117, 14251, 7129, 18195, 25302, 10248,
              23258, 3468, 26052, 20545, 21958, 5713, 346, 31194,
              8836, 25898, 8794, 17358, 1777, 8737,25038, 12483,
              10422, 5552, 1777, 8737, 3780, 16360,11685, 133,
              25115, 10840, 14130, 22010, 16081, 16414, 28580, 20845,
              23418, 22058, 24139, 9580, 173, 17075, 2016, 18131,
              19886, 22344, 21600, 25505, 27119, 19921,23312,16906,
              21563, 7891, 28250, 21321, 28327, 19237, 15313, 28649,
              24271, 8480, 26592, 25457, 9660, 7939, 10267, 20623,
              30499, 14423, 5839, 24179, 12846, 6598, 9284, 27858,
              24875, 17641, 1777, 8737, 18825, 19671, 31306, 11929,
              3576, 4630, 26664, 27572, 27011, 29164, 22763, 8992,
              3149, 7400, 8951, 29435, 2059, 3977, 16258, 30341,
              21541, 19004, 5865, 29526, 10536, 6941, 1777, 8737,
              17561, 11884, 2209, 6107, 10422, 5552, 19371, 21005,
              26521, 5803, 14884, 14280, 4328, 8635, 28250, 21321,
              28327, 19237, 15313, 28649
             };
void CalcCiphLen()
{
    int i = 0;
    for ( i = 0 ; cip[i]; i++) {}
    cipLen = i;
}
long fast (long x, long H, long n)
{

  long h;
  unsigned long long r;
  int bin[32];
  int i;

  r = x;
  i = 0;

  /* Converts H in Binary */
  while (H > 0)
    {

      if (H % 2 == 0)
    {
      bin[i] = 0;
    }
      else
    {
      bin[i] = 1;
    }

      H = H / 2;
      i++;

    }

  i--;              //t-1
  while (i > 0)
    {

      r = (r * r) % n;

      if (bin[--i] == 1)
    {
      r = (r * x) % n;
    }

    }

  return r;

}
void CalcD()
{

    for (long i = 2; i < p - 1; i++)
    {

        long test = fast (e1, i, p);
        if (test == e2)
        {
         //   printf ("\nd is %d", i);
           d = i;
            break;

        }

    }


}

void Decrypt(){
 for( int i = 0; i < cipLen;i+=2){
       // printf("\n %ld %ld", cip[i], cip[i+1]);
        long c2 = cip[i+1],c1 = fast(cip[i],p-1-d, p );
        //printf(" c1 = %ld, c2 = %ld", c1, c2);
        Dec[DecLen] = (c1*c2)%p ;
        DecLen++;
    }
   // for( int i = 0; i < DecLen; i++)
   //     printf("\n %ld", Dec[i]);
}
void Inv(long test){
   // printf("\n In Inv %ld", test);
    int i,ii, j,jj,k,kk, flag = 0;
    for( i = 0; i<26; i++)
    {
        for(j = 0; j<26; j++ )
        {
            for( k = 0; k < 26; k++)
            {
                if( ((i*26*26)+(j*26)+k) == test)
                {
                    printf(" %c%c%c ", i+65, j+65, k+65);
                    ii = i;
                    jj = j;
                    kk = k;
                    flag = 1;
                    break;

                }
            }
```

```c
        }
    }



}
int main()
{

    printf("\n Enter e1: ");
    scanf("%ld", &e1);
    printf("\n Enter e2: ");
    scanf("%ld", &e2);
    printf("\n Enter p: ");
    scanf("%ld", &p);
    // printf("\n e1 = %ld e2 = %ld p:%ld  ", e1,e2,p);
    CalcCiphLen();
    CalcD();
    Decrypt();
    for( int i = 0; i < DecLen; i++){
        Inv(Dec[i]);

    }


}
```