A Mini Project report on

RABIN CRYPTOSYSTEM

submitted in partial fulfillment of the course

CSE-1007: Introduction to Cryptography

By

18BCD7009	Rishabh Singh Bais
18BCN7057	Paduchuru Abhishek
18BCD7065	Ahmed Amjad Hussain
18BCN7054	E Rohit Reddy
18BCE7091	Pulkit Joshi

School of Computer Science & Engineering

VIT-AP UNIVERSITY

INAVOLU

OCTOBER, 2019

CONTENTS

List of figures

List of Tables

Introduction

The Rabin cryptosystem is an asymmetric cryptographic technique, whose security, like that of RSA, is related to the difficulty of factorization. However, the Rabin cryptosystem has the advantage that the problem on which it relies has been proved to be as hard as integer factorization, which is not currently known to be true of the RSA problem. It has the disadvantage that each output of the Rabin function can be generated by any of four possible inputs; if each output is a cipher text, extra complexity is required on decryption to identify which of the four possible inputs was the true plaintext. The process was published in January 1979 by Michael O. Rabin. The Rabin cryptosystem was the first asymmetric cryptosystem where recovering the entire plaintext from the cipher text could be proven to be as hard as factoring.

Theoretical study

Like all asymmetric cryptosystems, the Rabin system uses a key pair: a public key for encryption and a private key for decryption. The public key is published for anyone to use, while the private key remains known only to the recipient of the message.

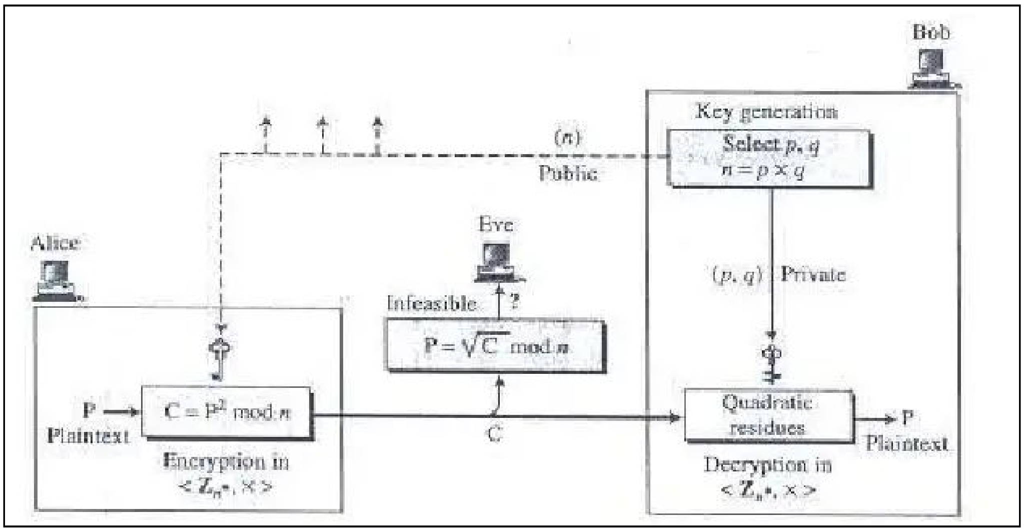


Figure 1: Rabin Encryption and Decryption

Key generation

The keys for the Rabin cryptosystem are generated as follows:

1. Choose two large distinct prime numbers p and q such p = 3 mod 4 and q = 3 mod 4
2. Compute $n = p*q$.

Then n is the public key and the pair (p,q) is the private key.

Encryption

A message *M* can be encrypted by first converting it to a number m<n using a reversible mapping, then computing $c = m^2 \bmod n$. The ciphertext is c.

Decryption

The message *m* can be recovered from the ciphertext c by taking its square root modulo n as follows.

Compute the square root c of modulo p and q using these formulas:

$m_p = c^{1/4(p+1)} mod\, p$

$m_q = c^{1/4(q+1)} mod\, q$

Use the extended Euclidean algorithm to find y_p and y_q such that $y_p.p + y_q.q = 1$.

Use the Chinese remainder theorem to find the four square roots of c modulo n :

$r_1 = (y_p..p.m_q + y_q.q.m_p) mod\, n$

$r_2 = n - r_1$

$r_3 = (y_p..p.m_q - y_q.q.m_p) mod\, n$

$r_4 = n - r_3$

One of these four values is the original plaintext m , although which of the four is the correct one cannot be determined without additional information.

Computing square roots

We can show that the formulas in step 1 above actually produce the square roots of as follows. For the first formula, we want to prove that $m_p{}^2 \equiv c mod\, p$ .Since $p \equiv 3 mod\, 4$ the exponent $1/4\,(p+1)$ is an integer. The proof is trivial if $c \equiv 0\ mod\, p$ so we may assume that does not divide . Note that $c \equiv m^2 mod\, pq$ implies that $c \equiv m^2 mod\, q$, so c is a quadratic residue modulo p . Then

$m_P{}^2 \equiv c^{1/2(p+1)} \equiv c.c^{1/2(p-1)} \equiv c.1\ mod\, p$

The last step is justified by Euler's criterion.

Example

As an example, take

$P = 7$ and $q = 11$, then $n=77$. Take $m = 20$ as our plaintext.

The ciphertext is thus $c = m^2 mod\, n = 400\ mod\ 77 = 15$.

Decryption proceeds as follows:

Compute

$m_p = c^{1/4(p+1)} mod\, p = 15^2\ mod\ 7 = 1$

And

$m_q = c^{¼(q+1)} mod\, q = 15^3\ mod\ 11 = 9$

Use the extended Euclidean algorithm to compute $y_p = -3$ and $y_q = 2$. We can confirm that $y_p.p + y_q.q = (-3.7) + (2.11) = 1$.

Compute the four plaintext candidates:

$r_1 = (-3.7.9 + 2.11.1) mod\ 77 = 64$

$r_2 = 77 - 64 = 13$

$r_3 = (-3.7.69 - 2.11.1) mod\ 77 = 20$

$r_4 = 77 - 20 = 50$

and we see that r_3 is the desired plaintext. Note that all four candidates are square roots of 15 mod 77. That $r_i^2 mod\ 77 = 15$ is, for each candidate, so each r_i encrypts to the same value, 15.

ASCII Table

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/

Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_

Decimal	Hexadecimal	Binary	Octal	Char
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Table 1: ASCII Table

Pseudo code

```
p= input (p)

q= input(q)

n=p*q

filename= input(Enter the file name in which plaintext is present)

call FileReading function and pass filename

plaintext = FileReading(filename)

print plaintext

FOR (i=0 to plaintext length)

        store each character in ch

        typecast ch in inetger and store in PlainText

        call Encrytion function and pass PlainText

        store return value in variable Cipher

        store Cipher in an integer array cip

END LOOP

print cip

store cip in string variable in ciphertext

filename=input(Enter the file name where the ciphertext is to be stored)

call FileWriting function and pass the filename and ciphertext

for(i=0 to plaintext length)

        call Decryption function and pass each value of cip array
```

```
END LOOP
print dec1,dec2,dec3,dec4
initialize decrypted_str
FOR (i=0 to plainttext length)
    IF(dec1 has character between A to Z or white space or between 0 to 9)
        THEN add that character to decrypted_str
    do same for dec2,dec3,dec3
END LOOP
filename=input("Enter the file name where decrypted text to be stored")
call FileWriting function and pass filename, decrypted_str
print decryted_str
FUNCTION FileReading(filename)
    scan the file
    read the plain text from file
    while (till last line )
        read each line
        store in a variable text
    END LOOP
    close file
    return text
END FUNCTION
```

```
FUNCTION FileWriting(filename,text)

        create a text file

        read the content from text

        print in the file

END FUNCTION


FUNCTION Encryption(PlainText)

        call FindT function and pass Plaintext,e,n

        store the return value in cipher

        return cipher

END FUNCTION

FUNCTION FindT(a,m,n)

        calculating a=(a^m) % n

        return a

END FUNCTION

FUNCTION Decrytion(Cipher)

        initialize P1,P2,P3,P4,a1,a2,a3,a4

        calculate a1 and a3 using FindT function

        calculate a2 = p - a1

        calculate b2 = q - b1

        calculate P1,P2,P3,P4 using ChineeseRemainderTheorem function
```

```
        store all the value in p1,p2,p3,p4 array
        convert p1,p2,p3,p4  to string and store in dec1,dec2,dec3,dec4
respectively
END FUNCTION
FUNCTION ChineeseRemainderTheorem(a,b,m1,m2)
        intialize M, M1, M2, M1_inv, M2_inv
        intialize result
        M = m1 * m2
        M1 = M / m1
        M2 = M / m2
        M1_inv = inverse(m1, M1)
        M2_inv = inverse(m2, M2)
        result = (a * M1 * M1_inv + b * M2 * M2_inv) % M
        return result
END FUNCTION
FUNCTION inverse(a,b)
        int inv=0
        intialize q, r, r1 = a, r2 = b, t, t1 = 0, t2 = 1
        while (r2 greater than 0)
                q = r1 / r2
                r = r1 - q * r2
```

```
        r1 = r2
        r2 = r
        t = t1 - q * t2;
        t1 = t2
        t2 = t
    END LOOP
    if (r1 == 1)
    inv = t1
    if (inv < 0)
    inv = inv + a
    return inv
END FUNCTION
```

Implementation

```
package rabinusingascii;
import java.io.*;
import java.util.*;
import java.lang.*;
public class AsciiRabin {
static int e = 2, n;
static int p, q;
static int[] cip=new int[100];
static char[] ch=new char[100];
static int[] newch=new int[100];
static int[] p1=new int[100];
 static int[] p2=new int[100];
static int[] p3=new int[100];
static int[] p4=new int[100];
static char ch1[]=new char[100];
static char ch2[]=new char[100];
static char ch3[]=new char[100];
 static char ch4[]=new char[100];
static String dec1="";
static String dec2="";
```

```
static String dec3="";
static String dec4="";
static int FindT(int a, int m, int n)
{
        int r;
        int[] y =new int[1] ;
        int[] x =new int[1] ;
        x[0]=a;
        y[0]=1;
        while (m > 0) {
                r = m % 2;
                FastExponention(r, n, y, x);
                m = m / 2;
        }
        return y[0];
}
static void FastExponention(int bit, int n, int[] y, int[] x)
{
        if (bit == 1)
                y[0] = (y[0] * x[0]) % n;
```

```
        x[0] = (x[0] * x[0]) % n;
}
static int Encryption(int PlainText)
{
        int cipher = FindT(PlainText, e, n);
        return cipher;
}
static int inverse(int a, int b)
{
        int inv=0;
        int q, r, r1 = a, r2 = b, t, t1 = 0, t2 = 1;
        while (r2 > 0) {
                q = r1 / r2;
                r = r1 - q * r2;
                r1 = r2;
                r2 = r;
                t = t1 - q * t2;
                t1 = t2;
                t2 = t;
        }
        if (r1 == 1)
```

```
            inv = t1;
        if (inv < 0)
            inv = inv + a;
        return inv;
    }
    static void Decryption(int Cipher)
    {
        int i=0;
       int P1, P2, P3, P4;
        int a1, a2, b1, b2;
        a1 = FindT(Cipher, (p + 1) / 4, p);
        a2 = p - a1;
        b1 = FindT(Cipher, (q + 1) / 4, q);
        b2 = q - b1;
        P1 = ChineseRemainderTheorem(a1, b1, p, q);
        P2 = ChineseRemainderTheorem(a1, b2, p, q);
        P3 = ChineseRemainderTheorem(a2, b1, p, q);
        P4 = ChineseRemainderTheorem(a2, b2, p, q);

      p1[i]=P1;
      p2[i]=P2;
```

```
        p3[i]=P3;
        p4[i]=P4;
        dec1+=(char)p1[i];
        dec2+=(char)p2[i];
            dec3+=(char)p3[i];
            dec4+=(char)p4[i];
      i++;
}
static int ChineseRemainderTheorem(int a, int b, int m1, int m2)
{
        int M, M1, M2, M1_inv, M2_inv;
        int result;
        M = m1 * m2;
        M1 = M / m1;
        M2 = M / m2;
        M1_inv = inverse(m1, M1);
        M2_inv = inverse(m2, M2);
        result = (a * M1 * M1_inv + b * M2 * M2_inv) % M;
        return result;
}
```

```
public static void FileWriting(String filename, String text) {
   PrintWriter outputstream=null;
   try {
      outputstream=new PrintWriter(new FileOutputStream(filename,false));
   }
   catch(FileNotFoundException e) {
      System.out.println("Error in opening file"+filename);
      System.exit(0);
   }
   outputstream.println(text);
   outputstream.close();
}

public static String FileReading(String filename) {
   Scanner inputstream=null;
   try {
      inputstream=new Scanner(new File(filename));
   }
   catch(FileNotFoundException e) {
      System.out.println("The file \""+filename+"\" could not be opened");
      System.exit(0);
```

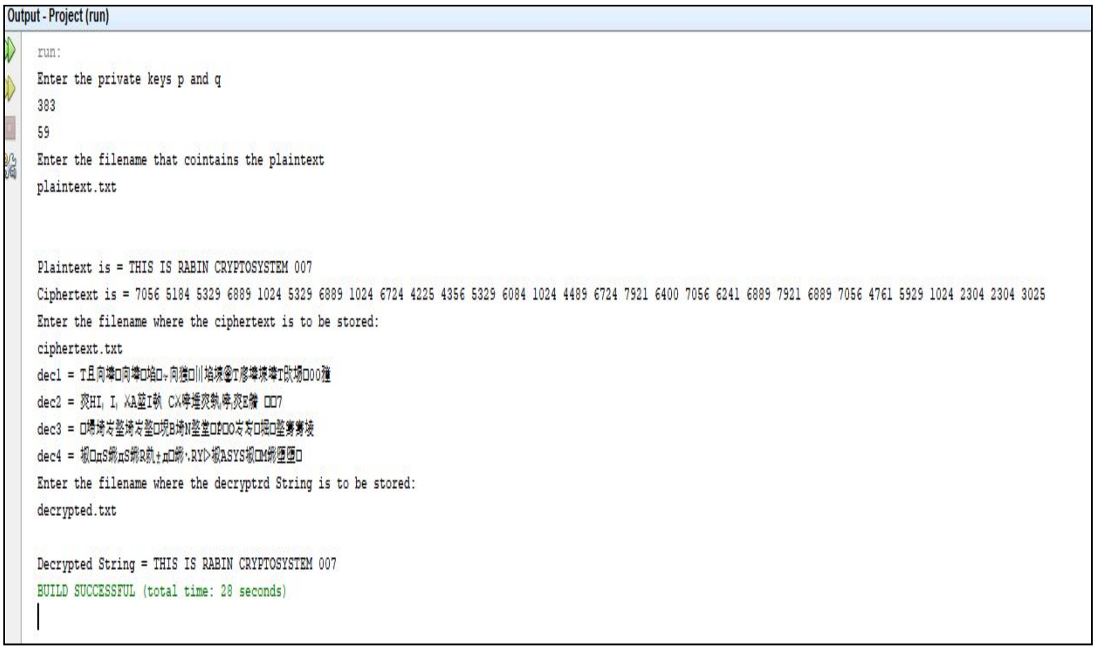
```
    }
    String text="";
    int c=0;
    System.out.println("\n");
    while(inputstream.hasNextLine()) {
        text=inputstream.nextLine();
    }
    inputstream.close();
    return text;
}
public static void main(String args[])
{
            Scanner sc=new Scanner(System.in);
        System.out.println("Enter the private keys p and q");
        p=sc.nextInt();
        q=sc.nextInt();
        n=p*q;
        System.out.println("Enter the filename that cointains the plaintext");
        sc=new Scanner(System.in);
            String filename=sc.nextLine();
        String plaintext=FileReading(filename);
```

```
System.out.println("Plaintext is = "+plaintext);
   for(int i=0;i<plaintext.length();i++) {
   char ch=plaintext.charAt(i);
   int PlainText=(int)ch;
      int Cipher = Encryption(PlainText);
      cip[i]=Cipher;
}
   System.out.print("Ciphertext is = ");
   String ciphertext="";
    for(int i=0;i<plaintext.length();i++) {
   System.out.print(cip[i]+" ");
   ciphertext+=cip[i]+" ";
}
System.out.println();
   System.out.println("Enter the filename where the ciphertext is to be
stored: ");
sc=new Scanner(System.in);
filename=sc.nextLine();
FileWriting(filename,ciphertext);
for(int i=0;i<plaintext.length();i++) {
   Decryption(cip[i]);
```

```
    }
   System.out.println("dec1 = "+dec1);
    System.out.println("dec2 = "+dec2);
    System.out.println("dec3 = "+dec3);
    System.out.println("dec4 = "+dec4);

    String decrypted_str="";

    for(int i=0;i<plaintext.length();i++) {
        if((dec1.charAt(i)>='A'&&dec1.charAt(i)<='Z')||(dec1.charAt(i)=='
')||(dec1.charAt(i)>='0'&&dec1.charAt(i)<='9')) {
            decrypted_str+=dec1.charAt(i);
        }
        if((dec2.charAt(i)>='A'&&dec2.charAt(i)<='Z')||(dec2.charAt(i)=='
')||(dec2.charAt(i)>='0'&&dec2.charAt(i)<='9')) {
            decrypted_str+=dec2.charAt(i);
        }
        if((dec3.charAt(i)>='A'&&dec3.charAt(i)<='Z')||(dec3.charAt(i)=='
')||(dec3.charAt(i)>='0'&&dec3.charAt(i)<='9')) {
            decrypted_str+=dec3.charAt(i);
        }
```

```
            if((dec4.charAt(i)>='A'&&dec4.charAt(i)<='Z')||(dec4.charAt(i)=='
')||(dec4.charAt(i)>='0'&&dec4.charAt(i)<='9')) {
                decrypted_str+=dec4.charAt(i);
            }
        }
        System.out.println("Enter the filename where the decryptrd String is to be
stored: ");
        sc=new Scanner(System.in);
        filename=sc.nextLine();
        FileWriting(filename,decrypted_str);
        System.out.println("\nDecrypted String = "+decrypted_str);
}
}
```

Results

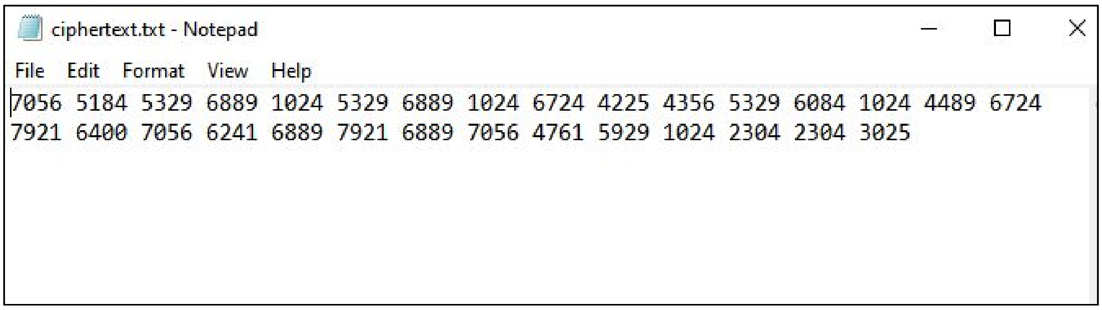


```
Output - Project (run)
run:
Enter the private keys p and q
383
59
Enter the filename that cointains the plaintext
plaintext.txt


Plaintext is = THIS IS RABIN CRYPTOSYSTEM 007
Ciphertext is = 7056 5184 5329 6889 1024 5329 6889 1024 6724 4225 4356 5329 6084 1024 4489 6724 7921 6400 7056 6241 6889 7921 6889 7056 4761 5929 1024 2304 2304 3025
Enter the filename where the ciphertext is to be stored:
ciphertext.txt
dec1 = [illegible]
dec2 = [illegible]
dec3 = [illegible]
dec4 = [illegible]
Enter the filename where the decryptrd String is to be stored:
decrypted.txt

Decrypted String = THIS IS RABIN CRYPTOSYSTEM 007
BUILD SUCCESSFUL (total time: 28 seconds)
```

Figure 2: Output in NetBeans IDE

```
plaintext.txt - Notepad
File Edit Format View Help
THIS IS RABIN CRYPTOSYSTEM 007
```

Figure 3: Reading from text file plaintext.txt



```
ciphertext.txt - Notepad
File Edit Format View Help
7056 5184 5329 6889 1024 5329 6889 1024 6724 4225 4356 5329 6084 1024 4489 6724
7921 6400 7056 6241 6889 7921 6889 7056 4761 5929 1024 2304 2304 3025
```

Figure 4: Writing text file ciphertext.txt

Figure 5: Writing text file in decrypted.txt

References

1. Buchmann, Johannes. Einführung in die Kryptographie. Second Edition. Berlin: Springer, 2001.
2. Menezes, Alfred; van Oorschot, Paul C.; and Vanstone, Scott A. Handbook of Applied Cryptography. CRC Press, October 1996.
3. Rabin, Michael. Digitalized Signatures and Public-Key Functions as Intractable as Factorization (in PDF). MIT Laboratory for Computer Science, January 1979.
4. Scott Lindhurst, An analysis of Shank's algorithm for computing square roots in finite fields. in R Gupta and K S Williams, Proc 5th Conf Can Nr Theo Assoc, 1999, vol 19 CRM Proc & Lec Notes, AMS, Aug 1999.
5. R Kumanduri and C Romero, Number Theory w/ Computer Applications, Alg 9.2.9, Prentice Hall, 1997. A probabilistic for square root of a quadratic residue modulo a prime.
6. Behrouz A. Forouzan. Cryptography & Network Security. McGraw-Hill, Inc. New York: NY, USA, 2008