Performance Analysis of Public key Cryptographic Systems RSA and NTRU

Narasimham Challa [†] ***and Jayaram Pradhan*** [††] ***,***

[†] **Faculty of MVGR College of Engineering, Vizianagaram, A.P., India.**
[††] **Professor of Berhampur University, Berhampur, Orissa, India.**

Summary

In many business sectors secure and efficient data transfer is essential. To ensure the security to the applications of business, the business sectors use Public Key Cryptographic Systems (PKCS). An RSA and NTRU system generally belongs to the category of PKCS. The efficiency of a public key cryptographic system is mainly measured in computational overheads, key size and bandwidth. In particular, the RSA algorithm is used in many applications. Although the security of RSA is beyond doubt, the evolution in computing power has caused a growth in the necessary key length. The fact that most chips on smart cards cannot process keys extending 1024 bit shows that there is a need for alternative. NTRU is such an alternative and it is a collection of mathematical algorithms based on manipulating lists of very small integers and polynomials. This allows NTRU to achieve high speeds with the use of minimal computing power. NTRU is the first secure public key cryptosystem not based on factorization or discrete logarithm problems. This means that given sufficient computational resources and time, an adversary, should not be able to break the key. The performance characteristics of NTRU and RSA are observed by implementing the algorithms for computation and comparing their experimental running times. Encryption and decryption speeds take O (n log (n)) operations. The RSA's complexity is O (n ^{3}) operations and it compares with NTRU's O (n log(n) operations. In this paper, we proposed and presented these two well known Public Key Cryptographic Systems, and were implemented to verify their performance for different text files of variable sizes.

Keywords

RSA, NTRU, Pubic Key, Private Key, Encryption, Decryption, Cipher Text.

1.0 Introduction

RSA [4] is one of the oldest and most widely used public key cryptographic systems. It was the first algorithm known to be suitable for signing as well as encryption [4], and one of the first great advances in public key cryptography. RSA is still widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys. NTRU [5] is latest in the line of PKCS. It is relatively new and was conceived by Jeffrey Hoff stein, Jill Pipher and Joseph. H. Silverman. NTRU uses polynomial algebra combined with clustering principle based on elementary mathematical theory. The security of NTRU comes from the interaction of polynomial mixing system with the independence of reduction modulo two relatively prime numbers. The basic collection of objects used by the NTRU Public Key Cryptosystem in the ring R that consists of all truncated polynomials of degree N-1 having integer coefficients **a** $=a_{0}+a_{1}X+a_{2}X^{2}+a_{3}X^{3}+\ldots+a_{N-2}X^{N-2}+a_{N-1}X^{N-1}$. Polynomials are added in the usual way. They are also multiplied more or less as usual, except that X^{N} is replaced by 1, X^{N+1} is replaced by X, X^{N+2} is replaced by X^{2} and so on. A full implementation of the NTRU Public Key Cryptosystem is specified by a number of parameters. *N* the polynomials in the truncated polynomial ring have degree *N*-1. *Q* large modulus, usually the coefficients of the truncated polynomials will be reduced to mod *q*. *p* small modulus. As the final step in decryption, the coefficients of the message are reduced to mod *p*.

The objective of this paper is to compare the performance of RSA and NTRU, which are readily available for commercial use. We proposed this work to test and verify the performance of these methods. There is no doubt that RSA provides security for large values of prime numbers, where as the NTRU provides the same with smaller values and their polynomials.

In the next section we describe the method in brief to convert the plain message file into binary file. In section three we describe RSA method to convert the plain text of different size text. In section four the NTRU method is described in the similar manner. In section five we compare the result of RSA and NTRU. Our conclusions have been presented in the section six.

2.0 Huffman encoding

David Huffman designed an optimal weighted binary in 1952 in his classic paper [1]. Subsequently

Manuscript received August 5, 2007
Manuscript revised August 20, 2007

that method was used for variable length encoding of the plain text. Huffman method can be implemented using a forest (i.e., collection of trees) each tree of which has its leaves labeled by characters , whose codes we describe to select and whose roots are labeled by the sum of the weights(frequency of characters) of all the leaf labels. Initially, each character is a one-node tree by itself and when the algorithm ends, there will be only one tree with all the characters at its leaves. In this tree, the path from the root to any leaf represents the code for the label of that leaf, according to left=0, right=1.

After the construction of the tree we can determine the code for each letter by starting at the leaf and moving to the parent repeatedly until we reach the root. Obviously we output "0" if the present position is left child of the parent and "1" if the present position is right child. The sequence of bits so outputted and reversed is the code for that character. Here it may be noted that the code is variable length code and optimal number of bits for a fixed text.

In brief frequently occurring symbols of the text message are assigned short code words where as rarely occurring symbols are assigned long code words. So for a given text file message as input we outputted as follows:

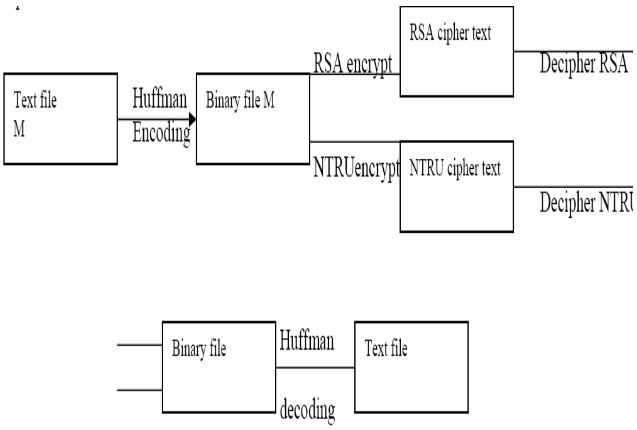

Fig1 Block diagram for overall procedure

3.0 RSA mathematical analysis

The RSA method is mainly based on integer and factoring as a one-way function. If the two large prime numbers p and q are given i.e. F (p, q), computing N is easy; the problem has a finite number of solutions (As shown in Fig 2). This can be treated as polynomial time solutions problem that is P problem [3].

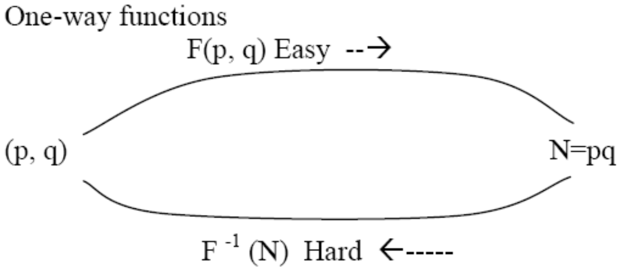

Fig 2 RSA mathematical complexity

The other side If N is given, computing p and q large polynomial is not so easy i.e. F^{-1} (N), the problem can be computed in non-polynomial time solutions, and can be considered as NP problem. The NP problem can be solved in polynomial time by providing trapdoor information (i.e, secret key). The security lies with the multiplicative exponent.

RSA Key generation

RSA public and private key pair can be generated by the following procedure. Choose two random prime numbers p and q such that the bit length of p is approximately equal to the bit length of q.
Compute n such that n = p * q.
Compute $\phi(n)$ such that $\phi(n) = (p-1)* (q-1)$.
Choose a random integer e, e < $\phi(n)$ and gcd (e, $\phi(n)$) = 1 then compute the integer d, such that e*d = 1 mod $\phi(n)$.
(n, e) is the public key, and d is the private key.

RSA Key generation implementation

```
main()
{
int p,q,f,g,t;
clrscr();
printf("enter two large primes");
scanf("%d\n%d",&p,&q);
fp=fopen("key.txt","w");
n=p*q; w=(p-1)*(q-1);
d=q+1; b:d++;
e=mul(d,w);
if(e<=0||e<log(n)/log(2)||e>q)
goto b;
printf("%d %d",e,d);
fprintf(fp,"%d",d);
fclose(fp);
fp=fopen("fs.txt","w");
fprintf(fp,"%d",n);
fclose(fp); }
mul(d,w)
{int x1,x2,x3,y1,y2,y3,q,t1,t2,t3;
x1=1; x2=0; x3=w; y1=0; y2=1;
y3=d;
a: if( y3==0)
{return 0;}
if(y3==1)
return y2;
q=x3/y3; t1=x1-q*y1; t2=x2-q*y2;
t3=x3-q*y3; x1=y1;x2=y2;x3=y3;
y1=t1;y2=t2;y3=t3;goto a;}
```

RSA Encryption

Person A transmits public key n & e to person B and keeps the private key secret [4].

B then wishes to send message m to A.

B first turns m into a number, such that the number <n.

B then computes the *cipher text* $c = \text{number}^{e} \bmod n$.

B then transmits c to A.

RSA Encryption implementation

```
main()
{
FILE *fp,*fhead,*fout;
long int i,j,a,temp=0,f,k,m;
int c; char  st[20];
char st1[20],st3[20],ch;
clock_t start,end,clk_tck;
fp=fopen("abc.txt","r");
fout=fopen("ciph.txt","w");
k=7; i=0; key1();
start=clock();
while(e>0)
{ g[i]=e%2; e=e/2; i++; }
e=i-1; k=7;j=0;
do
{while(j<8)
{ch=fgetc(fp);
if(ch==EOF)
break;
st1[j]=ch;
a=ch-48;
temp=temp+a*pow(2,k);
j++;k--;}
st1[j]='\0';
if(j<8)
{temp=0;i=0;
p=strlen(st1);p--;
while(p>0)
{a=st1[i]-48;
temp=temp+a*pow(2,p);
p--;i++;}
c=temp;}
c=temp;f=c;
c=pow1(e,n,f);
f=c;if(f==0)
strcpy(st,"0000000000000000");
else
{j=0;
while(f>0)
{p=f%2;
st[j]=p+48;
f=f/2;j++;}
st[j]='\0';}
f=strlen(st);
j=0;
while(f>0)
{f--;st3[j]=st[f];
j++;}
st3[j]='\0';p=strlen(st3);
strcpy(st1,"");
while(p<16)
{strcat(st1,"0");
p++;}
strcat(st1,st3);
strcpy(st3,st1);
fprintf(fout,"%s",st3);
k=7;temp=0;j=0;
} while(ch!=EOF);
p=strlen(st1) ;end=clock();
printf("%lftime for encryption\n",(double)(end-start)/18.2);}
pow1(e,n,f)
{long int d=1;
int a,i;a=f;
for(i=e;i>=0;i--)
{d=(d*d)%n;
if(g[i]==1)
d=(d * a)%n;}
if(a>n)
d=d+n;
return d; }
```

RSA Decryption

Person A can recover message m from c by using the private key d [4].

Decrypted value = c^{d} mod n.

Given decrypted value, A can recover the message m.

RSA Decryption implementation

```
main(){
FILE *fp,*fout;
char st[20],st1[20],ch;
long int i,j,temp,a,k,p,m,h,w;
```

```
int c;int f;clock_t start,end;
w=1;clrscr();key1();
start=clock();i=0;
while(d>0)
{g[i]=d%2;d=d/2;i++;}
o=i-1;fp=fopen("ciph.txt","r");
fout=fopen("deciph.txt","w");i=0 ;
while((ch=fgetc(fp))!=EOF)
{i++;}
i=i/16;w=i;rewind(fp);m=0;do
{j=0;temp=0; k=15;
while(j<16)
{ch=feetc(fp);if(ch==EOF)
{break;}
a=ch-48;
temp=temp+a*pow(2,k);
k-- ; j++;}
if(ch==EOF)
break;c=temp;f=c;
c=pow1(o,n,f);f=c;
if(f==0)
strcpy(st1,"00000000");
else
{j=0;while(f>0)
{p=f%2+48;st1[j]=p;f=f/2;j++;}
st1[j]='\0';p=strlen(st1);j=0;f=p;
while(j<f)
{p--;st[j]=st1[p];j++;}
st[j]='\0';
strcpy(st1,st);}
if(m<(w-1))
{f=strlen(st1);
strcpy(st,"");
if(f!=8)
{while(f<8)
{strcat(st,"0");f++;}}
strcat(st,st1);
strcpy(st1,st);}
fprintf(fout,"%s",st1);
m++;}while(ch!=EOF);
end =clock();
printf("%lfdecryptionime\n",(double)(end-start)/18.2);
fclose(fp);fclose(fout);}
pow1(o,n,f)
{long int b=1,a,i;a=f;
for(i=o;i>=0;i--)
{b=(b * b) %n;if(g[i]==1)b=(b*a)%n;}
if(a>n)
{
b=b+n; return b; }
```

3.1 RSA Performance

Experiments were conducted on different text files. The encryption method is implemented with the key size of 22 bits and the decryption method is implemented with the key size of 10 bits. The computing times are presented in the tabular form [11].

Text Size	Encryption	Decryption
128 bits	0.0549	0.0549
256 bits	0.1098	0.1098
512 bits	0.2197	0.1648
1K	0.3846	0.3296
2K	0.7142	0.6593
5K	1.7032	1.7032
10K	3.4020	3.4020

Table 1: RSA encryption and decryption execution timings

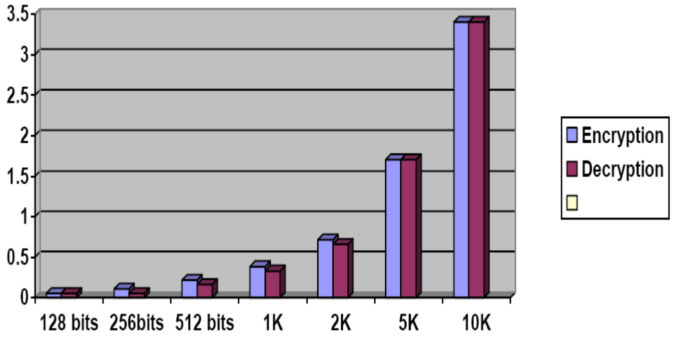

Fig 3 RSA performance

The detailed code of RSA method in order to output the above table taking for different text files is shown in above.

4.0 NTRU (Nth Degree Truncated Polynomial Ring Units)

NTRU mathematical analysis

NTRU theory [5] is mainly based on polynomials and not based on factoring as a one-way function. The one-way function [3] is described in Fig4.

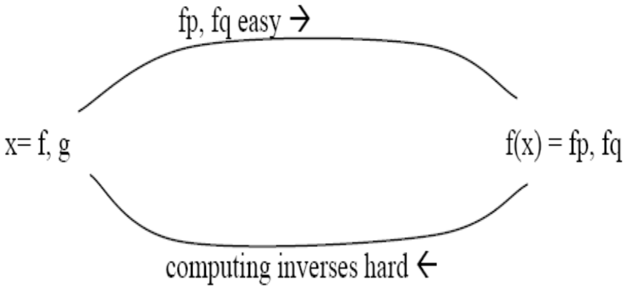

Fig4 NTRU mathematical complexity

Let f and g are two potential polynomials in the ring R^N, computing fp & fq the inverses with respect to modulo p and q respectively is easy. In other words, If x given, computing f(x) is easy i.e., P problem (As shown in Fig 4). But the reverse is hard i.e., NP problem. It is based on the shortest non-zero lattice vector problem. The NP problem can make easy with trap door function. The algorithms key generation, evaluation and inversion can be shown as

Key generation		(f, t)	G (random)	
Evaluation			c = f(x)	E (f, x)
Inversion	←		x= f^{-1} (c)	I (f, c, t)
	←			

Lattices v1, v2,……..,vn linearly independent in Rn
B = (V1, V2,………..,Vn) invertible matrix
L (B) = { Bx : x is an integer vector}

Closest Vector Problem
Given B and C belongs to R^N, find the closest path to c in L (B) and NP – hard.
Smallest Basis Problem
Given B, find B'of minimum size, so that L (B) = (B')
NP – time algorithm is known for any of them.

Key Generation

Let a person B wants to create a public/private key pair for the NTRU Public Key Cryptosystem. B first randomly chooses two small potential polynomials **f** and **g** in the ring of truncated polynomials **R**. A small polynomial is small relative to a random polynomial mod *q*. In a random polynomial, the coefficients will in general be randomly distributed mod *q*; in a small polynomial, the coefficients are much smaller than *q*. B must keep the values of the polynomials **f** and **g** private, since anyone who knows the value of either one of them will be able to decrypt messages sent to B. B's next step is to compute the inverse of **f** modulo *q* and the inverse of **f** modulo *p*. Thus B computes polynomials **f***q* and **f***p* with the property that **f***__f__*q* = 1 (modulo *q*) and **f***__f__*p* = 1 (modulo *p*). If by some chance these inverses do not exist, B will need to go back and choose another **f**. For information about computing inverses in the ring of truncated polynomials, Now B computes the product **h** = *p* **f***q****g** (modulo *q*).B's private key is the pair of polynomials **f** and **f***p*. B's public key is the polynomial **h**.
The CreateKey function
Creating the inverse polynomial of the secret key modulo q, Fq.
Creating the inverse polynomial of the secret key modulo p, Fp .
Creating the Public Key, h = p * ((Fq)*g) mod q .
Also the algorithm assumes q = 2^w so the reduction will be performed by extracting the lower w bits.

NTRU key generation implementation

```
CreateKey(N; q; p; f; g; h; Fp; Fq)
Require: p, q, N and random polynomials, f and g.
Inverse_Poly_ Fq(f; Fq;N; q)
Inverse_Poly _Fp(f; Fp;N; p)
StarMultiply(Fq; g; h;N; q)
for i = 0 to N - 1 do
if h[i] < 0 then
h[i] = h[i] + q  Make all coefficients in h   positive.
end if
h[i] = h[i]  * p mod q
end for
```

CreateKey returns the Public Key, h, and the inverse polynomial, Fp, through the argument list.

StarMultiply

This function outlined in algorithm performs the polynomial multiplication of a*b mod x^N - 1. As a note, the M in Step 9 is either p or q depending upon which one is passed into the function. In contrast to the guideline, Algorithm only executes Step 9 if the current coefficients of a[i] and b[j] are both non-zero. This, therefore, eliminates approximately a third of the operations, which are unnecessary. Also, for the case M = q, Algorithm assumes q = 2^w so the reduction is performed by extracting the lower w bits.

```
StarMultiply(a; b; c;N;M)
Require: N, the coefficient modulus, M, and the two polynomials to be multiplied a & b.
for k = N - 1 down to 0 do
c[k] = 0
j = k + 1
for i = N- 1 down to 0 do
if j = N then
j = 0
end if
if a[i] <> 0 and b[j]  <> 0 then
c[k] = c[k] + (a[i]  * b[j]) mod M
end if
j = j + 1
end for
end for
```

StarMultiply returns the product polynomial c through the argument list.

Random Polynomial

The RandPoly function, shown in algorithm, generates a random polynomial, r, whose coefficients are in the subset {-1, 0, 1}. The user specifies the number of ones (NumOnes) and the number of negative ones (NumNegOnes) that will make up the random polynomial, r. Basically, the algorithm works by randomly selecting a location (position) between 0 and N in the random polynomial vector, r. For each selected location, if the value is zero the algorithm replaces the zero with a 1 or a -1 until all the specified number of ones and negative ones have been entered into the vector.

RandPoly (r;N, NumOnes, NumNegOnes)

Require: N, NumOnes, NumNegOnes, and polynomial vector to be made random, r.

```
r = 0
while NumOnes <> 0 or NumNegOnes <> 0 do
  position = rand() mod N
  if r[position] = 0 then
    if NumOnes > 0 then
      r[position] = 1
      NumOnes = NumOnes - 1
    else if NumNegOnes > 0 then
      r[position] = -1
      NumNegOnes = NumNegOnes - 1
    end if
  end if
end while
```

RandPoly returns the newly generated random polynomial, r through the argument list.

Inverse_Poly_ F_q

The Inverse_Poly _Fq function in algorithm is responsible for generating the inverse polynomial of the secret key, f, modulo q. The first 40 lines of algorithm compute the inverse of the secret key modulo 2. Then, the last couple of lines in the algorithm finds the inverse polynomial modulo a power of 2, which is q. Algorithm is based on the pseudo-code for Inversion in (Z/2Z)[X]/(X^N - 1)" and Inversion in (Z/p^{rZ})[X]/(X^{N} - ^{1})".

Inverse_Poly_F $_q$(a; F_q;N; q)

Require:the polynomial to invert a(x), N,and q.

```
k = 0
b = 1
c = 0
f = a
g = 0  Steps 5-7 set g(x) = $x^N$ -1
g[0] = -1
g[N] = 1
loop
  while f[0] = 0 do
    for i = 1 to N do
      f[i-1] = f[i] {f(x) = f(x)/x}
      c[N + 1- i] = c[N - i] {c(x) = c(x) *x
    end for
    f[N] = 0
    c[0] = 0
    k = k + 1
  end while
  if deg(f) = 0 then
    goto Step 32
  end if
  if deg(f) < deg(g) then
    temp = f  {Exchange f and g}
    f = g
    g = temp
    temp = b {Exchange b and c}
    b = c
    c = temp
  end if
  f = f * g
  b = b * c
end loop
j = 0
k = k mod N
for i = N - 1 down to 0 do
  j = i - k
  if j < 0 then
    j = j + N
  end if
  $F_q$[j] = b[i]
end for
v = 2
while v < q do
  v = v * 2
  StarMultiply(a; $F_q$; temp;N; v)
  temp = 2 - temp mod v
  StarMultiply($F_q$ ; temp; $F_q$;N; v)
end while
for i = N - 1 down to 0 do
  if $F_q$[i] < 0 then
    $F_q$[i] = $F_q$[i] + q
  end if
end for
```

Inverse-Poly F_q returns the inverse polynomial, F_q, through the argument list.

Inverse_Poly_ F_p

The Inverse_Poly_ F_p function in Algorithm is responsible for generating the inverse polynomial of the secret key, f, modulo p. Algorithm is based off the pseudo-code for Inversion in (Z/pZ)[X]/(X^N - 1)”.

Inverse _Poly_ F $_p$(a; F_p;N; p)

Require: the polynomial to invert a(x), N, and p.

```
k = 0
b = 1
c = 0
f = a
g = 0   Steps 5-7 set g(x) = $x^N$ - 1
g[0] = -1
g[N] = 1
loop
```

```
 while f[0] = 0 do
 for i = 1 to N do
 f[i - 1] = f[i] {f(x) = f(x)/x}
 c[N + 1 - i] = c[N - i] {c(x) = c(x) * x}
 end for
 f[N] = 0
 c[0] = 0
 k = k + 1
 end while
 if deg(f) = 0 then
 goto  Step 33
 end if
 if deg(f) < deg(g) then
 temp = f {Exchange f and g}
 f = g
 g = temp
 temp = b{Exchange b and c}
 b = c
 c = temp
 end if
 u = f[0] * $g[0]^{-1}$ mod p
 f = f - u * g mod p
 b = b - u * c mod p
 end loop
 j = 0
 k = k mod N
 for i = N - 1 down to 0 do
 b[i] = f[0] $^{-1}$ * b[i] mod p
 j = i - k
 if j < 0 then
 j = j + N
 end if
 F $_q$[j] = b[i]
 end for
```

Inverse Poly Fp returns the inverse polynomial, Fp, through the argument.

Polynomial generation

```
randpoly()
{int r[11]={0},n,numones,numnegones;
int numzeros,position,i;
clrscr();printf("enter number");
scanf("%d",&n);
printf("enter number of ones");
scanf("%d",&numones);
printf("enter number of negative ones");
scanf("%d",&numnegones);
numzeros=n-(numones+numnegones);
while((numones!=0)||(numnegones!=0)||(numzeros!=0))
{position=rand()%n;
if(r[position]==0)
{if(numones>0)
{r[position]=1; numones--;}
else if(numnegones>0)
{r[position]=-1; numnegones--;}
else if(numzeros>0)
{r[position]=0; numzeros--; }}}
for(i=0;i<n;i++)
printf("%d\n",r[i]);}
```

NTRU Encryption

Let a person A wants to send a message to person B using B's public key h [5] 6]. Person A first puts the message in the form of a polynomial m whose coefficients are chosen modulo p, say between -p/2 and p/2 (in other words, m is a small polynomial mod q). Next the person A randomly chooses another small polynomial r. This is the blinding value, which is used to obscure the message. Then A uses the message m, and randomly chosen polynomial r, and B's public key h to compute the polynomial e = r*h + m (modulo q). The polynomial e is the encrypted message which A sends to B [7][8].

NTRU Encryption implementation

```
main()
{
FILE *fp,*fout;
int n=11,m[11]={-1,0,0,1,-1,0,0,0,-
1,1,1},i=0,e[11],q=32,y[12];
int k,h,v,g,w;
char st[8],st1[8],ch;
clock_t start,end;
fout=fopen("abc.txt","r");
fp=fopen("ntciph.txt","w");
key(); randpoly();
start=clock();
starmultiply(r,y,n,q);
for(i=0;i<11;i++)
y[i]=s[i];
do
{ i=0;
  do
 {ch=fgetc(fout);
 if(ch==EOF)
 break;     m[i]=ch-48;
i++;   }while(i<11);
 w=i;
 while(i<11)
{
m[i]=0;   i++;}
for(i=0;i<11;i++)
s[i]=y[i];
for(i=0;i<=w;i++)
{ s[i]=(s[i]+m[i])%q;
if(s[i]<0)
s[i]=s[i]+q;}
for(i=0;i<w;i++)
{k=0;
while(s[i]>0)
{st[k]=(s[i]%2)+48;
s[i]=s[i]/2;   k++;}
st[k]='\0';h=strlen(st);h--; g=0;
while(h>=0)
{  st1[g]=st[h];  g++;  h--;}
st1[g]='\0';h=strlen(st1);strcpy(st,"");
while(h<8)
{  strcat(st,"0");  h++;}
strcat(st,st1);
fprintf(fp,"%s",st);}
}while(ch!=EOF);
end=clock();
printf("%lf encryption time is",(double)(end-start)/18.2);}
```

NTRU Decryption

The person B has received A's encrypted message e and wants to decrypt [5] [6] it. The person B begins by using the private polynomial f to compute the polynomial a = f*e (mod q). Since B is computing a mod q, B can choose the coefficients of a to lie between –q/2 and q/2. In general, B will choose the coefficients of a to lie in an interval of length q. The specific interval depends on the form of the small polynomials. It is very important that B does this before performing the next step. B next computes the polynomial b = a (mod p), i.e., reduces each of the coefficients of a mod p. Finally B uses the other private polynomial fp to compute c = fp*b (modulo p). The polynomial c will be A's original message m [7] [8].

NTRU Decryption implementation

```
main()
{
FILE *fout,*fh;
int f[11]={-1,1,1,0,-1,0,1,0,0,1,-
1},e[11]={0,0,0,0,0,0,0,0,0,0,0},i=0,n=11,q=32,a[11],p=3
;
int fp[11]={1,2,0,2,2,1,0,2,1,2,0},k,j,z,y,w;
char ch,st1[20];
clock_t start,end;clrscr();
start=clock();
fout=fopen("ntciph.txt","r");
fh=fopen("ntdeciph.txt","w");
do
{for(i=0;i<=n-1;i++)
e[i]=0;i=0;
do
{k=0;
do
{ch=fgetc(fout);
if(ch==EOF)
break;st1[k]=ch;k++;
}while(k<8);
y=7;j=0;
while(st1[j]!='\0')
{z=st1[j]-48;
e[i]=e[i]+z*pow(2,y);
j++;y--;}
i++;
if(ch==EOF)
break;}while(i<11);
if(i<11)
w=i-2;
else
w=i-1;
starmultiply(f,e,n,q);
starmultiply(fp,d,n,p);
for(i=0;i<=w;i++)
fprintf(fh,"%d",d[i]);
}while(ch!=EOF);
end=clock();
printf("%lf decryption time is",(double)(end-
start)/18.2);}
starmultiply(int a[11],intb[11],int n,int m)
{int c[11],k,j,i;
for(k=n-1;k>=0;k--)
{
c[k]=0; j=k+1;
for(i=n-1;i>=0;i--)
{if(j==n)
j=0;
if(a[i]!=0&&b[j]!=0)
c[k]=(c[k]+a[i]*b[j])%m;
j++;}}
for(i=0;i<=n-1;i++)
{if(c[i]<-15)
d[i]=c[i]+m;
else if(c[i]>16)
d[i]=c[i]-m;
else
d[i]=c[i];}
for(i=0;i<=n-1;i++)
{d[i]=d[i]%3;
if(d[i]<-1)
d[i]=d[i]+3;
if(d[i]>1)
d[i]=d[i]-3;}}
```

4.1 NTRU Performance

Experiments were conducted on different text files. The encryption method is implemented with the key size of 51 bits and the decryption method is implemented with the key size of 20bits[11].

Text size	Encryption	Decryption
128 bits	0.0000001	0.0000001
256 bits	0.0000001	0.05490
512 bits	0.05494	0.05490
1K	0.10989	0.05490
2K	0.27472	0.05490
5K	0.65934	0.16484
10K	1.31868	0.36100

Table 2: NTRU encryption and decryption execution timings

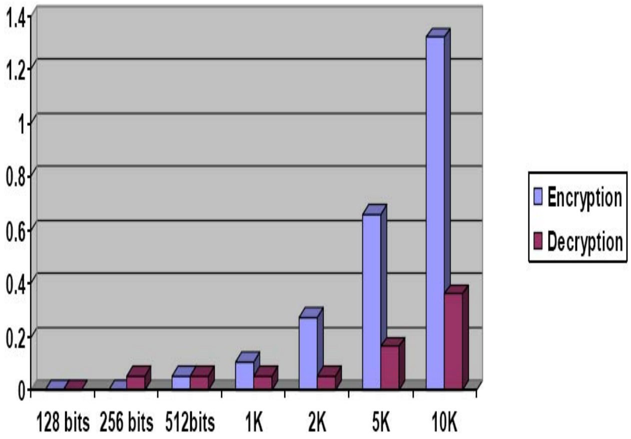

Fig 5 NTRU performance

The detailed code of NTRU method in order to output the above table taking different text files is shown in above.

Huffman Decoding

The resulting code string can be uniquely decoded [1] to get the original output of the run length encoder. Using Huffman decoding procedure, we have decoded the resulting optimized code into original text files.

5.0 Comparison of RSA and NTRU

Theoretically both RSA and NTRU are same in terms of P and NP problems. Particularly RSA is slow and NTRU is fast as per their claim. By the change of data structure RSA will be faster with same basic theory. RSA is based on the exponent multiplication and NTRU is based on exponent items of addition and subtraction.

Encryption Analysis of RSA and NTRU

Encryption performance characteristic of RSA and NTRU methods are shown in the figure [6]. Though the key size is more for NTRU, the NTRU provides better performance over RSA.

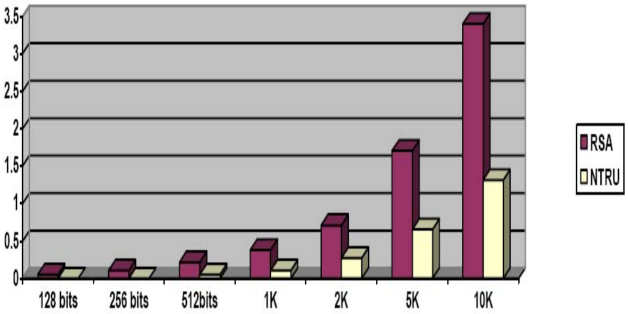

Fig 6 RSA and NTRU encryption performance

Decryption Analysis of RSA and NTRU

Decryption performance characteristic of RSA and NTRU methods are shown in Fig [7]. In this method also like encryption's performance the NTRU is giving better results over RSA.

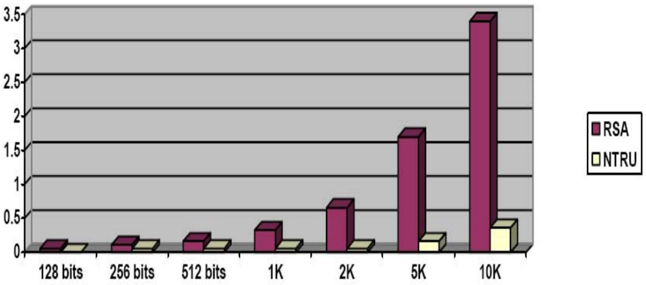

Fig 7 RSA and NTRU decryption performance

The results obtained for encryption and decryption of RSA and NTRU were given in seconds. As per the observations, the computational running times are efficient in NTRU public key cryptosystems (even though the NTRU key size is more) as compare to RSA. The decryption process can be optimized [6] by representing the inverse polynomial in the form f=1+p*f1, Where the polynomial f1 will be in the in ring R^N. Computing the inverse can be totally trivial. We have taken p as a number. If we consider the polynomials of the form p = X+2 then it solves all kinds of problems cleanly and elegantly [6]. There by the basic restriction of p be a number can be overwhelmed.

Generally, no other PKCS allows this kind of transformation.

6.0 Conclusions

The two-implemented systems RSA and NTRU are suitable for applications where it requires security based on the environment. Because of the RSA's time complexity O (n ^{3}) and that of NTRU's O (n log (n)), NTRU confirms its cryptography and delivers encryption, decryption and authentication at speeds of multiple times faster than RSA. NTRU is ideally suited for applications where high performance, high security and low power consumption are required. NTRU has its unprecedented performance advantages open up new options for security.

References

[1] A. Huffman "A method for the construction of minimum redundancy codes Proc. IRE, vol. 40, pp. 1098–1101, Sept. 1952.

[2] Latha Pillai, "Huffman Coding" EXILINX, Virtex Series, XAPP616 (v1.0) Apr 22, 2003.

[3] Whitefield Diffie, Martin E Hellman "New directions in cryptography" IEEE Information theory, June 23-25, 1975 and IEEE International Symposium on Information theory, Sweden, June21-24, 1976.

[4] R.L.Rivest, A.Shamir, L.Adleman "A method for obtaining digital signatures and Public-Key Cryptosystems", Communications of the ACM 21 (1978),120-126.

[5]Joffrey Hoffstein , Jill Pipher , Joseph H Silverman "NTRU-A Ring based public key cryptosystem" Lecture notes in Computer Science, Springer-Verlag, Berlin 1433(1998),267-288.

[6] Joffrey Hoffstein, Joseph H Silverman "Optimizations for NTRU" Proceedings of conference on Public key Cryptography and Computational number theory, Warsaw, De Gruyter ,2000 (Sep 11-15), 77-88.

[7] Collen Marie O'Rourke "Efficient NTRU Implementation" A thesis For Master of Science at Worcester Polytechnic Institute, Apr 2002.

[8] Karthik Thiagarajan "NTRU – A Public key ring based algorithm" A thesis for Master of Science, University of Texas, Dallas, 2003.

[9] NTRU Cryptosystem, Technical Reports 2002 available at http://www.ntru.com.

[10] J. Hoffstein, J. Pipher, J. H. Silverman "NTRU: A new high speed public key cryptography system in Algorithmic number theory" (ANTS III), Portland , OR, June 1998, Lecture Notes in Computer Science 1423 (J. P. Buhler, ed.) Springer-Verlag , Berlin 1998, 267-288.

[11] C. Narasimham, Jayaram Pradhan "Performance analysis: RSA and Truncated polynomials" Proceedings of the Internati€onal Conference CISTM'07, PP40, July 2007.

Mr. Narasimham Challa received his Master of Computer Applications Degree from Andhra University College of Engineering, India in the year 1998. He received his M. Tech Degree in Information Technology in 2003 from Punjabi University, Patiala, India. He has an experience of 14 years in teaching and about 18 months in software industry. Presently, he is working as Associate Professor in the Dept. of Computer Science and Engineering in MVGR College of Engineering, Vizianagaram, India. He has published and presented six papers in various national/international conferences.

J. Pradhan has joined in the department of Computer Science, Bhopal University in the year 1985 after completing M.Phil computer science from J.N University New-Delhi. He Joined in the Department of Computer Science Engineering & Application of REC Rourkela in the year 1986 where he completed his doctorial dissertation. In the year 1993 he joined in the Department of Computer Science Berhampur University as founder head of the department and working as Professor now. During last more than two-decade experience he has published many papers in national, international journal and conferences and offered different courses to DCA, MCA, BE, M.Tech students. His present interests are data structure, Algorithm analysis & design, Cryptography.