Proceeding of the 1st ICEENG conference, 24-26 March, 1998.

CE-1 313

MILITARY TECHNICAL COLLEGE
CAIRO-EGYPT

ICEENG 98

FIRST INTERNATIONAL CONF. ON
ELECTRICAL ENGINEERING

APPLICATION OF DIOPHANTINE EQUATIONS IN A CIPHER SYSTEM

M. S. IBRAHIM*, F. S. ABD RABOU**, H. Y. ZORKTA

ABSTRACT

In this paper an algorithm for data security within computer networks is introduced. The algorithm is based on a public key cipher system and Diophantine equations. In order to reduce the possibility of message length prediction, the algorithm is improved by using dummy bits technique. Moreover, some mathematical rules are applied to limit the length of the ciphering key. For the problem of truncation in floating point type representation of numbers, a proposed solution for a very long integer type representation is introduced.

Key words

Data Security, Cipher system, Public key, Diophantine equations .

1. INTRODUCTION

Public key cryptographic systems allow two users to communicate securely over an insecure channel without any prearrangement. Cryptographic systems which allow this type of communication are asymmetric in the sense that the sender and the receiver have different keys, one of which is computationally infeasible to derive from the other. These systems separate enciphering and deciphering capabilities and privacy is achieved without keeping the enciphering key secret, because it has no longer used for deciphering. Hence the enciphering key is published in addition to the enciphering and deciphering algorithms without compromising the security of the system, [1].

* Ass. Prof., Computer Dep., ** Chairman of Computer Dep., *** Ph.D. Candidate, Computer Dep. Military Technical College, Cairo, Egypt.

A number of public key cryptosystems have been proposed in [2,3,4]. These systems include: RSA; Merkle-Hellman Knapsack, McEliece, EIGamal, Chor-Rivest, and Elliptic Curve. These systems can be classified into two categories. One is based on hard number theoretic problems such as factoring, taking discrete logarithms, etc., while the other is related to NP-complete problems such as 0/1 knapsack and Diophantine equation problem [5]. Thus, the introduced cipher scheme is related to NP-complete problems, which means that, no known polynomial-time algorithm can solve these problems [2].

The realized algorithm achieves the following features:

- It is a public key cipher system, so it is suitable for networks environment.
- Keys can be easily generated.
- Encryption and decryption operations are relatively simple. In brief, to encrypt a message the sender is required to conduct a vector product of the message being sent and the enciphering key. On the other hand, the receiver can easily decrypt it by conducting several multiplication operations and modulus operations.

From the viewpoint of computation time, this algorithm is efficient because, it requires n multiplication operations and n-1 addition operations for encryption, and n multiplication operations and n modulus operation for decryption.

While implementing the algorithm, it was found that, it will be more interesting to leave the length of the message up to the user and don't be arrested to the condition [5], $m = n \times b$, where, m is the length of the message, n is the number of the pairs used to generate the key, and b is the number of bits in each submessage. Thus this condition of the length of the binary message should be n*b bits, which could be considered as a disadvantage of this algorithm. Therefore we have tried to release the algorithm from this disadvantage by using the dummy bits method. Because of the limitation of the computer representation of numbers, it was important to put down some conditions to guarantee that the computations are within the allowed range. Finally, this paper is organized as follows: The underlying mathematics of the used algorithm is presented in Section II. The dummy bits method is described in Section III. Section IV shows some mathematical rules used to implement this algorithm. Finally, conclusions are presented in Section V.

II. THE UNDERLYING MATHEMATICS OF THE USED ALGORITHM

This section describes the mathematics on which the cryptosystem is based. Let w be some positive integer and the domain D be a set of positive integers in the range[0,w], and $w=2^b-1$, where b is some positive integer. So w will be the maximum decimal number which represents the binary message with length b bits.

Assume that a sending message M with length n*b bits is broken up into n pieces of submessages, namely $m_1,m_2,...,m_n$, each of them with length b . bits, and each one can be represented by a decimal number m_i in the domain D.

Suppose that n pairs of integers (q_1,k_1), (q_2,k_2), ...,(q_n,k_n) are chosen according to the following conditions:

1) q_i's are pairwise relative primes; i.e., $\gcd(q_i,q_j)=1$, for $i\neq j$.

2) $k_i > w$ for i=1,2,...,n.

3) $q_i > k_i w$ ($q_i \bmod k_i$), and $q_i \bmod k_i \neq 0$, for i=1,2,..,n.

These n integer pairs (q_i,k_i)'s will be kept secret and used to decrypt messages. For convenience, the above three conditions are named as the DK-Conditions and they will be used as deciphering keys,[5]. Note that for the generation of pairwise relatively primes, one can consult, [6]. Furthermore, the following numbers are computed.

First, compute $R_i=q_i \bmod k_i$ and compute P_i's such that the following two conditions are satisfied:

1) $P_i \bmod q_i = R_i$, and

2) $P_j \bmod q_i = 0$ if $i \neq j$.

Since q_i's are pairwise relatively primes, one solution for P_i's satisfying these two conditions is that $P_i=Q_i b_i$ with $Q_i = \prod_{j\neq i} q_j$, and b_i is chosen such that $Q_i b_i \bmod q_i = R_i$.

Since Q_i and q_i are relatively prime, P_i's can be obtained by using the extended Euclid's algorithm, [2].

Secondly, compute $N_i=\lceil q_i /(k_i R_i)\rceil$, for i=1,2,...,n.

Finally compute $S_i = P_i N_i \bmod Q$, where

$$Q = \prod q_i \text{, for } i=1,2,...,n. \qquad (1)$$

That is, we have a vector $S =(s_1,s_2,...,s_n)$ with each component computed as above. After this, S can be used as the deciphering key for encryption messages. By conducting a vector product between $M =(m_1 ,m_2 ,...,m_n)$ and $S =(s_1,s_2,...,s_n)$; i.e.

$$C= E(S,M)= S*M = \sum m_i s_i \qquad (2)$$

a message M is transformed to its ciphertext C, where * denotes the vector product operation. Conversely, the i^{th} component m_i in M can be revealed by the following operation:

$$m_i=D((q_i,k_i),C)=\lfloor k_i C/q_i \rfloor \bmod k_i, \text{ for } i=1,2,...,n. \qquad (3)$$

For more details about the proof, one can consult [5].

The system constitute mainly three algorithms which described below.

Algorithm 1- Key Generating for Each User U:

step 1. Pick n pairs of positive integers (q_1,k_1), (q_2,k_2),...,and (q_n,k_n) such that the DK-conditions are satisfied.

step 2. Compute $R_i=q_i \bmod k_i$ for i=1,2,...,n.
Compute $Q_i = \prod_{j\neq i} q_i$
and $N_i=\lceil q_i /(k_i R_i)\rceil$, for i=1,2,...,n, and
compute $Q = \prod q_i$, for i=1,2,...,n.

step 3. Compute b_i's such that $Q_i b_i \bmod q_i = R_i$, for i=1,2,...,n.

step 4. Compute $P_i = Q_i b_i$ and $S_i = P_i N_i \bmod Q$, for i=1,2,...,n.

step 5. Publish the encryption key $Pk_u=(s_1,s_2,...,s_n)$ for user U.

step 6. Keep the private decryption key $Pr_u=((q_1,k_1),(q_2,k_2),...,(q_n,k_n))$ in secret.

step 7. Keep P_i, Q_i, b_i, N_i, and Q in secret or erase them.

Algorithm 2- Encryption Procedure for Sender A:

step 1. Encrypt $M = (m_1 ,m_2 ,...,m_n)$ by $C = E(S,M) = S * M$.

step 2. Send out the integer C as the ciphertext of message M.

step 3. Exit.

Algorithm 3- Decryption Procedure for Receiver B:

step 1. Compute the i^{th} component m_i of message M by computing
$m_i = D((q_i,k_i),C)=\lfloor k_iC/q_i \rfloor \bmod k_i$, for i=1,2,...,n.
step 2. Exit.

III. THE DUMMY BITS METHOD

The idea behind using the dummy bits method was the need to break the constraint of the length of the message which used in [5]. It was necessary to use a message of length equal to a multiple of (n*b). For example, assume n=3 and b=2 then the used message must be of length equal to 6*k, where k is a positive integer, representing the number of times the encryption process will last.

If the message was of length m=n*b+I, I=1,2,...,(n*b)-1, there will be I bits from the message will not be used within the encryption procedure. Fig.1, illustrates clearly this problem.

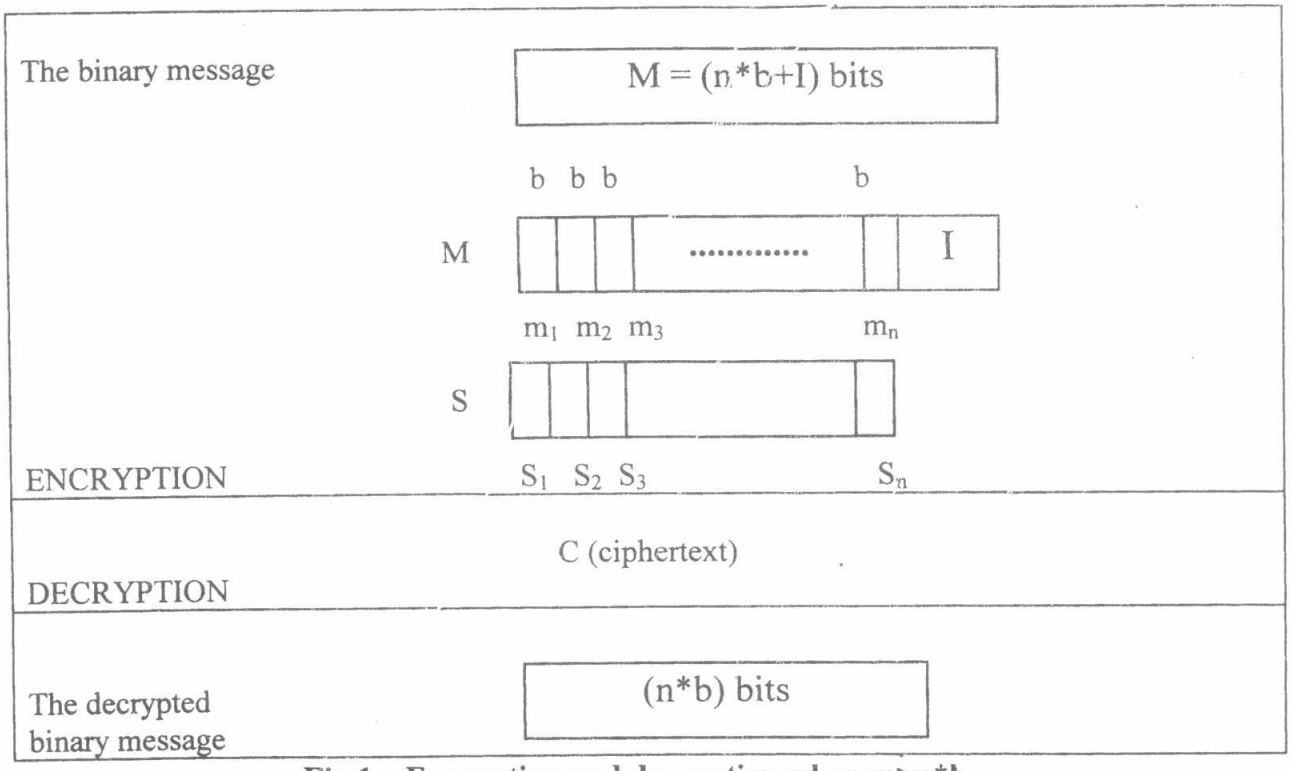

Fig.1. Encryption and decryption when m>n*b.

Assume that n=3, b=2, q_1=37, q_2=38, q_3=39, k_1=4, k_2=4, and k_3=4, then R_1=1, R_2=2, R_3=3 and Q_1=1482, Q_2=1443, Q_3=1406, and Q=54834, N_1=10, N_2=5, N_3=4, and b_1=19, b_2=36, b_3=21, and P_1=28158, P_2=51948, P_3=29526 and finally, S_1=7410, S_2=40404, S_3=8436. Now if the binary message was M=10110101, then C=144468 and it will be seen that only the first 6 bits (101101) can be recovered in the decryption process.

Dummy bits method depends on testing the length of the message. First of all, the length must be greater than n*b, otherwise, it will be meaningless. After the test, if the length was of multiple of n*b then there is no problem, else a number of zeros must be added to the end of the binary message to satisfy the condition. These dummy bits will not affect the work of the encryption procedure since, it will add zeros to the cipher text C. On the other hand, the decryption procedure will have again the same binary message and these dummy bits will be eliminated through the conversion process of the resulted binary value to the plaintext or the text_message. Fig.2, shows the method and the effect of its use.

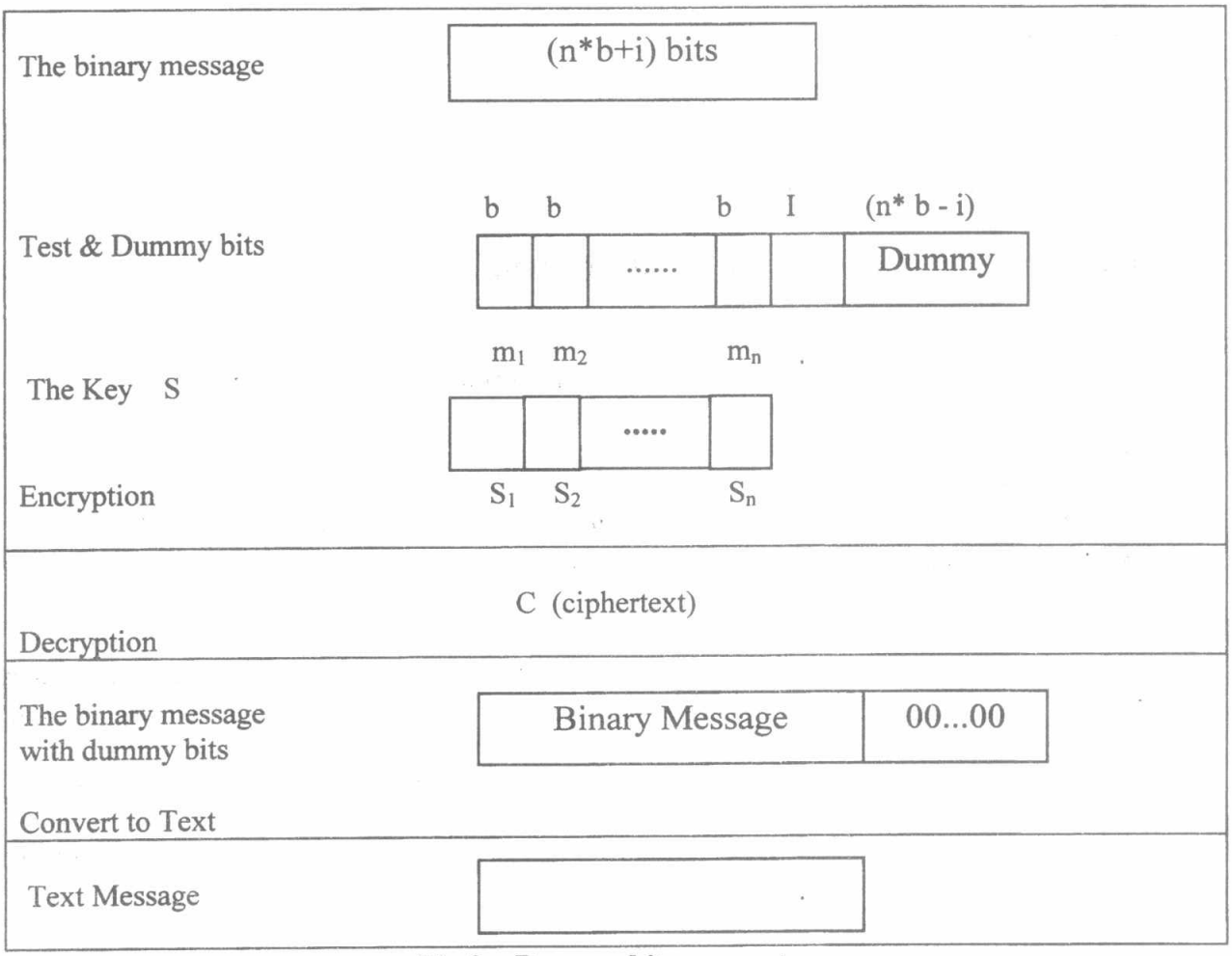

Fig.2. Dummy bits concept.

For instance, assume that n=5, b=3, q_1=57, q_2=65, q_3=73, q_4=89, q_5=97, k_1=8, k_2=8, and k_3=8, k_4=8, k_5=8 then, S_1=1.925288e+09, S_2=1.724252e+09, S_3=9.275727e+08, S4=5.771723e+08, S_5=2.142353e+09. If the message to encrypt was (egypt) then the binary length is 40 bits and the added bits are 5, M = 01100101011001110111100101110000011101000000, then C_1=3.488952e+10, C_2=3.101188e+10, C_3=1.39249e+10 and it will be seen clearly that the result message from the decryption process was (egypt).

IV. MAXIMUM NUMBER OF PAIRS COMPUTATION LIMITS

Referring to the algorithm in [5], and by using some mathematical rules, the suggested ranges for each parameter within the algorithm, and the most appropriate values for each of n and b were determined.

Using the third condition, $q_i > k_i w(q_i \bmod k_i)$, it is seen that, $1 \le q_i \bmod k_i \le k_i - 1$

With $q_i \bmod k_i = k_i - 1$ we have,

$$q_i > k_i * (k_i - 1) * w$$

$$\text{Max } q_i/w > \text{Max } k_i^2 - k_i$$

assume that $\text{Max } q_i / w = R$ then,

$$k_i^2 - k_i - R = 0$$

$$\text{Max } k_i < \frac{1+\sqrt{1+4R}}{2}.$$

And when $q_i \bmod k_i = 1$ then

$$q_i > k_i * w * 1 \quad \text{and} \quad \text{Max } k_i < R.$$

From the key generating procedure we can notice that

- $1 \le R_i = q_i \bmod k_i \le k_i - 1$
- $N_i = \lceil q_i / k_i R_i \rceil$

$$\text{Max } N_i = \lceil \text{Max } q_i / \text{Min } k_i * \text{Min } R_i \rceil = \lceil \text{Max } q_i / (w+1) \rceil$$

- $Q = \prod_{i=1}^{n} q_i$ then Max Q = maximum value allowed.

with the used type, i.e., q_i's with type integer, then Max Q =MaxINT =32767.

- $Q_i b_i \bmod q_i = R_i$

  $p_i = Q_i b_i$ then $\boxed{b_i \le q_i}$

Some selected pairs of (q_i,k_i) satisfy the DK-conditions but some of the calculated b_i's are greater than the corresponding q_i's. For example, assume that b=6, n=3 and the (q_i,k_i)'s = ((304291,64), (304391,65), (304491,65)), so , the calculated bi's =(431652,41732, 77021). It is seen that $b_1 > q_1$, and we have wrong results with these pairs. And by noticing all the examples with right results it is seen that, all b_i's must satisfy the condition $b_i \le q_i$. Thus, we suggest adding this condition to the DK-conditions, and make it a primary condition for selecting the pairs of (q_i,k_i).

- $S_i = N_i P_i \bmod Q$, $N_i P_i \le$ Max allowed value.

  $P_i, N_i \le$ SQRT(Max allowed value).

From the encryption procedure we can see that

$C = m_1 s_1 + m_2 s_2 + \ldots + m_n s_n$,we have Max m = w, and

$$\text{Max } \sum_i S_i = n * \text{Max Si, then}$$

$$\text{Max C} = w * n * \text{Max Si.}$$

Where Max C must satisfy the equation in decryption procedure

$m_i = \lfloor Ck_i/q_i \rfloor \bmod k_i$, and Max C * Max k_i = maximum allowed value.

Selecting appropriate values for qi's can control these conditions.

- Max Q can determine precisely the Max. number of n.
- b determine the beginning of the ranges for each of q_i and k_i.

For example, assume that the used type for q and k are real(Max value 1.7e+38), use b =2, so $w = 2^b - 1 = 3$. After a little work and with $R_i = k_i - 1$, you will have

$q_i \in [37 \div 1.08222e + 18]$

$k_i \in [4 \div 6.00618e + 08]$

$N_i \in [w = 3 \div 2.70555e + 17]$

$b_i \leq q_i$

$s_i \leq 4.80858e + 15$

$C < 1.44257e + 17$

$n \leq 9.$

With $R_i = 1$, you will have,

$q_i \geq 13$, and

$n \leq 10.$

When b=2 and n=10 the selected pairs of (q_i,k_i) are (13,4) (17,4) (21,4) (29,4) (37,4) (38,4) (41,4) (25,4) (43,4) (47,4).

After we have determined the maximum n, which was 10, and by using the same steps with different b's we want to select the appropriate value of b.

Table 1, shows maximum number of bits in each encryption step (n*b) against b and Max n.

Table 1. Maximum n*b

b	Max n	n*b
2	10	20
3	7	21
4	6	24
5	4	20
6	3	18
7	3	21
8	3	24
9	2	18

From the above table it is seen that the maximum n*b which we can take is 24 bits, three character each time. The problem that was found with floating point types is the truncation after a determined number of digits. So the result of 24 bits in each encryption step is not practical from view point of security. Only 2^{24} combination, if each one need 1μsec to find its corresponding ciphertext then after 16.7 sec, the intruder can find the message immediately. This problem will affect the decryption process even of the big range of representation with the floating point types.

To solve such a problem it was necessary to build a new type with its own declaration, definition and operation. For example, by using CLASS structure in C++,[7] one can build these appropriate very_long integers types. Here is a simple example about this kind of solution.

```
class vlong
{
    private:
        unsigned char data[N];
        .....
    public:
        vlong ( )
        { for (int i=0; i < N; ++i ) data[i]='\0'; }
        .....
};
```

It is seen that you can use N bytes to build this new type; Vlong,where N could be chosen according user requirments. In our application the constructor, vlong ( ), is initialized the Vlong type with zeros in the beginning.

By using overloading operations on the needed operators for our algorithm and by noticing the above conditions, one can control his work precisely. In the following we present operations on some operators.

First for the addition process:

If we have two Vlong variables such as v1 and v2 then the addition process will be performed according to the following for loop:

```
• for ( Counter = 0; Counter < N; ++ Counter )
    {
      Temp = v1[Counter] + v2[Counter];
      Res  = Result[Counter] + Temp%256;
      if (Res > 255)
          {
            Result[Counter] = Res % 256;
            Result[Counter+1]= Res / 256;
          }
      else
            Result[Counter] += Temp % 256;

      if (Temp > 255)
            Result[Counter+1] += Temp / 256;
    }
```

Second, the subtraction process:

If we have two Vlong variables such as v1 and v2 then the subtraction process will be performed according to the following steps:

```
• if ( v1 < v2 ) then Error (NoSubtract);

• for ( Counter = 0; Counter < N; ++ Counter )
    {
    if (v1[Counter ] < v2[ Counter ])
        {
        if (v1[Counter ]!=0)
            {
```

```
                    Temp = v1[Counter ]+256;
                    Result[Counter ]= Temp - v2[Counter ];
                    if (v1[Counter +1]= = 0)
                        {
                         for ( s= Counter +1; v1[s]= = 0 && s<N; s++);
                             v1[s]-=1;
                         for ( k= s-1; k>= Counter +1; --k)
                             v1[k]=255;
                        }
                     else
                             v1[Counter +1] -= 1;
                   }
                 else
                  {
                  for (s = Counter ; v1[s]= =0 && s<N ; s++);
                     v1[s]-=1;
                  for ( k= s-1; k>i ; --k)
                     v1[k]=255;
                  Temp = v1[Counter ]+256;
                  Result[Counter ]= Temp - v2[Counter ];
                  }
                }
        else
              Result[Counter ]= v1[Counter ]-v2[Counter ];
        Temp=0;
        }
```

Third for the multiplication process:

If we have two Vlong variables such as v1 and v2 then the multiplication process will be performed according to the following two nested loops:

```
• for ( k=0 ; k < v1.length() ; ++k)
    {
    Carry = Zero;
       for ( m=0 ; m < v2.length() ; ++m)
        {
            Temp = v1[k] * v2[m];
            Temp += Carry[m];
            Carry[m+1]= Temp / 256;
            Num = Result[m + Shift] + Temp%256;
                 if (Num > 255 )
                   {
                   Result[m + Shift+1 ] += Num / 256;
                    Result[m + Shift]   = Num % 256;
                   }
                  else
```

```
                Result[m + Shift]  = Num;
        }
        Result[m + Shift] += Carry[m];
        Shift++;
}
```

Finaly for the division process:

- if x1 = 0 then q = 0 , r = 0
- if x2 = 0 then Error (Division by Zero)
- if x1 =x2 then q = 1, r = 0
- if x1 <x2 then q = 0 , r = x1
- if x1 >x2 then
    1. Counter =1;
    2. Temp = Counter * x2;
    3. if (Temp ≥ x1) then { Result = Counter -1; Reminder =x1 - Result * x2; Stop;}
    4. else { Counter = Counter + 1; go to step 2;}

While for other operations and functions (e.g., / , % , floor, ceil) we use the following functions:

1- The operator / :

```
Vlong operator / (Vlong& x1, Vlong& x2)
{
Vlong q , r;
division (x1,x2,q,r);
return q;
}
```

2- The operator % :

```
Vlong operator % (Vlong& x1, Vlong& x2)
{
Vlong q , r;
division (x1,x2,q,r);
return r;
}
```

3- The function floor :

```
Vlong floor (Vlong& x1, Vlong& x2)
{
Vlong q , r;
division (x1,x2,q,r);
return q;
}
```

4- The function ceil:

```
Vlong ceil (Vlong& x1, Vlong& x2)
{
Vlong q , r;
division (x1,x2,q,r);
if (r = = 0)
```

```
        return q;
else
        return q+1;
}
```

As in the above example, one can redefine the needed operators to implement the suggested algorithm. When N is equal to 50, for example, then the Vlong type is an array of 50 bytes, so that the maximum allowed value is 2^{400} and when b=2 then w=2^b-1=3. k_i >w then Min k_i = 4. From $q_i > k_i$ w R_i then Min q_i > 12.
Max Q = maximum allowed value for the used Vlong type = 2^{400} = 2.58225e+120.
So, when b=2 and with Min q_i =13, we can conclude that n ≤ 100. This means that, Max (n*b)=200 bit =25 character in each encryption step.
With this new Vlong_integer type we have:

- Big range for all of q_i's and k_i's is achieved.
- As the number of encrypted bits in each step is increased, the total time needed for encryption of the whole message will be reduced, and so, the speed of the encryption process will increase.
- If an exhausted search is made to know the message from the public values C , n, S. It will take 5.10^{46} years to predict it.

V. CONCLUSIONS AND SUGGESTIONS

In this paper an improved algorithm for data security within computer networks is introduced. This improved algorithm achieves the following features:

- It is suitable for networks environment.
- Keys can be easily generated.
- Encryption and decryption operations are relatively simple.
- By using the dummy bits method, the length of the binary message becomes free to the user and is not restricted by the condition m = n*b.
- Due to the limitation of the computer representation of numbers, some important conditions are put to guarantee that the computations are within the allowed range.
- The selected pairs are guaranteed to give right results by introducing the new constraint $b_i \le q_i$.
- By using the suggested Vlong type, we can notice that:
  - Big range for all of q_i's and k_i's is achieved.
  - As the number of encrypted bits in each step is increased, the total time needed for encryption of the whole message will be reduced, and so, the speed of the encryption process will increase.
  - If an exhausted search is made to know the message from the public values C , n, S. And if N was equal 50 then it will take 5.10^{46} years to predict it.

CE-1 324

REFERENCES

[1] Muftic, S., Security Mechanisms For Computer Networks,·Muftic, S. / Ellis Horwood limited, England, (1989).

[2] Stinson, D.R., Cryptography Theory and Practice, CRC Press, Inc., Florida, (1995).

[3] Kaufman, Perlman, and Speciner, Network Security Private Communication in a Public World, Prentice Hall PTR Prentice-Hall, Inc., NewJersey, (1995).

[4] Jackson, K. M., Secure Information Transfer, Blackwell Scientific Publications, London, (1995).

[5] Lin, C.H., Chang,C.C., and Lee, R.C.T., " A New Public-Key Cipher System Based Upon the Diophantine Equations", *IEEE TRANSACTIONS on computers*, VOL. 44, NO. 1, January, (1995).

[6] Chang, C.C. and Shieh, J.C., "Pairwise Relatively Prime Generating Polynomials and Their Applications", *Proc. Int. Workshop on Discrete Algorithms and Complexity*, Kyushu, Japan, pp. 137-140, Nov. (1989).

[7] Ira, P., Object Oriented Programming Using C++, Benjamin/ Cummings Publishing Company, Inc., (1993).