Symetric encryption algorithms based on the mathematical structure underlying the three body problem

Samir Bouftass
email : crypticator@gmail.com

April 15, 2021

Abstract

The three body problem is the founding problem of deterministic chaos theory. In this article is proposed a new stream cipher algorithm based on a mathematical structure similar to that underlying the 3 body poblem. Is also proposed to use said structure for the design of new bloc ciphers and hash functions algorithms.

Keywords : 3 body problem, Henri Poincare, Chaos, Symetric encryption.

1 Introduction

The work of Henri Poincare (1854,1912) on three body problem had initiated the genesis of deterministic chaos theory . Practically this problem has still no solution because the underlying physico-mathematical configuration is capable of generating chaotic orbits very sensitive to initial conditions and with great complexities. that's why we think that this configuration or structure could be useful in designing secure cryptographic algorithms. In this article is proposed an algorithm of a stream cipher based on the mathematical structure underlying the 3 body problem in a 256 dimension space with a non linear physics.

2 Description of the algorithm

The physico-mathematical concepts used in the 3 body problem are :

- physical constants : the weights of the 3 bodies, the universal gravitational constant.
- the 3 body positions in space .
- the 3 gravitational fields of force to wich the 3 bodies are subjected.

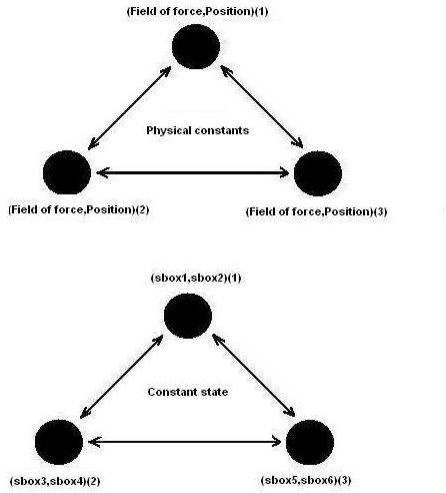
In our algorithm each of these concepts has its equivalent.
The physical constants are equivalant to secret key.
Algorithm's state is equivalant to

- A triplet of Sbox(8X8) , corresponding to the 3 bodies positions and reflecting 3 position vectors in a 256 dimensions space.

- A triplet of Sbox(8X8) , corresponding to the gravitational fields to wich the 3 bodies are subjected, and reflecting 3 non linear force operators .

Figure 1 illustrate this analogy.

Figure 1:



2.1 State intialization algorithm and pseudocode :

The state is constituted by a triplet of a couple of Sboxs (8 X 8) : (op1,pos1), (op2,pos2),(op3,pos3) changing along the enryption/decryption process. A part of the state remain constant. The three couples of Sboxs (8 X 8) : [position vector : pos(i=1..3) , field of force operator : op(i=1..3)] are corresponding to our three virtual bodies. The secret key or a part of it, is corresponding to the physical constants meaning weights and universal gravitational constant. The secret key size is of n * 384 bits , n being an integer. to initialize the state or the 3 Sboxs (8X8) couples ,the key is subdivised into 6 parts , each part helps construct or initialize a Sbox(8X8).

InitializeSbox(S[256], subkey[subkeylength]) pseucocode show how this inialization is performed. with each subkey a non encrypting Sbox(4X4) is encrypted by permutation, this last is used in an encryption of a non encrypting sbox(8X8) wich would be one of the sboxs of the state.

Figure 2 illustrate how the encryption of a non encrypting Sbox(8X8) is performed.

The two 4 least and most significant bits of each Sbox's bytes are encrypted by substitution with a sbox (4X4) after a one bit right rotation . this operation is to be repeated 9 times.

Figure 2:

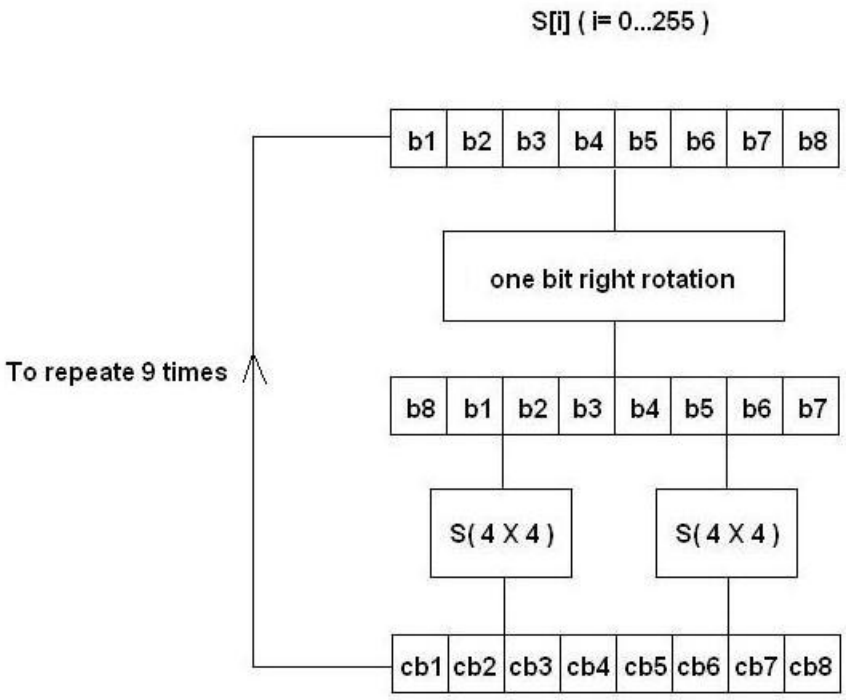


Figure 1 - b

Are following, InitializeState and InitializeSbox pseudocodes :

```
*****************************************************************
InitializeState(key[keylength])
Begin Procedure
    /* arguments = key[keylength] ( the secret key ).
    subkey1[keylength/6]
    subkey2[keylength/6]
    subkey3[keylength/6]
    subkey4[keylength/6]
    subkey5[keylength/6]
    subkey6[keylength/6]

    for i from 0 to keylength/6 - 1
        subkey1[i] := key[i]
    endfor
    for i from keylength/6 to keylength*2/6 - 1
        subkey2[i - keylength/6 ] := key[i]
    endfor
    for i from keylength*2/6 to keylength*3/6 - 1
        subkey3[i - keylength*2/6 ] := key[i]
    endfor
    for i from keylength*3/6 to keylength*4/6 - 1
        subkey4[i - keylength*3/6 ] := key[i]
    endfor
    for i from keylength*4/6 to keylength*5/6 - 1
        subkey5[i - keylength*4/6 ] := key[i]
    endfor
    for i from keylength*5/6 to keylength*6/6 - 1
        subkey6[i - keylength*5/6] := key[i]
    endfor
    /* position Sboxs[8X8]
    pos1[256]
    pos2[256]
    pos3[256]
    /* operators Sboxs[8X8]
    op1[256]
    op2[256]
    op3[256]
    pos1 = InitializeSbox( pos1, subkey1 )
    pos2 = InitializeSbox( pos2, subkey2 )
    pos3 = InitializeSbox( pos3, subkey3 )
    op1 = InitializeSbox( op1, subkey4 )
    op2 = InitializeSbox( op2, subkey5 )
    op3 = InitializeSbox( op3, subkey6 )
End Procedure
*****************************************************************
```

```
**********************************************************************
InitializeSbox( S[256], subkey[subkeylength] )
Begin Procedure

    /* arguments */
    /* S[256] : the Sbox[8X8] to be intialized
    /* subkey[subkeylength] : one of the 6 parts of the secret key
    /* Returned value */
    /* S[256] : the Sbox[8X8] to be intialized.

    S4X4[16]

    for i from 0 to 15
        S4X4[i] := i
    endfor

    for i from 0 to 16
        b := subkey[i] modulo 16;
        b1 := S4X4[i];
        S4X4[i] := S4X4[b];
        S4X4[b] :=b1;
    endfor

    for i from 0 to 255
        S[i] := i
     endfor

    for i from 0 to 255
        a:= i
        for j from 0 to 9
            S[j] : = OneBitRigntRotation(a);
            S[j] : = S4X4[S[j/16]] * 16 + S4X4[S[j modulo 16]] );
            a = S[j];
        endfor
     endfor

     Return( S[256] )
END Procedure
**********************************************************************
```

2.2 Pseudo random number generator algorithm and pseudocode :

the 3 Sbox couples representing the 3 bodies are changing along the encryption process.

- after a first C1 iteration the couple (op1,pos1) changes and then it changes after each N1 iteration. This change is about :

  – the bytes of sbox op1 are xorated with key[(i/N1) mod key.Length] and then encrypted with composite sbox pos2[pos3[]].key being the secret key and i the number of iterations performed .
  – whith Sbox op1, Sbox pos1 is encrypted successivily by substitution and permutation.

- after a first C2 iteration the couple (op2,pos2) changes and then it changes after each N2 iteration. This change is about :

  – the bytes of sbox op2 are xorated with key[(i/N2) mod key.Length] and then encrypted with composite Sbox pos3[pos1[]].key being the secret key and i the number of iterations performed.
  – whith Sbox op2, Sbox pos2 is encrypted successivily by substitution and permutation.

- after a first C3 iteration the couple (op3,pos3) changes and then it changes after each N3 iteration. This change is about :

  – the bytes of sbox op3 are xorated with key[(i/N3) mod key.Length] and then encrypted with composite Sbox pos1[pos2[]].key being the secret key and i the number of iterations performed.
  – whith Sbox op3, Sbox pos3 is encrypted successivily by substitution and permutation.

each iteration , a byte k is outputed :

k := op3[op2[op1[(pos1[i mod 256]+pos2[i mod 256]+pos3[i mod 256]) mod 256]]].

k is corresponding to a projection modulo 256 of the sum vector pos1+pos2+pos3, into the dimension (i mod 256) of a 256 dimension space. The output of the PRNG is used as a keystream.

Are following , the pseudo codes :

- EncDecloop() : encryption's, decryption's loop.

- ChangeOperatorSbox(position1[256],position2[256],operator3[256],byte kbyte) : the bytes of operator3 are xorated with kbyte and then encrypted by substitution with the composite Sbox position1[position2[]].

- ChangePositionSbox(position[256],operator[256]) : substitution encryption followed by a permutation one of position[256] with the help of operator[256].

```
**********************************************************************
EncDecloop(P[])
Begin Procedure

    /* arguments */
    /* P[ ] : the byte array corresponding to plain text.
    /* Returned value */
    /* C[ ] : the byte array corresponding to cipher text.
    /* in the case of decryption, C[] is the argument, and P[] the returned value.

    i := 0
    while i is inferior to the P[ ]s size in bytes :

        if i mode N1 equal C1
            op1 = ChangeOperatorSbox (pos2,pos3,op1,key[(i/N1)
            pos1 = ChangePositionSbox(pos1,op1)
        endif
         if i mode N2 equal C2
            op2 = ChangeOperatorSbox (pos3,pos1,op2,key[(i/N2)
            pos2 = ChangePositionSbox(pos2,op2)
        endif
        if i mode N3 equal C3
            op3 = ChangeOperateurSbox (pos1,pos2,op3,key[(i/N3)
            pos3 = ChangePositionSbox(pos3,op3)
        endif

        k := op3[op2[op1[(pos1[i mod 256]+pos2[i mod 256]+pos3[i mod 256]) mod 256 ]]].

        C[i] = k XOR P[i]
        i++

    endwhile
    Return (C[ ])

END Procedure
**********************************************************************
```

```
**********************************************************************
ChangeOperateurSbox(position1[256],position2[256],operator3[256],byte kbyte)
Begin Procedure

    /* arguments */
    /* position1[256], position2[256]: the 2 Sbox[8X8] position non associated with
    /* operator3[256], this last is the Sbox to be changed.
    /* kbyte : one of the secret key's byte.
    /* Returned value */
    /* operator3[256] : the Sbox to be changed .

    for i from 0 to 255
        operator3[i] = position1[position2[operator3[i] xor kbyte ]];
    endfor
    return (operator3)
END Procedure

**********************************************************************
**********************************************************************
ChangePositionSbox(position[256],operator[256])
Begin Procedure

    /* arguments */
    /* position[256]: the sbox position to be changed.
    /* operator[256] : the Sbox operator asociated with position[256].
    /* returned value */
    /* position[256]: the sbox position to be changed .

    for i from 0 to 255
        position[i] = operator[position[i]];
    endfor

    for i from 0 to 255
        a := i
        a1 := operator[position[i]];
        b = position[a];
        b1 := position[a1];
        position[a] = b1;
        position[a1] = b;
    endfor
    return (position)
END Procedure

**********************************************************************
```

3 Implementation :

In this web page : http://code.google.com/p/3bodyencryptor/ , is hosted a .Net/C sharp project of an encryption application (3BodyEncryptor) , in it is implemented a variant of the proposed stream cipher.
GUI is user friendly , to encrypt a file the user must provide a pass phrase wich the 384 first bits of its SHA2(512) hash are used as a key for the encryption . the files path is also to be provided by the user.
The algorithm is implemented in class file Enigtor40.cs. algorithm procedures had the same names in source code.
In order to achieve a good trade off between effeciency and security, in encryption/decryption loop we had fixed (N1,N2,N3) to (4096,1024,256) and (C1,C2,C3) to (2053,509,127).

4 Conclusion and perspectives :

In this article is proposed a stream cipher algorithm based on the physico-mathematical structure underlying famous 3 body problem in a 256 dimension space with a non linear physics.

As a non linear operation we have opted for :

- Xoration of a sboxs(8X8) bytes with a given byte of the secret key then followed by a substitution encryption with another sbox(8X8)
- Substitution encryption followed by a permutation one, of a sbox(8X8) with the help of another sbox.

The Question is to find other operations more performant in security and efficiency levels.
The strength of proposed algorithm is based on the mathematical structure underlying 3 bodies problem wich is capable of generating chaotic and complex orbites.
Figure 3-a illustrate an organigram of this algorithm's type.
it'is also possible to use this mathematical structure for the design of bloc ciphers, and hash functions algorithms.
Figures 3-b et 3-c illustrate respectively organigrams of a bloc cipher and a hash function based on the mathematical structure of the three bodies problem.
It is also possible to imagine and designe encryption algorithms based on the mathematical structure underlying N body problem (N superior to 3).

Figure 3:

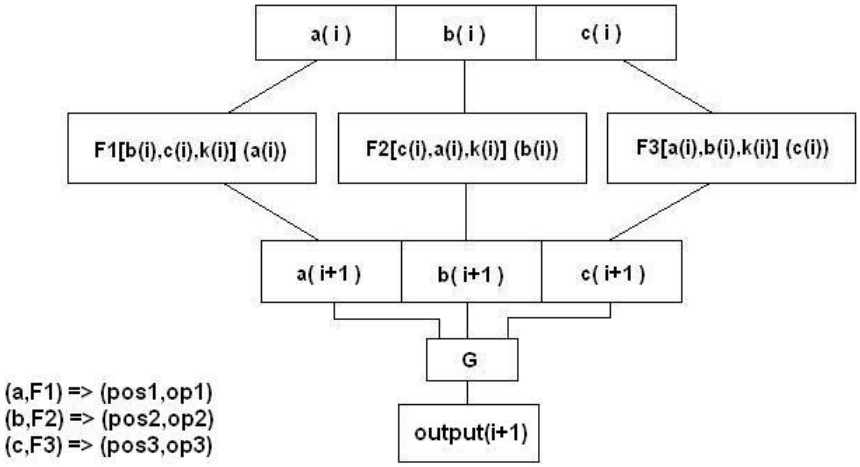


Figure 2 - a

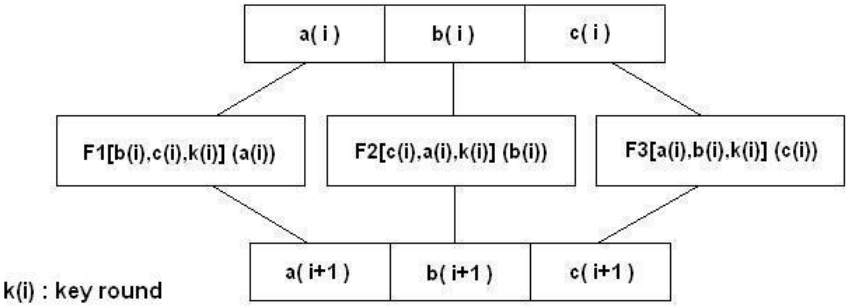


Figure 2 - b

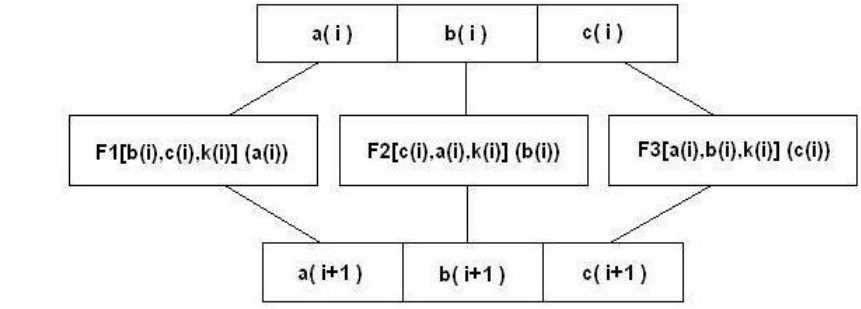


Figure 2 - c

References

[1] June Barrow-Green, *Poincare and the Three Body Problem - Volume 2* (1997), The American mathematical Society .

[2] Bruce Schneier, *Applied Cryptography Second Edition*,(1996), John Wiley and Sons.

[3] Ljupco Kocarev, Shiguo Lian, *Chaos-based Cryptography: Theory, Algorithms and Applications*, (2011), Springer Verlag Berlin Heidelberg.