Eng. & Technology, Vol.6, No.24, 2005

An Improvement of Serpent Algorithm

Saleh M. Al-Karaawy* Ashwaq T. Hashim

Received on 4/1/2005
Accepted on 11/9/2005

Abstract

An improvement to Serpent algorithm will be proposed. Serpent algorithm is a Feistel network, iterating a simple encryption function 32 times. The block size is 256 bits, and the key can be any length up to 256 bytes. The proposed algorithm is designed to take advantage of the powerful facility, which is supported by Serpent algorithm with overcoming its weaknesses, resulting in a much improved security/performance tradeoff over existing ciphers. As a result, the proposed algorithm offers better security than Serpent algorithm. The proposed algorithm is compact, simple and easy to understand. It increases the block size to overcome the matching cipher text and brute force attacks.

Key words: Serpent Algorithm, Encryption Techniques, Network Security.

تحسين خوارزمية سيربنت

الخلاصة

في هذا البحث سيقدم مقترح لتحسين خوارزمية سيربنت والتي هي عبارة عن شبكة فيستل حيث تعاد عملية التشفير 32 مرة. في هذه الخوارزمية يكون حجم البلوك 256 بت في حين يكون طول المفتاح أي طول يقترح لغاية 256 بايت. لقد صممت الخوارزمية المقترحة لتأخذ محاسن وقوة خوارزمية سيربنت وتلافي مساوئها وبالتالي الحصول على نسخة محسنة من ناحية الأداء/السرية في مجال التشفير. وفي النتيجة فان الخوارزمية المقترحة تقدم سرية أفضل من نسخة خوارزمية سيربنت. إن الخوارزمية المقترحة بسيطة وسهلة الفهم ومن ناحية اخرى فان حجم البلوك تم زيادته لتجنب عمليات الهجوم.

1. Introduction

Symmetric-key block ciphers have long been used as a fundamental cryptographic element for providing information security. Although they are primarily designed for providing data confidentiality, their versatility allows them to serve as a main component in the construction of many cryptographic systems such as pseudo random number generators, message authentication protocols, stream ciphers, and hash functions. There are many symmetric-key block ciphers, which offer different levels of security, flexibility, and efficiency. Among the many symmetric-key block ciphers currently available, some (such as DES, RC5, CAST, Blowfish, FEAL, SAFER, and IDEA) have received the greatest practical interest. Most symmetric-key block ciphers (such as DES, RC5, CAST, and Blowfish) are based on a "Feistel" network

---

* Dept. of Control & Systems Eng., University of Technology.

https://doi.org/10.30684/etj.24.6.11
University of Technology-Iraq, Baghdad, Iraq/2412-0758

construct and a "round function" [1]. The round function provides a basic encryption mechanism by composing several simple linear and non-linear operations such as Ex-ORing, substitution, permutetion, and modular arithmetic. The strength of a Feistel cipher depends heavily on the degree of diffusion and non-linearity properties provided by the round function. Many ciphers (such as DES and CAST) base their round functions on a construct called a "substitution box" (s-box) as a source of diffusion and non-linearity. Some ciphers (such as RC5) use bit-wise data-dependent rotations and a few other ciphers (such as IDEA) use multiplication in their round functions for diffusion [1]. Figure 1 illustrates the round of Serpent, while Fig. 2 represents the scheme of Serpent algorithm [2,3,4].

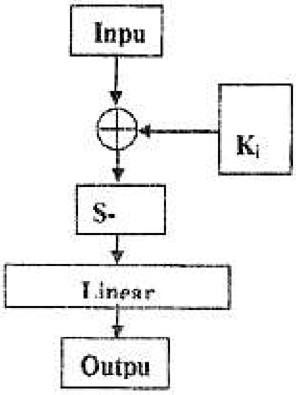


Fig. 1 A Round of Serpent

One of the main goals in secret key cryptography is the development of design criteria, so that we develop Serpent algorithm, which is one of the final list candidates, to become much suited to the next generation of 64-bit processor and to become more secure compared with the previous algorithm.

2. Key Generation

The serpent encryption requires 132 32-bits of key material. First the user supplied 256-bit key **K** is expanded to 33 128-bit subkeys $K_0, \ldots, K_{32}$, the key **K** will be written as eight 32-bit $w_{i-8} \ldots w_{i-1}$ and expand these into an intermediate key $w_0, \ldots, w_{131}$ by the following affine recurrence[5,6]:

$\{k_1, k_{34}, k_{67}, k_{100}\} = S_2\{w_1, w_{34}, w_{67}, w_{100}\}$

.

.

$\{k_{31}, k_{64}, k_{97}, k_{130}\} = S_4\{w_{31}, w_{64}, w_{97}, w_{130}\}$

$\{k_{32}, k_{65}, k_{98}, k_{131}\} = S_3\{w_{32}, w_{65}, w_{98}, w_{131}\}$

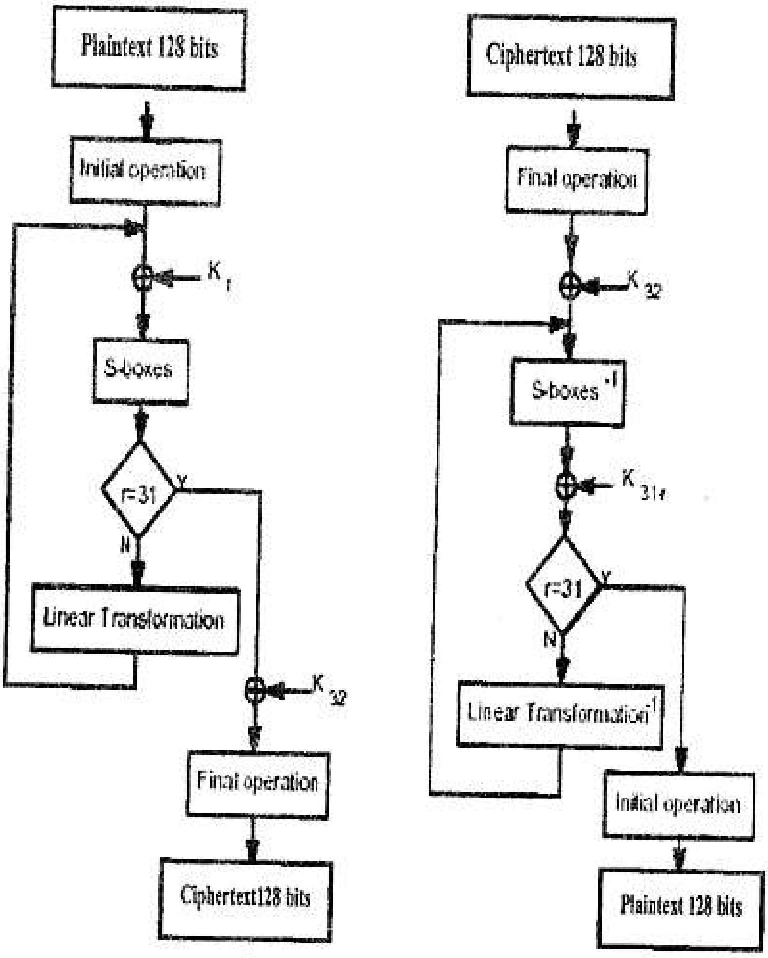
After that renumber the 32-bit values k_j as 128-bit subkeys k_i (for $i \in \{0 \ldots r\}$) as follows:

$$K_i = \{k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3}\}$$

The round keys are now calculated from the prekeys using the S-boxes. The S-boxes are used to transform the prekeys w_i into words k_i of round key by dividing the vector of prekeys into four sections and transformation the i^{th} words of each of the four sections using $S_i(r+3-i) \bmod r$. This can be seen simply for the default case **r=32** as follows:

$\{k_0, k_{33}, k_{66}, k_{99}\} = S_3\{w_0, w_{33}, w_{66}, w_{99}\}$

$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \Phi) <$



Where r =0.....31 (round number)

(a) The Encryption unit (b) The Decryption unit.

Fig. 2 Encryption and Decryption Units of Serpent Algorithm

Where **Φ**, is the fractional part of the golden ratio 0x9e3779b9 in hexadecimal. The underlying polynomial $(x^9 + x^7 + x^5 + x^3 + 1)$ is primitive, which together with the addition of the round index is chosen to ensure an even distribution of key bits throughout the rounds, and to eliminate weak keys and related keys.

The Initial Permutation (***IP***) is applied to the round key in order to place the key bits in the correct column, i.e., $\mathbf{K}_i$ ***IP*** *(K_i)*. Figure 3 illustrates the key generation algorithm [2].

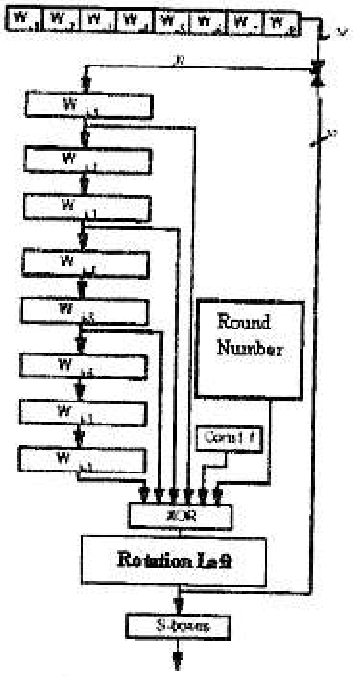


Fig. 3 The Key Generation Algorithm

3. The Improved Algorithm

Improved Serpent algorithm was intentionally designed to be extremely simple, to invite analysis shedding light on the security provided by extensive use of data-dependent rotations. To meet the requirements of the AES, a block cipher must handle 256-bit input/output blocks. While Serpent is an exceptionally fast block cipher, extending it to act on 256-bit blocks in the most natural manner would result in using two 128-bit working registers. The specified target architecture and languages for AES do not yet support 128-bit operations in an efficient manner. Thus, the design to use four 64-bit registers rather than two 128-bit registers in the P-function should be modified and still use the four 32-bit register by applying one round of previous Serpent algorithm to the left half of the proposed algorithm.

It is worth observing that with a cipher running at the rate of one terabit per second (that is, encrypting data at the rate of 10^{12} bits/second), the time required for 50 computers working in parallel to encrypt 2^{64} blocks of data is more than a year; to encrypt 2^{80} blocks of data is more than 98,000 years; and to encrypt 2^{128} blocks of data is more than 10^{19} years [7].

While having a data requirement of 2^{128} blocks of data for a successful attack might be viewed as sufficient in practical terms, the proposed algorithm aims to provide a much greater level of security. The community as a whole will decide which level of security a cipher; in particular an AES candidate should satisfy. Should this be less than a data requirement of 2^{256} blocks of data then, thereby providing an improvement in performance?

Figure 4 shows the structure of the improved algorithm, which in fact is a Feistel network. It consists of splitting the plaintext into two 128-bits halves. Feistel cipher is a special class of iterated block ciphers, where the ciphertext is calculated from the plaintext by repeated application of the same transformation or round function. The round function is applied to one half using a subkey and the output of F function is XORed with the other half. The two halves are then swapped. Each round follows the same pattern except for the last round where there is no swapping. A nice feature of a Feistel cipher is that encryption and decryption are structurally identical, through the subkeys used during encryption at each round and are taken in reverse order during decryption.

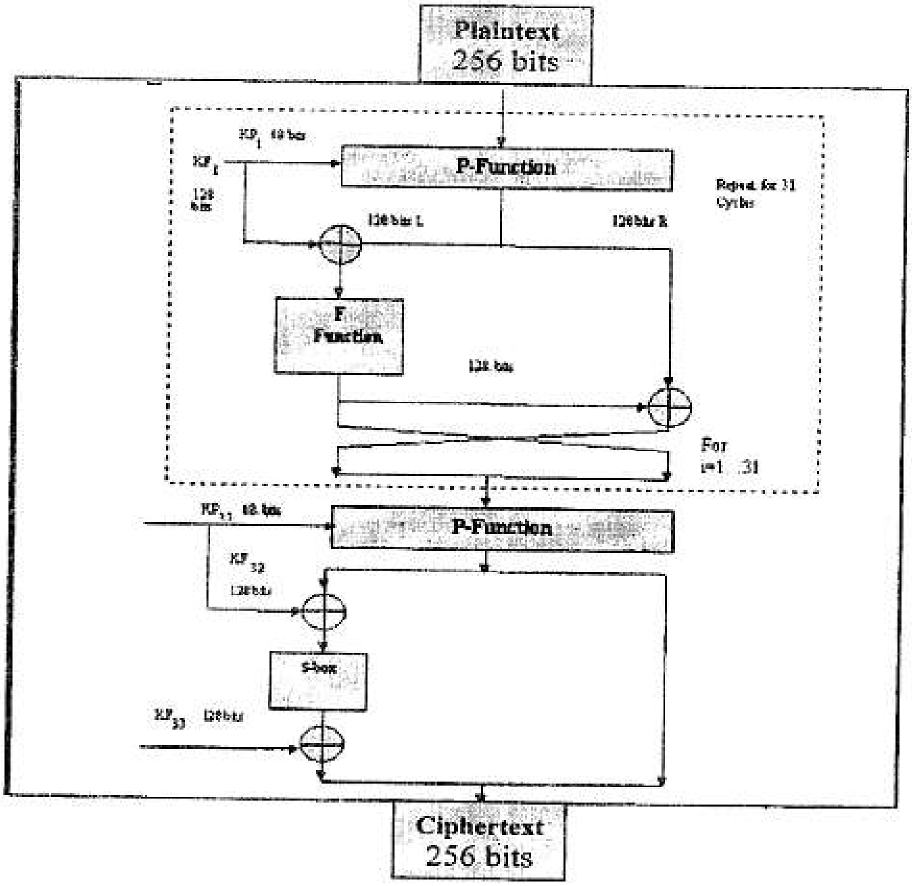


Fig. 4 The Structure of Improved Serpent Algorithm

3.1 Encryption and Decryption

The design of the proposed algorithm began with a consideration of Serpent as a potential candidate for an AES submission. Modifications were then made to meet the AES requirements, to increase security, and to improve performance.

Let IP represent Initial Permutation and L/R Left/Right shifting, the Encryption algorithm is used to encrypt the plaintext P using secret key K to produce Ciphertext C and can be described as follows:

Decryption Algorithm

```
1. A = Ciphertext C.
2. For i= r,.....,0
      2.1 swap L, R
      2.2 $R=R \oplus INV\text{-}LT(F(L\oplus KF_i))$
      When i=r step 2.2 is replaced by
            $R_i=R \oplus F(L\oplus KF_i) \oplus KF_{r+1}$
      2.3 $(L/R) = P(A, KP_{i,1}, KP_{i,2})$.
      2.4 $A=(L/R)$
3. Plaintext= $P(A, KP_{r,1}, KP_{r,2})$.
```

The decryption process is the same as the encryption process except that the round keys are applied in reverse order and use the inverse of linear transformation.

The decryption algorithm is described below, which is used to decrypt the Ciphertext C using secrete key K to produce the plaintext P.

3.2 The F-function

The F-function is, of 128-bit, and used to apply the Serpent algorithm to the left half of the proposed algorithm. This transformation appears to meet our security goals while taking advantage of previous Serpent algorithm that is efficiently implemented on most modern processors. Figure 5 illustrates the F-function of Serpent algorithm which represents the left half of the improved Serpent algorithm.

3.3 The P-Function

In Serpent algorithm (before improvement) the initial permutation and corresponding final permutation public and fixed table are used. They have no effect on the attacks but the fixed table makes the encrypted algorithm harder to explain and they do not affect the security.

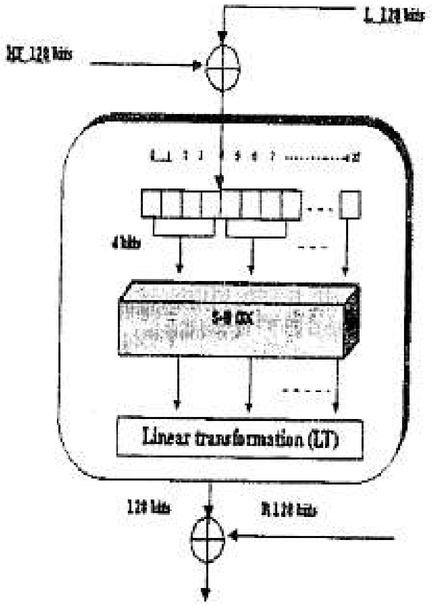


Figure 5 The F-function

In the improved algorithm key dependent initial permutation is applied as a reversible mixing function (which is more complicated reversible mixing function) before F function. It is used to overcome a Feistel structure weakness that each round transformation always keeps one half of the block constant so that the P-function would further confuse the entry values into the Feistel network and ensure a complete avalanche effect after the first two rounds.

The P-function is required to provide the necessary diffusion and confusion to the input block, such that additive differences will be destroyed as the key is changed.

This could provide a protection against linear and differential cryptanalysis. Figure 6 illustrates the P-function which has 256-bit input A and 256 A and 256-bit output D.

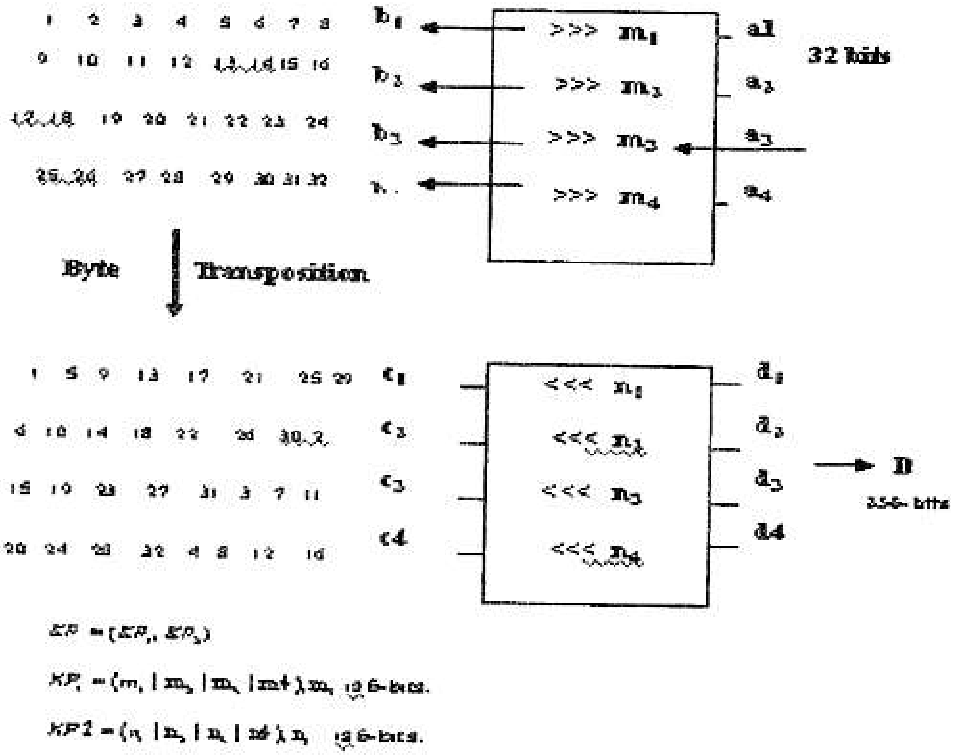
It adopts "byte transpo-sition" and 48-bit subkey ($KP_{i,1} \mid KP_{i,2}$) to control data rotations, where $i=0,\ldots,r-1$.

Let: $KP_1 = (m_1, m_2, m_3, m_4)$,
And $KP_2 = (n_1, n_2, n_3, n_4)$,

where m_j and n_j are 6-bit subkey and not equal to zero, $j= 1,\ldots, 4$.
The function:
$D = P(A, KP_1 \mid \mathrm{KP}_2)$
is defined by following:

i) Right rotation:
$b_j=a_j>>>m_j$,
for $j=1,\ldots,4$.

ii) Byte transposition:



Fig, 6 The P-function

Considering the block to be made up of bytes 1 to 32, these bytes are arranged in a rectangle, and shifted as follows:

From:

1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
4	8	12	16	20	24	28	32

To:

1	5	9	13	17	21	25	29
6	10	14	18	22	26	30	2
15	19	23	27	31	3	7	11
20	24	28	32	4	8	12	16

The transposition step ensures that the different bytes of each row do not interact with the corresponding byte in other rows.

iii) Left rotation:,

$d_j = c_j <<< n_j$

for $j=1,\ldots,4$.

It is clear that in this proposal each input word a_j affects all output words and consequently each output word is affected by all input words. In, *P*-function, the permutations are key dependent so that it could avoid linking plaintexts to input to the *F*-function and ciphertexts to input to the *F*-function in each round.

4. Evaluation of the Improved Algorithm

The improved Serpent algorithm increases the security of the original serpent algorithm by using block size of 256-bits. If the length of block is small, then the attack on the algorithm will be easy. All possibilities to the improvement algorithm of a data are 2^{256} blocks, thereby providing an improvement in performance. As described before, and referring to Fig.6, the improved Serpent algorithm increases the security by using the *P*-function, where each input word a_j affects all output words and each output word is affected by all input words. In, *P*-function, the permutations are key dependent so that it could avoid linking plaintexts to input to the first *F*-function and ciphertexts to input to the last *F*-function.

The improved algorithm is pertinent to the following types of attack and this is due to many reasons:

a) Differential cryptanalysis is largely theoretical. The enormous time and data requirements to mount a differential cryptanalytic attack put almost beyond the reach of everyone.

b) Key-dependent permutation function is used before the F-function such that the input bits are exchanged under the control of subkeys, so that the additive difference will be destroyed, as the bits are exchanged, this could provide protection against linear and differential cryptanalysis.

c) The previous Serpent S-boxes were well designed with respect to linear and differential cryptanalysis. So, the improvement algorithm uses the same S-boxes of the previous Serpent algorithm.

The block size of 256-bits makes Serpent algorithm vulnerable to the matching ciphertext attack, because after encryption of 2^{64} blocks, equal ciphertexts can be expected and information is leaked about plaintext. So that, the improved Serpent algorithm with 256-bits block size is resistant to matching ciphertext attacks and hence it is required for 2^{128} ciphertext.

4.1 Avalanche Effect

Horst Feistel referres to the avalanche effect as: "a small change in the key gives rise to a large change in the ciphertext" [8].

Avalanche effect is a property that is used to measure the strength of the algorithm. It is used for making statistical test on the ciphertext that is produced from encrypting variable plaintexts under the control of the key of length 256-bits ***Plaintext and Key in hexadecimal form:***

KEY=000000000000000000000000000
0000000000000000000000000000000

PT=00000000000000000000000000000
0000000000000000000000000000000
PT=44444444444444444444444444444
4444444444444444444444444444444
PT=11001100110011001100110011001
1001100110011001100110011001100
PT=55555555555555555555555555555
5555555555555555555555555555555
PT=01010101010101010101010101010
1010101010101010101010101010101
PT=fffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffff
fffffffff

Tables 1 and 2 illustrate that the number of blocks that have avalanche effect greater than 64 before improvement is 2 out of 6 and the average of avalanche effect is 61.1. After using the improved Serpent algorithm all the blocks have avalanche effect greater than 128 and the average of the avalanche effect is 132.6.

Let us consider another example to test Avalanche effect that assets the result.

Key in hexadecimal form:
Key= 3000000000000000000000
000000000000000000000
000000000000000000000

Plaintext in hexadecimal form:
PT = 00000000000000000000000
000000001
PT = 00000000000000000000000
000000004
PT = 00000000000000000000000
000000010
PT = 00000000000000000000000
000000040
PT = 00000000000000000000000
000000100
PT = 00000000000000000000000
000000400
PT = 00000000000000000000000
000001000
PT = 00000000000000000000000
000004000
PT = 00000000000000000000000
000010000
PT = 00000000000000000000000
000040000
PT = 00000000000000000000000
000100000
PT = 00000000000000000000000
000400000
PT = 00000000000000000000000
001000000
PT = 00000000000000000000000
004000000
PT = 00000000000000000000000
010000000
PT = 00000000000000000000000
040000000
PT = 00000000000000000000000
100000000
PT = 00000000000000000000000
400000000
PT = 00000000000000000000001
000000000
PT = 00000000000000000000004

000000000
PT = 00000000000000000000010
000000000
PT = 00000000000000000000040
000000000
PT = 00000000000000000000100
000000000
PT = 00000000000000000000400
000000000
PT = 00000000000000000001000
000000000
PT = 00000000000000000004000
000000000
PT = 00000000000000000010000
000000000
PT = 00000000000000000040000
000000000
PT = 00000000000000000100000
000000000
PT = 00000000000000000400000
000000000
PT = 00000000000000001000000
000000000
PT = 00000000000000004000000
000000

Figures 7 and 8 represent the avalanche effect on the ciphertext when only one bit is changed and the Serpent algorithm of 128-bit block size is performed before improvement and 256-bits block size after improvement. The figures show that the changes are 54 to 76 bits out of 128 bits when the algorithm before improvement is performed.

Table1. Avalanche before improvement

Block NO.	Ciphertext 128-bits in Hexadecimal	Weight
1	CT1=a82b6749f98dd998418195089104945 CT2=57d4866d80b663a39ded74039f1b1a3	66
2	CT1=47011f35136a421a93d39f06503869cc CT2=9e55874b386a40114419cecf515b04ab	57
3	CT1=4f8fc0128f44aa4886d33dd4a5443185 CT2=838687fbfe34864acea359f67c2cd24b	55
4	CT1=a1535c3cd908f7e833e00494c8163135 CT2=43548c00e90e77105ff7742f2dd08f53	60
5	CT1=55b0438b3c3a22a411094b68b5cf0e35 CT2=b7a1f80f4cb6d5002027af56b8312d17	60
6	CT1=9e200afef2a35222ff02b4446f0311dd CT2=c213e6b0dfb5ccfa4b945a92c3ac26c	69
Key1	0000000000000000000000000000000000000000000000000000000000000000	
Key2	1000000000000000000000000000000000000000000000000000000000000000	

Table 2 Avalanche effects after improvement

Block NO.	Ciphertext 128-bits in Hexadecimal	Weight
1	CT1=2b4940f22556032852270 7f68fc01a04b663246add75acd4de83aeb6940f768 CT2=7a3cd59334d5cfe25b31e8daf482f0e29c291db03be0a06828d83a1be5b8e45f	128
2	CT1=66ec237557a502f7e1fcb64557f0d4b4b89fcc2e6e2250198b570cbc78b1471e CT2=7a3cd559334d5cfe25b31e8daf482f0e29c291db03be0a06828d83a1be5b8e45f	136
3	CT1=aa3fbdb2d7a6f1e4cdf879ae4077eb83d0de30dcc4951d15a54612b2ab85c480 CT2=7a3cd59334d5cfe25b31e8daf482f0e29c291db03be0a06828d83a1be5b8e45f	134
4	CT1=df3ae4f225a37bdf332f954abb207521f67d6bc081b84810713b006760796272 CT2=7a3cd559334d5cfe25b31e8daf482f0e29c291db03be0a06828d83a1be5b8e45f	136
5	CT1=b9dc316a0445636ca67e19c87ac1da68ec6e1cd67169d5afbeb8598e360c52 CT2=7a3cd559334d5cfe25b31e8daf482f0e29c291db03be0a06828d83a1be5b8e45f	132
6	CT1=8b6b0f634e25183381aa485471a2a6196fdd1208cd09112845f3a757cc7996d5 CT2=7a3cd559334d5cfe25b31e8daf482f0e29c291db03be0a06828d83a1be5b8e45f	128
Key1	0000000000000000000000000000000000000000000000000000000000000000	
Key2	1000000000000000000000000000000000000000000000000000000000000000	

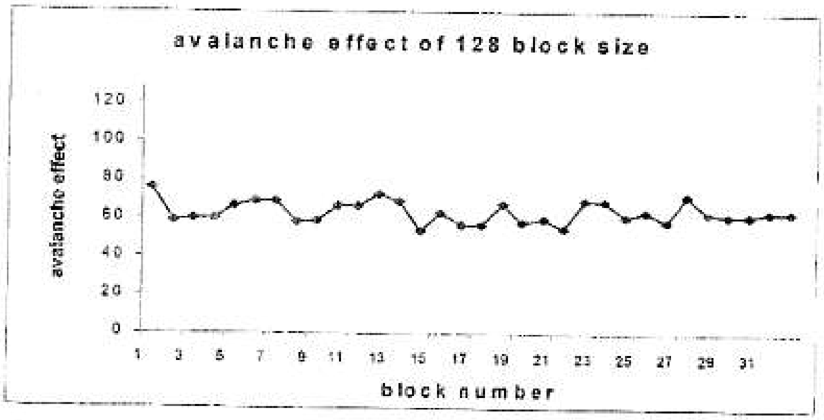


Figure 7 Avalanche effect on the ciphertext when only one bit is changed and the Serpent algorithm of 128-bit block size is performed

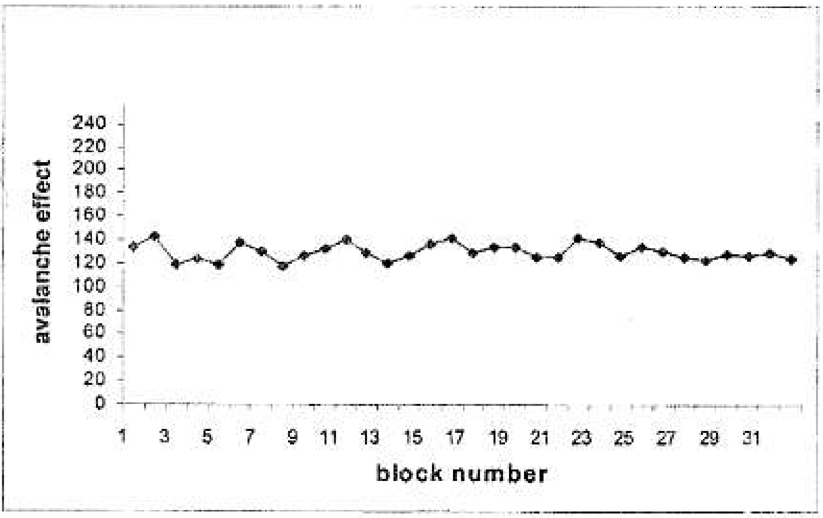


Figure 8 Avalanche effect on the ciphertext when only one bit is changed and the Improved Serpent algorithm of 256-bit block size is performed

Meanwhile, after the improvement the changes are 117 to 142 bits out of 256 bits. The number of blocks that have avalanche effect is larger than 64 (which is half of 128) after performing unimproved Serpent algorithm is 12 out of 32 blocks. From the figure it is clear that the number of blocks that have avalanche effect larger than 128 (which is half of 256) after performing improved Serpent algorithm is 17 out of 32 blocks. Moreover, we note that the average of avalanche effect before and after improvement is 63.5 and 129.3 respectively.

4.2 Randomness Test

Symmetric-key block ciphers are primarily designed for providing data confidentiality, their versatility allows them to serve as a main component in the construction of many cryptographic systems such as pseudo random number generators, message authentica-tion protocols, stream ciphers, and hash functions [8].

Three well-known tests [9] can be used to test the proposed algorithm, these tests are frequency, serial and auto-correlation. These tests have been made on the cipher text that is produced from encrypting of eight blocks of the block mentioned in Section 4.1 using Serpent algorithm before improvement and after improvement with block size 128-bits and 256-bits. Table 4 shows these tests.

4.3 Time Requirement

In this section the time requirements are computed for the algorithm before and after improvement.

This comparison represents the time for encryption and decryption using Serpent algorithm before and after improvement as shown in Table 3.

The following program part written in C, is used to compute the time required to encrypt file **example.txt** which contains a message of specific length.

```
clock_t   beg , end;
 if                    ((                    fp=fopen(
     "example.txt","rb"))==NULL)
     { printf("error: can not open file");
     exit(0);}
 beg=clock();
 while ( !feof ( fp ))
    {
 ENCRYPTION   PROCESS   FOR   ONE
          BLOCK
    }
end=clock();
fclose(fp);
printf("\n%8lf",(double)(end-
       beg)/CLK_TCK); // To get time in
       Second
```

Table 3. Speed comparisons of Serpent algorithm on a Pentium II PC before and after improvement

Algorithm	**Block size**	**Number of Bytes**	**Speed (Clocks per Sec.)**
Serpent	128-bits	1000	0.16
		10000	1.26
		100000	7.14
Improved Serpent	256-bits	1000	0.11
		10000	1.32
		100000	8.73

Table 4 . Randomness test of Serpent algorithm before and after improvement
Auto C. test = **Auto Correlation Test**

Block no.	***Randomness test***	***Serpent of 256 bits block size***	***Serpent of 128 bits block size***	***Degree of Freedom***
1	Freq. test	0.0765625	0.125000	With 1 <=3.84
	Serial test	0.0814767	0.02460	With 5 >= 14.1
	Auto C. test			With 1<=3.84
	d= 1	0.074254	0.017349	
	d= 2	0.093110	0.007448	
	d= 3	0.114186	0.001643	
	d= 4	0.137507	0.000032	
	d= 5	0.163100	0.002717	
	d= 6	0.190993	0.009806	
	d= 7	0.221213	0.021406	
	Freq. test	1.562500	3.12500	
	Serial test	2.464951	3.197835	

2	Auto C. test d= 1 d= 2 d= 3 d= 4 d= 5 d= 6 D= 7	0.000301 0.002260 0.006059 0.011721 0.019267 0.028720 0.040104	0.016101 0.008066 0.002749 0.000218 0.000539 0.003783 0.010023	
3	Freq. test	0.000000	0.781250	
	Serial test	0.105882	0.880167	
	Auto C. test d= 1 d= 2 d= 3 d= 4 d= 5 d= 6 D= 7	0.0360698 0.0325624 0.0292219 0.0260501 0.0230492 0.0202211 0.0175680	0.035800 0.022170 0.011721 0.004530 0.000677 0.000244 0.003315	
4	Freq. test	1.562500	0.000000	
	Serial test	1.617892	3.488189	
	Auto C. test d= 1 d= 2 d= 3 d= 4 d= 5 d= 6 D= 7	0.019966 0.030337 0.042924 0.057754 0.074852 0.094246 0.115964	0.868110 0.960317 1.058000 1.161290 1.270325 1.385246 1.506198	
5	Freq. test	0.0140625	0.031250	With 1 <=3.84
	Serial test	0.561336	0.685285	With 5 >= 14.1
	Auto C. test d= 1 d= 2 d= 3 d= 4 d= 5 d= 6 D= 7	0.023468 0.034357 0.047371 0.062534 0.079873 0.099414 0.121183	0.154589 0.193578 0.237316 0.285919 0.339506 0.398199 0.462124	With 1<=3.84
6	Freq. test	0.562500	0.500000	
	Serial test	3.312010	2.799213	
	Auto C. test d= 1 d= 2 d= 3 d= 4 d= 5 d= 6 D= 7	0.288350 0.322557 0.358829 0.397191 0.437669 0.480288 0.525073	0.600846 0.672355 0.748523 0.829463 0.915292 1.006131 1.102102	
	Freq. test	0.250000	1.125000	
	Serial test	2.710784	7.780512	

7	Auto C. test d= 1 d= 2 d= 3 d= 4 d= 5 d= 6 d= 7	0.324017 0.365019 0.408642 0.454918 0.503878 0.555554 0.609980	1.838584 1.964711 2.096142 2.233004 2.375430 2.523557 2.677526	
8	Freq. test	0.0265625	0.125000	
	Serial test	0.0757904	0.402559	
	Auto C. test d= 1 d= 2 d= 3 d= 4 d= 5 d= 6 d= 7	0.000831 0.003573 0.008251 0.014889 0.023510 0.034138 0.046797	0.045143 0.067209 0.093881 0.125272 0.161497 0.202673 0.248925	

Conclusions

The proposed algorithm is used in a large variety of applications including protection of the secrecy of login passwords, e-mail messages, and video transmissions (such as pay-per--view movies) and stored data files.

The block size can be increased to 256 bits instead of 128 bits by using round function in a Feistel construction; this makes the exhaustive key search and the matching ciphertext attack are infeasible. The proposed algorithm also uses key dependent function before and after each round instead of initial and final permutation which uses fixed tables. This gives the algorithm, a protection against differential and linear cryptanalysis.

The results obtained illustrate that the improved algorithm has the following features:

1- The same algorithm criteria for encryption and decryption with some key schedules can be used.

2- It is based on simple theory principles and simple arithmetic operations and easy to implement, easy to understand algorithm.

3- It adopts key–dependent permutation and substitution to provide protection against differential and linear cryptanalysis so, the improved algorithm is secure.

4- It uses the subkey to control data permutation and data substitution.

5-From the results obtained from the avalanche effect and random test, after measuring the strength of the proposed algorithm we can conclude that the proposed algorithm can be used to increase the security.

6- From the time require-ment section, we notice that although the improved algorithm uses P-function after each round, the time which is required to encrypt the same file after the improvement takes less time compared with previous Serpent algorithm. The reason for that is the encryption process encrypts double blocks instead of one block of previous Serpent algorithm.

References:

1. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas, L. O'Connor, M. Peyravian, D. Safford and N. Zunic, "Mars a candidate cipher for AES", First Advanced Encryption Standard (AES) Conference, Ventura, CA, 1998.

2. Anderson R., Biham E., and Knudsen L., "Serpent: A Proposal for the Advanced Encryption Standard," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

3. Serge Mister, "Properties of the Building Blocks of Serpent" http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html, May 15, 2000

4. Weeks B., Bean M., Rozylowicz T., and Ficke C., " Hardware Performance Simul-ations of Round 2 Advanced Encryption Standard Algorithms", *National Security Agency*, 2001

5. Anderson R., Biham E., and Knudsen L.," Serpent and Smart Cards." Third Smart Card Research and Advanced Applicat-ions Conference Proceedings, 1998, to appear. NIST AES Proposal, Jun 1998

6. Shakir M. " A New Feedback Symmetric Block Cipher Method" *Ph. D. Thesis, University of Tech., Baghdad*, 1997.

7. Bora P. and Ezajka T., "Implementation of the Serpent algorithm using altera FPGA devices", http://csrc.nist.gov/encryption/aes/round2/comments/20000513-pbora.pdf

8. Shiner B. "Applied Cryptography Second Edition Protocols. Algorithms, and Source, and Source Code in C", *John Wiley and Sons, Inc.*, 1996.

9. وسيم الحمداني ، "أنظمة التشفير". 1996.