Improving Keystream Generation for Sosemanuk Stream Cipher Using Twofish Block Cipher

Dr. Shant K. Avakian* & Rana S. Mohammed**

Received on:22/12/2008
Accepted on:5/3/2009

Abstract

This paper will introduce two proposal algorithms (Snowfish 1) and the (Snowfish 2) to improve the Sosemanuk stream cipher algorithm by benefiting from the efficient properties of the Twofish block cipher and also use its key schedule, key-dependent Sbox to increase the security, randomness and try to avoid the guess and determine attack of Sosemanuk. These two proposals use Twofish algorithm rather than Serpent algorithm which was used in the Sosemanuk and also they use key-dependent Sbox rather than static Sbox. They are similar in the same key length (128 to 256 bit), IV length (128 bit), LFSR length, FSM functions and output transformation.

In this paper will make a comparison between Snowfish 1, Snowfish 2 and Sosemanuk algorithms by using the tests of randomness, the structural tests and the complexity of the algorithm. These tests give results that show the two proposed algorithms have good results in increasing the security and randomness compared with Sosemanuk algorithm.

Keywords: twofish block cipher, sosemanuk stream cipher, key dependent s-box, LFSR, keystream.

تحسين مولد سلسلة المفتاح للتشفير التسلسلي Sosemanuk بأستخدام تشفير الكتلي Twofish

الخلاصة

في هذا البحث تم تقديم مقترحين لخوارزميتين الاول يسمى Snowfish 1 و الثاني يسمى Snowfish 2 لتحسين خوارزمية تشفير التسلسلي Sosemanuk من خلال الاستفادة من محاسن التشفير الكتلي Twofish و الذي يمتلك جدولة المفاتيح و Sbox جيدين في زيادة الأمنية و العشوائية و محاولة التغلب على هجوم التخمين و التحديد الموجود على Sosemanuk . إن هذين الاقتراحين يستخدمون خوارزمية Twofish بدلا من خوارزمية التشفير الكتلي Serpent الذي هو المستخدم في Sosemanuk . و كذلك يستخدمون Sbox الذي يعتمد على المفتاح بدلا من Sbox الذي يعتمد على قيم الثابتة أو جدول ثابت. هم متشابهين في استخدام نفس الطول للمفتاح (128 إلى 256 بت) و طول سلسلة الابتدائية 128 بت و طول مسجل الازاحة ذو دالة تغذية مرتدة خطية و دوال FSM و دالة المخرجات .

*[1]Education Collage, Computer Science Department, University of Mustansiryah/ Baghdad.
** Computer Science Department, University of Technology/Baghdad.

https://doi.org/10.30684/etj.27.10.7
2412-0758/University of Technology-Iraq, Baghdad, Iraq

في هذا البحث تم عمل مقارنة بين Snowfish 1 و Snowfish 2 و Sosemanuk باستخدام اختبارات العشوائية و الهيكلية و تعقيد الخوارزمية فنتائج الاختبارات تبين إن النظامين المقترحين لهما نتائج جيدة في تحقيق الأمنية و العشوائية مقارنة مع نظام إل Sosemanuk .

1.Inroduction

Snowfish aims to improve the Sosemanuk from the security, randomness, structure and complexity points depending on the advantages and good properties of twofish block cipher rather than serpent block cipher. This paper introduce two proposed stream cipher where the name of the first proposal is (Snowfish 1) and the second is (Snowfish 2).

These two algorithms are different in how to use key-dependent Sbox, Snowfish 1 has four 8×8 bit Sboxes and is called *Sbox128* which depends on one key ($R1_{t-1}$) and Snowfish 2 has four 8×8 bit Sboxes called *Sbox256* which has different structure from Sbox128 and depends on two keys ($R1_{t-1}$ and $R2_{t-1}$). Both of the proposed algorithms are similar in the use of Twofish block cipher as initialization phase with using all the same key length (128 to 256 bit), IV length (128 bit), LFSR length, FSM functions and output transformation.

Snowfish uses the same length of LFSR which is used in Sosemanuk because it has good property in defeating time memory data trade-off attacks [1].

Snowfish uses the same FSM functions which are used in Sosemanuk because the *Trans* function and the *mux* operation have good properties which are to avoid the existence of a linear relation between the least-significant bits of the inputs of *Trans* and the output of the FSM and also aim at increasing the complexity of fast correlation and algebraic attacks [1].

2. Twofish and Derivatives:

From Twofish , three primitives are defined called *Sbox128, Sbox256* and *twofish16*.

- **Snowfish's s-box**

Snowfish uses key dependent s-boxes which are more non-linear than *serpent1* (s_2) of Sosemanuk. In this paper two designs are proposed for the s-boxes:

Design of Sbox128

Snowfish 1 uses four different, bijective, key-dependent, 8-by-8-bit s-boxes which are the same as the s-box with 128 bit key size in twofish's round function with simple difference and will be called *Sbox128*. This difference is obtained after many tries to get more random keystream sequence. Snowfish's s-box uses 32 bit s_{t+9} as input with the use one key $R1_{t-1}$ and its output of 32 bits f_t. It is different from the design of s-box with 128 bit key size in Twofish's round function in the use of key dependent and the output as shown in Figure (1). Sbox128 uses one key rather than the use of two keys as in the s-boxes of Twofish's round function; also Sbox128 outputs 32 bit after performing q-permutation operations of four s-boxes and one XOR operation as shown in Figure (1), then it can produce one word

f_t by the following equations (1), where each $f_{t,i}$ is 8 bit:

$$\left.\begin{aligned} f_{t,0} &= q_1[q_0[q_0[s_{t+9,0}]]\oplus R1_{t-1,0}] \\ f_{t,1} &= q_0[q_0[q_1[s_{t+9,1}]]\oplus R1_{t-1,1}] \\ f_{t,2} &= q_1[q_1[q_0[s_{t+9,2}]]\oplus R1_{t-1,2}] \\ f_{t,3} &= q_0[q_1[q_1[s_{t+9,3}]]\oplus R1_{t-1,3}] \end{aligned}\right\} \quad \ldots\ldots\ldots\ldots 1$$

Design of Sbox256

Snowfish 2 uses four different, bijective, key-dependent, 8-by-8-bit S-boxes which are the same as the s-box with 256 bit key size in Twofish's round function with simple difference and will be called *Sbox256*. This difference is obtained after many tries to get more random keystream sequence compared with Sbox128. Snowfish's s-box uses 32 bit s_{t+9} as input with the use of two keys $R2_{t-1}$ and $R1_{t-1}$, and it outputs 128 bit $f_{t+3}, f_{t+2}, f_{t+1}, f_t$. It is different from the design of the s-box with 256 bit key size in Twofish's round function in the use of key dependent and the output as shown in Figure (2). Sbox256 uses two keys rather than the use of four keys as in s-boxes of Twofish's round function; also the Sbox256 outputs 128 bit after performing q-permutation operations and two XOR operations as shown in Figure (2), then it can produce four words $f_{t+3}, f_{t+2}, f_{t+1}, f_t$ by the following equations (2) where each f_{t+i} is 32 bit:

$$\left.\begin{aligned} &\text{To compute} \Rightarrow f_t \\ &\quad f_{t,0}(s_{t+9,0}) = q_1[s_{t+9,0}]\oplus R2_{t-1,0} \\ &\quad f_{t,1}(s_{t+9,1}) = q_0[s_{t+9,1}]\oplus R2_{t-1,1} \\ &\quad f_{t,2}(s_{t+9,2}) = q_0[s_{t+9,2}]\oplus R2_{t-1,2} \\ &\quad f_{t,3}(s_{t+9,3}) = q_1[s_{t+9,3}]\oplus R2_{t-1,3} \\ &\text{To compute} \Rightarrow f_{t+1} \\ &\quad f_{t+1,0}(f_{t,0}) = q_1[f_{t,0}] \\ &\quad f_{t+1,1}(f_{t,1}) = q_1[f_{t,1}] \\ &\quad f_{t+1,2}(f_{t,2}) = q_0[f_{t,2}] \\ &\quad f_{t+1,3}(f_{t,3}) = q_0[f_{t,3}] \\ &\text{To compute} \Rightarrow f_{t+2} \\ &\quad f_{t+2,0}(f_{t+1,0}) = q_0[f_{t+1,0}] \\ &\quad f_{t+2,1}(f_{t+1,1}) = q_1[f_{t+1,1}] \\ &\quad f_{t+2,2}(f_{t+1,2}) = q_0[f_{t+1,2}] \\ &\quad f_{t+2,3}(f_{t+1,3}) = q_1[f_{t+1,3}] \\ &\text{To compute} \Rightarrow f_{t+3} \\ &\quad f_{t+3,0}(f_{t+2,0}) = q_0[f_{t+2,0}]\oplus R1_{t-1,0} \\ &\quad f_{t+3,1}(f_{t+2,1}) = q_0[f_{t+2,1}]\oplus R1_{t-1,1} \\ &\quad f_{t+3,2}(f_{t+2,2}) = q_1[f_{t+2,2}]\oplus R1_{t-1,2} \\ &\quad f_{t+3,3}(f_{t+2,3}) = q_1[f_{t+2,3}]\oplus R1_{t-1,3} \end{aligned}\right\} \quad \ldots\ldots\ldots\ldots 2$$

- **Choice of the block cipher**.

The block cipher used in the IV setup is derived from Twofish for the following reasons compared with other AES block ciphers [2, 3]:

1. Performance as a function of key length
2. Software performance
3. Key setup plus encryption
4. Randomness.

Design of Twofish16 :

Since there is no attack of differential and linear on Twofish thus the output of three rounds is chosen freely; and then the outputs of round 8^{th} (4^{th} cycle), 12^{th} (6^{th} cycle), and 16^{th} (8^{th} cycle) are chosen based on Twofish's cycles to increase the security point.

3. Key Initialization and IV Injection

Snowfish needs key initialization process and IV injection of Twofish block cipher:

- *Key Schedule*

The key setup corresponds to the *Twofish16's* key schedule which is the same as the twofish's key schedule, which produces 40 32-bit subkeys rather than 100 32-bit in *serpent24*. Note that it is known previously that Twofish's key schedule produces at first 8 words subkey for whitening step and then produces 32 words subkey for round function step.

Twofish accepts any key length from 128 bits to 256 bits; hence, Snowfish may work with exactly the same keys.

- *IV Injection*

The IV is a 128-bit value. It is used as input to the *twofish16* block cipher, and operates in the same time with the key schedule by using round subkeys. *Twofish16* consists of 16 rounds and the outputs of the 8^{th} , 12^{th} and 16^{th} rounds are used. The outputs are denoted as follows:

$(Y_3^8, Y_2^8, Y_1^8, Y_0^8)$: Output of the 8^{th} round;

$(Y_3^{12}, Y_2^{12}, Y_1^{12}, Y_0^{12})$: Output of the 12^{th} round;

$(Y_3^{16}, Y_2^{16}, Y_1^{16}, Y_0^{16})$: Output of the 16^{th} round;

The output of each round consists of the four 32-bit. These values are used to initialize the Snowfish internal states with the following values as the equations (3) and see Figure (3) of the proposed 1 and (4) of the proposed 2:

$$\left.\begin{aligned}(s_7, s_8, s_9, s_{10}) &= (Y_3^8, Y_2^8, Y_1^8, Y_0^8)\\ (s_5, s_6) &= (Y_1^{12}, Y_3^{12})\\ (s_1, s_2, s_3, s_4) &= (Y_3^{16}, Y_2^{16}, Y_1^{16}, Y_0^{16})\\ R1_0 &= Y_0^{12}\\ R2_0 &= Y_2^{12}\end{aligned}\right\} \quad \ldots\ldots\ldots 3$$

4. Output Transformation

The output transformation process in Snowfish is similar to Sosemanuk by using the XOR operation between the output of LFSR and s-box. But the difference is in the use of s-box with its inputs and outputs, this cause an influence on output transformation process:

- *In case Sbox128*

The Sbox128 operates four loops in each time. In each time the four words outputs f_t of the $Sbox128(\times 4)$ are grouped and then are combined by XOR with the corresponding dropped values from the LFSR s_t to produce the output values z_t .

$$(z_{t+3}, z_{t+2}, z_{t+1}, z_t) = (f_{t+3}, f_{t+2}, f_{t+1}, f_t) \oplus (s_{t+3}, s_{t+2}, s_{t+1}, s_t) \quad \ldots\ldots 4$$

Figure (3) shows an overview of Snowfish 1.

- *In case Sbox256*

The Sbox256 is performing after four loops in each time. The four words f_t which output from the Sbox256 are grouped and then are combined by XOR with the

corresponding dropped values from the LFSR s_t, to produce the output values z_t as shown previously in Equation (4). Figure (4) gives an overview of Snowfish 2.

The following properties have also been taken into account in the choice of output transformation.

- Both nonlinear mixing operations involved in Snowfish (the *Trans* operation and the *s-box)* do not provide any correlation probability or linear property on the least significant bits that could be used to mount an attack.
- From an algebraic point of view, those operations are combined to produce nonlinear equations.
- No linear relation can be directly exploited on the least significant bit of the values $f_t, f_{t+1}, f_{t+2}, f_{t+3}$.

5. The Algorithm of snowfish stream cipher

This section will introduce the algorithm of the two proposals which have similar design of the input, key schedule phase, IV injection phase and encryption or decryption phase but they are different in keystream generation phase depending on the used Sbox. The figure (5) shows a flowchart of the algorithm in case keystream generation phase by using Sbox128 and figure (6) shows a flowchart of algorithm in case keystream generation phase by using Sbox256:

Input :

- The key with length (128 to 256 bits).
- The IV with length (128 bits).
- Plaintext or ciphertext

output :

- The keystream with length 128 bits for each time of period.

Key schedule phase:

Step1: The key is variable as described by the Twofish specification. Check the key if it is less than 128 or 192 or 256 to pad with zero.

Step2: Generate The Key-dependent S-boxes of *h* and *g* functions

Step3: The 8 whitening subkeys of *Twofish16* each with 32 bit are generated.

IV injection phase:

Step1: The 128-bit IV, transformed into four 32-bit words, in the (I_3, I_2, I_1, I_0) order; which is input to *Twofish16* with generate 32 round subkeys of *Twofish16* each with 32 bit.

Step2: The three outputs of round (8^{th}, 12^{th}, 16^{th}) are needed to initialize as equations (3):

1. The initial LFSR state (s_1 to s_{10}, in that order).

2. The initial FSM state ($R1_0$ and $R2_0$).

Keystream generation phase:

ü ***In case sbox128 (proposal 1) :***

Step1: Repeat ten* times (*t*) the following data:

1. Repeat the following four loops:

a. The new FSM state ($R1_t$ and $R2_t$).

b. The new LFSR state after the update (the dropped value s_t is also output).

* **Using ten times is enough for test vector**

c. The *sbox128* input as equation (1).

d. The *sbox128* output f_t as equation (1).

2. Apply output transformation between output of $sbox128(\times 4)$ and the corresponding dropped s_t as equation (4).

3. The keystream with length (128 bit) is output.

Step2: The total keystream output (1280 bits).

ü ***In case sbox256 (proposal 2)***

Step1: Repeat ten† times (*t*) the following data:

1. Repeat the following four loops:

a. The new FSM state ($R1_t$ and $R2_t$).

b. The new LFSR state after the update (the dropped value s_t is also output).

2. The *sbox256* input as equation (2).

3. The *sbox256* output ($f_{t+3}, f_{t+2}, f_{t+1}, f_t$) as equation (2).

4. Apply output transformation between output of *sbox256* and the corresponding dropped s_t as equation (4).

5. The keystream with length (128 bit) is output.

Step2: The total keystream output (1280 bits).

Encryption or Decryption phase:

- In case encryption : plaintext ⊕ keystream = ciphertext
- In case decryption : ciphertext ⊕ keystream = plaintext

† **Using ten times is enough for test vector**

End phase

6. Strengths and Advantages of Snowfish

The design of new synchronous stream cipher Snowfish is based on the Sosemanuk design and on the improvements of it. The strengths and advantages are from the following security points of view:

- Snowfish makes the high nonlinear correlation between the initial states also it avoids some potential weaknesses as the guess and determine attack proposed in [4] due to the particular use of $sbox128(\times 4)$ or *sbox256* rather than *serpent1*
- The chosen LFSR is designed to eliminate all potential weaknesses (particular decimation properties, linear relations...) [1].
- The mappings used in the Finite State Machine have been carefully designed in the following way [1].
  * The *Trans* function guarantees good properties of confusion and diffusion for a low cost in software. Moreover, this mapping prevents Snowfish from algebraic attacks.
  * The *mux* operation, that could be efficiently implemented, protects Snowfish from fast correlation attacks and algebraic attacks.

7. Apply Statistical Tests on the Keystream Bits

In this paper, the two statistical tests are applied to the generated keystream and these tests are the randomness test and the structural test.

7.1 The Randomness Tests

The randomness property of the stream cipher system is analyzed by using the statistical tests: Frequency, Serial, Poker, Runs and Autocorrelation tests. Different key sizes are used in these tests. The following section shows the results of test for three systems when different keys in size between 256 and 128 bit are applied with time from 1 to 10. The results show that the two proposals are more random than Sosemanuk cipher when they have values smaller and near to chi-square for each test and have less fail status in most results. Also they show that the proposal 2 is better than proposal 1 as shown in tables (1, 2, 3, 4, 5 and 6) because it has Sbox256 with more q-permutations and two key dependent sbox.

The following two examples are applied and the tables of each one show the results of randomness test on the result keystream with length (n) 128, 256 and 512 bits. Of these examples are:

Example 1 :

Key : " the university of technology "
IV : "computersecurity"

Example 2 :

Key : "informationsec "
IV : "computersecurity"

7.2 The Structural Tests

The structural property of the stream cipher system is analyzed by using the statistical tests: Key/Keystream correlation and IV/Keystream correlation tests. The same 100 key and 100 IV are used with size 128 bits which are generated randomly and use the same IV and key is "(theuniversity)" in Key/Keystream correlation and IV/Keystream correlation tests respectively. The table (7) shows the results of test of three systems. The results show that the two proposals are good and have the nearest values between (2 and 3) in two tests but in Sosemanuk is not good because it has 8.64 near to $X^{4}_{0.05}$ (9.488) in IV/Keysream Correlation.

For more, the Sosemanuk is good in Key/Keysream Correlation and not good in IV/Keysream Correlation;

The two proposals are good and close to Sosemanuk in Key/Keysream Correlation and are good in IV/Keysream Correlation and are better than Sosemanuk.

8. Time Requirement

Table (7) shows that the time of two proposed system has the same range of time compared to with the Sosemanuk such that the two proposals have efficiency in the speed. Note that this time is computed from begin to end of algorithm.

9. Measuring the Complexity

The strength of a cipher is determined by the computational complexity of the algorithms used to solve the cipher. The strength of an algorithm is measured by O(n) which means computing time of linear form.

Table (7) shows the results of computing time of two proposed system with Sosemanuk of the keystream generation phase only of algorithm. The result shows that the complexity of the proposed 1 increased by 4 and of the proposed 2 decreased by 2 compared with the keystream generation phase in sosemanuk.

10. Conclusions

1. The use of Twofish block cipher in key initialization and IV injection to initialize the states of LFSR is efficient because the speed, security and randomness of initialization are increase.

2. The use of key dependent s-box of Twofish with some modification on it such as building block in the proposed cipher rather than static s-box (s_2) of serpent is efficient because the nonlinear degree, randomness, and security of stream cipher are increased.

3. The use of LFSR with states which each one has length $q = 2^W$ is better than of $q = 2$ to make the length of LFSR appropriate to defeat time-memory data attacks, and at the same time let the period large where $T = q^l - 1$ when the feedback polynomial is primitive.

4. The LFSR is better than NLFSR when the low linear complexity of it can be destroyed by adding building blocks (FSM and key dependent S-boxes).

5. The two proposals are more random than Sosemanuk cipher. The proposal 2 is better than proposal 1 because it has Sbox256 with more q-permutations and two key dependent sbox. The two proposals have good structure of algorithm rather than Sosemanuk. The two proposals realize the speed and complexity of algorithm by benefiting from Twofish block cipher and its key dependent Sbox.

11. References

[1] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin and H. Sibert, " Sosemanuk, A Fast Software-Oriented Stream Cipher ", 2005.

[2] Bruce S. & Doug W., "A Performance Comparison of the five AES Finalists", 2000.

[3] Juan S. & Lawrence B., "Randomness Testing of the Advanced Encryption Standard Finalist Candidates", 2000.

[4] Hadi Ahmadi , Taraneh Eghlidos , Shahram Khazaei, "Improved Guess and Determine Attack on SOSEMANUK " , Tehran, Iran, 2006.

Table (1) The randomness test results of example 1 (n=128)

The system	Tests / Time no.	Frequency test	Serial test	Poker test	Run test	Autocorrelation test
Sosemanuk	1	1.53 Pass	1.20 Pass	6.11 Pass	12.04 pass	0.25 Pass
Snow fish (1st)		2.53 pass	3.54 pass	4.21 pass	13.66 pass	-0.25 pass
Snow fish (2nd)		0 pass	0.21 pass	2.18 pass	10.49 pass	1 pass
Sosemanuk	2	0.5 pass	2.799 pass	6.11 pass	31.04 fail	-2 pass
Snow fish (1st)		0.5 pass	1.60 pass	8.55 pass	9.96 pass	-0.5 pass
Snow fish (2nd)		0.5 pass	0.78 pass	5.70 pass	13.58 pass	-0.5 pass
Sosemanuk	3	1.53 pass	1.64 pass	7.33 pass	12.01 pass	-0.25 pass
Snow fish (1st)		0.5 pass	2.295 pass	8.142 pass	6.81 pass	0.5 pass
Snow fish (2nd)		2 pass	2.24 pass	4.21 pass	19.38 Pass	-2 pass
Sosemanuk	4	0.28 pass	0.37 pass	2.45 pass	14.96 pass	-1.25 pass
Snow fish (1st)		0.78 pass	1.195 pass	7.46 pass	9.21 pass	-0.25 pass
Snow fish (2nd)		0.5 pass	0.34 pass	7.46 pass	5.83 pass	0.5 pass
Sosemanuk	5	0.13 Pass	0.40 pass	6.11 pass	7.9997 pass	1 pass
Snow fish (1st)		0.03 pass	2.89 pass	6.92 pass	6.33 pass	0.75 pass
Snow fish (2nd)		0.28 pass	1.88 pass	5.03 pass	10.14 pass	0.75 pass
Sosemanuk	6	0 pass	2.86 pass	7.06 pass	24.01 fail	2.5 fail
Snow fish (1st)		0.13 pass	0.78 pass	2.32 pass	21.29 Pass	-0.5 pass
Snow fish (2nd)		0.13 pass	0.15 pass	8.55 pass	8.66 pass	1 pass
Sosemanuk	7	0.28 pass	2.07 pass	4.48 pass	9.18 pass	-0.25 pass
Snow fish (1st)		0.03 pass	2.89 pass	1.50 pass	11.41 pass	2.25 fail
Snow fish (2nd)		0.78 pass	3.97 pass	2.18 pass	15.73 pass	0.75 pass
Sosemanuk	8	**0 pass**	**1.79 pass**	**3.13 pass**	**8.82 pass**	**0 pass**
Snow fish (1st)		0.78 pass	1.13 pass	2.05 pass	10.41 pass	-1.75 pass
Snow fish (2nd)		0.28 pass	0.94 pass	9.09 pass	22.29 fail	-2.75 pass
Sosemanuk	9	0.5 pass	1.16 pass	1.64 pass	16.72 pass	-2.5 pass
Snow fish (1st)		0.3 pass	1.07 pass	2.05 pass	11.89 pass	-0.75 pass
Snow fish (2nd)		1.13 pass	1.17 pass	8.95 pass	15.01 pass	-1 pass
Sosemanuk	10	0.28 Pass	0.37 pass	2.05 pass	15.80 pass	-1.25 pass
Snow fish (1st)		1.53 Pass	1.77 pass	2.18 pass	12.95 pass	-0.25 pass
Snow fish (2nd)		0.28 pass	0.94 pass	3.67 pass	12.68 pass	-0.75 pass

Table (2) The randomness test results of example 1 (n=256)

The system	Tests / Time no.	Frequency test	Serial test	Poker test	Run test	Autocorrelation test
Sosemanuk	1-2	0.06 pass	1.15 pass	14.6 pass	24.46 fail	0.35 pass
Snow fish (1st)		0.57 pass	0.68 pass	6.83 pass	10.67 pass	0.71 pass
Snow fish (2nd)		0.25 pass	0.33 pass	7.69 pass	12.43 pass	-0.35 pass
Sosemanuk	3-4	1.28 pass	1.20 pass	6.74 pass	16.33 pass	-0.35 pass
Snow fish (1st)		1.33 pass	1.98 pass	9.46 pass	10.66 pass	-1.77 pass
Snow fish (2nd)		1.91 pass	1.68 pass	3.43 pass	18.06 pass	-0.71 pass
Sosemanuk	5-6	0.06 pass	2.95 pass	7.46 pass	16.45 pass	-0.35 pass
Snow fish (1st)		0.26 pass	0.47 pass	31.43 pass	15.65 pass	-0.71 pass
Snow fish (2nd)		0.02 pass	1.13 pass	6.17 pass	11.47 pass	-0.71 pass
Sosemanuk	7-8	0.06 pass	3.93 pass	1.86 pass	14.13 pass	1.77 pass
Snow fish (1st)		0.25 pass	3.27 pass	5.11 pass	11.55 pass	-0.18 pass
Snow fish (2nd)		0.77 pass	1.45 pass	9.26 pass	19.01 pass	0.71 pass
Sosemanuk	9-10	0 pass	0.11 pass	4.14 pass	16.72 pass	-0.71 pass
Snow fish (1st)		0.14 pass	1.04 pass	7.26 pass	17.63 pass	-0.35 pass
Snow fish (2nd)		1.57 pass	2.07 pass	6.31 pass	15.16 pass	-0.53 pass

Table (3) The randomness test results of example 1 (n=512)

The system	Tests / Time no.	Frequency test	Serial test	Poker test	Run test	Autocorrelation test
Sosemanuk	1-4	0.96 pass	1.12 pass	16.38 pass	21.76 fail	-0.88 pass
Snow fish (1st)		1.34 pass	2.27 pass	6.44 pass	12.69 pass	1.5 pass
Snow fish (2nd)		1.78 pass	1.57 pass	15.76 pass	11.03 pass	-1.88 pass
Sosemanuk	5-8	0.08 pass	0.14 pass	13.88 pass	7.94 pass	-0.13 pass
Snow fish (1st)		0.28 pass	3.00 pass	4.70 pass	17.13 pass	-0.75 pass
Snow fish (2nd)		0.64 pass	0.66 pass	5.10 pass	9.26 pass	0 pass

Table (4) the randomness test results of example 2 (n=128)

The system	Tests / Time no.	Frequency test	Serial test	Poker test	Run test	Autocorrelation test
Sosemanuk	1	fail	fail	14.24 pass	21.70 fail	-1.5 pass
Snow fish (1st)		0.5 pass	0.66 pass	4.08 pass	14.63 pass	1 pass
Snow fish (2nd)		0 pass	0.21 pass	2.18 pass	10.49 pass	1 pass
Sosemanuk	2	0.5 pass	3.996 pass	9.50 Pass	34.77 fail	1.5 pass
Snow fish (1st)		0.781 pass	1.13 pass	3.67 pass	17.80 fail	0.25 pass
Snow fish (2nd)		0.5 pass	0.78 pass	5.70 pass	13.58 pass	-0.5 pass
Sosemanuk	3	1.13 pass	2.49 pass	5.03 pass	18.46 pass	0 pass
Snow fish (1st)		0.03 pass	0.24 pass	4.49 pass	15.10 Pass	0.75 pass
Snow fish (2nd)		2 pass	2.24 pass	4.21 pass	19.38 Pass	-2 pass
Sosemanuk	4	0.03 pass	1.82 pass	2.86 pass	11.40 pass	-0.25 pass
Snow fish (1st)		1.13 pass	2.55 pass	10.99 pass	24.26 fail	-1 pass
Snow fish (2nd)		0.5 pass	0.34 pass	7.46 pass	5.83 pass	0.5 pass
Sosemanuk	5	1.13 pass	1.92 pass	5.03 pass	10.07 pass	1 pass
Snow fish (1st)		0.5 pass	0.59 pass	3.94 pass	10.32 pass	0.5 pass
Snow fish (2nd)		0.28 pass	1.88 pass	5.03 pass	10.14 pass	0.75 pass
Sosemanuk	6	0.28 pass	0.31 pass	9.90 pass	11.97 pass	-0.75 pass
Snow fish (1st)		2.53 pass	2.60 pass	7.74 pass	18.498 pass	0.75 pass
Snow fish (2nd)		0.13 pass	0.15 pass	8.55 pass	8.66 pass	1 pass
Sosemanuk	7	0.13 pass	1.60 pass	4.48 pass	9.51 pass	1 pass
Snow fish (1st)		2.53 pass	3.16 pass	6.11 pass	14.33 pass	-0.75 pass
Snow fish (2nd)		0.78 pass	3.97 pass	2.18 pass	15.73 pass	0.75 pass
Sosemanuk	8	**0.13 pass**	**1.47 pass**	**6.92 pass**	**14.87 pass**	**1.5 pass**
Snow fish (1st)		1.13 pass	1.80 pass	8.68 pass	28.91 fail	0 pass
Snow fish (2nd)		0.28 pass	0.94 pass	9.09 pass	22.29 fail	-2.75 pass
Sosemanuk	9	1.53 pass	3.34 pass	5.30 pass	11.24 pass	0.25 pass
Snow fish (1st)		0.13 pass	0.34 pass	1.51 pass	20.22 pass	1.5 pass
Snow fish (2nd)		1.13 pass	1.17 pass	8.95 pass	15.01 pass	-1 pass
Sosemanuk	10	0.13 pass	0.34 pass	5.70 pass	31.49 fail	1.5 pass
Snow fish (1st)		0.5 pass	0.47 pass	7.06 pass	6.63 pass	1 pass
Snow fish (2nd)		0.28 pass	0.94 pass	3.67 pass	12.68 pass	-0.75 pass

Table (5) the randomness test results of example 2 (n=256)

The system	Tests / Time no.	Frequency test	Serial test	Poker test	Run test	Autocorrelation test
Sosemanuk	1-2	0.77 pass	2.53 pass	11.54 pass	548.14 fail	-0.18 pass
Snow fish (1st)		1.28 pass	1.01 pass	6.74 pass	19.91 fail	0.53 pass
Snow fish (2nd)		0 pass	1.43 pass	3.69 pass	8.53 pass	1.41 pass
Sosemanuk	3-4	0.06 pass	0.55 pass	8.31 pass	16.30 pass	0.71 pass
Snow fish (1st)		0.25 pass	1.69 pass	2.60 pass	26.68 fail	-1.41 pass
Snow fish (2nd)		fail	fail	11.17 pass	17.98 pass	0.53 pass
Sosemanuk	5-6	3.54 pass	fail	18.46 pass	36.85 fail	1.41 pass
Snow fish (1st)		2.66 pass	2.69 pass	10 pass	9.45 pass	0 pass
Snow fish (2nd)		0.57 pass	1.06 pass	3.97 pass	17.20 fail	-0.53 pass
Sosemanuk	7-8	0.39 pass	0.50 pass	6.86 pass	25.84 pass	-1.59 pass
Snow fish (1st)		3.54 pass	3.77 pass	9.31 pass	24.65 fail	-0.53 pass
Snow fish (2nd)		1.57 pass	1.69 pass	8.26 pass	7.60 pass	-0.88 pass
Sosemanuk	9-10	0.25 pass	0.93 pass	3.74 pass	15.30 pass	-1.06 pass
Snow fish (1st)		0.06 pass	0.11 pass	2.26 pass	40.74 fail	-0.18 pass
Snow fish (2nd)		0.02 pass	0.72 pass	11.6 pass	142.18 fail	-1.06 pass

Table (6) the randomness test results of example 2 (n=512)

The system	Tests / Time no.	Frequency test	Serial test	Poker test	Run test	Autocorrelation test
Sosemanuk	1-4	0.79 pass	1.20 pass	17.36 pass	274.12 fail	-0.5 pass
Snow fish (1^{st})		0.39 pass	0.84 pass	16.01 pass	31.60 fail	-1.88 pass
Snow fish (2^{nd})		3.49 pass	3.93 pass	9.52 pass	15.40 pass	-0.38 pass
Sosemanuk	5-8	2.85 pass	3.92 pass	10.54 pass	45.02 fail	0.13 pass
Snow fish (1^{st})		fail	fail	13.91 pass	13.25 pass	0 pass
Snow fish (2^{nd})		2.28 pass	2.62 pass	5.61 pass	16.21 fail	0 pass

Table (7) The results of structural tests , the execution time and the complexity

The system	Key/ Keystream Correlation	IV/ Keystream Correlation	The execution time	The complexity
Sosemanuk	1.97 good	8.64 good	2 Sec.	$4n^2 + 3n$
Snowfish (1^{st})	2.41 good	2.67 good	2 Sec.	$8n^2 + 2n$
Snowfish (2^{nd})	2.62 good	2.10 good	2 Sec.	$2n^2 + 9n$

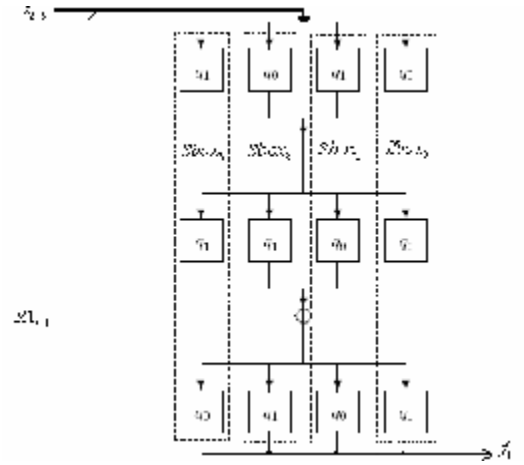

Figure (1) Overview of Sbox128

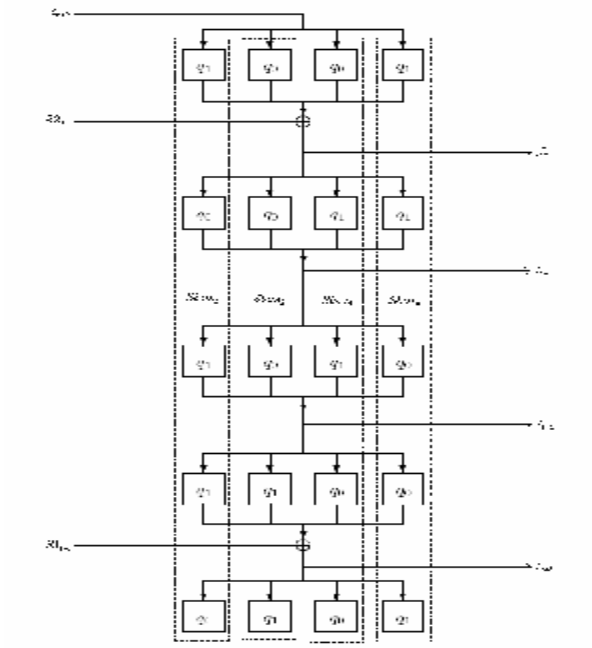

Figure (2) Overview of Sbox256

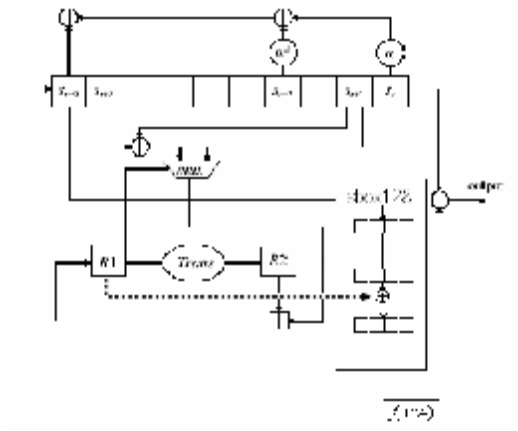

Figure (3) the proposal 1 of Snowfish

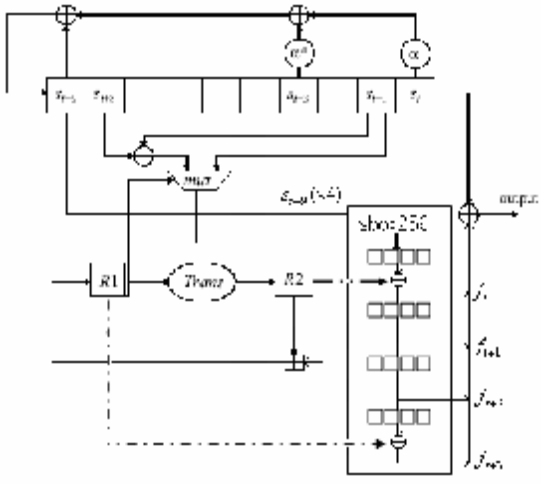

Figure (4) the proposal 2 of Snowfish

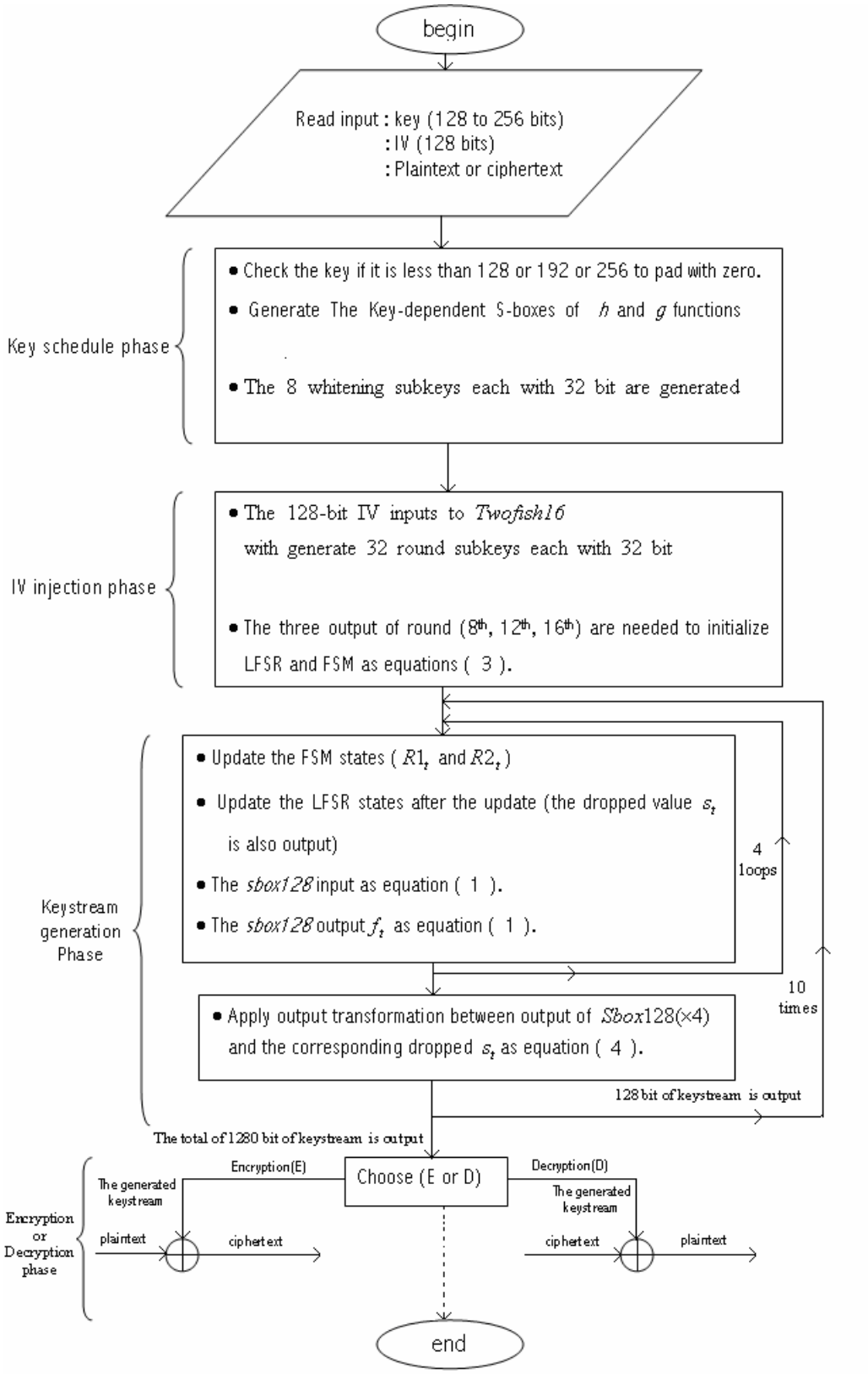

Figure (5) Flowchart of Snowfish 1 algorithm to encryption or decryption

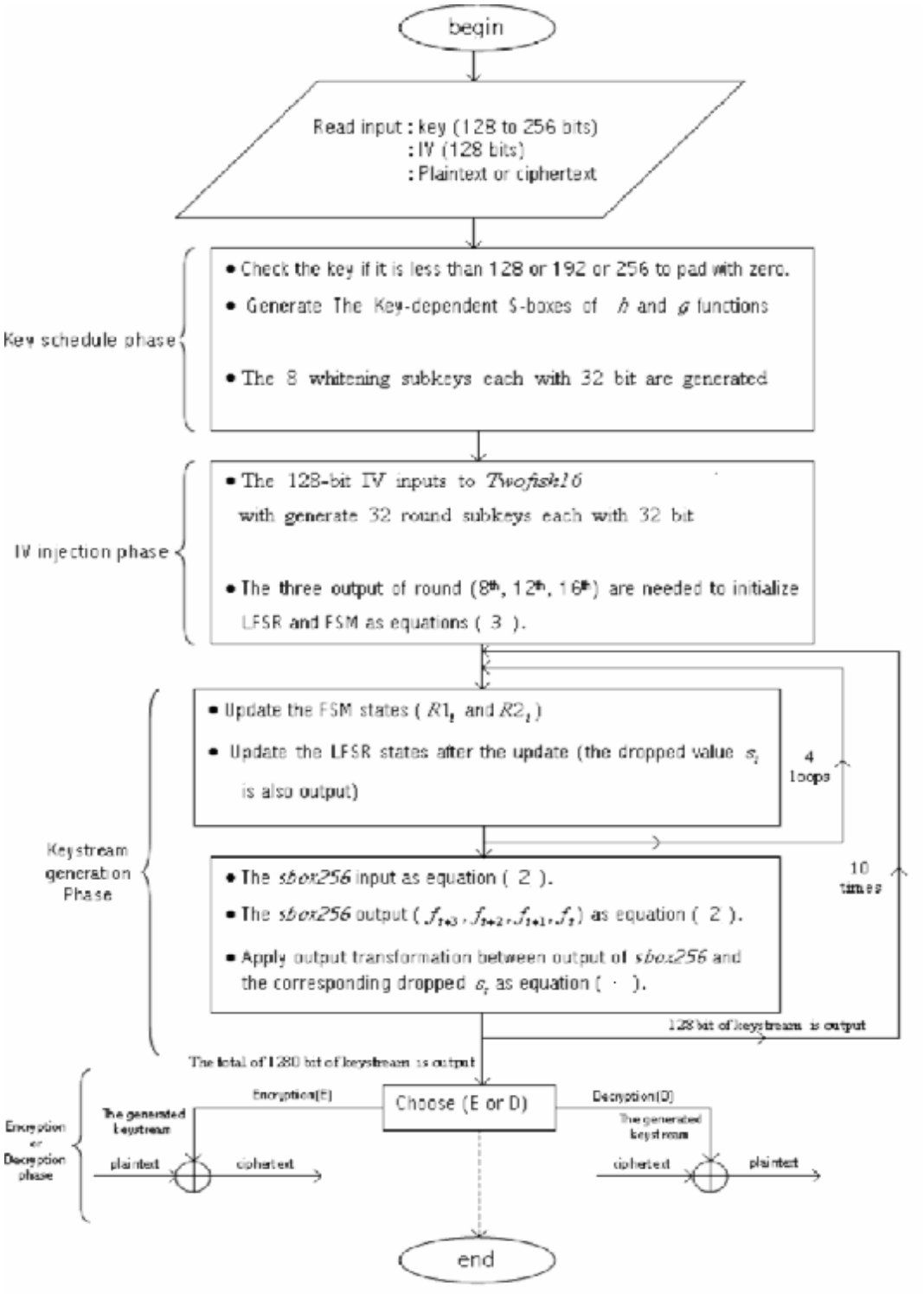

Figure (6) Flowchart of Snowfish 2 algorithm to e ncryption or decryption