2004－ＣＳＥＣ－26 (36)
2004／7／21

内部状態遷移型ストリーム暗号SSSMの提案

鵜川 三蔵† 大東 俊博† 白石 善明†† 森井 昌克†

† 徳島大学工学部知能情報工学科 〒770-8506 徳島市南常三島町 2-1
†† 近畿大学理工学部情報学科 〒577-8502 東大阪市小若江 3-4-1
E-mail: †{sanzo,ohigashi,morii}@is.tokushima-u.ac.jp, ††zenmei@info.kindai.ac.jp

あらまし 内部状態遷移型の新しい擬似乱数生成器SSSMを提案する．SSSMは32ビット演算CPUにおいて，高速に動作することができる．またSSSMは出力と更新を独立に行うため，高い安全性を有する．本稿ではSSSMのアルゴリズムと速度，そして安全性について述べる．

キーワード 擬似乱数生成器，ストリーム暗号，状態テーブル，内部状態推定法，NIST SP800-22

SSSM: A New Key Stream Generator with Time-variant Tables

Sanzo UGAWA†, Toshihiro OHIGASHI†, Yoshiaki SHIRAISHI††, and Masakatu MORII†

† Department of Information Science and Intelligent Systems, The University of Tokushima
2-1 Minamijyousanjima, Tokushima-shi, 770-8506, Japan.
†† Department of Informatics, Kinki University
3-4-1 Kowakae, Higashi-Osaka, 577-8502, Japan.
E-mail: †{sanzo,ohigashi,morii}@is.tokushima-u.ac.jp, ††zenmei@info.kindai.ac.jp

Abstract A new key stream generator with time-variant tables, named SSSM, is proposed. SSSM is high-speed and secure key stream generator operated by the unit of 32-bit. This paper gives the algorithm and analytical result of SSSM.

Key words keystream generator, stream cipher, time-variant table, internal-state reconstrunction method, NIST SP800-22

1. Introduction

The key stream generator is used in stream cipher. Many key stream generators consist of a number of possibly clocked linear feedback shift registers (LFSRs) combined by a function [1], [2]. In contrast, key stream generator with time-variant tables takes a design approach that is quite different from that of LFSR-based stream cipher. Key stream generator with time-variant tables is suitable for software implementation. There is RC4 [3] in the typical one of stream cipher using key stream generator with time-variant tables. RC4 is the most widely used stream cipher in any commercial products and standards; for example, Secure Sockets Layer standard(SSL)3.0 [4], Wi-Fi Protected Access(WPA) [5], and so on. Key stream generator with time-variant tables consists of table 2^n n-bit word and of some pointers n-bit word. The generator swaps the elements in the table and outputs n-bit word. A conventional key stream generator with time-variant tables operates by the unit of 8-bit.

In this paper, a new key stream generator with time-variant tables, named SSSM, is proposed. To make use of 32-bit operation CPU, SSSM consists of four time-variant tables 8-bit word and updates these tables at the same time. Throughput of SSSM is about twice of RC4. In addition, SSSM is secure for internal-state reconstruction because SSSM makes the table used for the update and output independent.

2. Key stream generator with time-variant tables

The key stream generator with time-variant tables updates tables and generates pseudo-random number every step. The table is updated by swapping the elements in the table. The swapped elements are selected by pointers. The pointers are also updated by the time-variant table. The time-variant table is initialized by using secret key at key scheduling part. Figure 1 shows the update of time-variant table. The key

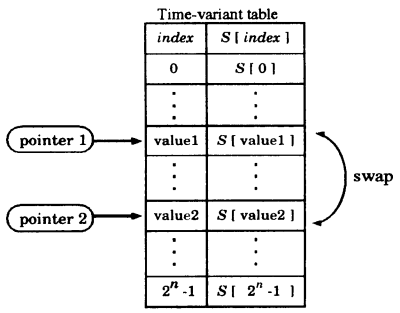

Fig. 1 update of time-variant table

stream generator with time-variant tables generates pseudo-random number with a simple operation that uses pointer and time-variant table.

2.1 RC4

RC4 [3] is a typical key stream generator of stream ciphers with time-variant table. Z_t denotes the output n-bit word of RC4 at time t. Then, the next-state and output functions of RC4 for every $t(t \geqq 1)$ are defined as

$$i_t = (i_{t-1} + 1) \bmod 2^n, \tag{1}$$

$$j_t = (j_{t-1} + S_{t-1}[i_t]) \bmod 2^n, \tag{2}$$

$$S_t[i] = \begin{cases} S_{t-1}[j_t], & i = i_t \\ S_{t-1}[i_t], & i = j_t \\ S_{t-1}[i], & i \neq i_t, j_t, \end{cases} \tag{3}$$

$$Z_t = S_t[(S_t[i_t] + S_t[j_t]) \bmod 2^n]. \tag{4}$$

RC4 uses two n-bit word pointers i_t and j_t. Two pointers i_0 and j_0 are initialized to zero. The pseudo-random number of n-bit word is generated by using updated pointer and time-variant table.

3. SSSM

SSSM is a new key stream generator with time-variant tables. SSSM consists of four time-variant tables and two pointers. Three of four time-variant tables are used to generate 24-bit pseudo-random number after update of four tables. One remaining table is used to update one pointer. Four time-variant tables is updated in common pointers.

3.1 Algorithm of SSSM

The algorithm of SSSM is described in Eqs. (5)–(8).

$$i_t = (i_{t-1} + 1) \bmod 256, \tag{5}$$

$$j_t = S_{x,t-1}[(j_{t-1} + S_{x,t-1}[i_t]) \bmod 256],$$
$$x = \lfloor t/256 \rfloor \bmod 4, \tag{6}$$

$$S_t[i] = \begin{cases} S_{t-1}[j_t], & i = i_t \\ S_{t-1}[i_t], & i = j_t \\ S_{t-1}[i], & i \neq i_t, j_t, \end{cases} \tag{7}$$

$$Z_{y,t} = S_{y,t-1}[(S_{y,t}[i_t] + S_{y,t}[j_t]) \bmod 256],$$
$$y \neq x, y \in \{0, 1, 2, 3\}. \tag{8}$$

SSSM uses four time-variant tables $S_{0,t}[i]$, $S_{1,t}[i]$, $S_{2,t}[i]$, and $S_{3,t}[i]$. $S_t[i]$ is a concatenated element of each $S_{0,t}[i]$, $S_{1,t}[i]$, $S_{2,t}[i]$, and $S_{3,t}[i]$, as shown by Eq. (9).

$$S_t[i] = S_{0,t}[i] \parallel S_{1,t}[i] \parallel S_{2,t}[i] \parallel S_{3,t}[i],$$
$$i = 0, 1, \ldots, 2^n - 1. \tag{9}$$

Two pointers i_t and j_t are used for the update of $S_t[i]$. The pointers i_0 and j_0 are initialized to zero. The pointer j_t is updated by adding element of time-variant table $S_{x,t}[i_t]$. When SSSM updates 256 times, x is updated by adding one. If $x = 4$, x is returned to zero.

3.2 SSSM Key Scheduling Algorithm

We explain SSSM Key Scheduling Algorithm. SSSM uses 128-bit key K. 128-bit key K is defined as sets of 16 8-bit element. K is shown $K[i](0 \leqq i \leqq 15)$ The time-variant tables are initialized by using the key K as follows:

for $i = 0$ *to* 255,

$S^*[i] = i$,

$i_0^* = 0$,

$j_0^* = 0$,

for $t = 0$ *to* 1023,

$i_t^* = (i_t^* + 1) \bmod 256$,

$j_t^* = S_{x,t-1}^*[(j_{t-1}^* + S_{0,t-1}^*[i_t^*] + K[i_t^* \bmod 16])]$,

$$S_t^*[i] = \begin{cases} S_{t-1}^*[j_t^*], i = i_t^* \\ S_{t-1}^*[i_t^*], i = j_t^* \\ S_{t-1}^*[i], i \neq i_t^*, j_t^*, \end{cases}$$

for $t = 1024$ *to* 2047,

$i_t^* = (i_t^* + 1) \bmod 256$,

$j_t^* = S_{0,t-1}^*[(j_{t-1}^* + S_{x,t-1}^*[i_t^*]$
$+ S_{x,t-1}^*[i_t^* \bmod 16]) \bmod 256]$,

$x = \lfloor t/256 \rfloor \bmod 4$,

$$S_{y,t}^*[i] = \begin{cases} S_{y,t-1}^*[j_t^*], i = i_t^* \\ S_{y,t-1}^*[i_t^*], i = j_t^* \\ S_{y,t-1}^*[i], i \neq i_t^*, j_t^*, \end{cases}$$

$y \neq x, y \in \{0, 1, 2, 3\}$.

$S_{i,2047}^*(0 \leqq i \leqq 3)$ is initial state $S_{i,0}$.

3.3 Update frequency of pointer in SSSM

SSSM uses four time-variant tables. These four tables we updated by two pointers at same time. In RC4, update of two pointers is needed three times to generate 24-bit pseudo-random number. On the other hand, SSSM updates two pointers only one time. The operation frequency of pointer of SSSM is 1/3 of the RC4. Figure 2 shows the four tables are updated by using two pointers.

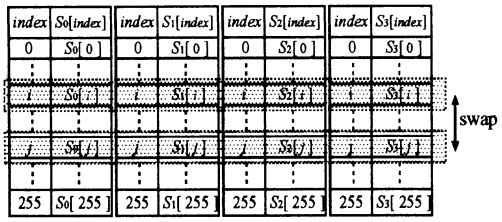

Fig. 2 Four tabless are updated by using two pointers

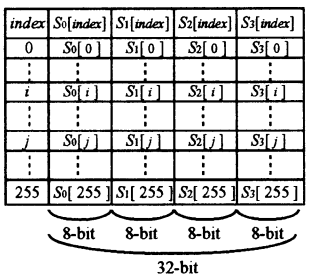

Fig. 3 Four 8-bit elements are substituted for 32-bit variables

Table 1 Evaluation platform specification

CPU	PentiumIII 1GHz
memory	512-MByte
OS	Vine Linux ver2.2.17
language	C
compiler	gcc ver2.95

3.4 32-bit operation

SSSM uses four time-variant tables $S_0[i]$, $S_1[i]$, $S_2[i]$, $S_3[i]$. Four 8-bit elements corresponding to the index are substituted for 32-bit variables. Table storing 256 32-bit variables is called $S_t[i]$. v Four table $S_0[i]$, $S_1[i]$, $S_2[i]$, and $S_3[i]$ are updated by common two pointers. It is shown that SSSM is very efficient in 32-bit operation CPU, since four tables can be updated in the same operation as one table using $S_t[i]$. This shows that SSSM is efficient in 32-bit operation CPU. Figure 3 shows four 8-bit elements are substituted for 32-bit variables.

4. Performance of SSSM and statistical test of the SSSM's output

We measured throughput of SSSM and tested NIST statistical test suite [6].

4.1 Performance of SSSM

Table 1 shows platform and language of evaluation for software implementation. Table 2 shows performance of SSSM and RC4. From the result, the performance of SSSM is about twice of RC4.

4.2 Statistical test of the SSSM's output

We tested SSSM's output sequence by the NIST statistical tests suite. This suite is called Special Publication 800-22(SP800-22) [6]. There is 16 kinds of tests in SP800-22. The test settings of Discrete Fourier Transform Test and Lempel-Ziv Compression Test of this suite are wrong [7]. According to [7], we corrected some parameters in Discrete Fourier Transform Test and used the modified tool. Lempel-Ziv Compression Test is not recommended for CRYPTREC to be put in a minimum set for test of a pseudo-random generator [12]. So, we did not adopt Lempel-Ziv Compression Test in our evaluation.. We got 1,000 1,000,000-bit outputs of SSSM for statistical test. The input parameters are listed in Table 3. As a result, these outputs passed all tests without Lempel-Ziv Compression Test.

Table 2 Performance in Mbits/sec

generator	performance
RC4	262 Mbits/sec
SSSM	556 Mbits/sec

Table 3 Parameters of SP800-22

Test Name	Block Length
Block Frequency	20,000
Non-overlapping Template Matching	9
Overlapping Template Matching	9
Universal(Initialization Steps)	7(1,280)
Serial	10
Approximate Entropy	10
Linear Complexity	500

5. Analysis of SSSM

We analyze cycles of SSSM and resistance of internal-state reconstruction method for SSSM.

5.1 Cycles of SSSM

According to [8], probability of entering a cycle not longer than X for an n-element random permutation is X/n. The cycle lengths in the output of the SSSM is scaled down to use M-element permutations. The total number of $256! \times 256^2 \times 4$ possible internal states of SSSM is determined by all possible configurations of permutation S and variables i, j and x. Probability of entering a cycle not longer than X by the SSSM is conjectured from this to be approximately $X/(256! \times 256^2 \times 4) \cong X/2^{1700}$. For example, the SSSM's output will enter a cycle not longer than 2^{850} is about $1/2^{850}$.

5.2 Internal-state reconstruction

Internal-state reconstruction method [9]~[11] is based on a tree-search algorithm with recursive process. The goal of an internal-state reconstruction is to reconstruct S_0 out of $Z_{y,t}$. In SSSM, time-variant tables to output pseudo-random number $Z_{y,t}$ and to update pointer j_t are independent. This means updated information of tables is not obtained from $Z_{y,t}$. Therefore, internal-state reconstruction cannot be used

applied to SSSM.

If pointer j_t is updated by Eq. (2), both Eqs. (10) and (11) are approved at probability $1/2^{16}$.

$$j = i + 1, \tag{10}$$

$$S_X[t+1] = 1. \tag{11}$$

It keeps approving Eqs. (10) and (11) until table for update of j_t is changed. When Eqs. (10) and (11) is approved, two pointer both i_t and j_t is specified. In SSSM, update of pointer j_t uses output function of VMPC stream cipher [13]. The function is Eq. (6). By using Eq. (6), SSSM is security for this attack.

6. Conclusions

In this paper, we proposed a new key stream generator SSSM. SSSM operates by the unit of 32-bit and improve throughput more than RC4. We plan to do other analyses to evaluate the safety of SSSM.

References

[1] R. A. Rueppel, Analysis and Design of Stream Ciphers, Springer-Verlag, Berlin, 1986.

[2] R. A. Rueppel, "Stream Ciphers," Contemporary Cryptology, G. J. Simmons, ed., pp.65–134, IEEE Press, New York, 1992.

[3] B. Schneier, Applied Cryptography, Wiley, New York, 1996.

[4] A. Freier, P. Karlton, and P. Kocher, "The SSL 3.0 protocol," Netscape Communications, Nov. 18, 1996.

[5] Wi-Fi Alliance,"Wi-Fi Protected Access", *available at http://www.weca.net/opensection/protected_access.asp*

[6] NIST, Special Publication 800-22, "A STATISTICAL TEST SUITE FOR RANDOM AND PSEUDORANDOM NUMBER GENERATORS FOR CRYPTOGRAPHICAPPLICATIONS", May 15, 2001.

[7] S. Kim, K. Umeno, and A. Hasegawa, "On the NIST Statistical Test Suite for Randomness",Tech. Report of IEICE, ISEC2003-87, pp.21-28, Dec. 2003.

[8] D. E. Knuth, "The Art of Computer Programming", vol.1. Fundamental Algorithms, Third Edition, Addison Wesley Longman, 1997.

[9] Y. Shiraishi, T. Ohigashi, and M. Morii, "Internal-State Reconstruction of a Stream Cipher RC4," IEICE Trans. on Fundamentals , vol.E86-A, no.10, pp.2636–2638, Oct. 2003.

[10] L.R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege, "Analysis Methods for (Alleged) RC4," Proc. of ASIACRYPT '98, Lecture Notes in Computer Science, vol. 1514, pp. 327–341, Springer-Verlag, 1998.

[11] J. Dj. Golić, "Iterative probabilistic cryptanalysis of RC4 keystream generator," Proc. ACISP2000, Lecture Notes in Computer Science, vol.1841, pp.220–233, Springer-Verlag, 2000.

[12] T. Kaneko, *http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/ documents/rep_ID0206.pdf*, in Japanese.

[13] B. Zoltak, "VMPC Stream Cipher", *available at http://www.vmpcfunction.com/cipher.html*