圧縮関数 ABSG を用いた高速乱数生成器の開発

仲神 秀彦　　森井 昌克

神戸大学大学院工学研究科
657-8501 神戸市灘区六甲台町 1-1

nakagami@stu.kobe-u.ac.jp, mmorii@kobe-u.ac.jp

あらまし 圧縮関数は入力系列からその部分系列を出力することで擬似乱数系列を生成する関数であり，以前から安全性に関する様々な考察が行われている．圧縮関数 ABSG は eSTREAM 提案暗号である DECIMv2 に実装されており，その安全性を保障している．しかし，圧縮関数 ABSG はその構造上，乱数生成速度に問題があると指摘されている．本稿では圧縮関数 ABSG の内部構造を利用した新たな高速乱数生成器の提案を行う．

Development of Fast Random Number Generator using Compression Function ABSG

Hidehiko Nakagami　　Masakatu Morii

Graduate School of Engineering, Kobe University
1-1 Rokkodai-Cho Nada-Ku Kobe-Shi 657-8501 Japan

nakagami@stu.kobe-u.ac.jp, mmorii@kobe-u.ac.jp

Abstract A compression function generates random numbers outputting a part of an input sequence, and the security has been studied by many researchers. A compression function ABSG is implemented in DECIMv2 that submitted to the ECRYPT stream cipher project (eSTREAM) and it guarantees the security. In the paper, we propose a fast random number generator using the internal structure of the compression function ABSG.

1 Introduction

A compression function generates random numbers outputting a part of an input sequence. In 1993, Shrinking Generator[1] which was designed by D. Coppersmith et al. is composed of two linear feedback shift registers (LFSR), one decides the output values of the other. In 1994, a simplified Shrinking Generator called Self-Shrinking Generator (SSG)[2] was designed by W. Meier et al.

One of the problems in those compression functions is the low speed of the generation of random numbers. In order to improve the speed, A. Gouget presented a new compression function called BSG in 2004[3], but it contained a weakness. Soon after the weakness was found, revised versions called ABSG and MBSG were proposed which speed were equal to BSG[4].

A compression function is used in a pseudo-random number generator and a stream cipher, and it increases the difficulty to recover the input sequence from the output one. For example, the compression function ABSG is implemented in DECIMv2[5] that has submitted to the ECRYPT stream cipher project (eSTREAM)[6]. The introduction of ABSG in a stream cipher increases the security, and it attracts many reseachers to evaluate the structure. Regretfully, it was pointed out that the generation of random numbers is slower than

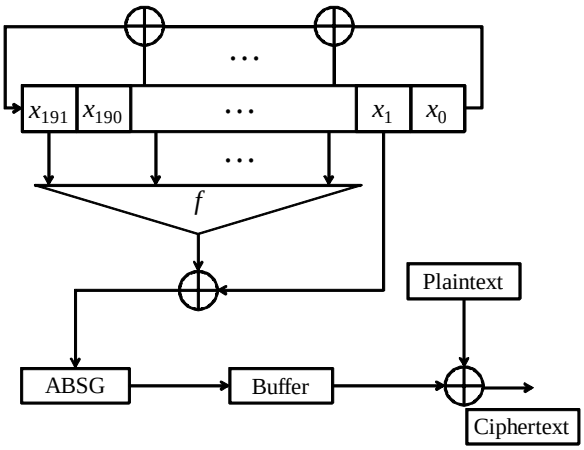

Figure 1: Structure of DECIMv2.

that of the other stream ciphers submitted the eSTREAM [7].

In this paper, we propose a fast random number generator using the compression function ABSG implemented in the stream cipher DECIMv2. We discuss about the speed of generating output sequence and the vulnerability reported in [8] that an attacker can recover the input sequence to compression function ABSG from output sequence if he knows the intervals of output bits in the input sequence. The compression function ABSG has not been cracked because it is difficult to decide the intervals of output bits in input sequence. Our random number generator consists of a LFSR and a function that utilizes the internal structure of ABSG.

2 Compression Function

A compression function is used in the pseudo-random number generator and the stream cipher, and it generates random numbers to output a part of an input sequences. In 2005, the compression function ABSG was designed by A. Gouget et al. It can generate random numbers with the same speed as BSG and it overcomes the weakness of BSG[4]. ABSG is implemented in the stream cipher DECIMv2, Figure 1 shows the structure of DECIMv2[5]. Table 1 shows the algorithm of the ABSG.

As shown in Table 2, the ABSG algorithm outputs a random sequence $(z_0,\ z_1, \ldots)$ using

Table 1: ABSG Algorithm.

Input: $(y_0,\ y_1, \ldots)$
Output: $(z_0,\ z_1, \ldots)$
Set: $i \leftarrow 0;\ j \leftarrow 0;$
Repeat the following steps:
1. $e \leftarrow y_i;\ z_j \leftarrow y_{i+1};$
2. $i \leftarrow i + 1;$
3. while$(y_i = \overline{e})\ i \leftarrow i + 1;$
4. $i \leftarrow i + 1;$
5. $j \leftarrow j + 1;$

an input one $(y_0,\ y_1, \ldots)$ by changing the internal states denoted by e, i, and j. Now, we focus on the behavior of a pointer e to explain the process of ABSG algorithm. The compression function ABSG selects a part of input sequence by the pointer e to output a random sequence, and the pointer e is incremented by a specific input value y_i pointed by i. From the step 3 in Table 2, we can see that the incrementation of i is also ruled by e. The non-linear relation between e and i makes it difficult to recover the input sequence from the output one in ABSG. Referring to [4], it is remarkable that the output rate per input sequence is approximated to 1/3.

3 Analysis of Compression Function ABSG

A compression function ABSG can generate an output sequence faster than SSG, and more secure than BSG. However, some problems is reported. In this section, we consider some problems in ABSG.

3.1 Speed of Generating Random Number

When a pseudo-random number generated by a compression function, the number of bits produced per unit of time depends on the output rate. The more the rate increases, the more output sequence can be gained and generating speed of random number is advanced.

The compression function ABSG decimates most part of the input sequence after the out-

put sequence has been generated because the output rate per input sequence of ABSG is approximated to 1/3. To generate a random number rapidly, it is necessary to increase the rate without degrading the difficulty to recover the input sequence from the output one in ABSG.

3.2 Recovery of Input Sequence

In this section, we show that an input sequence can be recovered from the output sequence of the compression function ABSG under the condition that the intervals of the selected input sequence as the output ones z_t are given[8].

The interval between z_t and z_{t+1} in the input sequence is denoted by $d(t, t+1)$ ($d(t, t+1) \geq 2$ because the pointer e can not move to the next bit). Suppose that an output bit z_t at a time t is selected from an input bit $y_{t'}$ at a time t', namely

$$y_{t'} = z_t, \tag{1}$$

and the next output bit is

$$y_{t'+d(t,t+1)} = z_{t+1}. \tag{2}$$

Considering the value of $d(t, t+1)$, we can classify the correlations related to $y_{t'}$ into the following two cases;

Case 1 $d(t, t+1) = 2$.
The pointer e moves to $y_{t'+1}$ from $y_{t'-1}$ because the step 3 is skipped and i is incremented by step 2 and step 4. So, $y_{t'-1}$ is equal to $y_{t'}$:

$$y_{t'-1} = y_{t'}. \tag{3}$$

Case 2 $d(t, t+1) > 2$.
The pointer e moves to $y_{t'+d(t,t+1)-1}$ from $y_{t'-1}$ because the step 3 is repeated $d(t, t+1)-2$ times. So, i is incremented $d(t, t+1)$ times by step2, step3, and step 4. Therefore, we can obtain the relations among $y_{t'-1}$ to $y_{t'+n}$ as follows:

$$y_{t'-1} = y_{t'+d(t,t+1)-2} \neq y_{t'}, \tag{4}$$
$$y_{t'+n} = y_{t'}, \tag{5}$$
$$n = 1,\ 2, \ldots,\ d(t, t+1) - 3.$$

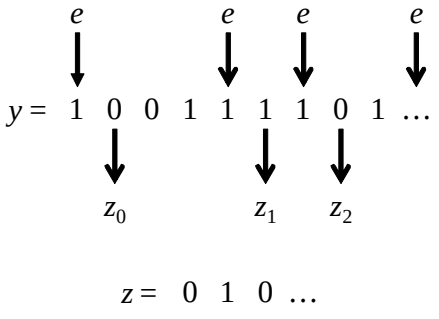

Figure 2: Behavior of Pointer e.

It is possible to recover an input sequence from an output sequence by using Eqs.(3)–(5) if the interval between z_t and z_{t+1} in the input sequence is given. For example, as shown in Fig. 2, if z_0, z_1, z_2 and their intervals in input sequence are given,

$$z_0 = 0,\ z_1 = 1,\ z_2 = 0,$$
$$d(0,1) = 4,\ d(1,2) = 2,\ d(2,3) = 3.$$

the input sequence y_1, y_5, and y_7 are obtained,

$$y_1 = 0,\ y_5 = 1,\ y_7 = 0. \tag{6}$$

From Case 2, y_0, y_1, y_2, and y_3 are obtained by $y_1 = 0$, $d(0,1) = 4$, and Eqs. (4)–(5).

$$\{y_0,\ y_1,\ y_2,\ y_3\} = \{1,\ 0,\ 0,\ 1\}. \tag{7}$$

In a similar way, from Case 1, y_4 and y_5 are obtained by $y_5 = 1$, $d(1,2) = 2$, and Eq. (3). And y_6, y_7, and y_8 are obtained by $y_7 = 0$, $d(2,3) = 3$, and Eqs. (4)–(5) from Case 2.

$$\{y_4,\ y_5\} = \{1,\ 1\}, \tag{8}$$
$$\{y_6,\ y_7,\ y_8\} = \{1,\ 0,\ 1\}. \tag{9}$$

In this way, input sequence $y = 100111101\ldots$ is recovered.

If the attacker can get the output sequence and the intervals of output bits in input sequence by side-channel attack, he can recover the input sequence. However, the compression function ABSG has not been cracked because it is difficult to decide the intervals of output bits in input sequence.

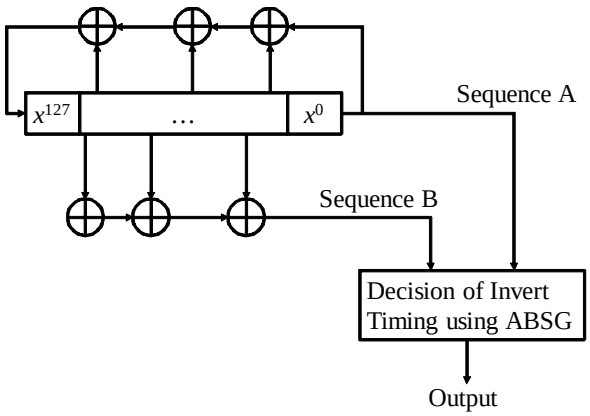

Figure 3: Random Number Generator.

4 Proposed Random Number Generator

In this section, we propose a fast random number generator using the compression function ABSG. As described in Sec. 3.2, the input sequence of ABSG can be recoverd from output sequence if the intervals of output bits in the input sequence is known. The interval is however difficult to obtain from the output sequence, and until now, no crack report has been publishded. For the above reason, the proposed random number generator applies the pointer e that decides the output timing in compression function ABSG.

Figure 3 shows the structure of the proposed random number generator, which is composed of a 128-bit LFSR and a decision function that controls the output sequence using the pointer e. Our generator produces two sequences : a sequence A is related to the polynomial of LFSR and a sequence B is extracted from the registers using a specific polynomial determined by a transition matrix $\mathbf{T}$. The internal states of LFSR are updated by XORing variables of tap positions derived from an 128-degree primitive polynomial.

The matrix $\mathbf{T}$ is designed such that the sequence B is the δ-bit shifted sequence A[9], and the parameter δ can be freely determined. The detail of the determination of δ is discussed in Sec. 5. A transition matrix to generate a polynomial $f(x) = 1 + a_1 x + a_2 x^2 + \ldots + a_L x^L$ is given as follows:

$$\mathbf{T} = \begin{bmatrix} a_1 & a_2 & \cdots & & a_L \\ 1 & 0 & \cdots & & 0 \\ 0 & 1 & \cdots & & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ 0 & & \cdots & 1 & 0 \end{bmatrix}. \tag{10}$$

$\mathbf{T}^m$, $\overline{\mathbf{T}}^m$, and $\overline{\mathbf{T}}_1^m$ $(m = 1,\ 2, \ldots)$ are defined as follows:

$$\mathbf{T}^m = \begin{bmatrix} a_{1,1}^m & a_{1,2}^m & \cdots & a_{1,L}^m \\ a_{2,1}^m & a_{2,2}^m & \cdots & a_{2,L}^m \\ \vdots & \vdots & \ddots & \vdots \\ a_{L,1}^m & a_{L,2}^m & \cdots & a_{L,L}^m \end{bmatrix}, \tag{11}$$

$$\overline{\mathbf{T}}^m = \begin{bmatrix} a_{1,L}^m & a_{1,L-1}^m & \cdots & a_{1,1}^m \\ a_{2,L}^m & a_{2,L-1}^m & \cdots & a_{2,1}^m \\ \vdots & \vdots & \ddots & \vdots \\ a_{L,L}^m & a_{L,L-1}^m & \cdots & a_{L,1}^m \end{bmatrix} \tag{12}$$

$$= \begin{bmatrix} \overline{\mathbf{T}}_1^m \\ \overline{\mathbf{T}}_2^m \\ \vdots \\ \overline{\mathbf{T}}_L^m \end{bmatrix}. \tag{13}$$

Using the initial states of LFSR, $\mathbf{x}_0 = (x_0^0,\ x_0^1,\ \ldots,\ x_0^L)$, the internal states at time l, $\mathbf{x}_l = (x_l^0,\ x_l^1, \ldots,\ x_l^L)$ are represented as follows:

$$\mathbf{x}_l = \overline{\mathbf{T}}^l \mathbf{x}_0,\ l = 1,\ 2, \ldots. \tag{14}$$

From Eq. (14), the output bit x_k^0 at time k is given as follows:

$$x_k^0 = \overline{\mathbf{T}}_1^k \mathbf{x}_0,\ k = 1,\ 2, \ldots. \tag{15}$$

Next, the sequence A and sequence B are input to the decision function. Table 2 shows the algorithm of the decision function. We denote the sequence A and sequence B by $A = (a_0,\ a_1, \ldots)$ and $B = (b_0,\ b_1, \ldots)$, respectively. Our pointer e decides the positions that a bit is flipped, while that of ABSG determines the output timing.

The pointer e indicates the invert timing of the sequence B, and the invert timing is determined by the sequence A in our algorithm. Figure 4 shows the example of output.

Table 2: Algorithm of Decision Function.

Input : $(a_0,\ a_1, \ldots)$
$(b_0,\ b_1, \ldots)$
Output : $(z_0,\ z_1, \ldots)$
Set : $i \leftarrow 0$;
Repeat the following steps:
1. $e \leftarrow a_i;\ z_i \leftarrow b_i$;
2. $i \leftarrow i+1;\ z_i \leftarrow \overline{b_i}$;
3. while$(a_i = \overline{e})$ $i \leftarrow i+1;\ z_i \leftarrow b_i$;
4. $i \leftarrow i+1$;

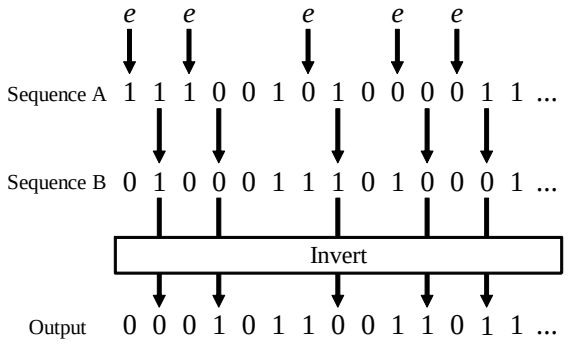

Figure 4: Output Example.

5 Consideration

In this section, we consider the following cases about δ which is the phase shifting between the sequence A and sequence B.

Case 1 $\delta = 0$.
The sequence B is the same as the sequence A. Even if only one invert timing is leaked, the whole input sequence can be recovered using the correlations with the timing.

Case 2 $\delta > 0$.
Assuming that a certain part of the sequence A is leaked in some way. Then, a part of the sequence B can be recovered from the output sequence and the part of the sequence A by synchronizing with δ. Similarly, the corresponding part of sequence A can be recovered because the sequence B is equivalent to δ-bit shifted sequence A.

From the above consideration, it is necessary to make δ sufficiently large. In order to avoid the synchronization attack, we need to update the internal states before outputting δ bits.

6 Conclusion

In this paper, we proposed the new random number generator using the compression function ABSG implemented in the stream cipher DECIMv2. The interval is difficult to obtain from the output sequence, and until now, no crack report has been publishded. Then, taking the advantage of the compression function ABSG, we design the decision function that controls the output sequence. We considered the phase shifting δ between two sequences prodeced by LFSR, and designed our random number generator to avoid the synchronization attack.

The security analysis of our random number generator and the comparison with other generators are left for our future work.

Acknowledgement

We thank Dr. Minoru Kuribayashi for useful discussions.

References

[1] D. Coppersmith, H. Krawczyk, and Y. Mansour, "The Shrinking Generator," CRYPTO 93, LNCS vol. 773, pp.22–39, Springer 1993.

[2] W. Meier and O. Staffelbach, "The Self-Shrinking Generator," Eurocrypt 94, LNCS vol. 950, pp.205–214, Springer 1994.

[3] A. Gouget and H. Sibert, "The Bit Search Generator," In The State of the Art of Stream Cipher: Workshop Record, Brugge, Belgium, pp.60–68, 2004.

[4] A. Gouget, H. Sibert, C. Berbain, N. Courtois, B. Debraize, and C. Mitchell, "Analysis of the Bit-Search Generator and Sequence Compression Techniques,"

FSE2005, LNCS vol. 3557, pp.196–214, Springer 2005.

[5] C. Berbain, O. Billet, A. Canteaut, N. Courtois, B. Debraize, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert, "DECIM v2," eSTREAM, available at
`http://www.ecrypt.eu.org/stream/p3ciphers/decim/decim_p3.pdf`

[6] eSTREAM, ECRYPT Stream Cipher Project, IST-2002-507932, available at
`http://www.ecrypt.eu.org/stream/`

[7] T. Good and M. Benaissa, "Hardware performance of eStream phase- stream cipher candidates," SASC2008 Workshop Record, pp.163–173, available at
`http://www.ecrypt.eu.org/stvl/sasc2008/SASCRecord.zip`

[8] H. Nakagami, R. Teramura, and M. Morii, "On the Security of the Compression Function ABSG on DECIM v2," CSS2008, pp.427–432, 2008.

[9] H. Saitoh, M. Mohri, Y. Fukuta, and Y. Shiraishi, "Improvement of LFSR Initial State Reconstruction Algorithm in Fast Correlation Attack Using Dynamically Constructed Parity Check Equations," Tech. Rep. of IEICE, ISEC2008-142, pp.259–266, 2009.