

# Analysis of DECIM v2 using the timing ABSG output is dropped

Rei Onga and Masakatu Morii

Graduate School of Engineering, Kobe University,  
1-1 Rokkodai, Nada, Kobe, Hyogo, 657-8501, Japan.  
onga@stu.kobe-u.ac.jp, mmorii@kobe-u.ac.jp

**Abstract.** DECIM v2 is a stream cipher proposed by Berbain et al., and it uses the compression function ABSG which guarantees the security of DECIM v2. It also uses a buffer to ensure that the keystream of a constant length is generated each time. In the conventional works, the candidates of ABSG input sequence is reduced by referring to its output one. But in DECIM v2, there is no report that can recover the exact value of input, because of the characteristic of the buffer. In this paper, we propose a new technique to recover the exact value of the bits input to ABSG. Our technique can recover about 8.33% of input sequence of ABSG. We assume that we can get the timing such that the ABSG output is dropped. This assumption enables us to correlate the timing of the ABSG input and that of the buffer output. Observing the buffer output, we can recover the ABSG input.

**Keywords:** cryptanalysis, stream cipher, DECIM v2, ABSG, compression function

## 1 Introduction

A compression function is used by a pseudo-random generator and a stream cipher in order to guarantee these security. For this reason, the recovery of the inputs of compression function inputs from its outputs must be difficult. Researchers have been analyzing the compression function.

The Shrinking Generator (SG) is a compression function proposed by Coppersmith et al. in 1993 [1], which uses two linear feedback shift registers (LFSR). One LFSR controls the outputs of the other one. The Self-Shrinking Generator (SSG) was proposed by Meier et al. in 1994 [2]. The structure of SSG is simpler than SG because SSG requires only one LFSR. Bit-Search Generator (BSG), which is faster than SSG, was proposed by Gouget et al. in 2004 [3]. But BSG requires only an  $3n$ -bit on average. Gouget et al. reported that there was a weakness in BSG and proposed two compression functions ABSG and MBSG [4].

DECIM v2 is a stream cipher proposed by Berbain et al. [5] which applies the compression function ABSG. Since ABSG does not output a random sequence at

a constant speed, DECIM v2 uses a buffer to make it constant without degrading the security level.

As for the evaluation of the security of DECIM v2, ABSG was analyzed by Nakagami et al. [6] and Loe et al. [7]. Nakagami et al. recovered the ABSG input sequence using the interval of inputs which determine the value of ABSG output. But in DECIM v2, we can not get this interval because the ABSG outputs are stored in a buffer and a keystream is generated from the stored values at a constant rate. Loe et al. reduced the candidates of the ABSG input sequence from the ABSG output one. They assumed that they can get the timing such that the buffer becomes full in the initialization algorithm. It implies the Side Channel Attack against ABSG. On the estimation of an 128-bit input sequence,  $2^{128}$  candidates of input sequence are reduced to  $2^{80}$  when they estimate 128-bit input sequence.

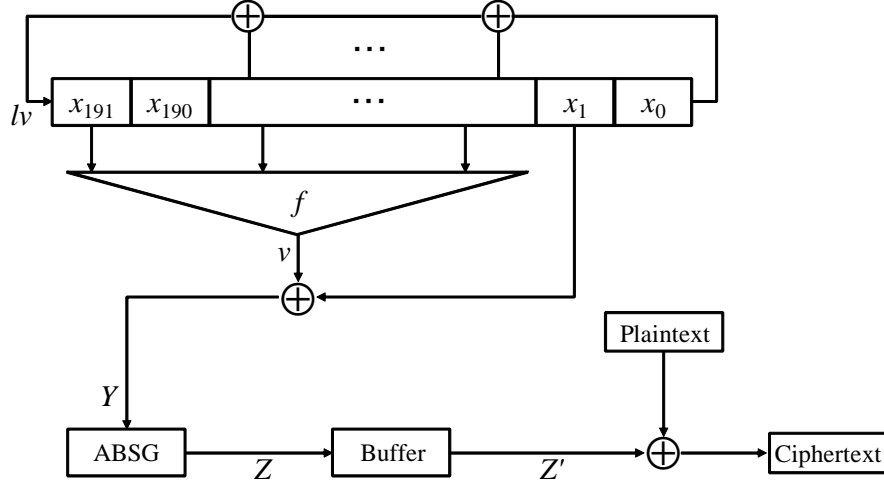
In this paper, we propose a new technique to recover the exact value of bits input to ABSG. When ABSG outputs a bit, a buffer stores it if the buffer is not full, otherwise, the bit is discarded. Here, we introducing a new assumption that we can get this timing. This assumption enables us to recover a particular bit input to ABSG by correlating the timing of the ABSG input and that of the buffer output. Observing a buffer output, we can recover the exact value of an ABSG input. In this situation, we could recover about 8.33% of the input sequence of ABSG, and we succeeded to recover 8.12–8.54% of it in our simulation.

This paper is organized as follows: the algorithm of the DECIM v2 is described in Sect. 2. In Sect. 3, we describe the mechanism of ABSG and review the conventional analyses of the ABSG. In Sect. 4, we propose our technique. In Sect. 5, we show the result of a simulation recovering the ABSG inputs by using our technique. In Sect. 6, we summarize this paper.

## 2 The Stream Cipher DECIM v2

DECIM v2 uses an 80-bit length secret key  $K$  and an 64-bit length public initialization vector  $IV$ . The internal state consists of an 192-bit linear feedback shift register (LFSR)  $S = (x_0, x_1, \dots, x_{191})$ , where  $x_i$  is an 1-bit variable. In addition, DECIM v2 has a nonlinear filter function  $f$ , an irregular decimation mechanism which called ABSG, and a buffer. The buffer consists of an 32-bit internal state  $B = (b_0, b_1, \dots, b_{31})$ , where  $b_i$  is an 1-bit variable.

The algorithm of DECIM v2 can be divided into three parts; pseudo-random generation algorithm, buffer initialization algorithm, Key/IV setup algorithm. In the pseudo-random generation stage, a pseudo-random sequence which is called keystream is generated from the bits which are stored in a buffer. A ciphertext is obtained by XORing a plaintext and the keystream. In the buffer initialization stage, the buffer stores the ABSG outputs until it becomes full. In the Key/IV setup stage, the internal state of LFSR is initialized by a secret key and an initialization vector.



**Fig. 1.** The structure of pseudo-random generation algorithm and buffer initialization algorithm.

## 2.1 Pseudo-Random Generation Algorithm

The structure of the pseudo-random generation algorithm and the buffer initialization algorithm is shown in Fig. 1. The internal state of LFSR at time  $t$  is denoted by  $S_t = (x_{0,t}, x_{1,t}, \dots, x_{191,t})$ , and that of buffer is denoted by  $B_{\tilde{t}} = (b_{0,\tilde{t}}, b_{1,\tilde{t}}, \dots, b_{31,\tilde{t}})$ . The pseudo-random generation algorithm has two functions  $PRGA\_Nextstate$  and  $Buffer\_Output$ .  $PRGA\_Nextstate$  updates the internal state, and  $Buffer\_Output$  generates the keystream from the buffer.

The process of  $PRGA\_Nextstate$  is summarized as follows.

**Step1** Updating the internal state of LFSR.

The internal state is updated by the following equation:

$$x_{i,t+1} = \begin{cases} x_{i+1,t} & \text{for } 0 \leq i \leq 190 \\ lv_t & \text{for } i = 191 \end{cases}, \quad (1)$$

where

$$\begin{aligned} lv_t = & x_{189,t} \oplus x_{188,t} \oplus x_{169,t} \oplus x_{156,t} \oplus x_{155,t} \\ & \oplus x_{132,t} \oplus x_{131,t} \oplus x_{94,t} \oplus x_{77,t} \oplus x_{46,t} \\ & \oplus x_{17,t} \oplus x_{16,t} \oplus x_{5,t} \oplus x_{0,t}. \end{aligned} \quad (2)$$

**Step2** Updating the nonlinear filter function.

The nonlinear filter function  $f$  outputs an 1-bit variable  $v$  using the internal

state as follows:

$$\begin{cases} X = (x_{191,t} + x_{186,t} + x_{178,t} + x_{172,t} + x_{162,t} \\ \quad + x_{144,t} + x_{111,t} + x_{104,t} + x_{65,t} + x_{54,t} \\ \quad + x_{45,t} + x_{28,t} + x_{13,t}) \bmod 4 \\ v_t = \begin{cases} 0 & \text{for } X = 0, 3 \\ 1 & \text{for } X = 1, 2 \end{cases} \end{cases} \quad (3)$$

**Step3** Generating the input sequence of ABSG.

An 1-bit variable  $y_t$  is obtained from the outputs of the nonlinear filter function  $f$  and the internal state of LFSR:

$$y_t = v_t \oplus x_{1,t}. \quad (4)$$

**Step4** Storing the output sequence of ABSG in a buffer.

The variables  $y_t$  are sequentially input to ABSG, and ABSG outputs  $Z = (z_0, z_1, \dots)$ . The rate of output is approximately 1-bit per 3-bit of input. These outputs are stored in a buffer subsequently if the buffer is not full, otherwise, the output is discarded.

Remember that ABSG receives  $y_t$  produced by Eq. (4) at the timing  $t$ , and compresses the values to produce an output sequence  $z_{t'}$  ( $t' = 0, 1, 2, \dots$ ) which timing  $t'$  is not synchronized to  $t$ . Now, we suppose that at a time  $t$  the ABSG output  $z_{t'}$  is stored in the buffer  $b_{31,\tilde{t}}$ ;

$$b_{31,\tilde{t}} = z_{t'}, \quad (5)$$

$$\tilde{t} = \left\lfloor \frac{t}{4} \right\rfloor. \quad (6)$$

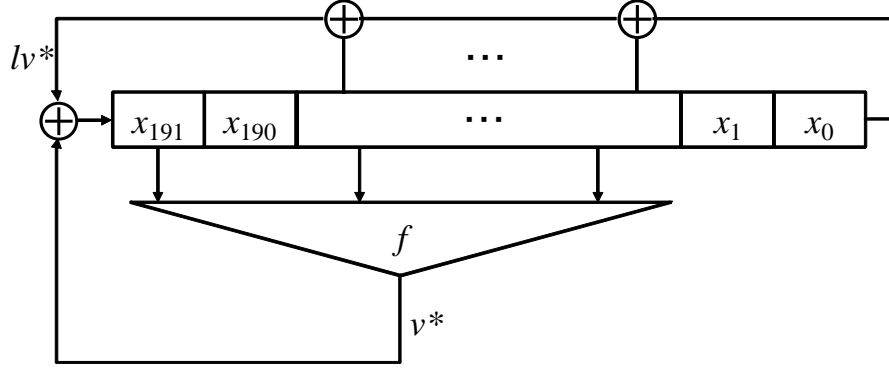
After 30 cycles, the stored value is shifted to  $b_{0,\tilde{t}+30}$ , and it is output from the buffer at the next cycle.

## 2.2 Buffer Initialization Algorithm

The buffer initialization algorithm fills up a buffer after finishing the Key/IV setup algorithm. This algorithm has a functions *PRGA\_Nextstate*. *PRGA\_Nextstate* works the same way as explained in Sect. 2.1. After executing *PRGA\_Nextstate* four times, this algorithm ascertains whether the buffer is full or not. If the buffer is full, this algorithm finishes and the internal state of the buffer at that time is used as an initial state of the pseudo-random generation algorithm.

## 2.3 Key/IV Setup Algorithm

The structure of the Key/IV setup algorithm is shown Fig. 2. The internal state of the Key/IV setup algorithm at time  $t$  is denoted by  $S_t^* = (x_{0,t}^*, x_{1,t}^*, \dots, x_{191,t}^*)$ .



**Fig. 2.** The structure of Key/IV setup Algorithm.

First,  $S^*$  is initialized by  $K$  and  $IV$  as follows:

$$x_{i,0}^* = \begin{cases} K_i & \text{for } 0 \leq i \leq 79 \\ K_{i-80} \oplus IV_{i-80} & \text{for } 80 \leq i \leq 143 \\ K_{i-80} \oplus IV_{i-144} \oplus IV_{i-128} \oplus IV_{i-112} \oplus IV_{i-96} & \text{for } 144 \leq i \leq 159 \\ IV_{i-160} \oplus IV_{i-128} \oplus 1 & \text{for } 160 \leq i \leq 191 \end{cases} \quad (7)$$

Next,  $S_{t-1}^*$  is updated by the following  $KSA\_Nextstate$ .

$$x_{i,t+1}^* = \begin{cases} x_{i+1,t}^* & \text{for } 0 \leq i \leq 190 \\ lv_t^* \oplus v_t^* & \text{for } i = 191 \end{cases}, \quad (8)$$

where  $lv_t^*$  and  $v_t^*$  are obtained from  $S_t^*$  by using Eq. (2)(3).

After executing  $KSA\_Nextstate$  768 times, the derived  $S_{768}^*$  is set to  $S_0$ , which is an initial state of the buffer initialization algorithm.

### 3 ABSG

In this section, we describe the mechanism of ABSG and review the conventional analyses about the ABSG.

#### 3.1 Mechanism of ABSG

DECIM v2 gets the compressed pseudo-random sequence by inputting the sequence generated by the values of LFSR and the nonlinear filter function  $f$  to ABSG. ABSG is a bit generator that searches for a pattern in the input bitstream and outputs a shorter bitstream. The algorithm of ABSG is shown in Fig. 3.

Input: $(y_0, y_1, \dots)$ Set: $t \leftarrow 0; t' \leftarrow 0;$ Repeat the following steps: 1. $e \leftarrow y_t, z_{t'} \leftarrow y_{t+1}$ 2. $t \leftarrow t + 1;$ 3. while $(y_t = \bar{e}) t \leftarrow t + 1;$ 4. $t \leftarrow t + 1;$ 5. output $z_{t'};$ 6. $t' \leftarrow t' + 1;$
---

**Fig. 3.** The algorithm of ABSG.

### 3.2 Conventional Analyses

Nakagami et al. have reported that they can recover the exact value of the input sequence to ABSG if the interval of inputs which determine the value of ABSG output is known. [6]. When the value of the ABSG output  $z_{t'}$  is derived from that of the ABSG input  $y_t$ , the value of  $y_{t+m}$  ( $m \neq 1$ ) which determines that of  $z_{t'+1}$  can be recovered as follows:

**Case 1** When  $m = 2$ .

$$y_{t-1} = y_t. \tag{9}$$

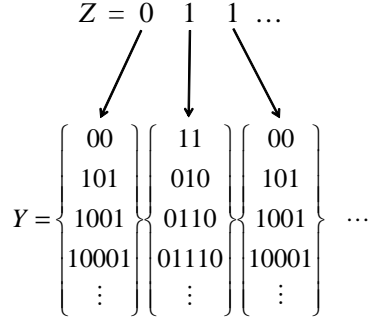
**Case 2** When  $3 \leq m$ .

$$y_{t-1} = y_{t+m-2} \neq y_t. \tag{10}$$

$$y_{t+n} = y_t \text{ for } n = 1, 2, \dots, m - 3. \tag{11}$$

Eq.(9)–(11) show that it is possible to recover the input sequence  $y_{t-1} - y_{t+m-2}$  if we know the variable  $m$ . For example, when  $z_{t'} = 0$  and  $m = 2$ , we can recover  $\{y_{t-1}, y_t\} = \{0, 0\}$ . Also,  $z_{t'} = 1$  and  $m = 3$ , we can recover  $\{y_{t-1}, y_t, y_{t+1}\} = \{1, 0, 1\}$ . Fig. 4 shows this situation. It is noticed that the introduction of a buffer makes it difficult to obtain  $m$ .

Loe et al. have reported that it is possible to reduce the candidates of input sequences [7]. If we know the timing when the buffer initialization algorithm finishes, we can reduce the candidates of input sequences. The timing informs us to determine the number of inputs  $L$  required to fill up the buffer in the the buffer initialization algorithm. Because the size of the buffer  $P$  is known, we can get the number of candidates inputs  $C$  by calculating the total number of the



**Fig. 4.** Candidate inputs.

combinations as follows:

$$C = \begin{cases} \binom{L-P-1}{L-2P} & \text{for } L = L' \\ \binom{L-P-1}{L-2P} \sum_{1 \leq p \leq (L'-L)/2} \binom{L'-L-p-1}{L'-L-2p} & \text{for } L < L' \\ \min \left( \binom{L-P-1}{L-2P}, \sum_{1 \leq p \leq L'/2} \binom{L'-p-1}{L'-2p} \right) & \text{for } L > L' \end{cases}, \quad (12)$$

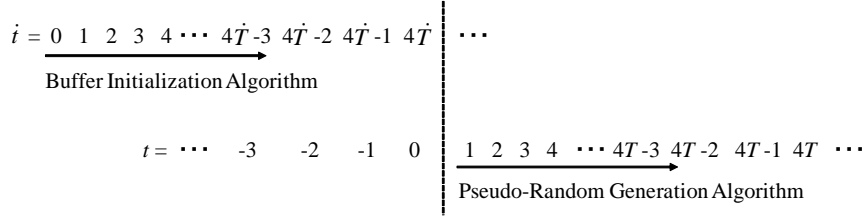
where  $L'$  is the number of inputs which we want to know. In their result,  $2^{128}$  of input sequences is reduced to about  $2^{80}$  candidates when they estimate an 128-bit input sequence and  $P = 32$ .

## 4 Proposed Analysis

In this section, we propose a new analyzing technique. Using our technique, we can recover the exact value of the ABSG inputs in DECIM v2, if the timing the buffer initialization algorithm finishes and the timing that ABSG output is dropped, we call this timing “drop timing”, are known. First, we consider the drop timing. Then, we describe how to recover the ABSG inputs.

### 4.1 The ABSG Dropped Timing

DECIM v2 generates the input sequence of ABSG  $Y$  by using the value of LFSR and the nonlinear filter function  $f$ . ABSG generates the compressed sequence  $Z$  from  $Y$ . A buffer stores  $Z$  and generates a keystream  $Z'$ . When the buffer is full, the newly computed bit is not added into the queue, i.e. it is dropped. In the buffer initialization algorithm, for every four times executing  $PRGA\_Nextstate$



**Fig. 5.** The timing each algorithm starts.

it checks if the buffer is full or not. So, the algorithm must finish at a certain time  $t = 4T$ , and then the pseudo-random generation algorithm starts at the time  $t = 1$ . Fig. 5 shows this situation.

We consider the drop timing. This timing could be restricted by using the following two properties of DECIM v2.

**Property 1.** The minimum interval of the ABSG outputs is two clocks.

Referring to Fig. 3, at the step2 and step4 the cycle is incremented. If  $y_t = e$  at the step3, the incrementation is omitted. Therefore, the minimum interval is derived when  $y_t = e$  at the step3, and the number of cycles is two.

**Property 2.** The buffer  $b_{31,\bar{t}}$  becomes empty at  $t = 4T$  for  $T = 1, 2, \dots$

Since a keystream is output from  $b_{0,\bar{t}}$  at the timing  $t = 4T$ , the stored buffer values are shifted at the same timing as follows;

$$b_{i,\bar{t}+1} = b_{i+1,\bar{t}} \text{ for } 0 \leq i \leq 30. \quad (13)$$

After  $b_{31,\bar{t}}$  becomes empty at  $t = 4T$ , we suppose the case such that ABSG outputs  $z_{t'}$  during the timing  $t \in \{4T + 1, 4T + 2, 4T + 3, 4T + 4\}$ . When ABSG outputs  $z_{t'}$  at  $t = 4T + 1$ , it may output further one bit  $z_{t'+1}$  at timing  $t \in \{4T + 3, 4T + 4\}$  from the property 1. In such case, the bit  $z_{t'+1}$  is discarded as  $z_{t'}$  has been already assigned to  $b_{31,\bar{t}}$ . Similarly, when ABSG outputs  $z_{t'}$  at  $t = 4T + 2$ ,  $z_{t'+1}$  may be output at timing  $t = 4T + 4$  and it is discarded.

There are the other cases that the drop of ABSG outputs is occurred. Though the buffer initialization algorithm finishes at  $t = 0$ , the buffer may have been already full at timing  $t \in \{-3, -2, -1\}$  because the algorithm checks for every four times if the buffer is full or not. If the buffer becomes full at  $t = -3$ , an ABSG output at  $t \in \{-1, 0\}$  is discarded. However, DECIM v2 does not generate a keystream at  $t = 0$ ,  $b_{31,\bar{t}}$  does not become empty at that times. Now, if the buffer becomes full at  $t = -3$ , the ABSG output at  $t \in \{-1, 0, 1, 2, 3, 4\}$  is discarded. Similarly, the other cases can be enumerated. Table 1 summarize the cases of timing  $t$ .

## 4.2 Recovering The ABSG Inputs

We describe how to recover the exact value of ABSG inputs by using the drop timing at Case 1–6.

**Table 1.** The classification of the timing  $t$ .

	$b_{31,\bar{i}} \rightarrow \in z_{t'}$	$z_{t'+1}$ is dropped
Case.1	$4T + 1$	$4T + 3, 4T + 4$
Case.2	$4T + 2$	$4T + 4$
Case.3	$-3$	$-1, 0, 1, 2, 3, 4$
Case.4	$-2$	$0, 1, 2, 3, 4$
Case.5	$-1$	$1, 2, 3, 4$
Case.6	$0$	$2, 3, 4$

**Case 1** The buffer stores  $z_{t'}$  in  $b_{31,\bar{i}}$  at  $t = 4T + 1$

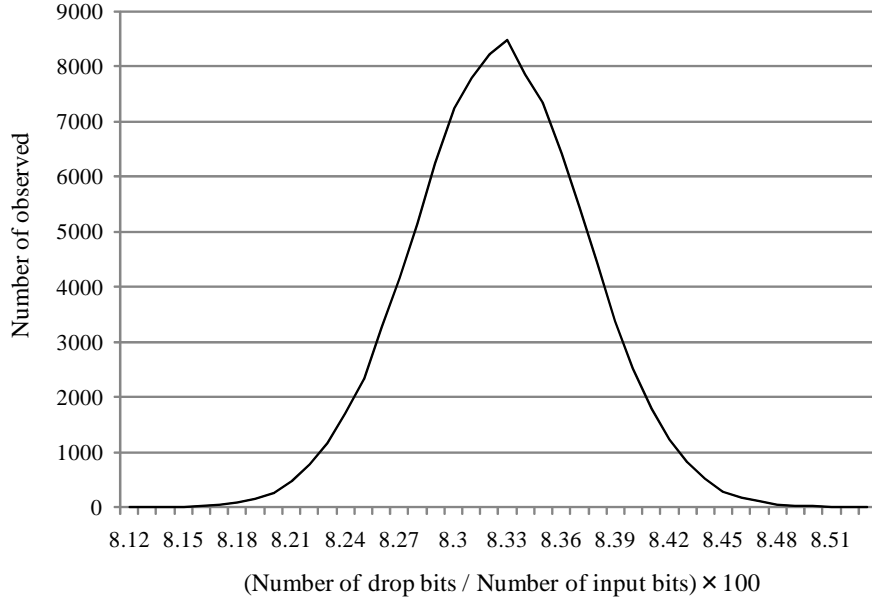
When the buffer stores  $z_{t'}$  in  $b_{31,\bar{i}}$  at  $t = 4T + 1$ ,  $z_{t'+1}$  may be output at  $t \in \{4T + 3, 4T + 4\}$ . Now, we suppose  $z_{t'+1}$  is discarded at  $t = 4T + 3$ . At the time, the value of  $y_{4T+3}$  is assigned to  $e$ , and that of  $y_{4T+4}$  to  $z_{t'+2}$  in ABSG. Until  $y_t = e$ , ABSG does not output  $z_{t'+2}$ , we define the timing  $t = \hat{t}$ . By the way, the buffer outputs a bit as a keystream and updates their internal state given by Eq. (13) at  $t = 4T + 4$ . Because property 1 in Sect. 4.1 shows  $\hat{t} \geq 4T + 5$ ,  $z_{t'+2}$  is stored in the buffer as  $b_{31,\bar{i}}$ . Since the value is output as  $z'_{T+32}$ , the observation of  $z'_{T+32}$  gives us the exact value of  $y_{4T+4}$ . Fig. 6 shows this algorithm.

- |  |
|--|
| <ol style="list-style-type: none"> <li>1. Observing the timing <math>z_{t'+1}</math> is dropped,</li> <li>2. ABSG assign the value of <math>y_{4T+4}</math> to <math>z_{t'+2}</math>,</li> <li>3. Buffer generates <math>z'_{T+1}</math>,</li> <li>4. ABSG outputs <math>z_{t'+2}</math>,</li> <li>5. Buffer stores <math>z_{t'+2}</math> as <math>b_{31,\bar{i}}</math>,</li> <li>6. Observing <math>z'_{T+32}</math>,</li> <li>7. Recovering <math>y_{4T+4}</math> from <math>z'_{T+32}</math>.</li> </ol> |
|--|

**Fig. 6.** How to recover exact value of ABSG inputs.

Next, we suppose  $z_{t'+1}$  is discarded at  $t = 4T + 4$ . We can recover the ABSG input by the similar way described above. At the time, the value of  $y_{4T+4}$  is assigned to  $e$ , and that of  $y_{4T+5}$  to  $z_{t'+2}$ . The buffer stores  $z_{t'+2}$  in  $b_{31,\bar{i}}$ , and outputs the value as  $z'_{T+32}$ . Therefore we can recover the exact value of  $y_{4T+5}$  by observing  $z'_{T+32}$ . At Case 2, we can recover the ABSG input by the similar way.

For example, we suppose an output of ABSG  $z_{t'+1}$  is dropped at  $t = 11$ . At the time, ABSG assigns the value of  $y_{11}$  to  $e$ , and that of  $y_{12}$  to  $z_{t'+2}$ . By the way, the buffer outputs a bit and updates at  $t = 12$ . Hence,  $z_{t'+2}$  is stored in the buffer as  $b_{31,3}$ . Buffer outputs this value as  $z'_{34}$ , the observation of  $z'_{34}$  gives us to recover the exact value of  $y_{12}$ .



**Fig. 7.** Result of the simulation.

**Case 3** The buffer becomes full at  $t = -3$

When the buffer becomes full at  $t = -3$ ,  $z_{t'+1}$  may be output at  $t \in \{-1, 0, 1, 2, 3, 4\}$ . We can recover the ABSG input by the similar way described at Case 1. Supposing the last time that ABSG output  $z_{t'+1}$  is dropped at  $\hat{t}$  is during the timing  $t \in \{-1, 0, 1, 2, 3, 4\}$ . At the time, the value of  $y_{\hat{t}}$  is assigned to  $e$ , and that of  $y_{\hat{t}+1}$  to  $z_{t'+2}$ . Then, the buffer stores  $z_{t'+2}$  in  $b_{31,1}$ , and outputs it as  $z'_{32}$ . Therefore, we can recover the exact value of  $y_{\hat{t}+1}$  by observing  $z'_{32}$ .

The recovery procedure at Case 4-6 is analogy to the procedure at Case 3, so the description is omitted.

## 5 Simulation Results

We show a numerical result of recovering the ABSG inputs by using our technique. In this numerical experiment, we use 100000 patterns of secret keys and generate an 10000-byte keystream. We investigated the number of ABSG inputs which could be recovered per all ABSG inputs. Fig. 7 shows the result. Using our technique, we could recover one ABSG input by a drop timing. Therefore a horizontal axis indicates the number of ABSG inputs which could be recovered per all ABSG inputs. As you can see in Fig. 7, we could recover 8.12–8.54% of the ABSG inputs. We could recover in average about 8.33% of the ABSG inputs.

## 6 Conclusion

DECIM v2 is a stream cipher which has submitted to ISO/IEC 18033-4. In this research, we analyzed the security of DECIM v2, and focused on the compression function ABSG and the buffer.

In this paper, we proposed a new technique to recover the exact value of bits input to ABSG by using the timing when ABSG output is dropped. In DECIM v2, some ABSG output may be dropped because the speed of ABSG outputting is faster than that of buffer outputting. We suppose that we can get this drop timing as the Side Channel Attack. First, we showed that we could restrict the timing ABSG output is dropped. Second, we gave our technique to recover the ABSG inputs. Finally, we showed a simulation result to estimate how many bits could be recovered. The result implied that we can recover in average about 8.33% of the ABSG inputs.

## References

1. D. Coppersmith, H. Krawczyk, and Y. Mansour, "The Shrinking Generator," *Proc. CRYPTO'93*, Lecture Note in Computer Science, vol. 773, pp.22–39, 1993.
2. W. Meier and O. Staffelbach, "The Self-Shrinking Generator," *Proc. EURO-CRYPT'94*, Lecture Note in Computer Science, vol. 950, pp.205–214, 1994.
3. A. Gouget and H. Sibert, "The Bit Search Generator" In *The State of the Art of Stream Ciphers: 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," *Proc. FSE2005*, Lecture Note in Computer Science, vol. 3557, pp.196–214, 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/stream/p3ciphers/decim/decim\\_p3.pdf](http://www.ecrypt.eu/stream/p3ciphers/decim/decim_p3.pdf)
6. H. Nakagami, R. Teramura, and M. Morii, "On the Security of the Compression Function ABSG on DECIM v2," *Proc. Computer Security Symposium 2008 (CSS2008)*, 2008, (in Japanese).
7. C. W. Loe and K.Khoo, "Side Channel Attacks on Irregularly Decimated Generators" *Proc. ICISC 2007*, Lecture Note in Computer Science, vol. 4817, pp.116–130, 2007.