Distinguishing Attack Against TPypy

Yukiyasu Tsunoo[1], Teruo Saito[2], Takeshi Kawabata[2], and Hiroki Nakashima[2]

[1] NEC Corporation
1753 Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa 211-8666, Japan
tsunoo@BL.jp.nec.com
[2] NEC Software Hokuriku, Ltd.
1 Anyoji, Hakusan, Ishikawa 920-2141, Japan
{t-saito@qh,t-kawabata@pb,h-nakashima@vt}.jp.nec.com

Abstract. TPypy is a tweaked version of the Py stream cipher algorithm submitted to eSTREAM. Py uses a kind of processing referred to as a 'rolling array', the mixing of two types of array and one variable, to generate the keystream. TPypy is proposed as a highly secure stream cipher that fixes all of the previously identified weaknesses of Py.

This paper reports a significant bias in the pseudo-random generation algorithm of TPypy that can be exploited to distinguish the keystream obtained from multiple arbitrary secret key and initial vector pairs from a truly random number sequence using about 2^{199} words.

Keywords: distinguishing attack, ECRYPT, eSTREAM, Py family of stream ciphers, TPypy.

1 Introduction

Many stream ciphers have been proposed over the past 20 years. Most of them are constructed using a linear feedback shift register (LFSR), which is easily implemented in hardware, but the software implementations are mostly slow. In 1987, Rivest designed the RC4 stream cipher, which is suited to software implementation [11]. RC4 has been implemented for many applications, including the Secure Socket Layer (SSL) and Wired Equivalent Privacy (WEP), and is one stream cipher that is widely used around the world.

In the past few years, several modified RC4 algorithms have been proposed. One of them is the Py stream cipher proposed by Biham and Seberry for eSTREAM in 2005 [1,3]. The secret key is up to 32 bytes long and the initial vector (IV) is up to 16 bytes long. Both are selectable in multiples of one byte. An 8-byte keystream is generated at each time. However, from a security standpoint, the designers limited the keystream that can be generated for one secret key and IV pair to 2^{64} bytes. Py employs processing called a 'rolling array' to generate a keystream while mixing two arrays and one variable.

For the analysis of Py, a number of distinguishing attacks that focus on weaknesses in the pseudo-random generation algorithm (PRGA) have been proposed [6,9,10]. None of those methods, however, threaten the security of Py because of the limit on the length of the keystream. Nevertheless, the designers have further proposed Pypy, a model in which security can be guaranteed without a limit on the length of the keystream [4]. Pypy includes the modification that a 4-byte keystream is generated at each time.

Since 2006, however, a number of key recovery attacks that exploit weakness in the IV schedule have been reported [7,13,14,15]. Those methods pose a security problem for Py, Py6, and Pypy. Accepting those reports, Biham and Seberry tweaked the IV schedule and proposed a secure model as TPy, TPy6, and TPypy [5]. On the basis of various analyses [7,9,14], however, the Py family of stream ciphers was not selected as a phase 3 candidate by eSTREAM.

Here, we report a significant bias in the output sequence of the newly proposed TPypy PRGA based on a detailed analysis. Exploiting that bias allows the keystream that can be obtained with mulitple arbitrary secret key and IV pairs to be distinguished from a truly random number sequence with about 2^{199} words. Furthermore, our method succeeds with a greatly smaller amount of data than results that have been reported previously [8,12].

The Py family of stream ciphers is explained in Section 2. Section 3 explains the distinguishing attack on TPypy and Section 4 is the conclusion.

2 Py Family of Stream Ciphers

In this section, we explain the Py family of stream ciphers. For a more detailed description refer to the proposal papers [3,4,5].

2.1 Proposals and Analyses of the Py Family of Stream Ciphers

In this section, we summarize the flow of the Py family of stream cipher proposals as well as analyses of them.

In 2005, Biham and Seberry proposed Py [3] to improve implementability and security. Py has an 8-bit index array P and a 32-bit array Y for mixing data as an internal state. It changes these arrays by a process known as 'rolling array'. An evaluation of the implementability of Py by Biham et alia showed Py to be about 2.5 times as fast as RC4 on a 32-bit processor. Py6 was also proposed at the same time as a model that has fast initialization. For security reasons, however, Biham et alia limited the keystreams that can be generated by one secret key and IV pair to 2^{64} bytes in Py and 2^{40} bytes in Py6.

Nevertheless, in 2006 Paul et alia proposed a distinguishing attack against Py [9]. By their method, Py output can be distinguished from a truly random number sequence using a keystream of about $2^{89.2}$ bytes. In the same year, Crowley increased the efficiency of the method of Paul et alia with respect to amount of data by applying a hidden Markov model [6]. By Crowley's method, Py can be distinguished from a truly random number sequence with a keystream of about 2^{72} bytes. In the same year, Paul et alia also showed that the same method could be applied to Py6 [10]. With the method of Paul et alia, Py6 can be distinguished from a truly random number sequence with a keystream of about $2^{68.61}$ words (64 bits/word).

The security standard of Py and Py6 are that attack is not possible with a keystream of less than 2^{64} bytes for Py and of less than 2^{40} bytes for Py6, so the method of [6,9,10] does not go as far as to threaten the security Py or Py6. Nevertheless, Biham et alia proposed at the FSE 2006 rump session a further modified algorithm as Pypy, to which the method of [6,9,10] cannot be applied, even with keystreams larger than 2^{64} bytes [4]. Although Pypy is basically the same algorithm as Py, only 32 bits of data are output as the keystream, which is half that of Py.

After that, Wu and Preneel proposed a key recovery attack that exploits the weakness in the IV schedule of Py and Pypy that equivalent keystreams can be output from different IV [13,14]. By that method, the Py and Pypy with a 16-byte secret key and a 16-byte IV can be broken by using 2^{24} chosen IV and assuming 72-bit keys. After that, Isobe et alia showed that an improvement in the efficiency of the Wu et alia method allows 16-byte secret key and 16-byte IV Py and Pypy to be broken by using 2^{24} chosen IV and a 48 bit key [7]. In 2007, Wu et alia further improved the efficiency of their attack and reported that 16-byte secret key and 16-byte IV Py and Pypy can be broken by using 2^{23} chosen IV and a 24-bit key [15].

To prevent attacks that exploit the weakness of the IV schedule, Biham et alia improved the IV schedule and proposed TPy, TPy6, and TPypy [5]. TPy, TPy6, and TPypy inherit the respective security standards of Py, Py6, and Pypy, so TPypy can be considered the model that has the highest security of the Py family of stream ciphers.

Recently, Kogiso and Shimoyama proposed a distinguishing attack against Pypy [8]. By their method, Pypy can be distinguished from a truly random number sequence with a keystream of about 2^{220} words. Because Pypy and TPypy have the same PRGA structure, this attack can also be applied to TPypy. Sekar et alia also proposed a distinguishing attack against TPypy and TPy [12] that can distinguish TPypy from a truly random number sequence with a keystream of about 2^{281} words.

2.2 Description of TPypy

TPypy has arrays P and Y, and 32-bit variable s. P is an array of 256 bytes that contains a permutation of all values from 0 to 255, and Y is an array of 260 32-bit words indexed from -3 to 256. TPypy uses a process called a 'rolling array' to mix the data of arrays P and Y and variable s to generate the keystream. The keystream is output 32 bits at a time. The encryption generates a keystream whose length is the number of bytes of the input plaintext, with the ciphertext generated by a taking the bit-wise exclusive-OR of the plaintext.

TPypy consists of roughly three phases: the key schedule, which performs initialization with the secret key; the IV schedule, which performs initialization with the IV; and the keystream generating PRGA. In the analysis we report here, we are concerned only with the structure of the PRGA, so we omit explanation

```
Input: Y[-3,...,256], P[0,...,255], a 32-bit variable s
Output: 32-bit random output

   /* update and rotate P */
1. swap(P[0], P[Y[185] & 0xFF]);
2. rotate(P);

   /* Update s */
3. s += Y[P[72]] – Y[P[239]];
4. s = ROTL32(s, ((P[116] + 18) & 31));

   /* Update 4 bytes (least significant byte first) */
5. output((s ⊕ Y[-1]) + Y[P[208]]);

   /* Update and rotate Y */
6. Y[-3] = (ROTL32(s, 14) ⊕ Y[-3]) + Y[P[153]];
7. rotate(Y);
```

Fig. 1. PRGA of TPypy

related to the key schedule and the IV schedule. The TPypy PRGA is shown in Fig. 1. TPypy is a modification of the initialization of Pypy, so it has the same PRGA structure as Pypy.

In Fig. 1, the $\oplus$ symbol refers to a bit-wise exclusive-OR operation. The bit-wise AND operation is denoted as &. Addition and subtraction with modulus 2^{32} are denoted as $+$ and $-$. ROTL32(X, i) denotes i-bit rotation of word X to the left. The exchange of entry 0 and entry j of array P is denoted as swap$(P[0], P[j])$. The notation rotate(P) means a cyclic rotation of the elements of array P by one position.

In this paper, the keystream at time t is defined as O_t. In the same way, the arrays P and Y, and the internal variable s at time t are denoted as P_t, Y_t, s_t. After completion of the IV schedule, it becomes P_0, Y_1, s_0, and the output at time $t = 1$ is as follows.

$$O_1 = (s_1 \oplus Y_1[-1]) + Y_1[P_1[208]]$$

Bit n of word X is defined as $X_{(n)}$. Here, $n = 0$ means the least significant bit.

3 Distinguishing Attack

In this section, we explain a bias that exists in the TPypy output sequence and show that the bias can be exploited to distinguish the output from a truly random number sequence.

3.1 Bias in Output Sequence

In this section, we show that Theorem 1 holds for the TPypy output sequence.

Theorem 1. *When the following conditions hold,* $O_{1(0)} \oplus O_{3(0)} \oplus O_{6(0)} \oplus O_{7(0)} = 0$ *necessarily holds.*

C1. $P_6[208] = 254$
C2. $P_7[208] = 255$
C3. $P_2[116] \equiv -18 \ (mod\ 32)$
C4. $P_2[72] = P_3[239] + 1$
C5. $P_2[239] = P_3[72] + 1$
C6. $P_4[116] \equiv -18 \ (mod\ 32)$
C7. $P_4[72] = P_5[239] + 1$
C8. $P_4[239] = P_5[72] + 1$
C9. $P_3[116] \equiv P_5[116] \equiv -18 \ (mod\ 32)$ or $P_3[116] \equiv P_5[116] \equiv 0 \ (mod\ 32)$
C10. $P_7[116] \equiv -18 \ (mod\ 32)$
C11. $P_1[208] = 4$
C12. $P_3[208] = 3$
C13. $P_3[153] = P_7[72] + 4$
C14. $P_5[153] = P_7[239] + 2$

Proof. First, from the TPypy output generation equation, we derive the following.

$$\begin{aligned} O_1 &= (s_1 \oplus Y_1[-1]) + Y_1[P_1[208]] \\ O_3 &= (s_3 \oplus Y_3[-1]) + Y_3[P_3[208]] \\ O_6 &= (s_6 \oplus Y_6[-1]) + Y_6[P_6[208]] \\ O_7 &= (s_7 \oplus Y_7[-1]) + Y_7[P_7[208]] \end{aligned}$$

Thus, if $Z_{(0)} = O_{1(0)} \oplus O_{3(0)} \oplus O_{6(0)} \oplus O_{7(0)}$, we have

$$\begin{aligned} Z_{(0)} = s_{1(0)} &\oplus Y_1[-1]_{(0)} \oplus Y_1[P_1[208]]_{(0)} \\ \oplus s_{3(0)} &\oplus Y_3[-1]_{(0)} \oplus Y_3[P_3[208]]_{(0)} \\ \oplus s_{6(0)} &\oplus Y_6[-1]_{(0)} \oplus Y_6[P_6[208]]_{(0)} \\ \oplus s_{7(0)} &\oplus Y_7[-1]_{(0)} \oplus Y_7[P_7[208]]_{(0)} \end{aligned} \quad (1)$$

Here, when conditions C1 and C2 hold, the following relations are derived from the values of A' and B' for the state transitions of array Y in Fig. 2.

$$A' = Y_6[P_6[208]] = Y_6[254] = (ROTL32(s_3, 14) \oplus Y_1[-1]) + Y_3[P_3[153]] \quad (2)$$
$$B' = Y_7[P_7[208]] = Y_7[255] = (ROTL32(s_5, 14) \oplus Y_3[-1]) + Y_5[P_5[153]] \quad (3)$$

From the relation of Eq. (2) and Eq. (3), Eq. (1) is as follows.

$$\begin{aligned} Z_{(0)} = s_{1(0)} &\oplus s_{3(18)} \oplus Y_3[P_3[153]]_{(0)} \oplus Y_1[P_1[208]]_{(0)} \\ \oplus s_{3(0)} &\oplus s_{5(18)} \oplus Y_5[P_5[153]]_{(0)} \oplus Y_3[P_3[208]]_{(0)} \\ \oplus s_{6(0)} &\oplus s_{7(0)} \oplus Y_6[-1]_{(0)} \oplus Y_7[-1]_{(0)} \end{aligned} \quad (4)$$

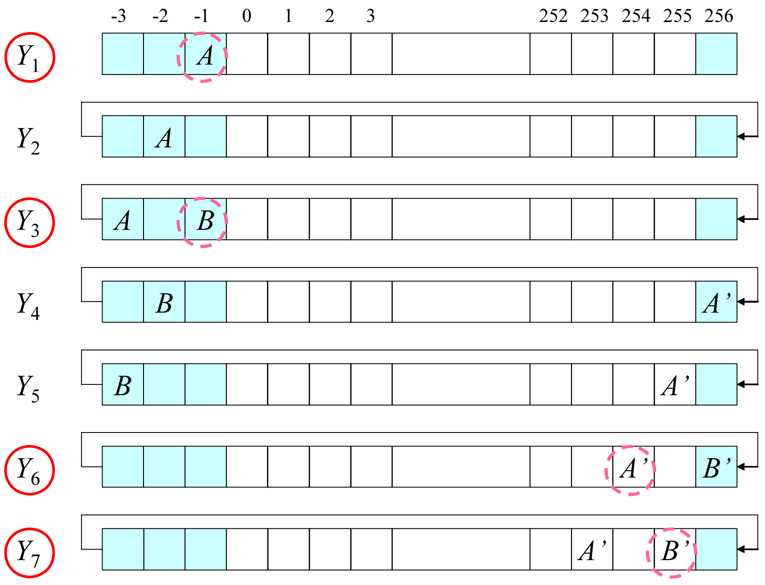

Fig. 2. State transitions of array Y

Next, from the update equation for variable s of TPypy, we derive the following.

$$s_3 = ROTL32(s_2 + Y_3[P_3[72]] - Y_3[P_3[239]], ((P_3[116] + 18) \ \& \ 31))$$
$$s_2 = ROTL32(s_1 + Y_2[P_2[72]] - Y_2[P_2[239]], ((P_2[116] + 18) \ \& \ 31))$$

Thus, when conditions C3 through C5 hold,

$$P_2[116] + 18 \equiv -18 + 18 \equiv 0 \ (mod\ 32)$$
$$Y_2[P_2[72]] = Y_2[P_3[239] + 1] = Y_3[P_3[239]]$$
$$Y_2[P_2[239]] = Y_2[P_3[72] + 1] = Y_3[P_3[72]]$$

also hold, so the following relation is obtained.

$$s_3 = ROTL32(s_1, ((P_3[116] + 18) \ \& \ 31)) \tag{5}$$

In the same way, we derive the following from the update equation of variable s of TPypy.

$$s_5 = ROTL32(s_4 + Y_5[P_5[72]] - Y_5[P_5[239]], ((P_5[116] + 18) \ \& \ 31))$$
$$s_4 = ROTL32(s_3 + Y_4[P_4[72]] - Y_4[P_4[239]], ((P_4[116] + 18) \ \& \ 31))$$

Thus, when conditions C6 to C8 hold, the following also hold.

$$P_4[116] + 18 \equiv -18 + 18 \equiv 0 \ (mod\ 32)$$
$$Y_4[P_4[72]] = Y_4[P_5[239] + 1] = Y_5[P_5[239]]$$
$$Y_4[P_4[239]] = Y_4[P_5[72] + 1] = Y_5[P_5[72]]$$

Thus, the following relation is obtained.

$$s_5 = ROTL32(s_3, ((P_5[116] + 18)\ \&\ 31)) \tag{6}$$

In Eq. (5) and Eq. (6), taking the case when $P_3[116] \equiv P_5[116] \equiv -18\ (mod\ 32)$ of the two conditions in C9 holds as condition C9a, the following relation holds.

$$s_5 = s_3 = s_1 \tag{7}$$

Therefore, when condition C9a holds, which is to say when Eq. (7) holds, the following relations are satisfied.

$$s_{3(0)} = s_{1(0)} \tag{8}$$

$$s_{5(18)} = s_{3(18)} \tag{9}$$

Also, taking the case when $P_3[116] \equiv P_5[116] \equiv 0\ (mod\ 32)$ of the two conditions in C9 holds as condition C9b, the following relations hold.

$$s_3 = ROTL32(s_1, 18) \tag{10}$$

$$s_5 = ROTL32(s_3, 18) \tag{11}$$

Therefore, when condition C9b holds, that is to say when both Eq. (10) and Eq. (11) hold, the following relations are satisfied.

$$s_{3(18)} = s_{1(0)} \tag{12}$$

$$s_{5(18)} = s_{3(0)} \tag{13}$$

In other words, when condition C9 holds, either both Eq. (8) and Eq. (9) hold, or both Eq. (12) and Eq. (13) hold, so Eq. (4) becomes as follows in either case.

$$\begin{aligned} Z_{(0)} = {} & Y_3[P_3[153]]_{(0)} \oplus Y_1[P_1[208]]_{(0)} \oplus Y_5[P_5[153]]_{(0)} \oplus Y_3[P_3[208]]_{(0)} \\ & \oplus s_{6(0)} \oplus s_{7(0)} \oplus Y_6[-1]_{(0)} \oplus Y_7[-1]_{(0)} \end{aligned} \tag{14}$$

Also, when condition C10 holds, the following relation is satisfied.

$$s_7 = s_6 + Y_7[P_7[72]] - Y_7[P_7[239]]$$

Thus, Eq. (14) becomes as follows.

$$\begin{aligned} Z_{(0)} = {} & Y_3[P_3[153]]_{(0)} \oplus Y_1[P_1[208]]_{(0)} \oplus Y_5[P_5[153]]_{(0)} \oplus Y_3[P_3[208]]_{(0)} \\ & \oplus Y_6[-1]_{(0)} \oplus Y_7[-1]_{(0)} \oplus Y_7[P_7[72]]_{(0)} \oplus Y_7[P_7[239]]_{(0)} \end{aligned} \tag{15}$$

Finally, when conditions C11 through C14 hold, the following hold.

$$\begin{aligned} Y_1[P_1[208]] &= Y_1[4] = Y_6[-1] \\ Y_3[P_3[208]] &= Y_3[3] = Y_7[-1] \\ Y_3[P_3[153]] &= Y_3[P_7[72] + 4] = Y_7[P_7[72]] \\ Y_5[P_5[153]] &= Y_3[P_7[239] + 2] = Y_7[P_7[239]] \end{aligned}$$

Thus, from Eq. (15), the following relation necessarily holds.

$$Z_{(0)} = O_{1(0)} \oplus O_{3(0)} \oplus O_{6(0)} \oplus O_{7(0)} = 0 \tag{16}$$

This completes the proof. □

3.2 Probability of Distinguisher Existing and Amount of Data Required

In this section, we consider the probability that Eq. (16), which is used as the distinguisher, holds. If the TPypy output sequence is a truly random number sequence, the probability that the Eq. (16) distinguisher holds is 2^{-1}. The probability that Eq. (16) holds depends on the structure of the TPypy PRGA and does not depend on the key schedule or the IV schedule. Therefore, in the following estimation, we take it that variable s and arrays P and Y are independent and follow a uniform distribution after completion of the IV schedule.

First, we consider conditions C3, C6, C9, C10, which concern the number of rotations for updating variable s. When condition C3 holds, the relation $P_2[116] \equiv -18\ (mod\ 32)$ holds, which is to say $P_2[116] \in \{14, 46, 78, 110, 142, 174, 206, 238\}$. Therefore, we take the probability that condition C3 holds to be $Pr[C3]$ and get the following.

$$Pr[C3] = \frac{8}{256}$$

In the same way, when the relation $P_3[116] \equiv -18\ (mod\ 32)$ or $P_3[116] \equiv 0\ (mod\ 32)$ holds, $P_3[116] \in \{14, 46, 78, 110, 142, 174, 206, 238\}$ or $P_3[116] \in \{0, 32, 64, 96, 128, 160, 192, 224\}$. However, it is clear from the TPypy updating equation for array P that the value of $P_2[116]$ can transition only to $P_3[115]$ or $P_3[255]$ as a result of the restriction imposed by condition C3. Considering this restriction condition, the probability that conditions C3, C6, C9, and C10 hold simultaneously can be estimated in the following way according to Bayes' theorem.

$$\begin{aligned} Pr[C3 \cap C6 \cap C9 \cap C10] &= Pr[C3 \cap C6 \cap C9a \cap C10] \\ &\quad + Pr[C3 \cap C6 \cap C9b \cap C10] \\ &= \frac{8}{256} \cdot \frac{7}{255} \cdot \frac{6}{254} \cdot \frac{5}{253} \cdot \frac{4}{252} \\ &\quad + \frac{8}{256} \cdot \frac{8}{255} \cdot \frac{7}{254} \cdot \frac{7}{253} \cdot \frac{6}{252} \end{aligned}$$

Next, consider that an entry of array P has a particular value, and that there is a relationship between the entries of array P. We assumed that these conditions occur independently with probability of approximately 2^{-8}.[1] Also, if for each condition there are multiple patterns for the combinations for which terms cancel in the $Z_{(0)}$ relationship, the number of combinations is also taken into account. Taking conditions C11 and C12 for example, in Eq. (15), $Y_1[P_1[208]]$ and $Y_6[-1]$, and $Y_3[P_3[208]]$ and $Y_7[-1]$ cancel out, but $Y_3[P_3[208]]$ and $Y_6[-1]$, $Y_1[P_1[208]]$ and $Y_7[-1]$ canceling out is another possibility. However, when considering the number of combinations for which the various terms can cancel in the $Z_{(0)}$ relationship equation, the following constraints apply.

[1] Actually, the constraint condition of the previous time applies, but it is difficult to accurately evaluate all of the conditions, so in this work we performed an approximate evaluation.

- For the combinations of $Y_6[-1]$ and $(Y_7[-1], Y_7[P_7[72]], Y_7[P_7[239]])$, the terms cannot cancel.
- For variable pairs at the same time t, the terms cannot cancel.

Also, because conditions C4 and C5, and conditions C7 and C8 are subject to the constraint that Eq. (7) must be satisfied in units of the word because Eq. (9) holds when condition C9a holds, their respective combinations are limited to one case. Taking these constraint conditions into consideration, the numbers of combinations that are possible for the various conditions are listed in Table 1. Specific examples of all of the conditions are given in the appendix A.

Table 1. Numbers of combinations of the conditions

Conditions	Combinations
$C1 \cap C2$	1
$C4 \cap C5 \cap C9a$	1
$C4 \cap C5 \cap C9b$	2
$C7 \cap C8 \cap C9a$	1
$C7 \cap C8 \cap C9b$	2
$C11 \cap C12 \cap C13 \cap C14$	24

Therefore, defining event E for which all of the conditions C1 through C14 hold, the probability that event E holds, $Pr[E]$, is as follows.

$$\begin{aligned}
Pr\left[E\right] &= Pr\left[\bigcap_{i=1}^{14} Ci\right] \\
&= Pr\left[\bigcap_{i=1}^{2} Ci\right] \times \left\{ \begin{array}{l} Pr\left[\left(\bigcap_{i=3}^{8} Ci\right) \cap C9a \cap C10\right] \\ +Pr\left[\left(\bigcap_{i=3}^{8} Ci\right) \cap C9b \cap C10\right] \end{array} \right\} \times Pr\left[\bigcap_{i=11}^{14} Ci\right] \\
&= 1 \cdot \left(\frac{1}{256}\right)^2 \times \left\{ \begin{array}{l} 1 \cdot \left(\frac{1}{256}\right)^2 \times 1 \cdot \left(\frac{1}{256}\right)^2 \times \frac{8}{256} \cdot \frac{7}{255} \cdot \frac{6}{254} \cdot \frac{5}{253} \cdot \frac{4}{252} \\ +2 \cdot \left(\frac{1}{256}\right)^2 \times 2 \cdot \left(\frac{1}{256}\right)^2 \times \frac{8}{256} \cdot \frac{8}{255} \cdot \frac{7}{254} \cdot \frac{7}{253} \cdot \frac{6}{252} \end{array} \right\} \\
&\quad \times 24 \cdot \left(\frac{1}{256}\right)^4 \\
&\approx 2^{-99.04}
\end{aligned}$$

Here, when any of the conditions C1 through C14 are not satisfied, assuming that the probability that Eq. (16) holds is ideally 2^{-1}, the probability that Eq. (16) holds for the TPypy output sequence, $Pr[Z_{(0)} = 0]$, is as follows.

$$Pr\left[Z_{(0)} = 0\right] = Pr\left[Z_{(0)} = 0 \mid E\right] \cdot Pr\left[E\right] + Pr\left[Z_{(0)} = 0 \mid E^c\right] \cdot Pr\left[E^c\right]$$

$$= 1 \cdot 2^{-99.04} + \frac{1}{2} \cdot (1 - 2^{-99.04})$$
$$= \frac{1}{2} \cdot (1 + 2^{-99.04})$$

This is large compared to the 2^{-1} probability for a truly random number sequence.

Here, we regard N samples as independent binary sequences that follow the biased distribution D_{BIAS}. In addition, denoting a uniform distribution as D_{UNI}, the amount of data N required for constructing the optimal distinguisher can be obtained from the following Theorem 2 [2].

Theorem 2. *Taking the input to an optimal distinguisher to be a binary random variable z_i $(0 \le i \le N-1)$ that follows D_{BIAS}, to achieve an advantage greater than 0.5 requires at least the number of samples from the optimal distinguisher derived by the following equation.*

$$N = 0.4624 \times M^2 \qquad \textit{where}$$
$$P_{D_{BIAS}}[z_i = 0] - P_{D_{UNI}}[z_i = 0] = \frac{1}{M}$$

Thus, from Theorem 2, the amount of data N that is required to distinguish the TPypy output sequence from a uniform distribution can theoretically be estimated as $2^{198.96}$. Here, Theorem 1 constructs a distinguisher for $Z_{(0)} = O_{1(0)} \oplus O_{3(0)} \oplus O_{6(0)} \oplus O_{7(0)} = 0$, but it is clear that the same relation holds for any time t $(t \ge 1)$.

$$O_{t(0)} \oplus O_{t+2(0)} \oplus O_{t+5(0)} \oplus O_{t+6(0)} = 0$$

Therefore, when about 2^{199} words of the keystream obtained with multiple arbitrary secret key and IV pairs are collected, it is possible to distinguish the TPypy output sequence from a truly random number sequence. Therefore, the countermeasure of discarding the first few words of the keystream is ineffective against the method described in this paper.

4 Conclusion

We have reported a bias in the output sequence of TPypy, which has the highest security of the Py family of stream ciphers. That bias can be exploited to distinguish the keystream obtained with multiple arbitrary secret key and IV pairs from a truly random number sequence by using about 2^{199} words. This method can also be applied to Pypy in exactly the same way. Furthermore, our method is powerful in that it succeeds with a greatly smaller amount of data that results that have been reported previously.

Acknowledgements. The authors would like to thank the anonymous referees of SAC 2007 whose helpful remarks that improved the paper.

References

1. eSTREAM, the ECRYPT Stream Cipher Project, available at `http://www.ecrypt.eu.org/stream/`
2. Baignères, T., Junod, P., Vaudenay, S.: How Far Can We Go Beyond Linear Cryptanalysis? In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 432–450. Springer, Heidelberg (2004)
3. Biham, E., Seberry, J.: Py (Roo): A Fast and Secure Stream Cipher Using Rolling Arrays. eSTREAM, the ECRYPT Stream Cipher Project, Report 2005/023 (2005)
4. Biham, E., Seberry, J.: Pypy: Another Version of Py. eSTREAM, the ECRYPT Stream Cipher Project, Report 2006/038 (2006)
5. Biham, E., Seberry, J.: Tweaking the IV Setup of the Py Family of Stream Ciphers – The Ciphers TPy, TPypy, and TPy6. eSTREAM, the ECRYPT Stream Cipher Project, Report 2007/038 (2007)
6. Crowley, P.: Improved Cryptanalysis of Py. In: SASC 2006 - Stream Ciphers Revisited, Workshop Record, pp. 52–60 (2006)
7. Isobe, T., Ohigashi, T., Kuwakado, H., Morii, M.: How to Break Py and Pypy by a Chosen-IV Attack. In: SASC 2007 - The State of the Art of Stream Ciphers, Workshop Record, pp. 340–352 (2007)
8. Kogiso, M., Shimoyama, T.: A Distinguishing Attack on the Stream Cipher Pypy. In: Symposium on Cryptography and Information Security - SCIS, IEICE Technical Report, 2A2-2 (2007) (in Japanese)
9. Paul, S., Preneel, B., Sekar, G.: Distinguishing Attacks on the Stream Cipher Py. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 405–421. Springer, Heidelberg (2006)
10. Paul, S., Preneel, B.: On the (In)security of Stream Ciphers Based on Arrays and Modular Addition. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 69–83. Springer, Heidelberg (2006)
11. Schneier, B.: Applied Cryptography, 2nd edn. John Wiley & Sons, Chichester (1996)
12. Sekar, G., Paul, S., Preneel, B.: Weaknesses in the Pseudorandom Bit Generation Algorithms of the Stream Ciphers TPypy and TPy. eSTREAM, the ECRYPT Stream Cipher Project, Report 2007/037 (2007)
13. Wu, H., Preneel, B.: Attacking the IV Setup of Py and Pypy. eSTREAM, the ECRYPT Stream Cipher Project, Report 2006/050 (2006)
14. Wu, H., Preneel, B.: Key Recovery Attack on Py and Pypy with Chosen IVs. eSTREAM, the ECRYPT Stream Cipher Project, Report 2006/052 (2006)
15. Wu, H., Preneel, B.: Differential Cryptanalysis of the Stream Ciphers Py, Py6 and Pypy. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 276–290. Springer, Heidelberg (2007)

A Specific Examples of Conditions

Specific examples of the combinations that are possible for the conditions listed in Table 1 are shown in Table 2, Table 3, and Table 4. For conditions C1 and C2, the single conditions shown by Theorem 1 are omitted.

Table 2. Specific examples of conditions $C4 \cap C5 \cap C9$

$C4$	$C5$	$C9$
$P_2[72] = P_3[239] + 1$	$P_2[239] = P_3[72] + 1$	$P_3[116] \equiv P_5[116] \equiv -18\ (mod\ 32)$
$P_2[72] = P_3[239] + 1$	$P_2[239] = P_3[72] + 1$	$P_3[116] \equiv P_5[116] \equiv 0\ (mod\ 32)$
$P_2[72] = P_3[72] + 1$	$P_2[239] = P_3[239] + 1$	$P_3[116] \equiv P_5[116] \equiv 0\ (mod\ 32)$

Table 3. Specific examples of conditions $C7 \cap C8 \cap C9$

$C7$	$C8$	$C9$
$P_4[72] = P_5[239] + 1$	$P_4[239] = P_5[72] + 1$	$P_3[116] \equiv P_5[116] \equiv -18\ (mod\ 32)$
$P_4[72] = P_5[239] + 1$	$P_4[239] = P_5[72] + 1$	$P_3[116] \equiv P_5[116] \equiv 0\ (mod\ 32)$
$P_4[72] = P_5[72] + 1$	$P_4[239] = P_5[239] + 1$	$P_3[116] \equiv P_5[116] \equiv 0\ (mod\ 32)$

Table 4. Specific examples of conditions $C11 \cap C12 \cap C13 \cap C14$

$C11$	$C12$	$C13$	$C14$
$P_1[208] = 4$	$P_3[208] = 3$	$P_3[153] = P_7[72] + 4$	$P_5[153] = P_7[239] + 2$
$P_1[208] = 4$	$P_3[208] = 3$	$P_3[153] = P_7[239] + 4$	$P_5[153] = P_7[72] + 2$
$P_1[208] = 4$	$P_3[153] = 3$	$P_3[208] = P_7[72] + 4$	$P_5[153] = P_7[239] + 2$
$P_1[208] = 4$	$P_3[153] = 3$	$P_3[208] = P_7[239] + 4$	$P_5[153] = P_7[72] + 2$
$P_1[208] = 4$	$P_5[153] = 1$	$P_3[208] = P_7[72] + 4$	$P_3[153] = P_7[239] + 4$
$P_1[208] = 4$	$P_5[153] = 1$	$P_3[208] = P_7[239] + 4$	$P_3[153] = P_7[72] + 4$
$P_3[208] = 2$	$P_1[208] = 6$	$P_3[153] = P_7[72] + 4$	$P_5[153] = P_7[239] + 2$
$P_3[208] = 2$	$P_1[208] = 6$	$P_3[153] = P_7[239] + 4$	$P_5[153] = P_7[72] + 2$
$P_3[208] = 2$	$P_3[153] = 3$	$P_1[208] = P_7[72] + 6$	$P_5[153] = P_7[239] + 2$
$P_3[208] = 2$	$P_3[153] = 3$	$P_1[208] = P_7[239] + 6$	$P_5[153] = P_7[72] + 2$
$P_3[208] = 2$	$P_5[153] = 1$	$P_1[208] = P_7[72] + 6$	$P_3[153] = P_7[239] + 4$
$P_3[208] = 2$	$P_5[153] = 1$	$P_1[208] = P_7[239] + 6$	$P_3[153] = P_7[72] + 4$
$P_3[153] = 2$	$P_1[208] = 6$	$P_3[208] = P_7[72] + 4$	$P_5[153] = P_7[239] + 2$
$P_3[153] = 2$	$P_1[208] = 6$	$P_3[208] = P_7[239] + 4$	$P_5[153] = P_7[72] + 2$
$P_3[153] = 2$	$P_3[208] = 3$	$P_1[208] = P_7[72] + 6$	$P_5[153] = P_7[239] + 2$
$P_3[153] = 2$	$P_3[208] = 3$	$P_1[208] = P_7[239] + 6$	$P_5[153] = P_7[72] + 2$
$P_3[153] = 2$	$P_5[153] = 1$	$P_1[208] = P_7[72] + 6$	$P_3[208] = P_7[239] + 4$
$P_3[153] = 2$	$P_5[153] = 1$	$P_1[208] = P_7[239] + 6$	$P_3[208] = P_7[72] + 4$
$P_5[153] = 0$	$P_1[208] = 6$	$P_3[208] = P_7[72] + 4$	$P_3[153] = P_7[239] + 4$
$P_5[153] = 0$	$P_1[208] = 6$	$P_3[208] = P_7[239] + 4$	$P_3[153] = P_7[72] + 4$
$P_5[153] = 0$	$P_3[208] = 3$	$P_1[208] = P_7[72] + 6$	$P_3[153] = P_7[239] + 4$
$P_5[153] = 0$	$P_3[208] = 3$	$P_1[208] = P_7[239] + 6$	$P_3[153] = P_7[72] + 4$
$P_5[153] = 0$	$P_3[153] = 3$	$P_1[208] = P_7[72] + 6$	$P_3[208] = P_7[239] + 4$
$P_5[153] = 0$	$P_3[153] = 3$	$P_1[208] = P_7[239] + 6$	$P_3[208] = P_7[72] + 4$