This paper appears under the same title
in the proceedings of *FSE 2006* (Matt Robshaw, ed.), *LNCS*, Springer, 2006, pp. 405-421.

Distinguishing Attacks on the Stream Cipher Py[⋆]

Souradyuti Paul[‡], Bart Preneel[‡], and Gautham Sekar[‡†]

[‡]Katholieke Universiteit Leuven, Dept. ESAT/COSIC,
Kasteelpark Arenberg 10,
B–3001, Leuven-Heverlee, Belgium
[†]Birla Institute of Technology and Science, Pilani, India,
Dept. of Electronics and Instrumentation, Dept. of Physics.
{souradyuti.paul, bart.preneel}@esat.kuleuven.be,
gautham.sekar@gmail.com

Abstract. The stream cipher Py designed by Biham and Seberry is a submission to the ECRYPT stream cipher competition. The cipher is based on two large arrays (one is 256 bytes and the other is 1040 bytes) and it is designed for high speed software applications (Py is more than 2.5 times faster than the RC4 on Pentium III). The paper shows a statistical bias in the distribution of its output-words at the 1st and 3rd rounds. Exploiting this weakness, a distinguisher with advantage greater than 50% is constructed that requires $2^{84.7}$ randomly chosen key/IV's and the first 24 output bytes for each key. The running time and the data required by the distinguisher are $t_{ini} \cdot 2^{84.7}$ and $2^{89.2}$ respectively (t_{ini} denotes the running time of the key/IV setup). We further show that the data requirement can be reduced by a factor of about 3 with a distinguisher that considers outputs of later rounds. In such case the running time is reduced to $t_r \cdot 2^{84.7}$ (t_r denotes the time for a single round of Py). The Py specification allows a 256-bit key and a keystream of 2^{64} bytes per key/IV. As an ideally secure stream cipher with the above specifications should be able to resist the attacks described before, our results constitute an academic break of Py. In addition we have identified several biases among pairs of bits; it seems possible to combine all the biases to build more efficient distinguishers.

Update (29th March 2008). The attack described in this paper for Py also applies as-is to recently proposed strengthened version TPy [4].

[⋆] This work was supported in part by the Concerted Research Action (GOA) Mefisto 2000/06 and Ambiorix 2005/11 of the Flemish Government and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

1 Introduction

The cipher Py, designed by Biham and Seberry [3], was submitted to the ECRYPT project [8] as a candidate for Profile 1 which covers software based stream ciphers suitable for high-speed applications. In the last couple of years a growing interest has been noticed among cryptographers to design fast and secure stream ciphers because of weaknesses being found in many *de facto* standards such as the RC4 and also due to the failure of the NESSIE project [13] to find a stream cipher that met its very stringent security requirements. The current stream cipher, namely Py, is one of the attempts in this direction.

Py is the most recent addition to the class of stream ciphers whose design principles are motivated by that of the RC4 (see [10, 11, 14–16]). Like RC4, Py also uses the technique of random shuffle to update the internal state. In addition, Py uses a new technique of rotating all array elements every round with a minimal running time. The high performance (it is 2.5 times faster than the RC4 on Pentium III) and its apparent security make this cipher very attractive for selection to the Profile 1 of the ECRYPT project.

This paper identifies several biased pairs of output bits of Py at rounds t and $t+2$ (where $t > 0$). The weaknesses originate from the non-uniformity of the distributions of carry bits in modular addition used in Py. Using those biases, we have constructed a class of distinguishers. We show that the best of them works successfully with $2^{84.7}$ randomly chosen key/IV's, the first 24 bytes for each key (i.e., a total of $2^{89.2}$ bytes) and running time $t_{ini} \cdot 2^{84.7}$ where t_{ini} is the running time of the key/IV setup of Py. We also show that a simple adjustment to the above distinguisher reduces the number of key/IV's, the data complexity and the running time to $2^{28.7}$, a total of $2^{87.7}$ bytes and $t_r \cdot 2^{84.7}$ respectively, where t_r is the running time of a single round of Py. Note that the allowable key-size and keystream length of Py are 256 bits and 2^{64} bytes respectively. Therefore, these results imply that – even if our attack has a larger total complexity – Py fails to provide the security level expected from an ideal stream cipher with the parameter sizes of Py. Therefore, we believe that our results present a theoretical break of the cipher; see Sect. 9 for an elaborate discussion on this issue. It is important to note that the weaknesses of Py which are described in this paper, still cannot be implemented in practice in view of the its high time complexity. However, the individual distinguishers open the possibility to combine them in order to generate more efficient distinguishers.

2 Description of Py

Py is a synchronous stream cipher which normally uses a 32-byte key (however, the key can be of any size from 1 byte to 256 bytes) and a 16-byte initial value or IV (IV can also be of any size from 1 byte to 64 bytes). The allowable keystream length per key/IV is 2^{64} bytes. Py works in three phases – a key setup algorithm, an IV setup algorithm and a round function which generates two output-words (each output-word is 4 bytes long). The internal state of Py contains two S-boxes

Y, P and a variable s. Y contains 260 elements each of which is 32 bits long. The elements of Y are indexed by [-3, -2,..., 256]. P is a permutation of the elements of $\{0, ..., 255\}$. The main feature of the stream cipher Py is that the S-boxes are updated like 'rolling arrays' [3]. The technique of 'rolling arrays' means that, in each round of Py, *(i)* one or two elements of the S-boxes are updated (line 1 and 7 of Algorithm 1) and *(ii)* all the elements are cyclically rotated by one position toward the left (line 2 and 8 of Algorithm 1). In our analysis, we have assumed that, after the key/IV setup, Y, P and the variable s are uniformly distributed and independent. Under this assumption we analyzed the round function of Py (or Pseudorandom Bit Generation Algorithm) which is described in Algorithm 1. See [3] for a detailed description of the key/IV setup algorithms.

The inputs to Algorithm 1 are $Y[-3, ..., 256]$, $P[0, ..., 255]$ and s, which are obtained after the key/IV setup. Lines 1 and 2 describe how P is updated and rotated. In the update stage, the 0th element of P is swapped with another element in P, which is accessed indirectly, using $Y[185]$. The next step involves a cyclic rotation by one position, of the elements in P. This implies that the entry in $P[0]$ becomes the entry in $P[255]$ in the next round and the entry in $P[i]$ becomes the entry in $P[i-1]$ ($\forall i \in \{1, 2, ..., 255\}$). Lines 3 and 4 of Algorithm 1 indicate how s is updated and its elements rotated. Here, the '$ROTL32(s, x)$' function implies a cyclic left rotation of s by x bit-positions. The output-words (each 32-bit) are generated in lines 5 and 6. The last two lines of the algorithm explain the update and rotation of the elements of Y. The rotation of Y is carried in the same manner as the rotation of P.

Algorithm 1 Single Round of Py

```
Input: $Y[-3, ..., 256]$, $P[0, ..., 255]$, a 32-bit variable $s$
Output: 64-bit random output
    /*Update and rotate $P$*/
 1: swap ($P[0]$, $P[Y[185] \& 255]$);
 2: rotate ($P$);
    /* Update s*/
 3: $s += Y[P[72]] - Y[P[239]]$;
 4: $s = ROTL32(s, ((P[116] + 18) \& 31))$;
    /* Output 8 bytes (least significant byte first)*/
 5: output $((ROTL32(s, 25) \oplus Y[256]) + Y[P[26]])$;
 6: output $((\quad s \quad \oplus Y[-1]) + Y[P[208]])$;
    /* Update and rotate $Y$*/
 7: $Y[-3] = (ROTL32(s, 14) \oplus Y[-3]) + Y[P[153]]$;
 8: rotate($Y$);
```

2.1 Notation and Convention

As Py uses different types of internal and external states (e.g. integer arrays, 32-bit integer) and they are updated every round, it is important to denote all the states and rounds in a simple but consistent way. In every round of Py, the S-box P and the variable s are updated before the output generation (see Algorithm 1). The other S-box, namely Y, is updated after the output generation.

1. In the beginning of any round i, the components of the internal state are denoted by P_{i-1}, s_{i-1} but Y_i.
2. At the end of any round i, the internal state is updated to P_i, s_i and Y_{i+1}. (If the above two conventions are followed, we have P_i, s_i and Y_i in the formulas for the generation of the output-words in round i (line 5 and 6 of Algorithm 1)).
3. The nth element of the arrays Y_i and P_i, are denoted by $Y_i[n]$ and $P_i[n]$ respectively. The jth bit of $Y_i[n]$, $P_i[n]$ and s_i are denoted by $Y_i[n]_{(j)}$, $P_i[n]_{(j)}$ and $s_{i(j)}$ respectively (following the convention that the least significant bit is the 0th bit).
4. The output-words generated in line 5 and line 6 of Algorithm 1 are referred to as the '1st output-word' and the '2nd output-word' respectively.
5. $O_{l,m}$ denotes the lth ($l \in \{1, 2\}$) output-word generated in the mth round of Py. $O_{l,m(j)}$ denotes the jth bit of $O_{l,m}$. For example, $O_{1,3(5)}$ denotes the 5th bit of the 1st output-word in round 3.
6. The '+' operator denotes *addition modulo* 2^{32} except when it is used to increment elements of P (particularly in expressions of the form $P_i[n] = P_j[m] + 1$, where '+' denotes *addition over* $\mathbb{Z}$). Similarly, '-' and '$\oplus$' denote *subtraction modulo* 2^{32} and bitwise *exclusive-or*.
7. $P[A]$ denotes the probability of occurrence of the event A.
8. $[a, b]$ denotes the set of all integers between a and b including both.
9. A Pseudorandom Bit Generator will be denoted by PRBG.

2.2 Assumption

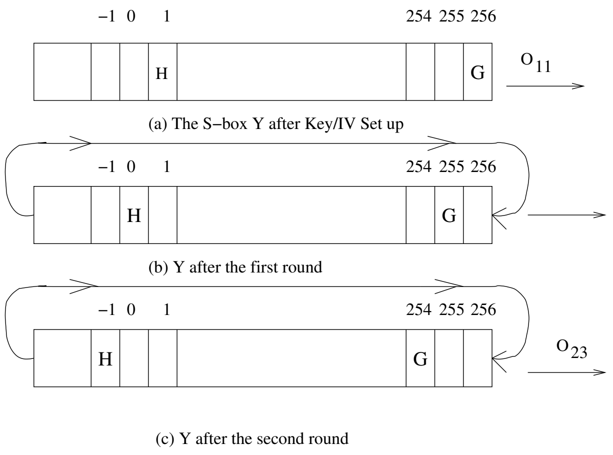
We assume that the key setup and the IV setup algorithms of Py are perfect, i.e., after the execution of them, the permutation P, the elements of Y and the s are uniformly distributed and independent. When we are interested in the analysis of the mixing of bits of the internal state by the PRBG, the above assumption is reasonable, particularly when it is difficult to derive any relation between inputs and outputs of the key/IV setup algorithm. Apart from that the assumption is in agreement with a claim made in Sect. 6.4 of [3] that the key/IV setup leaks no statistical information on the internal state.

3 Motivational Observation

Our main observation is that, if certain conditions on the elements of the S-box P are satisfied then the least significant bit (lsb) of the 1st output-word at the 1st round is equal to the lsb of the 2nd output-word at the 3rd round.

Theorem 1. $O_{1,1(0)} = O_{2,3(0)}$ *if the following six conditions on the elements of the S-box P are simultaneously satisfied.*

1. $P_2[116] \equiv -18 (\bmod 32)$ *(event A),*
2. $P_3[116] \equiv 7 (\bmod 32)$ *(event B),*
3. $P_2[72] = P_3[239] + 1$ *(event C),*
4. $P_2[239] = P_3[72] + 1$ *(event D),*
5. $P_1[26] = 1$ *(event E),*
6. $P_3[208] = 254$ *(event F).*



−1 0 1 254 255 256
H
G
O11
(a) The S−box Y after Key/IV Set up
−1 0 1 254 255 256
H
G
(b) Y after the first round
−1 0 1 254 255 256
H
G
O23
(c) Y after the second round

Fig. 1. (a) $P_1[26] = 1$ (condition 5): G and H are used in $O_{1,1}$, (b) Y_2 (i.e., Y after the 1st round), (c) $P_3[208] = 254$ (condition 6): G and H are used in $O_{2,3}$

Proof. The formulas for the $O_{1,1}$, $O_{2,3}$ and s_2 are given below (see Sect. 2):

$$O_{1,1} = (ROTL32(s_1, 25) \oplus Y_1[256]) + Y_1[P_1[26]]\,, \tag{1}$$
$$O_{2,3} = (s_3 \oplus Y_3[-1]) + Y_3[P_3[208]]\,, \tag{2}$$
$$s_2 = ROTL32(s_1 + Y_2[P_2[72]] - Y_2[P_2[239]], ((P_2[116] + 18) \bmod 32))\,. \tag{3}$$

• Condition 1 (i.e., $P_2[116] \equiv -18 (\bmod 32)$) reduces (3) to

$$s_2 = s_1 + Y_2[P_2[72]] - Y_2[P_2[239]]\,.$$

• Condition 2 (i.e., $P_3[116] \equiv 7 (\bmod 32)$) together with Condition 1 implies

$$s_3 = ROTL32((s_1 + Y_2[P_2[72]] - Y_2[P_2[239]] + Y_3[P_3[72]] - Y_3[P_3[239]]), 25)\,.$$

• Condition 3 and 4 (that is, $P_2[72] = P_3[239] + 1$ and $P_2[239] = P_3[72] + 1$) reduce the previous equation to

$$s_3 = ROTL32(s_1, 25)\,. \tag{4}$$

From (1), (2), (4) we get:

$$O_{1,1} = (ROTL32(s_1, 25) \oplus Y_1[256]) + Y_1[P_1[26]]\,, \quad (5)$$
$$O_{2,3} = (ROTL32(s_1, 25) \oplus Y_3[-1]) + Y_3[P_3[208]]\,. \quad (6)$$

In Fig. 1, conditions 5 and 6 are described. According to the figure,

$$H = Y_1[P_1[26]] = Y_3[-1]\,, \quad (7)$$
$$G = Y_1[256] = Y_3[P_3[208]]\,. \quad (8)$$

Applying (7) and (8) in (5) and (6) we get,

$$O_{1,1(0)} \oplus O_{2,3(0)} = Y_1[256]_{(0)} \oplus Y_1[P_1[26]]_{(0)} \oplus Y_3[-1]_{(0)} \oplus Y_3[P_3[208]]_{(0)} = 0\,.$$

This completes the proof. □

4 Bias in the Distribution of the 1st and the 3rd Outputs

In this section, we shall compute $P[O_{1,1(0)} \oplus O_{2,3(0)} = 0]$ using the results of Sect. 3. We now recall the six events (or conditions) A, B, C, D, E, F as described in Theorem 1. First, we shall compute $P[A \cap B \cap C \cap D \cap E \cap F]$. The elements involved in the calculation of the probability are $P_1[26]$, $P_2[72]$, $P_2[116]$, $P_2[239]$, $P_3[72]$, $P_3[116]$, $P_3[208]$, and $P_3[239]$. Now we observe that Algorithm 1 ensures that all the above elements occupy unique indices in round 1. We calculate the probabilities step by step using Bayes' rule under the assumption described in Sect. 2.2.

1. $P[E] = \frac{1}{256}$,
2. $P[E \cap F] = P[F|E] \cdot P[E] = \frac{1}{255} \cdot \frac{1}{256}$,
3. $P[A \cap E \cap F] = P[A|E \cap F] \cdot P[E \cap F] = \frac{8}{254} \cdot \frac{1}{255} \cdot \frac{1}{256}$,
4. $P[A \cap B \cap E \cap F] = P[B|A \cap E \cap F] \cdot P[A \cap E \cap F] = \frac{8}{253} \cdot \frac{8}{254} \cdot \frac{1}{255} \cdot \frac{1}{256}$,
5. Similarly, $P[A \cap B \cap C \cap E \cap F] = \frac{247}{251 \cdot 252} \cdot \frac{8}{253} \cdot \frac{8}{254} \cdot \frac{1}{255} \cdot \frac{1}{256}$,
6. $P[A \cap B \cap C \cap D \cap E \cap F] \approx \frac{244}{249 \cdot 250} \cdot \frac{247}{251 \cdot 252} \cdot \frac{8}{253} \cdot \frac{8}{254} \cdot \frac{1}{255} \cdot \frac{1}{256} \approx 2^{-41.9}$.

Under the assumption of randomness and uniformity of the distributions of the S-box elements and of s after the key/IV setup, if any of the six events – described in Theorem 1 – does not occur then $P[O_{1,1(0)} \oplus O_{2,3(0)} = 0] = \frac{1}{2}$ (see Appendix A for a justification for that). That is,

$$P[O_{1,1(0)} \oplus O_{2,3(0)} = 0 | (A \cap B \cap C \cap D \cap E \cap F)^c] = \frac{1}{2}\,.$$

We denote the event $A \cap B \cap C \cap D \cap E \cap F$ by L and its complement by L^c. Therefore,

$$\begin{aligned} P[O_{1,1(0)} \oplus O_{2,3(0)} = 0] &= P[O_{1,1(0)} \oplus O_{2,3(0)} = 0 | L] \cdot P[L] \\ &\quad + P[O_{1,1(0)} \oplus O_{2,3(0)} = 0 | L^c] \cdot P[L^c] \\ &= 1 \cdot 2^{-41.91} + \frac{1}{2} \cdot (1 - 2^{-41.91}) \\ &= \frac{1}{2} \cdot (1 + 2^{-41.91}) \,. \end{aligned} \tag{9}$$

Note that, if Py had been an ideal PRBG then the above probability would have been exactly $\frac{1}{2}$.

5 The Distinguisher

A *distinguisher* is an algorithm which distinguishes a stream of bits from a perfectly random stream of bits, that is, a stream of bits that has been chosen according to the uniform distribution. There are several ways a cryptanalyst may try to distinguish between a string, generated by an insecure pseudorandom bit generator, and one from a perfectly random source. In one case, she selects a *single* key/IV randomly and produce keystream, seeded by the chosen key/IV, long enough to distinguish it from random with high success probability. This attack scenario is rather common and such *distinguisher* is called a *regular distinguisher*. In a different scenario, to build a distinguisher, the adversary may use *many* randomly chosen key/IV's rather than a single key and a few *specified* bytes from each of the keystreams generated by those key/IV's. The *distinguisher*, so constructed, is called a *prefix distinguisher*. A bias present in the output at time t in a single stream may hardly be detected by a *regular distinguisher* but a *prefix distinguisher* can easily discover the anomaly with a few bytes. This fact was nicely demonstrated by Mantin and Shamir [12] to detect a strong bias toward zero in the second output byte of RC4. In addition to that, there exist *hybrid distinguishers* that may fall between the above two extreme cases, that is, the adversary may use *many* key/IV's and for each key/IV she collects *long* keystream. The idea of constructing distinguishers using *many* randomly chosen key/IV's has been a well studied subject. Goldreich has shown that a distribution which is *computationally indistinguishable* from the *uniform distribution* based on a *single sample* is also *computationally indistinguishable* from the *uniform distribution* based on *multiple samples* [9].

The distinguishers that we construct in this section and Sect. 6, using the bias described in Sect. 4, are *prefix distinguishers*. In Sect. 7, we build a *regular distinguisher*; however, the number of outputs needed for this distinguisher exceeds the allowable keystream length per key/IV. In Section 8, we propose a *hybrid distinguisher* mainly to reduce the time cost of our *prefix distinguisher*.

Algorithm 2. The *prefix distinguisher* that separates Py from random is described in Algorithm 2. The input to the algorithm is a realization of the

Algorithm 2 *A Distinguisher* separating Py from Random

```
Input: An $n$-bit sequence $(z_1, z_2, z_3, \cdots, z_n)$
Output: Whether the sequence is random or generated by Py
 1: Compute LLR $= \sum_i \log(\frac{P_0[z_i]}{P_1[z_i]})$;
 2: If LLR $\geq 0$ then return 1 (i.e., "The sequence is from Py")
      else 0 (i.e., "The sequence is random");
```

sequence of binary random variables $(z_1, z_2, z_3, \cdots, z_n)$. The adversary first generates n key/IV pairs $X_1, X_2, X_3, \cdots, X_n$ randomly and then computes $z_i = O_{1,1(0)} \oplus O_{2,3(0)}$ for all X_i, $1 \leq i \leq n$. Using the results obtained by Baignères, Junod and Vaudenay [1], it can be shown that Algorithm 2 is an *optimal distinguisher*. Given a fixed number of samples, an *optimal distinguisher* attains the *maximum advantage*. Note that the random variables z_i's are independent of each other and each of them follows the distribution computed in Sect. 4 (call the distribution D_0). Let the uniform distribution on alphabet $[0, 1]$ be denoted by D_1. In Algorithm 2, $P_0[z_i]$ (shorthand for $P_{\mathsf{D}_0}[z_i]$) denotes the probability of occurrence of z_i when chosen according to D_0 (similarly $P_1[z_i]$ and $P_{\mathsf{D}_1}[z_i]$).

Let the Algorithm 2, the sequence of variables $(z_1, z_2, z_3, \cdots, z_n)$ and the quantity $\sum_i \log(\frac{P_0[z_i]}{P_1[z_i]})$ be denoted by $\mathcal{F}$, Z and LLR respectively. Now we will compute the *advantage* of $\mathcal{F}$ (the advantage of this distinguisher has been independently calculated by Paul Crowley [6]). The *advantage* of a distinguisher – a measure indicating the efficiency of an algorithm to distinguish a distribution from another – is given by the following formula [1]:

$$\mathrm{Adv}^n_{\mathcal{F}} = \left| P_{\mathsf{D}_0^n}[\mathcal{F}(Z) = 1] - P_{\mathsf{D}_1^n}[\mathcal{F}(Z) = 1] \right| . \tag{10}$$

Following the results in [1], it can be shown that for large n,

$$P_{\mathsf{D}_0^n}[\mathcal{F}(Z) = 1] = P_{\mathsf{D}_0}[\mathrm{LLR} \geq 0] \approx \Phi\left(\frac{\sqrt{n}\mu_0}{\sigma_0}\right) ,$$
$$P_{\mathsf{D}_1^n}[\mathcal{F}(Z) = 1] = P_{\mathsf{D}_1}[\mathrm{LLR} \geq 0] \approx \Phi\left(\frac{\sqrt{n}\mu_1}{\sigma_1}\right) .$$

where Φ is the standard normal distribution function expressed as,

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{1}{2}u^2}\, du.$$

If the two distributions D_0 and D_1 are close (i.e.,$\left|P_0[z] - P_1[z]\right| \ll P_1[z]$) then

$$\mu_0 \approx -\mu_1 \approx \frac{1}{2} \sum_{z \in [0,1]} \frac{(P_0[z] - P_1[z])^2}{P_1[z]} \text{ and } \sigma_0^2 \approx \sigma_1^2 \approx \sum_{z \in [0,1]} \frac{(P_0[z] - P_1[z])^2}{P_1[z]} .$$

The above equations suggest that, for a given n, using the known distributions D_0 and D_1, the *advantage* of Algorithm 2 can be computed from (10). Some simple calculations show that, if $P_0[0] - P_1[0] = \frac{1}{M}$, then, to ensure the *advantage* of the distinguisher to be greater than 0.5, the required number of samples is

$$n = 0.4624 \cdot M^2 \,. \tag{11}$$

In the present case $P_0[0] - P_1[0] = \frac{1}{2^{42.9}}$ (see Sect. 4). Therefore, from (11), $n = 0.4624 \cdot (2^{42.9})^2 = 2^{84.7}$ samples (i.e., as many randomly chosen key/IV's) can distinguish Py from random with an advantage that exceeds 0.5. The time cost to build this distinguisher is $t_{ini} \cdot 2^{84.7}$ where t_{ini} is the running time of the key/IV setup of Py. Note that, for each key/IV, we collect the first 24 bytes of the keystream. Therefore, the number of bytes required to establish the distinguisher is $2^{84.7} \cdot 24 = 2^{89.2}$.

6 Biases among other Pairs of Bits and Distinguishers

In Sect. 4, we have showed a bias in $(O_{1,1(0)}, O_{2,3(0)})$. In this section, we show that the bias is present in $(O_{1,1(i)}, O_{2,3(i)})$, where $0 \leq i \leq 31$; however, the bias gradually reduces as i increases. From (1) and (2), we get:

$$\begin{aligned} O_{1,1(i)} &= ROTL32(s_1, 25)_{(i)} \oplus Y_1[256]_{(i)} \oplus Y_1[P_1[26]]_{(i)} \oplus c_{1(i)} \,, \\ O_{2,3(i)} &= s_{3(i)} \oplus Y_3[-1]_{(i)} \oplus Y_3[P_3[208]]_{(i)} \oplus c_{3(i)} \,, \end{aligned}$$

where $0 \leq i \leq 31$ and c_1, c_3 are the carry terms in (1) and (2) respectively.

A Special Case. If all the 6 conditions of Theorem 1 are satisfied, $O_{1,1}$ and $O_{2,3}$ can be written in the following form (see Theorem 1):

$$O_{1,1} = (S \oplus G) + H \,, \tag{12}$$
$$O_{2,3} = (S \oplus H) + G \,, \tag{13}$$

which implies that

$$O_{1,1(i)} \oplus O_{2,3(i)} = c_{1(i)} \oplus c_{3(i)}, \qquad 0 \leq i \leq 31 \,,$$

where the carries $c_{1(i)}$ and $c_{3(i)}$ can be calculated from the following recursive relations (note that $c_{1(0)} = c_{3(0)} = 0$),

$$\begin{aligned} c_{1(i)} &= c_{1(i-1)}(S_{(i-1)} \oplus G_{(i-1)}) \oplus c_{1(i-1)} H_{(i-1)} \oplus \\ &\qquad H_{(i-1)}(S_{(i-1)} \oplus G_{(i-1)}) \,, \end{aligned} \tag{14}$$
$$\begin{aligned} c_{3(i)} &= c_{3(i-1)}(S_{(i-1)} \oplus H_{(i-1)}) \oplus c_{3(i-1)} G_{(i-1)} \oplus \\ &\qquad G_{(i-1)}(S_{(i-1)} \oplus H_{(i-1)}) \,. \end{aligned} \tag{15}$$

Computing $P[O_{1,1(i)} \oplus O_{2,3(i)} = 0]$. Note that

$$\begin{aligned}
P[O_{1,1(i)} \oplus O_{2,3(i)} = 0] &= P[O_{1,1(i)} \oplus O_{2,3(i)} = 0|L] \cdot P[L] \\
&+ P[O_{1,1(i)} \oplus O_{2,3(i)} = 0|L^c] \cdot P[L^c] \\
&= \underbrace{P[c_{1(i)} \oplus c_{3(i)} = 0|L]}_{p_i} \cdot P[L] \\
&+ \underbrace{P[O_{1,1(i)} \oplus O_{2,3(i)} = 0|L^c]}_{X_i} \cdot P[L^c] \,, \qquad (16)
\end{aligned}$$

where $i \in [0, 31]$ and the event L is $A \cap B \cap C \cap D \cap E \cap F$. Note that four components are involved in (16); they are $P[L]$, $P[L^c]$, p_i and X_i. Next, we show how to determine these four quantities.

1,2. Computing $P[L]$ and $P[L^c]$: the results in Sect. 4 show that $P[L] = 2^{-41.9}$ and $P[L^c] = (1 - 2^{-41.9})$.

3. Computing p_i: now we recursively compute $P[c_{1(i)} \oplus c_{3(i)} = 0|L]$, denoted by p_i in (16) (similarly p_{i-1} should be understood), from the following equation derived directly from (14) and (15).

$$\begin{aligned}
c_{1(i)} \oplus c_{3(i)} = (c_{1(i-1)} \oplus c_{3(i-1)})(S_{(i-1)} \oplus G_{(i-1)} \oplus H_{(i-1)}) \oplus \\
S_{(i-1)}(G_{(i-1)} \oplus H_{(i-1)}) \,. \qquad (17)
\end{aligned}$$

Note that the variables G, H, S are uniformly distributed and independent. The truth table for (17) is shown in Table 1. From Table 1, using Bayes' rule, we obtain the following recursion to compute p_i,

$$p_i = \frac{p_{i-1}}{2} + \frac{1}{4} \,.$$

We already know that $p_0 = 1$ (i.e., $P[O_{1,1(0)} \oplus O_{2,3(0)} = 0|L] = 1$). Therefore, solving the above recurrence relation, finally we get

$$p_i = \frac{1}{2} + \frac{1}{2^{i+1}} \,, \quad 0 \le i \le 31 \,. \qquad (18)$$

4. Computing X_i: according to the results obtained in Appendix A it is reasonable to assume that

$$X_i = \frac{1}{2} \,, \qquad \text{for all } i \in [0, 24] \cup [26, 31] \,.$$

General Expression. Using the above results, recalling (16), we find,

$$P[O_{1,1(i)} \oplus O_{2,3(i)} = 0] = \frac{1}{2}(1 + 2^{-(41.9+i)}) \,, \qquad (19)$$

where $i \in [0, 24] \cup [26, 31]$. It is also reasonable to assume (due to the event L' as described in Appendix A) that

$$\begin{aligned}
P[O_{1,1(25)} \oplus O_{2,3(25)} = 0] &\ge \frac{1}{2}(1 + 2^{-(41.9+25)}) \\
&\ge \frac{1}{2}(1 + 2^{-66.9}) \,.
\end{aligned}$$

Table 1. Truth table for (17). The last column in each row indicates the probability of the occurrence of that row

$c_{1(i-1)} \oplus c_{3(i-1)}$	$S_{(i-1)}$	$B_{(i-1)}$	$A_{(i-1)}$	$c_{1(i)} \oplus c_{3(i)}$	Probability
0	0	0	0	0	$p_{i-1}/8$
0	0	0	1	0	$p_{i-1}/8$
0	0	1	0	0	$p_{i-1}/8$
0	0	1	1	0	$p_{i-1}/8$
0	1	0	0	0	$p_{i-1}/8$
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	0	$p_{i-1}/8$
1	0	0	0	0	$(1-p_{i-1})/8$
1	0	0	1	1	
1	0	1	0	1	
1	0	1	1	0	$(1-p_{i-1})/8$
1	1	0	0	1	
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	1	

From (19), one may see that $P[O_{1,1(i)} \oplus O_{2,3(i)} = 0]$ attains the maximum value if $i = 0$. Our distinguisher, described in Sect. 4 and 5, exploits the case if $i = 0$. Equation (19) suggests that many distinguishers can be generated using different $(O_{1,1(i)}, O_{2,3(i)})$'s rather than only $(O_{1,1(0)}, O_{2,3(0)})$, however, the amount of bias decreases as i increases (i.e., we get the most effective distinguisher if $i = 0$). For example, if $i = 1$,

$$P[O_{1,1(1)} \oplus O_{2,3(1)} = 0] = \frac{1}{2}(1 + 2^{-42.91})\,.$$

For the above case, taking the 1st bits of $O_{1,1}$ and $O_{2,3}$, the number of samples (i.e., the number of key/IV's) required to establish a distinguisher with advantage exceeding 0.5 is $2^{86.7}$ (see (11)). Similarly, if we consider $i = 2$ then the number of required samples is $2^{88.7}$.

7 Generalizing the Bias at Rounds t and $t + 2$: A Distinguisher Using a Single Keystream

Under assumptions similar to those in Sect. 2.2, the results of Sect. 3 and Sect. 4 are valid even if we consider any rounds t and $t+2$ ($t > 0$) instead of just rounds 1 and 3. In other words, instead of $(O_{1,1(0)}, O_{2,3(0)})$, one can show that the bias exists even in the distribution of $(O_{1,t(i)}, O_{2,(t+2)(i)})$. Now, we state a theorem which is the generalized version of Theorem 1.

Theorem 2. $O_{1,t(0)} = O_{2,(t+2)(0)}$ *if the following six conditions on the elements of the S-box P are simultaneously satisfied.*

1. $P_{t+1}[116] \equiv -18 (\bmod 32)$,
2. $P_{t+2}[116] \equiv 7 (\bmod 32)$,
3. $P_{t+1}[72] = P_{t+2}[239] + 1$,
4. $P_{t+1}[239] = P_{t+2}[72] + 1$,
5. $P_t[26] = 1$,
6. $P_{t+2}[208] = 254$.

Using the above theorem and the techniques used before, it is easy to show that (see (9))

$$P[O_{1,t(0)} \oplus O_{2,(t+2)(0)} = 0] = \frac{1}{2}(1 + 2^{-41.91}) .$$

The fact that the above probability is valid, $\forall\, t > 0$, allows us to generate a *regular distinguisher* with the number of rounds $2^{84.7}$ of a *single keystream* (see Sect. 5 for a definition of a *regular distinguisher*). This means that $2^{84.7} \times 2^3 = 2^{87.7}$ bytes of *a single* stream generated by a randomly chosen key/IV are sufficient to distinguish Py from random with success probability greater than 0.5. The work-load here is also comparable to $2^{87.7}$. However, this attack is rendered ineffective because the amount of required bytes falls outside the allowable keystream length of 2^{64} bytes.

8 A More Efficient Hybrid Distinguisher

The results of Sect. 5 and Sect. 7 lead us in a natural way to build a *hybrid* distinguisher by making a trade-off between the number of key/IV's and output bytes per key/IV. It is apparent from the previous discussion that, to realize our distinguisher, we need $2^{84.7}$ pairs of internal states (recall that the internal state of Py consists of the arrays P, Y and a 32-bit integer s) with each pair being separated by one round. Then, under the assumption that the first state of each pair is randomly generated, those pairs can be used to build a distinguisher. As the allowable number of rounds per key/IV is $2^{64-8} = 2^{56}$, the number of required key/IV's is $2^{84.7-56} = 2^{28.7}$ to construct this *hybrid distinguisher*. The main difference between the *prefix distinguisher* in Sect. 5 and this *hybrid distinguisher* is that the running time to build this *hybrid distinguisher* is much smaller, as it requires the key/IV setup to run only for $2^{28.7}$ times compared to $2^{84.7}$ times for the previous *prefix distinguisher*. Therefore, the time and the data complexity of this distinguisher are $t_r \cdot 2^{84.7}$ and $2^{87.7}$ bytes respectively, where t_r is the running time of a single round of Py. Furthermore, this *hybrid* distinguisher does not breach the cipher specifications.

9 Do Our Distinguishers Break the Cipher Py?

The subject of what constitutes a break of a *practical stream cipher* or a PRBG is a highly contentious issue even if the area is quite well developed in theory. Theoretically, a cryptographically strong pseudorandom bit generator (CSPRBG) is an algorithm $\mathcal{A}$ that, on being given a random seed k as input, generates a sequence of pseudo-random bits a_1, a_2, $a_3, \cdots$. The function $\mathcal{A}$ possesses the following properties (see Blum and Micali [5]):

1. Each bit a_i can be produced in time polynomial in the length of seed k.
2. Given the algorithm $\mathcal{A}$ and the first s output bits generated by an unknown seed k, it is *computationally infeasible* to predict the $s+1$st bit with biased probability. The s is polynomial in the length of the seed k.

We see that, theoretically, a PRBG is studied according to how it behaves when the length of *seed* is increased asymptotically. The major problem in fitting the analyses of practical stream ciphers into the above framework is that, most of the ciphers work with *fixed sized* keys and keystream bits (e.g. Py allows 256-bit key and 2^{64} bytes of keystream per key/IV pair). Such constraints make the asymptotic analyses of practical stream ciphers impossible. For a practical PRBG with a *fixed sized key* (such as Py), given the first s output bits generated by an unknown key/IV, the $s+1$st bit can be predicted with a high probability with running time bounded above by a trivial exhaustive search. As there is no non-trivial upper bound on the running time of a distinguishing attack on a stream cipher (or PRBG) with a fixed sized key, any legal distinguishing attack with running time less than exhaustive search constitutes an academic break of the cipher.[1] Therefore, our attacks from Sect. 5, Sect. 6 and Sect. 8 imply a theoretical break of Py. However, it should be noted that each of the attacks presented in the paper requires a workload larger than 2^{85} and therefore, poses no practical threats to the cipher.

Do our distinguishing attacks on Py violate the designers' claims?

The stream cipher Py is claimed by the designers to have up to 256-bit security (see Appendix A of [3]). In the authors' words, "The security claims are for keys up to 256 bits (32 bytes) and IVs up to 128 bits (16 bytes)". 256-bit is also the category of security level under which Py is included in the ECRYPT project [7]. According to the discussion on the definition of *n-bit security* of a perfectly secure stream cipher, it is clear that this claim is compromised by our attacks.

However, in Sect. 6.1 of [3], the authors claim,"There are no distinguishing attacks that succeed given less than 2^{64} bytes of key stream with a complexity less than of exhaustive search." It is understood from [2], that those 2^{64} bytes, as mentioned in the claim, may be generated by many keys rather than a single key. Under this interpretation, our attacks do not violate this claim, since our best attack requires $2^{87.7}$ bytes of output.

As a result we conclude that two claims, mentioned above, contradict each other with respect to the attacks mentioned in this paper. At this point, we leave it to the reader to decide on the implications of our distinguishers.

[1] A *legal distinguishing attack* is the one which does not violate the specified parameters of the cipher.

10 Future Work

One could try to combine the individual biases of the pairs of bits presented here to develop a more sophisticated distinguisher with fewer output bytes. Paul Crowley has reduced the time and output bytes of our distinguisher to 2^{72} each, by analyzing our observation in Sect. 3 using a Hidden Markov Model [6]. A plausible strategy consists of identifying many more correlations between internal and external states of Py in order to reduce the time and data complexity of the distinguisher.

11 Conclusion and Remarks

The paper presented several weaknesses of the stream cipher Py. We discovered a class of distinguishers for the cipher, the best of which works with $2^{87.7}$ bytes and comparable time. We also showed that the output stream of Py with a recommended keystream length of 2^{64} bytes, contains biases at different points – this fact can be exploited to build more effective distinguishers. These results break the cipher Py academically. However, the data complexity for the best distinguishing attack falls well beyond the time complexity what is feasible today. Therefore, these weaknesses pose no practical threat to the security of the cipher at this moment. However, the shortened version of Py, known as Py6, may contain more serious weaknesses than the ones described here, but the complete description of Py6 has not been provided in [3].

Acknowledgments

We thank Paul Crowley for providing us with useful references. It is a great pleasure to thank Eli Biham for his comments on our work. Thanks are due to Hongjun Wu and Jongsung Kim for numerous valuable discussions. We also acknowledge the constructive comments of the anonymous reviewers of FSE'06.

References

1. T. Baignères, P. Junod and S. Vaudenay, "How Far Can We Go Beyond Linear Cryptanalysis?," *Asiacrypt 2004* (P. Lee, ed.), vol. 3329 of *LNCS*, pp. 432–450, Springer-Verlag, 2004.
2. E. Biham, Personal Communication, Dec. 2005.
3. E. Biham, J. Seberry, "Py (Roo): A Fast and Secure Stream Cipher using Rolling Arrays," eSTREAM, ECRYPT Stream Cipher Project, Report 2005/023, 2005.
4. E. Biham, J. Seberry, "Tweaking the IV Setup of the Py Family of Ciphers – The Ciphers Tpy, TPypy, and TPy6," Published on the author's webpage at `http://www.cs.technion.ac.il/ biham/`, January 25, 2007.
5. M. Blum, S. Micali, "How to Generate Cyptographically Strong Sequence of Psudo-random Bits," *Siam Journal of Computing*, vol. 13, No. 4, pp. 850–864, November 1984.

6. P. Crowley, "Improved Cryptanalysis of Py," *Workshop Record of SASC 2006 – Stream Ciphers Revisited*, ECRYPT Network of Excellence in Cryptology, February 2006, Leuven (Belgium), pp. 52–60.
7. Daniel. J. Bernstein, "Comparison of 256-bit stream ciphers at the beginning of 2006," *Workshop Record of SASC 2006 – Stream Ciphers Revisited*, ECRYPT Network of Excellence in Cryptology, pp. 70–83.
8. Ecrypt, http://www.ecrypt.eu.org.
9. O. Goldreich, "Lecture Notes on Pseudorandomness–Part-I," Department of Computer Science, Wiezmann Institute of Science, Rehovot, Israel, January 23, 2001.
10. G. Gong, K. C. Gupta, M. Hell, Y. Nawaz, "Towards a General RC4-Like Keystream Generator," *First SKLOIS Conference, CISC 2005* (D. Feng, D. Lin, M. Yung, eds.), vol. 3822 of *LNCS*, pp. 162–174, Springer-Verlag, 2005.
11. Robert J. Jenkins Jr., "ISAAC," *Fast Software Encryption 1996* (D. Gollmann, ed.), vol. 1039 of *LNCS*, pp. 41–49, Springer-Verlag, 1996.
12. I. Mantin, A. Shamir, "A Practical Attack on Broadcast RC4," *Fast Software Encryption 2001* (M. Matsui, ed.), vol. 2355 of *LNCS*, pp. 152–164, Springer-Verlag, 2001.
13. NESSIE: New European Schemes for Signature, Integrity and Encryption, http://www.cryptonessie.org.
14. Souradyuti Paul, Bart Preneel, "A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher," *Fast Software Encryption 2004* (B. Roy, ed.), vol. 3017 of *LNCS*, pp. 245–259, Springer-Verlag, 2004.
15. Hongjun Wu, "A New Stream Cipher HC-256," *Fast Software Encryption 2004* (B. Roy, ed.), vol. 3017 of *LNCS*, pp. 226–244, Springer-Verlag, 2004.
16. Bartosz Zoltak, "VMPC One-Way Function and Stream Cipher," *Fast Software Encryption 2004* (B. Roy, ed.), vol. 3017 of *LNCS*, pp. 210–225, Springer-Verlag, 2004.

A Uniformity of Bits If L Does Not Occur

We first write the general formula to calculate $Z = O_{1,1} \oplus O_{2,3}$.

$$O_{1,1} = (ROT32(s, 25) \oplus G) + H\,, \tag{20}$$
$$O_{2,3} = (ROT32(ROT32(s + I - J\,, r) + K - L\,, l) \oplus M) + N\,, \tag{21}$$

where
$s = s_1$, $G = Y_1[256]$, $H = Y_1[P_1[26]]$, $I = Y_2[P_2[72]]$, $J = Y_2[P_2[239]]$, $r = P_2[116] + 18 \bmod 32$, $K = Y_3[P_3[72]]$, $L = Y_3[P_3[239]]$, $l = P_3[116] + 18 \bmod 32$, $M = Y_3[-1]$, $N = Y_3[P_3[208]]$.

Below we isolate 18 cases, divided into 4 groups, where the relation between internal and external states is not trivial. The symbol '/' is used to mean 'or'. Note that the equalities in each group are satisfied if they do not violate the condition of uniqueness of permutation elements of S-box P. The notation $A \Leftrightarrow B$ signifies that the A and B are identical elements in two different rounds of the S-box Y (i.e., $A = B$ but their indices may be changed in different rounds).

1. $I \Leftrightarrow N/M$, $J \Leftrightarrow M/N$, $K \Leftrightarrow G/H$, $L \Leftrightarrow H/G$ (a total of 4 cases). See Fig.2.

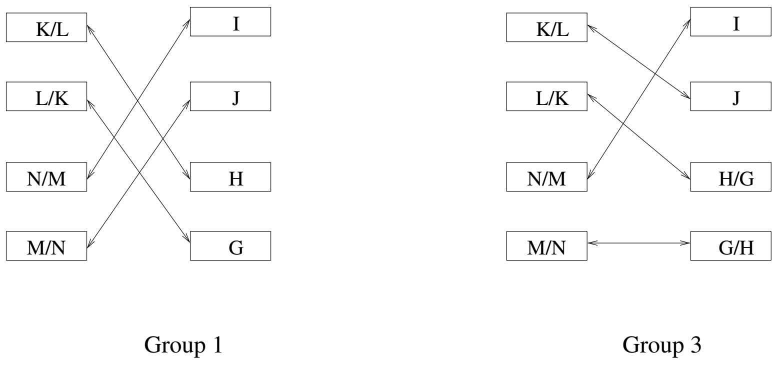


Fig. 2.

2. $I \Leftrightarrow K/L$, $J \Leftrightarrow L/K$, $M \Leftrightarrow H$, $N \Leftrightarrow G$ (a total of 2 cases). See Fig.3.
3. $I \Leftrightarrow N/M$, $J \Leftrightarrow K/L$. The G is identical to one of the remaining two elements (so is the H) (a total of 6 cases). See Fig.2.
4. Similar to the above, $J \Leftrightarrow M/N$, $I \Leftrightarrow L/K$. The G is identical to one of the remaining two elements (so is the H) (a total of 6 cases).

Fact 1 *After the key/IV setup, if the permutation P falls outside all of the 18 cases described above then $O_{1,1}$ and $O_{2,3}$ are independent and uniformly distributed over $[2^{32}-1, 0]$.*

Now we denote $O_{1,1(0)} \oplus O_{2,3(0)}$ by R_0.

Theorem 3. *After the key/IV setup, let the S-box P be one of the 16 cases described in Groups 1,3 and 4. Then*

$$Prob[R_0 = 0 \,|\, P] = \frac{1}{2}.$$

Proof. Now we prove the theorem by considering Groups 1, 3 and 4 separately. **Group 1** (see Fig. 2). For this group, R_0 can be written in the following form,

$$R_0 = s_{(7)} \oplus s_{(w)} \oplus (G_{(0)} \oplus H_{(0)} \oplus K_{(u)} \oplus L_{(u)}) \\ \oplus (I_{(w)} \oplus J_{(w)} \oplus M_{(0)} \oplus N_{(0)}) \oplus C.$$

Note that the C is a nonlinear function of several bits of s, M, N, H, G. Now we take three possible subcases.

1. If $u \neq 0$ then R_0 is uniformly distributed since C is independent of $K_{(u)}$ and $L_{(u)}$.
2. If $u = 0$, $w \neq 0$ then R_0 is uniformly distributed since C is independent of $I_{(w)}$ and $J_{(w)}$.

3. If $u = 0$, $w = 0$ then $R_0 = s_{(7)} \oplus s_{(0)}$. Therefore, R_0 is uniformly distributed.

Group 3 (see Fig. 2). R_0 can be written in the following form,

$$\begin{aligned} R_0 = s_{(7)} \oplus s_{(w)} \oplus (G_{(0)} \oplus N_{(0)}) \oplus (K_{(u)} \oplus J_{(w)}) \\ \oplus (H_{(0)} \oplus L_{(u)}) \oplus (M_{(0)} \oplus I_{(w)}) \oplus C. \end{aligned}$$

Of the 6 cases in Group 3, we are considering *only* the following case where $I \Leftrightarrow M$, $J \Leftrightarrow K$, $G \Leftrightarrow N$ and $H \Leftrightarrow L$. In a similar way as above we divide this case into three subcases.

1. If $u \neq 0$ then C is independent of $L_{(u)}$ and thus R_0 is uniformly distributed.
2. If $u = 0$, $w \neq 0$ then R_0 is uniformly distributed since C is independent of $J_{(w)}$.
3. If $u = 0$, $w = 0$ then $R_0 = s_{(7)} \oplus s_{(0)}$. Therefore, R_0 is uniformly distributed.

All the other 5 cases of this group can be proved in a similar fashion.
Group 4. Proof for this group is similar to that for Group 3. $\square$

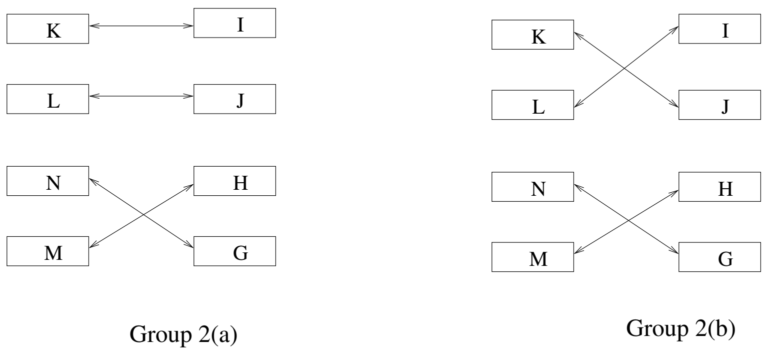


Fig. 3. Group 2(a): $I \Leftrightarrow K$, $J \Leftrightarrow L$, $M \Leftrightarrow H$, $N \Leftrightarrow G$; Group 2(b): $I \Leftrightarrow L$, $J \Leftrightarrow K$, $M \Leftrightarrow H$, $N \Leftrightarrow G$

Discussion. From Fact 1 and Theorem 3, it is clear that, if P does not fall within Group 2 then $Prob[R_0 = 0|P] = \frac{1}{2}$. The probability of the occurrence of Group 2 is approximately 2^{-31}. Therefore, for a fraction of $(1 - 2^{-31})$ of all cases, R_0 is uniformly distributed. Sect. 4 shows that, for the event L occurring with probability $2^{-41.9}$, $Prob[R_0 = 0 \,|\, P = L] = 1$.

Therefore, we are able to prove that, for a fraction of $(1 - 2^{-31.001})$ of cases, there exists a bias in R_0 toward zero. It is, however, nontrivial to determine the distribution of R_0 for the remaining fraction of $2^{-31.001}$ of the cases, because of vigorous mixing of bits in a nonlinear way. Our experiments suggest that it is very unlikely that the positive bias generated in a large fraction of $(1 - 2^{-31.01})$

of the cases can be compensated by a very minuscule fraction of $2^{-31.001}$ of them. According to a small number of experiments that we carried out, a slight bias toward zero was detected for that remaining fraction of $2^{-31.001}$ also. However, we ignored that bias and assumed R_0 to be uniformly distributed for those cases in building the distinguishers described in the paper.

In addition to the event L, for which R_i is biased toward zero $\forall i \in [1, 31]$ (see Sect. 6), we also identify another event L', for which R_{25} is again biased toward zero (all other R_i's are uniformly distributed individually). The event L' occurs when $P_2[116] \equiv -18 (\bmod 32)$, $P_3[116] \equiv 7 (\bmod 32)$, $P_2[72] = P_3[72] + 1$, $P_2[239] = P_3[239] + 1$, $P_1[26] = 1$, $P_3[208] = 254$ (see Group 2(a) of Fig. 3). Using similar arguments as above, it can be shown that R_i is uniformly distributed over [0, 1] for the rest of the cases.