*Journal of Universal Computer Science, vol. 18, no. 12 (2012), 1741-1756*
*submitted: 15/9/11, accepted: 15/2/12, appeared: 28/6/12 © J.UCS*

# New Results of Related-key Attacks on All Py-Family of Stream Ciphers

**Lin Ding**
(Information Science and Technology Institute, Zhengzhou, China
dinglin_cipher@163.com)

**Jie Guan**
(Information Science and Technology Institute, Zhengzhou, China
guanjie007@163.com)

**Wen-long Sun**
(Information Science and Technology Institute, Zhengzhou, China
swl881010@126.com)

**Abstract:** The stream cipher TPypy has been designed by Biham and Seberry in January 2007 as the strongest member of the Py-family of stream ciphers. At Indocrypt 2007, Sekar, Paul and Preneel showed related-key weaknesses in the Py-family of stream ciphers including the strongest member TPypy. Furthermore, they modified the stream ciphers TPypy and TPy to generate two fast ciphers, namely RCR-32 and RCR-64, in an attempt to rule out all the attacks against the Py-family of stream ciphers. So far there exists no attack on RCR-32 and RCR-64. In this paper, we show that the related-key weaknesses can be still used to construct related-key distinguishing attacks on all Py-family of stream ciphers including the modified versions RCR-32 and RCR-64. Under related keys, we show distinguishing attacks on RCR-32 and RCR-64 with data complexity $2^{139.3}$ and advantage greater than 0.5. We also show that the data complexity of the distinguishing attacks on Py-family of stream ciphers proposed by Sekar et al. can be reduced from $2^{193.7}$ to $2^{149.3}$. These results constitute the best attacks on the strongest members of the Py-family of stream ciphers Tpypy, RCR-32 and RCR-64. By modifying the key setup algorithm, we propose two new stream ciphers TRCR-32 and TRCR-64 which are derived from RCR-32 and RCR-64 respectively. Based on our security analysis, we conjecture that no attacks lower than brute force are possible on TRCR-32 and TRCR-64 stream ciphers.

**Keywords:** Cryptanalysis, Related-key Attack, Distinguishing Attack, Py-family of Stream Ciphers, TRCR-32, TRCR-64.
**Categories:** D.4.6, E.3, K.6.5

## 1 Introduction

RC4 was designed by Rivest in 1987. It has inspired the design of a number of fast stream ciphers, such as ISAAC [Robert, 96], Py [Biham, 05] and MV3 [Keller, 07]. Being the most widely used software stream cipher, RC4 is extremely simple and efficient. At the time of the invention of RC4, its array based design was completely different from the previous stream ciphers mainly based on linear feedback shift registers.

The stream ciphers Py [Biham, 05] and Py6 [Biham, 05], designed by Biham and Seberry, were submitted to the eSTREAM project for analysis and evaluation in the category of software based stream ciphers. However, due to several cryptanalytic attacks on them [Paul, 06, Crowley, 06], a strengthened version Pypy [Biham, 06] was proposed to rule out those attacks. The ciphers were promoted to the 'Focus' ciphers of the Phase II of the eSTREAM project. The impressive speed of the ciphers made them the forerunners in the competition. However, at Eurocrypt 2007, Wu and Preneel showed key recovery attacks against the ciphers Py, Pypy, Py6 with chosen *IV*s [Wu, 07]. This attack was subsequently improved by Isobe et al. [Isobe, 06]. Distinguishing attacks were reported against Py6 with $2^{68.6}$ data and comparable time by Paul and Preneel [Paul, 06]. These three attacks force the designers to again go for modifications. As a result, three new ciphers TPypy, TPy and TPy6 were built, which can very well be viewed as the strengthened versions of the previous ciphers Py, Pypy and Py6 where the above attacks should not apply [Biham, 07]. Among all the members of the Py-family of stream ciphers, the TPypy is conjectured to be the strongest. The ciphers are normally used with 32-byte keys and 16-byte initial values (or *IV*). However, the key size may vary from 1 to 256 bytes and the IV from 1 to 64 bytes. The ciphers were claimed by the designers to be free from related-key and distinguishing attacks [Biham, 05, Wu, 07, Biham, 07].

For the analysis of TPypy, TPy and TPy6, several distinguishing attacks have been proposed.

- Sekar, Paul and Preneel published distinguishing attacks on Py, Pypy, TPy and TPypy with data complexities $2^{281}$ each [Sekar, 07a].
- **(at ISC 2007).** Sekar, Paul and Preneel showed new weaknesses in the stream ciphers TPy and Py [Sekar, 07b]. Exploiting these weaknesses distinguishing attacks on the ciphers are constructed where the best distinguisher requires $2^{268.6}$ data and comparable time.
- **(at WEWoRC 2007).** Sekar, Paul and Preneel mounted distinguishing attacks on TPy6 and Py6 with $2^{224.6}$ data and comparable time each [Sekar, 07c].
- **(at SAC 2007).** Yukiyasu Tsunoo et al. proposed a distinguishing attack on Tpypy that requires $2^{199}$ words of keystreams [Tsunoo, 07].
- **(at Indocrypt 2007).** Sekar, Paul and Preneel presented related-key distinguishing attacks on TPypy, TPy, Pypy and Py, whose data complexity is $2^{193.7}$ [Sekar, 07d]. Moreover, they have modified Tpypy and TPy to design two new ciphers RCR-32 and RCR-64 which were claimed to be free from all attacks excluding brute force ones.

Compared with other results, the paper [Sekar, 07d] constitutes the best attack on the strongest member of the Py-family of stream ciphers Tpypy. So far there exist no attacks on RCR-32 and RCR-64. In this paper, we show that the related-key weaknesses can be still used to construct related-key distinguishing attacks on all Py-family of stream ciphers including the modified versions RCR-32 and RCR-64. Under related keys, we show distinguishing attacks on RCR-32 and RCR-64 with data complexity $2^{139.3}$ and advantage greater than 0.5. We also show that the data complexity of the distinguishing attacks on Py-family of stream ciphers proposed by Sekar et al. can be reduced from $2^{193.7}$ to $2^{149.3}$. These results constitute the best attacks on the strongest members of the Py-family of stream ciphers Tpypy, RCR-32 and

RCR-64. By modifying the key setup algorithm, we propose two new stream ciphers TRCR-32 and TRCR-64 which are derived from RCR-32 and RCR-64 respectively. Based on our security analysis, we conjecture that no attacks lower than brute force are possible on TRCR-32 and TRCR-64 stream ciphers.

This paper is organized as follows. In Section 2, the structure of all Py-family of stream ciphers is briefly described and the previous related-key attacks are discussed. In Section 3, related-key attacks against RCR-32 and RCR-64 are presented. In Section 4, the improved related-key attacks against Py-family of stream ciphers are presented. Two new stream ciphers TRCR-32 and TRCR-64 are proposed in Section 5. Section 6 concludes this paper.

## 2 Preliminaries

### 2.1 Brief Description of Py-family of Stream Ciphers

In each of the Py-family of stream ciphers two rolling arrays have been used. One array P is of 256 bytes that contains a permutation of all the values from 0 to 255 and second array Y is an array of size 260 where each word is of 32 bit and is indexed from -3 to 256. Each of the Py-family of stream ciphers is composed of three parts: (1) a key setup algorithm, (2) an IV setup algorithm and (3) a round function or pseudorandom bit generation algorithm (PRBG). The first two parts are used for the initial one-time mixing of the secret key and the IV. These parts generate a pseudorandom internal state composed of (1) a permutation P of 256 elements, (2) a 32-bit array Y of 260 elements and (3) a 32-bit variable s. The key/IV setup uses two intermediate variables: (1) a fixed permutation of 256 elements denoted by *internal_permutation* and (2) a variable *EIV* whose size is equal to that of the *IV*. The round function, which is executed iteratively, is used to update the internal state (i.e., P, Y and s) and to generate pseudorandom output bits. The key setup algorithms of the TPypy, the TPy, the Pypy and the Py are identical. Notation for different parts of the four ciphers is provided in Table 2.

| **Part** | **RCR-32** | **RCR-64** | **TPypy** | **TPy** | **Pypy** | **Py** |
|---|---|---|---|---|---|---|
| Key Setup | $KS$ | $KS$ | $KS$ | $KS$ | $KS$ | $KS$ |
| *IV* Setup | $IVS_1$ | $IVS_1$ | $IVS_1$ | $IVS_1$ | $IVS_2$ | $IVS_2$ |
| Round Function | $RF_3$ | $RF_4$ | $RF_1$ | $RF_2$ | $RF_1$ | $RF_2$ |

*Table 1: Description of the ciphers RCR-32, RCR-64, TPypy, TPy, Pypy and Py*

Due to space constraints, the $KS$, the $IVS_1$, the $IVS_2$, the $RF_1$ and the $RF_2$, as mentioned in Table 2, are described in Appendix A. The $RF_3$ and $RF_4$ are described in Appendix B. The details of the algorithms can also be found in [Biham, 05, Wu, 07, Biham, 07, Sekar, 07d].

In this paper, the notation and the convention followed are described below.

- The outputs generated when $key_1$ and $key_2$ are used are denoted by O and Z respectively.
- $O^{a}_{(b)}$ (or $Z^{a}_{(b)}$) denotes the $b$-th bit ($b$=0 is the least significant bit or lsb) of

the second output word generated at round a when $key_1$ (or $key_2$) is used. We do not use the first output word anywhere in our analysis.

- $P_1^a, Y_1^a$ and $s_1^a$ are the inputs to the PRBG at round a when $key_1$ is used. It is easy to see that when this convention is followed the $O^a$ takes a simple form: $O^a = \left(s \oplus Y^a[-1]\right) + Y^a[P^a[208]]$. The same applies to $key_2$.
- $P_1^a[b], Y_1^a[b]$ denote the $b$-th elements of arrays $P_1^a$ and $Y_1^a$ respectively, when $key_1$ is used. The same applies to $key_2$.
- $P_1^a[b]_{(i)}, Y_1^a[b]_{(i)}$ denote the $i$-th bits of elements $P_1^a[b]$ and $Y_1^a[b]$ respectively, when $key_1$ is used. The same applies to $key_2$.
- The operators '+', '-' and '$\oplus$' denote addition modulo $2^{32}$, subtraction modulo $2^{32}$ and bitwise exclusive-or respectively, $\cap$ denotes set intersection and $\cup$ denotes set union.

## 2.2 Sekar et al.'s Attacks on Py-family of Stream Ciphers at Indocrypt 2007

In paper [Sekar, 07d], Sekar, Paul and Preneel presented related-key distinguishing attacks on TPypy, TPy, Pypy and Py, whose data complexity is $2^{193.7}$. They show that, when used with the identical $IV$s of 16 bytes each, if two long keys $key_1$ and $key_2$ of 256 bytes each, are related in the following manner,

**$C$1.** $key_1[16] \oplus key_2[16] = 1$,

**$C$2.** $key_1[17] \neq key_2[17]$ and

**$C$3.** $key_1[i] = key_2[i]$, $\forall i \notin \{16,17\}$

Then, they showed that the relation between two keys ($C$1-$C$3) can be traced through various parts of the Py-family of stream ciphers (i.e., TPypy, TPy, Pypy and Py). That is, using the above relation, they exploited the weaknesses of the key setup algorithms of Py-family of stream ciphers, and propagated through the IV setup algorithms and finally induced biases in the keystream outputs. Here, the process of trace is skipped, and we consider only the result of this process. Let $D$ denote the event $Y_1[i] = Y_2[i]$ (where $-3 \le i \le 12$) after IV setup. Then we have a proposition from [Sekar, 07d].

***Proposition 1** [Sekar, 07d].* Under the relation between the keys ($C$1-$C$3), the event $D$ after key setup and IV setup occurs with probability

$$\Pr(D) = 2^{-28.4} \cdot \left(\frac{255}{256}\right)^{16} = 2^{-28.5}$$

As shown in paper [Sekar, 07d], in the first 17 rounds of IV setup part-2 (see Algorithm 2 of Appendix A), the differences in $key$ [16] and $key$ [17] between $key_1$ and $key_2$ causes the internal state $s$ to be different, and then causes $EIV$ to be different in the following round and hence $P_1 \neq P_2$. In the subsequent rounds, the mixing becomes more random. Finally, at the end of IV setup, $Y_1[i] = Y_2[i]$ (where $-3 \le i \le 12$), $P_1 \neq P_2$, $s_1 \neq s_2$ and $Y_1[i] \neq Y_2[i]$ (where $13 \le i \le 256$). Thus, the internal state $P$, $s$ and $Y[i]$ (where $13 \le i \le 256$) after IV setup can be considered to be uniformly distributed and independent.

After the process of trace, they finally induced biases in the least significant bit of the outputs at the 1st and the 3rd rounds. That is,

$$\Pr\left(O^1_{(0)} \oplus O^3_{(0)} \oplus Z^1_{(0)} \oplus Z^3_{(0)} = 0\right) = \frac{1}{2}\left(1 + \frac{1}{2^{96.4}}\right)$$

Therefore, using Theorem 6 of [Baigneres, 04], they showed that the number of samples required to establish an optimal distinguisher with advantage greater than 0.5 is $2^{193.7}$.

## 3 Related-key Distinguishing Attacks on RCR-32 and RCR-64

In this section, we will present our related-key distinguishing attacks on RCR-64 and RCR-32 stream ciphers. The key/IV setup algorithms of RCR-64 and RCR-32 are identical with those of TPy and TPypy. The round function of RCR-64 and RCR-32 are also very similar to those of the TPy and TPypy. The only changes in the round function are that: the variables rotation of the quantity *s* is replaced by a constant rotation of 19. Single round of RCR-64 and RCR-32 are shown in algorithm 4 of Appendix B.

Since the first 32-bit keystream word of $RF_3$ at each round is skipped, thus, they consider only the round function $RF_3$ of Algorithm 4 (see Appendix B). Let $s^1_1, P^1_1, Y^1_1$ (or $s^1_2, P^1_2, Y^1_2$) denote the internal state *s*, *P* and *Y* after IV setup when $key_1$ (or $key_2$) is used. At the end of any round $i(i \geq 1)$, the internal state is updated to $s^{i+1}_1, P^{i+1}_1, Y^{i+1}_1$ (or $s^{i+1}_2, P^{i+1}_2, Y^{i+1}_2$) when $key_1$ (or $key_2$) is used. In our attacks on RCR-64 and RCR-32, we will induce biases in the outputs at the 1st and the 2nd rounds, not 1st and the 3rd rounds. The formulas for the least significant bit of the outputs generated at rounds 1 and 2 when $key_1$ (the output words are denoted by *O*) and $key_2$ (the output words are denoted by *Z*) are used are given below.

$$O^1_{(0)} = s^2_{1(0)} \oplus Y^1_1[-1]_{(0)} \oplus Y^1_1[P^2_1[208]]_{(0)} \quad (1)$$

$$O^2_{(0)} = s^3_{1(0)} \oplus Y^2_1[-1]_{(0)} \oplus Y^2_1[P^3_1[208]]_{(0)} \quad (2)$$

$$Z^1_{(0)} = s^2_{2(0)} \oplus Y^1_2[-1]_{(0)} \oplus Y^1_2[P^2_2[208]]_{(0)} \quad (3)$$

$$Z^2_{(0)} = s^3_{2(0)} \oplus Y^2_2[-1]_{(0)} \oplus Y^2_2[P^3_2[208]]_{(0)} \quad (4)$$

Recall the round function $RF_3$ of Algorithm 3. When the event *D* occurs, $Y^1_1[-1] = Y^1_2[-1]$ and $Y^2_1[-1] = Y^2_2[-1]$ are always satisfied. Thus,

$$Y^1_1[-1] = Y^1_2[-1] \Rightarrow Y^1_1[-1]_{(0)} = Y^1_2[-1]_{(0)}$$

$$Y^2_1[-1] = Y^2_2[-1] \Rightarrow Y^2_1[-1]_{(0)} = Y^2_2[-1]_{(0)}$$

Let $C_1, C_2, C_3$ and $C_4$ denote $Y^1_1[P^2_1[208]]_{(0)}, Y^2_1[P^3_1[208]]_{(0)}, Y^1_2[P^2_2[208]]_{(0)}$ and $Y^2_2[P^3_2[208]]_{(0)}$, respectively. In order to make $C_1 \oplus C_2 \oplus C_3 \oplus C_4 = 0$ (denoted by event *G*) to be satisfied with a high probability, some conditions on the elements of $P_1$ and $P_2$ should be simultaneously satisfied. Concluded from a large number of experiments, it is determined that when the two conditions $P^2_1[208] = P^3_1[208] + 1$ (denoted by event $U_1$) and $P^2_2[208] = P^3_2[208] + 1$ (denoted by event $U_2$) are simultaneously satisfied, the probability that the event $G$ occurs is quite close to 1. Running simulation, it is determined that

$$\Pr(G) = \Pr(U_1 \cap U_2) \approx 2^{-8} \cdot 2^{-8} = 2^{-16}$$

The formulas for $s_1^3$ and $s_2^3$ is given below:

$$s_1^3 = RTOTL32(s_1^2 + Y_1^2[P_1^3[72]] - Y_1^2[P_1^3[239]], 19) \qquad (5)$$

$$s_2^3 = RTOTL32(s_2^2 + Y_2^2[P_2^3[72]] - Y_2^2[P_2^3[239]], 19) \qquad (6)$$

Let $c_1 = Y_1^2[P_1^3[72]] - Y_1^2[P_1^3[239]]$ and $c_2 = Y_2^2[P_2^3[72]] - Y_2^2[P_2^3[239]]$ . Let $\delta$ and $\gamma$ represent the carry bits from the additions in the equations (5) and (6), respectively. Thus, we know $s_1^2 + c_1 = s_1^2 \oplus c_1 \oplus \delta$ and $s_2^2 + c_2 = s_2^2 \oplus c_2 \oplus \gamma$ .

Therefore,

$$\begin{aligned} & s_{1(0)}^2 \oplus s_{1(0)}^3 \oplus s_{2(0)}^2 \oplus s_{2(0)}^3 \\ &= s_{1(0)}^2 \oplus s_{1(19)}^2 \oplus c_{1(19)} \oplus \delta_{(19)} \oplus s_{2(0)}^2 \oplus s_{2(19)}^2 \oplus c_{2(19)} \oplus \gamma_{(19)} \\ &= s_{1(0)}^2 \oplus s_{1(19)}^2 \oplus s_{2(0)}^2 \oplus s_{2(19)}^2 \oplus c_{1(19)} \oplus c_{2(19)} \oplus \delta_{(19)} \oplus \gamma_{(19)} \end{aligned}$$

Where

$$\delta_{(19)} = s_{1(18)}^2 c_{1(18)} \oplus \delta_{(18)} s_{1(18)}^2 \oplus \delta_{(18)} c_{1(18)}$$

$$\gamma_{(19)} = s_{2(18)}^2 c_{2(18)} \oplus \gamma_{(19)} s_{2(18)}^2 \oplus \gamma_{(19)} c_{2(18)}$$

We now state the following proposition.

***Proposition 2.*** $s_{1(0)}^2 \oplus s_{1(0)}^3 \oplus s_{2(0)}^2 \oplus s_{2(0)}^3 = 0$ when the following conditions are simultaneously satisfied.

1. $s_{1(0)}^2 \oplus s_{1(19)}^2 \oplus s_{2(0)}^2 \oplus s_{2(19)}^2 \oplus \delta_{(19)} \oplus \gamma_{(19)} = 0$ (event $E_1$)
2. $P_1^3[72] = P_2^3[72] = a \in \{-3, -2, \cdots, 11\}$ ,

   $P_1^3[239] = P_2^3[239] = b \in \{-3, -2, \cdots, 11\}$ and $b \neq a$ (event $E_2$)

***Proof.*** Recall the round function $RF_3$ of Algorithm 4.

Since it has $Y_1[i] = Y_2[i]$ (where $-3 \le i \le 12$ ) at the end of IV setup, and then the event $E_2$ implies $Y_1^2[P_1^3[72]] = Y_2^2[P_2^3[72]]$ and $Y_1^2[P_1^3[239]] = Y_2^2[P_2^3[239]]$ . That is, $c_1 = c_2 \Rightarrow c_{1(19)} = c_{2(19)}$ .

Thus,

$$\begin{aligned} & s_{1(0)}^2 \oplus s_{1(0)}^3 \oplus s_{2(0)}^2 \oplus s_{2(0)}^3 \\ &= s_{1(0)}^2 \oplus s_{1(19)}^2 \oplus c_{1(19)} \oplus \delta_{(19)} \oplus s_{2(0)}^2 \oplus s_{2(19)}^2 \oplus c_{2(19)} \oplus \gamma_{(19)} \\ &= s_{1(0)}^2 \oplus s_{1(19)}^2 \oplus s_{2(0)}^2 \oplus s_{2(19)}^2 \oplus \delta_{(19)} \oplus \gamma_{(19)} \end{aligned}$$

Therefore, condition 1 and condition 2 together imply $s_{1(0)}^2 \oplus s_{1(0)}^3 \oplus s_{2(0)}^2 \oplus s_{2(0)}^3 = 0$ .

This completes the proof. □

Since the internal state $P$, $s$ and $Y[i]$ (where $13 \le i \le 256$ ) after IV setup can be considered to be uniformly distributed and independent, and then we know

$$\Pr(E_1) \approx 2^{-1} \text{ and } \Pr(E_2) \approx 2^{-8} \cdot \frac{15}{256} \cdot 2^{-8} \cdot \frac{14}{256} = 2^{-23.7} .$$

The two events $E_1$ and $E_2$ are assumed to be independent to facilitate calculation of bias. The actual value without independence assumption is in fact more, making the attack marginally stronger. Let $E$ denote the event $E_1 \cap E_2$ . Hence,

$$\Pr(E) = \Pr(E_1 \cap E_2) \approx \Pr(E_1) \cdot \Pr(E_2) = 2^{-24.7} .$$

Therefore, from Proposition 1 and equations (1-4), we observe that $O^1_{(0)} \oplus O^2_{(0)} \oplus Z^1_{(0)} \oplus Z^2_{(0)} = 0$ holds when the following events simultaneously occur.

$$D, G \text{ and } E.$$

In the following, we calculate the probability that $O^1_{(0)} \oplus O^2_{(0)} \oplus Z^1_{(0)} \oplus Z^2_{(0)} = 0$ is satisfied. Let $L$ denote the event $(D \cap G \cap E)$ . Thus,

$$\Pr(L) \approx \Pr(D) \cdot \Pr(G) \cdot \Pr\left(E\right) = 2^{-28.5} \cdot 2^{-16} \cdot 2^{-24.7} = 2^{-69.2}$$

Assuming randomness of the outputs when event $L$ does not occur, we have

$$\Pr\left(O^1_{(0)} \oplus O^2_{(0)} \oplus Z^1_{(0)} \oplus Z^2_{(0)} = 0\right) = 2^{-69.2} \cdot 1 + \frac{1}{2} \cdot \left(1 - 2^{-69.2}\right) = \frac{1}{2}\left(1 + 2^{-69.2}\right)$$

To compute the number of samples required to establish an optimal distinguisher with advantage greater than 0.5, we use the following equation from [Paul, 06, Baigneres, 04].

$$N = 0.4624 \cdot \frac{1}{p^2}$$

Here, $p = 2^{-70.2}$ . Therefore, the number of samples is $2^{139.3}$ .

Therefore, the number of samples required for our distinguishing attack is $2^{139.3}$ .

## 4 Improved Related-key Distinguishing Attacks on Py-family Stream Ciphers

In this section, we will apply our related-key attacks on Py-family of stream ciphers (i.e., TPypy, TPy, Pypy and Py) to improve the attacks presented by Sekar et al.

Since the first 32-bit keystream word of $RF_1$ at each round is skipped, thus, we consider only the round function $RF_1$ of Algorithm 3 (see Appendix A).

The formulas for $s^3_1$ and $s^3_2$ is given below:

$$s^3_1 = RTOTL32(s^2_1 + Y^2_1[P^3_1[72]] - Y^2_1[P^3_1[239]], (P^3_1[116] + 18) \bmod 32) \quad (7)$$

$$s^3_2 = RTOTL32(s^2_2 + Y^2_2[P^3_2[72]] - Y^2_2[P^3_2[239]], (P^3_2[116] + 18) \bmod 32) \quad (8)$$

Let $e_1 = Y^2_1[P^3_1[72]] - Y^2_1[P^3_1[239]]$ , $e_2 = Y^2_1[P^3_1[72]] - Y^2_1[P^3_1[239]]$ , $d_1 = (P^3_1[116] + 18) \bmod 32$ and $d_2 = (P^3_2[116] + 18) \bmod 32$ . Let $\beta$ and $\varepsilon$ represent the carry bits from the additions in the equations (7) and (8), respectively. Thus, we know $s^2_1 + e_1 = s^2_1 \oplus e_1 \oplus \beta$ and $s^2_2 + e_2 = s^2_2 \oplus e_2 \oplus \varepsilon$ . Without loss of generality, set lsb of $\beta$ and $\varepsilon$ (denoted by $\beta_{(0)}$ and $\varepsilon_{(0)}$ ) be 0.

Therefore,

$$s^2_{1\,(0)} \oplus s^3_{1\,(0)} \oplus s^2_{2(0)} \oplus s^3_{2(0)}$$

$$= s^2_{1\,(0)} \oplus s^2_{1\,(d_1)} \oplus e_{1(d_1)} \oplus \beta_{(d_1)} \oplus s^2_{2(0)} \oplus s^2_{2(d_2)} \oplus e_{2(d_2)} \oplus \varepsilon_{(d_2)}$$
$$= s^2_{1\,(0)} \oplus s^2_{1\,(d_1)} \oplus s^2_{2(0)} \oplus s^2_{2(d_2)} \oplus e_{1(d_1)} \oplus e_{2(d_2)} \oplus \beta_{(d_1)} \oplus \varepsilon_{(d_2)}$$

Where

$$\beta_{(d_1)} = \begin{cases} 0, & d_1 = 0 \\ s^2_{1\,(d_1-1)} e_{1(d_1-1)} \oplus \beta_{(d_1-1)} s^2_{1\,(d_1-1)} \oplus \beta_{(d_1-1)} e_{1(d_1-1)}, & 1 \le d_1 \le 31 \end{cases}$$
$$\varepsilon_{(d_2)} = \begin{cases} 0, & d_2 = 0 \\ s^2_{1\,(d_2-1)} e_{1(d_2-1)} \oplus \varepsilon_{(d_2-1)} s^2_{1\,(d_2-1)} \oplus \varepsilon_{(d_2-1)} e_{1(d_2-1)}, & 1 \le d_2 \le 31 \end{cases}$$

Similarly, we now state the following proposition.

**Proposition 2.** $s^2_{1\,(0)} \oplus s^3_{1\,(0)} \oplus s^2_{2(0)} \oplus s^3_{2(0)} = 0$ when the following conditions are simultaneously satisfied.

1. $d_1 = d_2$ (event $F_1$)
2. $s^2_{1\,(0)} \oplus s^2_{1\,(d_1)} \oplus s^2_{2(0)} \oplus s^2_{2(d_2)} \oplus \beta_{(d_1)} \oplus \varepsilon_{(d_2)} = 0$ (event $F_2$)
3. $P_1^3[72] = P_2^3[72] = a \in \{-3,-2,\cdots,11\}$,
   $P_1^3[239] = P_2^3[239] = b \in \{-3,-2,\cdots,11\}$ and $b \ne a$ (event $F_3$)

**Proof.** Recall the round function $RF_3$ of Algorithm 4.

Since it has $Y_1[i] = Y_2[i]$ (where $-3 \le i \le 12$) at the end of IV setup, and then the condition $F_3$ implies $Y_1^2[P_1^3[72]] - Y_1^2[P_1^3[239]] = Y_2^2[P_2^3[72]] - Y_2^2[P_2^3[239]]$. Conditions $F_1$ and $F_3$ together imply $e_{1(d_1)} \oplus e_{2(d_2)} = 0$.

Thus,

$$s^2_{1\,(0)} \oplus s^3_{1\,(0)} \oplus s^2_{2(0)} \oplus s^3_{2(0)}$$
$$= s^2_{1\,(0)} \oplus s^2_{1\,(d_1)} \oplus s^2_{2(0)} \oplus s^2_{2(d_2)} \oplus c_{1(d_1)} \oplus c_{2(d_2)} \oplus \delta_{(d_1)} \oplus \gamma_{(d_2)}$$
$$= s^2_{1\,(0)} \oplus s^2_{1\,(d_1)} \oplus s^2_{2(0)} \oplus s^2_{2(d_2)} \oplus \delta_{(d_1)} \oplus \gamma_{(d_2)}$$

Therefore, condition 1, condition 2 and condition 3 together imply $s^2_{1\,(0)} \oplus s^3_{1\,(0)} \oplus s^2_{2(0)} \oplus s^3_{2(0)} = 0$.

This completes the proof. □

Since the internal state $P$, $s$ and $Y[i]$ (where $13 \le i \le 256$) after IV setup can be considered to be uniformly distributed and independent, and then we know

$$\Pr(F_1) \approx 2^{-5},\ \Pr(F_2) \approx 2^{-1} \text{ and } \Pr(F_3) \approx 2^{-8} \cdot \frac{15}{256} \cdot 2^{-8} \cdot \frac{14}{256} = 2^{-23.7}.$$

Let $F$ denote the event $F_1 \cap F_2 \cap F_3$. Hence,

$$\Pr(F) = \Pr(F_1 \cap F_2 \cap F_3) \approx \Pr(F_1) \cdot \Pr(F_2) \cdot \Pr(F_3) = 2^{-29.7}.$$

Therefore, we observe that $O^1_{(0)} \oplus O^2_{(0)} \oplus Z^1_{(0)} \oplus Z^2_{(0)} = 0$ holds when the following events simultaneously occur.

$D$, $G$ and $F$.

Let $Q$ denote the event $(D \cap G \cap F)$. Then, we get

$$\Pr(Q) \approx \Pr(D) \cdot \Pr(G) \cdot \Pr(F) = 2^{-28.5} \cdot 2^{-16} \cdot 2^{-29.7} = 2^{-74.2}$$

Assuming randomness of the outputs when event $Q$ does not occur, we have

$$\Pr\left(O^1_{(0)} \oplus O^2_{(0)} \oplus Z^1_{(0)} \oplus Z^2_{(0)} = 0\right) = 2^{-74.2} \cdot 1 + \frac{1}{2} \cdot \left(1 - 2^{-74.2}\right) = \frac{1}{2}\left(1 + 2^{-74.2}\right)$$

To compute the number of samples required to establish an optimal distinguisher with advantage greater than 0.5, we use the following equation from [Paul, 06, Baigneres, 04].

$$N = 0.4624 \cdot \frac{1}{p^2}$$

Here, $p = 2^{-75.2}$. Therefore, the number of samples is $2^{149.3}$. Compared with paper [Sekar, 07d], in our related-key attacks on Py-family of stream ciphers, we induce biases in the outputs at the 1st and the 2nd rounds, not 1st and the 3rd rounds. We make a more accurate evaluation on the probability that $s^2_{1(0)} \oplus s^3_{1(0)} \oplus s^2_{2(0)} \oplus s^3_{2(0)} = 0$ holds. Hence, our attacks improve the attacks proposed in [Sekar, 07d] obviously in terms of data complexity.

## 5 New Stream Ciphers: TRCR-32 and TRCR-64

### 5.1 Comparison of Our Results with Previous Attacks

The stream cipher TPypy has been designed by Biham and Seberry in January 2007 as the strongest member of the Py-family stream ciphers. At Indocrypt 2007, Sekar, Paul and Preneel showed related-key weaknesses in the Py-family of stream ciphers including the strongest member TPypy. Furthermore, they modified the stream ciphers TPypy and TPy to generate two fast ciphers, namely RCR-32 and RCR-64, in an attempt to rule out all the attacks against the Py-family of stream ciphers. So far there exist no attacks on RCR-32 and RCR-64.

| Attacks | Py6 | Py | Pypy | TPy6 | TPy | TPypy | RCR-32 | RCR-64 |
|---|---|---|---|---|---|---|---|---|
| [Paul, 06] | X | $2^{89.2}$ | X | X | $2^{89.2}$ | X | X | X |
| [Crowley, 06] | X | $2^{72}$ | X | X | $2^{72}$ | X | X | X |
| [Wu, 07] | $<2^{24}$ | $2^{24}$ | $2^{24}$ | X | X | X | X | X |
| [Isobe, 06] | $<2^{24}$ | $2^{24}$ | $2^{24}$ | X | X | X | X | X |
| [Paul, 06] | $2^{68.6}$ | X | X | $2^{68.6}$ | X | X | X | X |
| [Sekar, 07a] | X | $2^{281}$ | $2^{281}$ | X | $2^{281}$ | $2^{281}$ | X | X |
| [Sekar, 07b] | X | $2^{268.6}$ | X | X | $2^{268.6}$ | X | X | X |
| [Sekar, 07c] | $2^{224.6}$ | X | X | $2^{224.6}$ | X | X | X | X |
| [Tsunoo, 07] | X | X | X | X | X | $2^{199}$ | X | X |
| [Sekar, 07d] | X | $2^{193.7}$ | $2^{193.7}$ | X | $2^{193.7}$ | $2^{193.7}$ | X | X |
| Related key(this paper) | X | $2^{149.3}$ | $2^{149.3}$ | X | $2^{149.3}$ | $2^{149.3}$ | $2^{139.3}$ | $2^{139.3}$ |

*Table 2: Comparison of our results with previous attacks on Py-family of stream ciphers (Note: 'X' denotes that the attack does not work.)*

In Section 3 and 4, we show that the related-key weaknesses can be still used to construct related-key distinguishing attacks on all Py-family of stream ciphers including the modified versions RCR-32 and RCR-64. Under related keys, we show distinguishing attacks on RCR-32 and RCR-64 with data complexity $2^{139.3}$ and advantage greater than 0.5. We also show that the data complexity of the distinguishing attacks on Py-family of stream ciphers proposed by Sekar et al. can be reduced from $2^{193.7}$ to $2^{149.3}$. These results constitute the best attacks on the strongest members of the Py-family of stream ciphers Tpypy, RCR-32 and RCR-64. It is shown that the above attacks also work on the other members TPy, Pypy and Py.

Table 3 summarizes the attacks on Py-family of stream ciphers. Compared with previous attacks, our results constitute the best attacks on the strongest members of the Py-family of stream cipher Tpypy. We also introduce the first attack on RCR-32 and RCR-64, which shows the modifications made by Sekar, Paul and Preneel to generate RCR-32 and RCR-64 are not reasonable. In the next subsection, we will present our modifications to improve all Py-family of stream ciphers.

## 5.2 New Proposal for Key Setup Algorithm

In [Sekar, 07d], Sekar, Paul and Preneel showed related-key weaknesses in the Py-family of stream ciphers, and then made simple modifications to the ciphers Tpypy and TPy to build RCR-32 and RCR-64 respectively. In their modified designs, the key scheduling algorithms of RCR-32 and RCR-64 are identical with those of TPypy and TPy. They modified the round function of TPypy and TPy to build RCR-32 and RCR-64 respectively. Our attacks on RCR-32 and RCR-64 show their modifications are not reasonable. In this subsection, we propose a new proposal for key setup algorithm which is similar to the original.

Our attacks can succeed mainly because of the weaknesses of the key setup algorithms of Py-family of stream ciphers. Hence, modifying the key setup algorithm is a more reasonable choice. According to Proposition 1, under the relation between the keys ($C1$-$C3$), the event $D$ after key setup and IV setup occurs with probability $2^{-28.5}$. This show the initialization of Py-family of stream ciphers is quite bad in terms of the completeness property. Recall the key setup algorithm (see Algorithm 1 of Appendix A). Each element of array Y has been updated only once in key setup algorithm, which makes the event $D$ after key setup and IV setup occur with high probability.

In this subsection, we propose two new stream ciphers, TRCR-32 and TRCR-64 derived from RCR-32 and RCR-64, which are shown to be secure against all the existing attacks on RCR-32 and RCR-64.The IV setup algorithms and round functions of TRCR-32 and TRCR-64 are identical with those of RCR-32 and RCR-64. The only changes in the key setup algorithms are that: the array Y is updated twice not once. The new proposal for key setup algorithm of all Py-family of stream ciphers is shown in Algorithm 5.

**Algorithm 5 Key Setup Algorithm of TRCR-32 and TRCR-64**

**Require:** A key, an IV and an *initial permutation*
**Ensure:** An array Y [−3, . . . , 256] and a 32-bit variable *s*

```
keysizeb = size of key in bytes;
ivsizeb = size of IV in bytes;
YMININD=-3;
YMAXIND=256;
s = internal_permutation[keysizeb-1];
s = (s<<8) | internal_permutation[(s ^(ivsizeb-1))&0xFF];
s = (s<<8) | internal_permutation[(s ^ key[0])&0xFF];
s = (s<<8) | internal_permutation[(s ^ key[keysizeb-1])&0xFF];
for(j=0; j<keysizeb; j++)  /* Part-1*/
{
    s = s + key[j];
    s0 = internal_permutation[s&0xFF];
    s = ROTL32(s, 8) ^ (u32)s0;
}
/* Tweak - Initialize the array Y */
for(i=YMININD, j=0; i<=YMAXIND; i++)  /* Part-2*/
{
    s = s + key[j];
    s0 = internal_permutation[s&0xFF];
    Y(i) = s = ROTL32(s, 8) ^ (u32)s0;
    j = j+1 mod keysizeb;
}
/* Tweak - Update the array Y */
for(i=YMAXIND, j= YMININD; i>=YMININD; i--)  /* Part-3*/
{
    s = s +Y[j];
    s0 = internal_permutation[s&0xFF];
    Y(i) = s = ROTL32(s, 8) ^ (u32)s0;
    j++;
}
```

## 5.3 Security Analysis

In this section we justify how the new stream ciphers TRCR-32 and TRCR-64 should be able to resist several common attacks against array-based stream ciphers.

*(i)* *Resistance to Distinguishing Attacks, Differential attacks, Algebraic attacks and Guess-and-Determine Attacks*: Since the IV setup algorithms and round functions of TRCR-32 and TRCR-64 are identical with those of RCR-32 and RCR-64, these attacks are no longer applicable in new stream ciphers.

*(ii)* *Resistance to Related-key attacks [Sekar, 07d, this paper]*: When tracing the relation between two keys ($C1$-$C3$) through various parts of the Py-family of stream ciphers, we find that the internal state (i.e., P, Y and s) are uniformly distributed at random after key/IV setup algorithms. At the end of Part-2 of

Algorithm 5, we have $P_1 = P_2$, $s_1 = s_2$ and $Y_1[i] = Y_2[i]$ (where $i \neq 13$). In the Part-3 of Algorithm 5, we use one element of array Y to update the other element of array Y, instead of the key. The difference in arrays $Y_1$ and $Y_2$ (i.e., $Y_1[13] \neq Y_2[13]$) causes the internal state *s* to be different, and then causes the array Y to be different. In the subsequent rounds of IV setup algorithm, the mixing becomes more random. Finally, at the end of IV setup, $Y_1 \neq Y_2$, $P_1 \neq P_2$ and $s_1 \neq s_2$. Hence, the new stream ciphers TRCR-32 and TRCR-64 have much better completeness properties than the original RCR-32 and RCR-64 stream ciphers, and then the internal state *P*, *s* and *Y* after IV setup algorithm can be considered to be uniformly distributed and independent. Therefore, the outputs generated in the keystream generation algorithm are not expected to be correlated. Hence, the new stream ciphers TRCR-32 and TRCR-64 are expected to be free from any correlations between the outputs.

Based on our security analysis, we conjecture that no attacks lower than brute force are possible on TRCR-32 and TRCR-64 stream ciphers.

## 6 Conclusions

In this paper, we show that the related-key weaknesses can be still used to construct related-key distinguishing attacks on all Py-family of stream ciphers including the modified versions RCR-32 and RCR-64. Under related keys, we show distinguishing attacks on RCR-32 and RCR-64 with data complexity $2^{139.3}$ and advantage greater than 0.5. We also show that the data complexity of the distinguishing attacks on Py-family of stream ciphers proposed by Sekar et al. can be reduced from $2^{193.7}$ to $2^{149.3}$. These results constitute the best attacks on the strongest members of the Py-family of stream ciphers Tpypy, RCR-32 and RCR-64. By modifying the key setup algorithm, we propose two new stream ciphers TRCR-32 and TRCR-64 which are derived from RCR-32 and RCR-64 respectively. Based on our security analysis, we conjecture that no attacks lower than brute force are possible on TRCR-32 and TRCR-64 stream ciphers.

We hope our results can be helpful in evaluating the security of Py-family stream ciphers against related-key distinguishing attacks and we look forward to further work in evaluating TRCR-32 and TRCR-64 against other kinds of cryptanalytic attacks.

**Acknowledgements**

I would like to thank David G. Rosado, Eduardo Fernández-Medina, Luis Enrique Sánchez, Jan Jurjens and anonymous reviewers for their helpful comments on this paper. This work was sponsored in part by the Science and Technology on Communication Security Laboratory Foundation of China under Grant No.9140C110202110C1101.

## References

[Baigneres, 04] Baigneres, T., Junod, P., Vaudenay, S.: How Far Can We Go Beyond Linear Cryptanalysis?, In Proc. Int. Conf. on Asiacrypt, December 2004, 432–450.

[Biham, 05] Biham, E., Seberry, J.: Py (Roo): A Fast and Secure Stream Cipher using Rolling Arrays, eSTREAM, ECRYPT Stream Cipher Project, Report 2005/023, 2005.

[Biham, 06] Biham, E., Seberry, J.: Pypy (Roopy): Another Version of Py, Ecrypt Record, March 2006.

[Biham, 07] Biham, E., Seberry, J.: Tweaking the IV Setup of the Py Family of Ciphers – The Ciphers Tpy, TPypy, and TPy6, January 2007 http://www.cs.technion.ac.il/biham/

[Crowley, 06] Crowley, P.: Improved Cryptanalysis of Py, In Record of Int. Workshop of SASC 2006 - Stream Ciphers Revisited, ECRYPT Network of Excellence in Cryptology, February 2006, 52-60.

[Isobe, 06] Isobe, T., Ohigashi, T., Kuwakado, H., Morii, M.: How to Break Py and Pypy by a Chosen-IV Attack, eSTREAM, ECRYPT Stream Cipher Project, Report 2006/060.

[Keller, 06] Keller, N., Miller, S., Mironov, I., Venkatesan, R.: MV3: A new word based stream cipher using rapid mixing and revolving buffers, In Proc. Int. Conf. on Topics in Cryptology-CT-RSA, February 2007, 1-19.

[Paul, 06] Paul, S., Preneel, B., Sekar, G.: Distinguishing Attacks on the Stream Cipher Py. In Proc. Int. Conf. on Fast Software Encryption, February 2006, 405-421.

[Paul, 06] Paul, S., Preneel, B.: On the (In)security of Stream Ciphers Based on Arrays and Modular Addition, In Proc. Int. Conf. on Asiacrypt, December 2006, 69-83.

[Robert, 96] Robert, J.: ISAAC. In Proc. Int. Conf. on Fast Software Encryption, February 1996, 41–49.

[Sekar, 07a] Sekar, G., Paul, S., Preneel, B.: Weaknesses in the Pseudorandom Bit Generation Algorithms of the Stream Ciphers TPypy and TPy, February 2007 http://eprint.iacr.org/2007/075.pdf

[Sekar, 07b] Sekar, G., Paul, S., Preneel, B.: New Weaknesses in the Keystream Generation Algorithms of the Stream Ciphers TPy and Py. In Proc. Int. Conf. on Information Security, June 2007, 249-262.

[Sekar, 07c] Sekar, G., Paul, S., Preneel, B.: New Attacks on the Stream Cipher TPy6 and Design of New Ciphers the TPy6-A and the TPy6-B, In Western European Workshop on Research in Cryptology-WEWoRC, July 2007, 127-141.

[Sekar, 07d] Sekar, G., Paul, S., Preneel, B.: Related-key Attacks on the Py-family of Ciphers and an Approach to Repair the Weaknesses. In Proc. Int. Conf. on Indocrypt, December 2007, 58-72.

[Tsunoo, 07] Tsunoo, Y., Saito, T., Kawabata, T., Nakashima, H.: Distinguishing Attack against TPypy. In Proc. Int. Conf. on Selected Areas in Cryptography, August 2007, 396-407.

[Wu, 07] Wu, H., Preneel, B.: Differential Cryptanalysis of the Stream Ciphers Py, Py6 and Pypy, In Proc. Int. Conf. on Eurocrypt, May 2007, 276-290.

## Appendix A Various Parts of Py-family of Stream Ciphers

**Algorithm 1 Key setup: *KS***

**Require:** A key, an IV and an *initial permutation*
**Ensure:** An array Y [−3, . . . , 256] and a 32-bit variable *s*

```
keysizeb = size of key in bytes;
ivsizeb = size of IV in bytes;
YMININD=-3;
YMAXIND=256;
s = internal_permutation[keysizeb-1];
s = (s<<8) | internal_permutation[(s ^(ivsizeb-1))&0xFF];
s = (s<<8) | internal_permutation[(s ^ key[0])&0xFF];
s = (s<<8) | internal_permutation[(s ^ key[keysizeb-1])&0xFF];
for(j=0; j<keysizeb; j++)
{
    s = s + key[j];
    s0 = internal_permutation[s&0xFF];
    s = ROTL32(s, 8) ^ (u32)s0;
}
/* Again */
for(j=0; j<keysizeb; j++)
{
    s = s + key[j];
    s0 = internal_permutation[s&0xFF];
    s ^= ROTL32(s, 8) + (u32)s0;
}
/* Initialize the array Y */
for(i=YMININD, j=0; i<=YMAXIND; i++)
{
    s = s + key[j];
    s0 = internal_permutation[s&0xFF];
    Y(i) = s = ROTL32(s, 8) ^ (u32)s0;
    j = j+1 mod keysizeb;
}
```

**Algorithm 2 The *IV* setup algorithms of $IVS_1$ and $IVS_2$ - initialization of P and *EIV***

**Require:** The Y, the *s* from the key setup algorithm and the *IV*
**Ensure:** Rolling arrays P[0, . . . , 255], *EIV* [0, . . . , ivsizeb − 1], the variable *s*

```
/* Create an initial permutation */
u8 v= iv[0] ^ ((Y(0)>>16)&0xFF);
u8 d=(iv[1 mod ivsizeb] ^ ((Y(1)>>16)&0xFF))|1;
for(i=0; i<256; i++)
{
    P(i)=internal_permutation[v];
    v+=d;
}
/* Now P is a permutation */
/* Initialize s */
s = ((u32)v<<24) ^ ((u32)d<<16) ^ ((u32)P(254)<<8) ^ ((u32)P(255));
s ^= Y(YMININD)+Y(YMAXIND);
for(i=0; i<ivsizeb; i++)
{
    s = s + iv[i] + Y(YMININD+i);
    u8 s0 = P(s&0xFF);
    EIV(i) = s0;
    s = ROTL32(s, 8) ^ (u32)s0;
}
/* Again, but with the last words of Y, and update EIV */
for(i=0; i<ivsizeb; i++)
{
    s = s + iv[i] + Y(YMAXIND-i);
    /*s = s + EIV((i+ivsizeb-1)mod ivsizeb) + Y(YMAXIND-i); for IVS1.*/
    u8 s0 = P(s&0xFF);
    EIV(i) += s0;
    s = ROTL32(s, 8) ^ (u32)s0;
}

for(i=0; i<260; i++)  //IV setup part-2
{
    u32 x0 = EIV(0) = EIV(0) ^ (s&0xFF);
    rotate(EIV);
    swap(P(0), P(x0));
    rotate(P);
    Y(YMININD)=s=(s ^ Y(YMININD))+Y(x0);
    /*s=ROTL32(s,8)+Y(YMAXIND);Y(YMININD)+=s^Y(x0); for IVS1.*/
    rotate(Y);
}
s=s+Y(26)+Y(153)+Y(208);
if(s==0)
  s=(keysizeb*8)+((ivsizeb*8)<<16)+0x87654321;
```

**Algorithm 3 Round functions: *RF*$_1$ and *RF*$_2$**

```
Require: Y [−3, ..., 256], P[0, ..., 255], a 32-bit variable s
Ensure: 32-bit random output (for RF_1) or 64-bit random output (for RF_2)
/*Update and rotate P*/
swap (P[0], P[Y [185]&255]);
rotate (P);
/* Update s*/
s+ = Y [P[72]] − Y [P[239]];
s = ROTL32(s, ((P[116] + 18)&31));
/* Output 4 or 8 bytes (least significant byte first)*/
output ((ROTL32(s, 25) □ Y [256]) + Y [P[26]]);/* This step is skipped for RF_1.*/
output (( s□Y [−1]) + Y [P[208]]);
/* Update and rotate Y */
Y [−3] = (ROTL32(s, 14) □ Y [−3]) + Y [P[153]];
   rotate(Y );
```

## Appendix B Round Function of RCR-32 and RCR-64

**Algorithm 4 Round functions of RCR-32 and RCR-64: *RF*$_3$ and *RF*$_4$**

```
Require: Y [−3, ..., 256], P[0, ..., 255], a 32-bit variable s
Ensure: 64-bit random output (for RCR-64) or 32-bit random output (for RCR-32)
/*Update and rotate P*/
swap (P[0], P[Y [185]&255]);
rotate (P);
/* Update s*/
s+ = Y [P[72]] − Y [P[239]];
s = ROTL32(s, 19); /*Tweak - the variable s undergoes a constant, non-zero rotation
(c = 19).*/
/* Output 4 or 8 bytes (the least significant byte first)*/
output ((ROTL32(s, 25) □ Y [256]) + Y [P[26]]);/* This step is skipped for RF_3.*/
output (( s □Y [−1]) + Y [P[208]]);
/* Update and rotate Y */
Y [−3] = (ROTL32(s, 14) □ Y [−3]) + Y [P[153]];
rotate(Y );
```