How Not to Combine RC4 States

Subhadeep Banik[1] and Sonu Jha[2]

[1] DTU Compute, Technical University of Denmark, Lyngby 2800, Denmark
subb@dtu.dk
[2] National Informatics Center, Sector V, Salt Lake, Kolkata 91, India
jhasonu1987@yahoo.com

Abstract. Over the past few years, an attractive design paradigm has emerged, that aims to produce new stream cipher designs, by combining one or more independently produced RC4 states. The ciphers so produced turn out to be faster than RC4 on any software platform, mainly because the average number of internal operations used in the cipher per byte of keystream produced is usually lesser than RC4. One of the main efforts of the designers is to ensure that the existing weaknesses of RC4 are not carried over to the new ciphers so designed. In this work we will look at two such ciphers RC4B (proposed by Zhang et. al.) and Quad-RC4/m-RC4 (proposed by Maitra et. al.). We will propose distinguishing attacks against all these ciphers, and look at certain design flaws that made these ciphers vulnerable.

Keywords: RC4, RC4B, Quad-RC4, m-RC4, Distinguishing Attacks, Stream Cipher.

1 Introduction

From over the past two decades, RC4 has been one of the most extensively used stream ciphers in many popular protocols like WEP, TLS, TCP etc. The reason behind the popularity of this byte oriented stream cipher was the simplicity of its design. Using a very few number of operations, RC4 is able to provide fast enough encryption in software. It is not very surprising that such an elegant cipher wrapped in just 4 lines of code was going to gain the attention of the researchers from all over the world. As a result, several attempts have been made to cryptanalyze this stream cipher (see [7, 8]). Apart from the analysis point of view, there has also been several proposals of new RC4-like stream ciphers by introducing some number of modifications on the original RC4 design paradigm. The major motivations behind these new proposals were to protect the cipher against some well known cryptanalytic results shown on the RC4 stream cipher keeping also in mind that the average number of operational steps taken by those new introduced designs in order to encrypt the data is not much more than the number of steps taken by RC4 itself. For example, the RC4+ stream cipher [6] proposed by Maitra et. al. introduced a modified version of RC4 with a complex 3-phase key schedule and a more complex output function in order to protect the new design against the above mentioned well known attacks with

R.S. Chakraborty et al. (Eds.): SPACE 2015, LNCS 9354, pp. 95–112, 2015.
DOI: 10.1007/978-3-319-24126-5_6

the speed being marginally slower in software compared to RC4. Similarly there were interesting stream cipher proposals (such as VMPC [16], GGHN [3] etc.) with the introduction of various modifications to achieve faster encryption in software and to protect the design of the cipher against the potential vulnerabilities reported in literature. Nevertheless, some distinguishing attacks on all of the above mentioned ciphers have already been reported [2, 9, 15].

An interesting advancement proposed by some researchers towards the modifications in the RC4 design has been to increase the number of RC4 states, i.e., to increase the number of permutations in order to make the output generation dependent on more random variables which minimizes the correlation between the bytes produced. This approach has the added advantage that the number of steps performed per keystream byte produced may be made smaller than RC4 itself. This makes the ciphers designed under this paradigm faster than RC4 in software. Ciphers like RC4A [12], RC4B [4], Quad-RC4 [11] etc. have been introduced to fulfill such needs. In this paper we will concentrate on the analysis of two stream ciphers namely RC4B and Quad-RC4. The RC4B stream cipher is similar to RC4's exchange shuffle model i.e. RC4A. It also uses two different arrays of permutations. The Key-Scheduling Algorithm (KSA) of RC4B is same as that of RC4A. The Pseudo-Random Keystream Generation Algorithm (PRGA) of RC4B differs slightly from that of RC4A. In order to prevent the strong distinguisher biases [4, 14], the authors of RC4B choose to mix the two array's states. A detailed description of the RC4B stream cipher will be given in Section 2.

Quad-RC4 was first presented at a session in Indocrypt 2012 [5]. Its design focuses on building a 32-bit RC4 for a 32-word machine, however the basic 8-bit RC4 is used as a building block at every round of keystream generation. The KSA of Quad-RC4 is similar to the 3-layer KSA+ of RC4+. Since Quad-RC4 uses four different 8-bit identity permutations, the authors run the KSA+ routine on four identity permutations independently to generate four scrambled permutations over $\mathbb{Z}_{256}$. In the PRGA, the four scrambled pseudo random permutations are merged into a single array of size 256 where each element is a 32 bit number. The output byte is then produced following a certain number of operations. See Section 3 for a detailed description of the Quad-RC4 stream cipher. A description of the m-RC4 stream cipher will also be given in section 3.3 in which the authors propose a model of combining m number of different 8-bit pseudo random permutations.

1.1 Contribution and Organization of the Paper

In this paper, we analyze the security of RC4B and Quad-RC4 stream ciphers by mounting distinguishing attacks on them. The RC4B stream cipher uses two independent RC4 states in its encryption scheme. In Section 2, we will show that the probability that the first two output bytes produced by RC4B are both 0 is approximately $\frac{2}{N^2}$ ($N = 256$) which is twice the expected probability in the ideal case. In Section 3, we will show that any r-th 4-byte output word Z_r (for $r \geq 1$) produced by the Quad-RC4 stream cipher is biased and the probability

that it is equal to 0 is around $\frac{3}{N^2}$. Since in the ideal case, this probability should have been $\frac{1}{N^4}$, this represents a huge bias in the distribution. The authors of Quad-RC4 also proposed a scheme of combining m number of independent RC4 states (m-RC4) which would produce output bytes of size m bytes. We will also analyze the m-RC4 stream cipher by mounting distinguishing attacks and show that this design is still vulnerable for any m. In fact we will show that for the case of m being even (Section 3.4), the r-th output word Z_r produced by the stream cipher is biased towards 0. Furthermore in Section 3.5, we will show for the case of any m in general ($m > 2$), the probability of the first two output bytes Z_1 and Z_2 being equal is also biased. Lastly, in Section 3.6, we will discuss some flaws in the design of these stream ciphers which made them vulnerable to distinguishing attacks. We tabulate some experimental results in Section 4. We will conclude the paper in Section 5.

2 Description and Analysis of the RC4B Stream Cipher

In this section we give a detailed description of RC4B stream cipher. In addition we also analyze the stream cipher by mounting distinguishing attack on its first two output bytes Z_1 and Z_2.

2.1 Description of RC4B

The RC4B stream cipher is similar to the RC4A [12], and uses two RC4 states namely S_1 and S_2. RC4B uses the same Key Scheduling Algorithm (KSA) as RC4 and RC4A. The KSA routine is used to construct two permutations S_1 and S_2 using two keys K_1 and K_2 respectively (K_2 is usually derived as some pseudorandom function of K_1). The PRGA of RC4B is different from RC4A. Unlike RC4A, RC4B mixes the two arrays of the state. The arrays in RC4A evolve independent of the other, i.e. the *index pointers* used to update the array S_1 are generated by S_1 itself, and similarly the *index pointers* used to update S_2 are generated by S_2. This makes the cipher design vulnerable to distinguishing attacks. Therefore in RC4B this trend is reversed, the elements to be swapped in a particular array is determined by the other array. Algorithm 1 describes the PRGA of RC4B.

The key scheduling algorithm or KSA takes an array S to derive a permutation of $\{0, 1, 2, \ldots, N-1\}$ using a variable size key K. The byte-length of the Secret Key K is denoted by l. Please note that all the addition operations are done in the integer ring $\mathbb{Z}_{256}$. Three byte indices i, j_1 and j_2 are used. After performing the KSA on S_1 and S_2, the PRGA begins which produces two pseudo-random bytes Z_1 and Z_2 using the two permutations derived from KSA. The authors claim that RC4B generates keystreams faster than RC4 itself, since the number of operations performed per byte of keystream produced is lesser than that of RC4. The state space of RC4B is $N! \cdot N! \cdot N^3$ which is approximately 2^{2388} since $N = 256$. Hence it is hard to perform state recovery attack on RC4B.

Input: Pseudorandom Permutations S_1,S_2
Output: Keystream bytes Z_1, Z_2

$i \leftarrow 0$, $j_1 \leftarrow 0$, $j_2 \leftarrow 0$;
while Keystream is required **do**
  $i \leftarrow i + 1$;
  $j_1 \leftarrow j_1 + S_2[i]$;
  Swap($S_1[i]$,$S_1[j_1]$);
  $Z_1 = S_2[S_1[i] + S_1[j_1]]$;
  $j_2 \leftarrow j_2 + S_1[i]$;
  Swap($S_2[i]$,$S_2[j_2]$);
  $Z_2 = S_1[S_2[i] + S_2[j_2]]$;
end

Algorithm 1. PRGA

2.2 Analysis of RC4B

In this Subsection, we analyze the RC4B stream cipher. We refer to the PRGA Algorithm 1 of RC4B. Let the initial states of RC4B PRGA be denoted by S_1 and S_2.

Lemma 1. *Let S_1 and S_2 be random permutations on $\{0, 1, 2, \ldots, 255\}$. If $S_1[1] = 0$, $S_2[1] = X$, $S_1[X] = Y$ and $S_2[Y] = 0$, (where $X \neq 1$, $Y \neq 0$ are any two byte values) then the first two output bytes Z_1 and Z_2 are always 0.*

Proof. According to the PRGA algorithm described in Algorithm 1, initially $i = j_1 = j_2 = 0$. In the next step, the index i is incremented as $i = i + 1$. The secret index j_1 is incremented as

$$j_1 = j_1 + S_2[i] = 0 + S_2[1] = X. \tag{1}$$

After following the swap operation, $S_1[1] = Y$ and $S_1[X] = 0$. In the next step, the first output byte Z_1 is produced as

$$Z_1 = S_2[(S_1[1] + S_1[X])] = S_2[Y + 0] = 0. \tag{2}$$

The second secret index j_2 is incremented as

$$j_2 = j_2 + S_1[i] = 0 + S_1[1] = Y. \tag{3}$$

After following the next swap operation, $S_2[1] = 0$ and $S_2[Y] = X$. Thereafter, the next output byte Z_2 is given as

$$Z_2 = S_1[(S_2[1] + S_2[Y])] = S_1[X + 0] = 0. \tag{4}$$

Hence the first 2 output bytes Z_1 and Z_2 are always 0.

□

Theorem 1. *Let S_1 and S_2 be random permutations on $\{0, 1, 2, \ldots, 255\}$. The probability that $Z_1 = Z_2 = 0$ is given by the equation $\mathsf{Pr}[Z_1 = Z_2 = 0] = \frac{2}{N^2} - \frac{1}{N^4}$ where $N = 256$.*

Proof. Let E denote the event "$S_1[1] = 0$, $S_2[1] = X$, $S_1[X] = Y$ and $S_2[Y] = 0$". Then we have,

$$\mathsf{Pr}[\mathsf{E}] = \frac{1}{(N-1)} \cdot \frac{1}{(N-1)} \approx \frac{1}{N^2}.$$

From Lemma 1, we know $\mathsf{Pr}[Z_1 = Z_2 = 0|\mathsf{E}] = 1$. By standard randomness assumptions and by performing extensive computer experiments using up to 2^{25} keys, we have verified $\mathsf{Pr}[Z_1 = Z_2 = 0|\mathsf{E}^c] = \frac{1}{N^2}$ (E^c denotes the complement of the Event E). Hence the final probability is given as

$$\begin{aligned}\mathsf{Pr}[Z_1 = Z_2 = 0] &= \mathsf{Pr}[Z_1 = Z_2 = 0|\mathsf{E}] \cdot \mathsf{Pr}[\mathsf{E}] + \\ &\quad \mathsf{Pr}[Z_1 = Z_2 = 0|\mathsf{E}^c] \cdot \mathsf{Pr}[\mathsf{E}^c] \\ &= 1 \cdot \frac{1}{N^2} + \frac{1}{N^2} \cdot (1 - \frac{1}{N^2}) \\ &= \frac{2}{N^2} - \frac{1}{N^4}.\end{aligned} \tag{5}$$

□

For an ideal cipher, the probability $\mathsf{Pr}[Z_1 = Z_2 = 0]$ should be only $\frac{1}{N^2}$, so we can see that in RC4B, this probability is twice that of an ideal cipher. We now state the following theorem from [7], which outlines the number of output samples required to distinguish two distributions X and Y.

Theorem 2. *(Mantin-Shamir [7]) Let X, Y be distributions, and suppose that the event e happens in X with probability p and in Y with probability $p(1 + q)$. Then for small p and q, $O\left(\frac{1}{pq^2}\right)$ samples suffice to distinguish X from Y with a constant probability of success.*

Distinguishing RC4B from Random Sources. Let X be the probability distribution of Z_1, Z_2 in an ideal random stream, and let Y be the probability distribution of Z_1, Z_2 in streams produced by RC4B for randomly chosen keys. Let the event e denote $Z_1 = Z_2 = 0$, which occurs with probability of $\frac{1}{N^2}$ in X and $\frac{2}{N^2} - \frac{1}{N^4} \approx \frac{2}{N^2}$ in Y. By using the Theorem 2 with $p = \frac{1}{N^2}$ and $q = 1$, we can conclude that we need about $\frac{1}{pq^2} = N^2 = 2^{16}$ output samples to reliably distinguish the two distributions.

3 Description and Analysis of Quad-RC4 and *m*-RC4 Stream Ciphers

In this section we describe Quad-RC4 and m-RC4 stream ciphers. We also demonstrate distinguishing attacks on Quad-RC4 and m-RC4 by proving biases in their output bytes.

3.1 Description of Quad-RC4

The rationale behind the design of Quad-RC4 was the optimal utilization of the resources on the modern processors which are mostly 32 bits. The authors take a single l-byte key ($16 \leq l \leq 30$) which is used to drive 4 different key scheduling in parallel to obtain 4 different permutations over $\mathbb{Z}_{256}$. Two byte indices i and j are used where j is kept secret. The authors of Quad-RC4 argue that their scheme is more secure than the basic RC4 encryption scheme. The key scheduling for Quad-RC4 is same as the KSA+ of the stream cipher RC4+ [6]. The KSA+ consists of 3 layers: the basic scrambling of the first layer of KSA+ is similar to the RC4 KSA. Algorithms 2 and 3 describes the other two layers namely IV Scrambling and Zig-Zag Scrambling of the KSA+ respectively. All the addition operations are performed in $\mathbb{Z}_{256}$ and $\oplus$ denotes the bitwise `XOR`. The array V used in the Algorithm 2 is of length $N = 256$ and is defined as

$$V[i] = \begin{cases} IV[\frac{N}{2} - 1 - i] & \text{if } \frac{N}{2} - l \leq i \leq \frac{N}{2} - 1 \\ IV[i - \frac{N}{2}] & \text{if } \frac{N}{2} \leq i \leq \frac{N}{2} + l - 1 \\ 0 & \text{otherwise} \end{cases}$$

Input: S, K, V
Output: Scrambled S

---

for $i = N/2 - 1$ to 0 **do**
  $j = (j + S[i]) \oplus (K[i] + V[i])$;
  Swap($S[i]$,$S[j]$);
end

for $i = N/2 to N - 1$ **do**
  $j = (j + S[i]) \oplus (K[i] + V[i])$;
  Swap($S[i]$,$S[j]$);
end

Algorithm 2. Mix-IV

Input: S, K
Output: Scrambled S

---

for $y = 0$ to $N - 1$ **do**
  if $y \equiv 0 \mod 2$ **then**
    $i = \frac{y}{2}$;
  end
  else
    $y = N - \frac{y+1}{2}$;
  end
  $j = (j + S[i] + K[i])$;
  Swap($S[i]$,$S[j]$);
end

Algorithm 3. Zig-Zag

Let S_1, S_2, S_3 and S_4 denote the 4 pseudo random permutations over $\mathbb{Z}_{256}$ produced after running the key scheduling algorithm. They are merged into a single array S of size 256, where the i-th entry of S is an 32-bit number formed by concatenating the 4 bytes $S_1[i], \ldots, S_4[i]$. Algorithm 4 describes the PRGA of Quad-RC4. Please note that $\ll$ and $\gg$ denotes left and right bitwise shifts respectively. The $|$ and & signs represent bitwise *OR* and *AND* whereas $\oplus$ represents the bitwise `XOR`.

Input: 4 pseudo random permutations over $\mathbb{Z}_{256}$
Output: 32-bit output words

$i = j = 0$;
for $i = 0$ to 255 **do**
  $S[i] = (S_1[i] \ll 24)|(S_2[i] \ll 16)|(S_3[i] \ll 8)|S_4[i]$;
end
while Keystream is required **do**
  $i = (i + 1) \mod 256$;
  $j = (j + S_4[i]) \mod 256$;
  Swap($S[i]$, $S[j]$);
  $t = (S[i] + S[j]) \mod 2^{32}$;
  $t_1 = t$&`0xFF`;
  $t_2 = (t \gg 8)$&`0xFF`;
  $t_3 = (t \gg 16)$&`0xFF`;
  $t_4 = (t \gg 24)$&`0xFF`;
  Output $Z = S[t_1] \oplus S[t_2] \oplus S[t_3] \oplus S[t_4]$;
  $\{a, b\}$ =Next pair of permutations in turn;
  Swap($S_a[i]$,$S_a[t_a]$);
  Swap($S_b[i]$,$S_b[t_b]$);
end

Algorithm 4. Quad-RC4 PRGA Routine

The authors introduce some additional swaps to break the symmetry in the swaps of the individual permutations. Two permutations S_a and S_b are selected at every round. Thereafter the i-th and the t_a-th bytes of S_a and the i-th and the t_b-th bytes of S_b are swapped. Note that t_a and t_b are the a-th and the b-th bytes of $t = t_1 \| \ldots \| t_4$. Note that 2 permutations out of 4 can be selected in 6 ways.

3.2 Analysis of Quad-RC4

In this subsection we present the analysis of the Quad-RC4 stream cipher by demonstrating a distinguishing attack. We refer to the PRGA of Quad-RC4 presented in Algorithm 4.

Some Notations: Let the initial states of four 8-bit RC4 state permutations be represented as S_1, S_2, S_3 and S_4. Let S denote the 256-element array whose i-th entry is formed by concatenating the i-th bits of these four RC4 states. Let $t_1, \ldots, t_4$ denotes variables of size 1 byte each and $t = t_1 \| \ldots \| t_4$ be a 32-bit variable. Before proceeding, let us define the event E_τ described as follows.

The Event: *Consider the four elements t_1, t_2, t_3, t_4. Partition these elements into 2 groups G_1, G_2 (there are $\frac{\binom{4}{2}}{2} = 3$ ways to do so). The event E_τ will be said to have occurred if one of the two conditions are satisfied:*

1. All the t_i's are equal i.e. $t_1 = t_2 = t_3 = t_4$.
2. All the t_i's are not equal but the two elements in G_1 are equal and the two elements in G_2 are also equal.

Lemma 2. *If the event E_τ described above occurs, the 4-byte output word Z_r in any PRGA round r is always equal to* 0.

Proof. In any PRGA round r of Quad-RC4, the variable t is assigned a value as follows

$$t = (S[i] + S[j]) \mod 2^{32}$$

In the subsequent operations, $t_1, \ldots, t_4$ are assigned the values

$$t_1 = t\&\texttt{0xFF},\ t_2 = (t \gg 8)\&\texttt{0xFF},\ t_3 = (t \gg 16)\&\texttt{0xFF},\ t_4 = (t \gg 24)\&\texttt{0xFF}.$$

Now the event E_τ can occur in 2 ways:

1. If $t_1 = t_2 = t_3 = t_4 = a$ (say). In that case $Z_1 = S[a] \oplus S[a] \oplus S[a] \oplus S[a] = 0$.
2. If two elements of G_1 are equal and the two elements of G_2 are also equal. Without loss of generality assume that G_1 consists of t_1, t_2 and G_2 of t_3, t_4 and $t_1 = t_2 = a$ and $t_3 = t_4 = b$. In this case

$$Z_r = S[a] \oplus S[a] \oplus S[b] \oplus S[b] = 0 \oplus 0 = 0$$

So it is evident that the value of Z_r is always going to be 0 under the occurrence of E_τ. □

In general, the number of ways one can divide 4 things in 2 groups where each group has 2 things is $\frac{\binom{4}{2}}{2}$. For each such group, say (t_1, t_2) and (t_3, t_4) as shown earlier, let's have $t_1 = t_2 = a$ and $t_3 = t_4 = b$. Here a can take N values and b can take $N-1$ values ($N = 256$) giving $N \cdot (N-1) \cdot \frac{\binom{4}{2}}{2} = 3N^2 - 3N$ total ways the elements can be partitioned. Also $t_1 = t_2 = t_3 = t_4$ can occur in exactly N ways, and so the event E_τ can occur in total $3N^2 - 3N + N = 3N^2 - 2N$ ways. The probability that E_τ occurs if we choose the indices t_1, t_2, t_3, t_4 randomly is therefore given as $\mathsf{Pr}[E_\tau] = \frac{3N^2-2N}{N^4}$.

Theorem 3. *The probability that the 4-byte output word Z_r, produced by Quad-RC4 in any PRGA round r, is equal to* 0 *is given by the equation* $\mathsf{Pr}[Z_r = 0] \approx \frac{3}{N^2}$.

Proof. We have established that $\mathsf{Pr}[E_\tau] = \frac{3N^2-2N}{N^4}$. Since Z_r is always 0 under the event E_τ, therefore we have $\mathsf{Pr}[Z_r = 0|E_\tau] = 1$. By the results of the extensive computer experiments performed using 2^{30} keys, we have verified that $\mathsf{Pr}[Z_r = 0|E_\tau^c] = \frac{1}{N^4}$. Therefore the final probability can be given as

$$\begin{aligned} \mathsf{Pr}[Z_r = 0] &= \mathsf{Pr}[Z_r = 0|E_\tau] \cdot \mathsf{Pr}[E_\tau] + \mathsf{Pr}[Z_r = 0|E_\tau^c] \cdot \mathsf{Pr}[E_\tau^c] \\ &= 1 \cdot \frac{3N^2 - 2N}{N^4} + \frac{1}{N^4} \cdot (1 - \frac{3N^2 - 2N}{N^4}) \\ &= \frac{3N^2 - 2N + 1}{N^4} \approx \frac{3}{N^2}. \end{aligned} \tag{6}$$

□

Thus we see a huge bias present in the output words produced Quad-RC4. In an ideal cipher, the probability of Z_r being equal to 0 was required to be $\frac{1}{N^4}$. Thus in Quad-RC4, the value of $\mathsf{Pr}[Z_r = 0]$ is $3N^2 \approx 2^{17.5}$ times that of an ideal cipher. It is clear that the design paradigm of the stream cipher is vulnerable. This has long term implications in the broadcast scenario, in which a single plaintext is encrypted by several randomly generated secret keys, and broadcast over a network. The attacker can perform a ciphertext only plaintext recovery attack if Quad-RC4 is used for encryption. For example if an attacker can collect around 2^{20} broadcast ciphertexts, then following our analysis we can say that any r-th 4-byte word of the plaintext would have been encrypted with around $\frac{3 \cdot 2^{20}}{N^2} \approx 48$ zeroes. Thus the attacker can do a simple statistical test: the most frequent r-th ciphertext word is also likely to be the r-th plaintext word. Since the attack works for any r, it makes the entire plaintext easily recoverable. In the upcoming subsection we will generalize our results for m-RC4 and present our analysis for odd and even m.

3.3 Description of m-RC4

The m-RC4 stream cipher is similar to the Quad-RC4 stream cipher explained in the previous section with the difference being the number of different 8-bit RC4 states to be combined here is m and the output bytes produced through a suitable function h which takes the quantities $S[t]$, $S[i]$, $S[j]$ and $S[t_p]$ where $p = 1, \ldots, m$. However the authors argue that if h simply returns the bitwise `XOR`

Input: m pseudo random permutations over $\mathbb{Z}_{256}$
Output: $8m$-bit output words

$i = j = 0$;
for $i = 0$ to 255 **do**
  $S[i] = (S_1[i] \ll 8(m-1)) | \ldots | (S_{m-1}[i] \ll 8) | S_m[i]$;
end
while Keystream is required **do**
  $i = (i + 1) \mod 256$;
  $j = (j + S_m[i]) \mod 256$;
  $t = (S[i] + S[j]) \mod 2^{8m}$;
  $t_1 = t \& \texttt{0xFF}; t_2 = (t \gg 8) \& \texttt{0xFF}; \ldots t_m = (t \gg 8(m-1)) \& \texttt{0xFF}$;
  Swap($S[i]$, $S[j]$);
  Output $Z = h(S[t], S[i], S[j], S[t_1], S[t_2], \ldots, S[t_m])$;
  $\{a, b\}$ =Next pair of permutations in turn;
  Swap($S_a[i]$,$S_a[t_a]$);
  Swap($S_b[i]$,$S_b[t_b]$);
end

Algorithm 5. m-RC4 PRGA Routine

of its quantities, then the keystreams produce is not perfectly random. Thus the design of h with good randomness properties is left by the authors as an open problem. The PRGA of m-RC4 takes m different pseudo random permutations over $\mathbb{Z}_{256}$ produced by applying m number of key scheduling namely KSA+. Let $S_1, \ldots, S_m$ be the pseudo random permutations produced. They are merged into a single array S of size 256, where the i-th entry of S is an $8m$-bit number formed by concatenating the m many bytes $S_1[i], \ldots, S_m[i]$. Algorithm 5 describes the PRGA of the m-RC4 stream cipher.

3.4 Analysis for Even m

In this subsection we present an analysis for even m. The analysis is similar to that for Quad-RC4. Before presenting the analysis, we will go through some general notations which will be used by us during the proofs.

Some Notations and Assumptions: Let the initial states of m 8-bit RC4 state permutations be represented as $S_1, S_2, S_3, \ldots, S_m$. Let S denote the 256-element array whose i-th entry is formed by concatenating the i-th bits of these m RC4 states. Let $t_1, \ldots, t_m$ denotes variables of size 1 byte each and $t = t_m \| \ldots \| t_1$ be an $8m$-bit variable. Since the design of an output function (namely h in Algorithm 5) with good randomness properties is left by the authors as an open problem, in our analysis we will assume that as in Quad-RC4, the function simply returns the bitwise `XOR` of $S[t_1], S[t_2], \ldots, S[t_m]$.

The Event: *We will start with the definition of the event E_τ for m-RC4, along similar lines as in Quad-RC4. Consider the m elements $t_1, t_2, \ldots, t_m$. Partition these elements into k groups $G_1, G_2, \ldots, G_k$ ($1 \leq k \leq \frac{m}{2}$), such that the cardinality of each group G_i is even. The event E_τ will be said to have occurred if the following occurs:*

1. *The elements in each of the $G_i's$ are equal.*

Note that when $k = 1$, it denotes the degenerate case when all the $t_i's$ are equal. So this definition is consistent with the definition of E_τ given for Quad-RC4.

So, there are exactly $f(\frac{m}{2})$ ways in which the groups G_i can be formed, where f denotes the well-known partition function (i.e. the number of ways of writing an integer as sum of positive integers) [1]. For example, when $m = 6$, the number of partitions are $f(3) = 3$, i.e., 6, $4 + 2$ and $2 + 2 + 2$. In the first case we have $t_1 = t_2 = \ldots = t_6$ and it can happen in N ways where $N = 256$. In the second case, we need to divide the 6 elements into 2 groups of 4 and 2. Without loss of generality let us have $t_1 = t_2 = t_3 = t_4 = a$ and $t_5 = t_6 = b$. The number of ways we can divide 6 elements in groups of 4 and 2 is $\binom{6}{2} = 15$ and the number of ways we can select a, b is $N \cdot (N-1)$. So the total number of ways is $15 \cdot N \cdot (N-1)$. Lastly we have 2 elements in each of the 3 groups and the number of ways it can happen is given as $\frac{\binom{6}{2} \cdot \binom{4}{2}}{3!} = 15$ and the number of ways the elements of

each group can be selected is $N \cdot (N-1) \cdot (N-2)$ and so the total number of combinations is $15 \cdot N \cdot (N-1) \cdot (N-2)$. Therefore the total number of ways for all the partitions can be given as $15 \cdot N \cdot (N-1) \cdot (N-2) + 15 \cdot N \cdot (N-1) + N$, and so the for $m = 6$, we have

$$\mathsf{Pr}[E_\tau] = \frac{15 \cdot N \cdot (N-1) \cdot (N-2) + 15 \cdot N \cdot (N-1) + N}{N^6}$$

Similarly, when $m = 8$, the number of partitions are $f(4) = 5$ which are 8, $6+2$, $4+2+2$, $4+4$ and $2+2+2+2$. For an arbitrary even value of $m = 2p$, it is difficult to analytically determine the value of $\mathsf{Pr}[E_\tau]$. However, we can find a lower bound for this probability. In the case of $m = 2p$, we know that total partition is denoted by $f(p)$. However, the dominant partition, that is the one that will contribute the maximum number of combinations will be

$$\underbrace{2+2+\ldots+2}_{p \text{ times}}.$$

We call this event as E_m which denotes the situation when $t_1, \ldots, t_m$ are divided into p groups having 2 elements each and elements of any given group are equal to each other. The number of ways of dividing $m = 2p$ items in p groups of 2 each is

$$\frac{\binom{2p}{2} \cdot \binom{2p-2}{2} \cdot \ldots \cdot \binom{2}{2}}{p!} = \frac{2p!}{p!2^p} = \prod_{i=1}^{p}(2i-1) = B_m \text{ (say)}. \tag{7}$$

B_m is therefore the product of the first $\frac{m}{2} = p$ odd integers. So the number of ways in which the event E_m can occur is given by

$$B_m \cdot N \cdot (N-1) \cdot (N-2) \ldots (N-p+1) \approx B_m \cdot N^p. \tag{8}$$

Therefore we have

$$\mathsf{Pr}[E_\tau] \approx \mathsf{Pr}[E_m] = \frac{B_m \cdot N^p}{N^m} = \frac{B_m}{N^p}$$

Lemma 3. *If the event E_τ described above occurs, the any m-byte output word Z_r, produced in PRGA round r, is always equal to* 0.

Proof. The proof is similar to Lemma 2, and follows from the definition of E_τ. If E_τ occurs then the t_i's are divided into k groups each having an even number of elements which are equal. In that case the output byte Z_r is given as

$$Z_r = (S[a_1] \oplus \cdots S[a_1]) \oplus (S[a_2] \oplus \cdots S[a_2]) \oplus \cdots \oplus (S[a_k] \oplus \cdots S[a_k]) = 0. \tag{9}$$

□

Theorem 4. *The probability that $Z_r = 0$ in m-RC4, is given by the equation $\mathsf{Pr}[Z_r = 0] = \frac{B_m}{N^p}$ where $\frac{m}{2} = p$.*

Proof. We know that $\mathsf{Pr}[E_\tau] = \frac{B_m}{N^p}$. Since Z_r is always 0 under the event E_τ, therefore we have $\mathsf{Pr}[Z_r = 0|E_m] = 1$. By the results of the extensive computer experiments performed using 2^{30} keys, we have verified $\mathsf{Pr}[Z_r = 0|E_m^c] = \frac{1}{N^m}$. Therefore the final probability can be given as

$$\begin{aligned} \mathsf{Pr}[Z_r = 0] &= \mathsf{Pr}[Z_r = 0|E_m] \cdot \mathsf{Pr}[E_m] + \mathsf{Pr}[Z_r = 0|E_m^c] \cdot \mathsf{Pr}[E_m^c] \\ &= 1 \cdot \frac{B_m}{N^p} + \frac{1}{N^m} \cdot (1 - \frac{B_m}{N^p}) \\ &= \frac{B_m}{N^p}. \end{aligned} \tag{10}$$

Note that in an ideal cipher the probability $\mathsf{Pr}[Z_r = 0]$ was required to be $\frac{1}{N^m} = \frac{1}{N^{2p}}$. So in m-RC4 this probability is $B_m \cdot N^p$ times that of an ideal cipher. For example in 10-RC4, this figure is $945 \cdot 2^{40} \approx 2^{50}$. So as m increases the design becomes increasingly weaker. □

3.5 Analysis for General m

Previously we demonstrated our analysis for m-RC4 in case when m was an even number. In this subsection we will present our analysis in case when m is any integer greater than equal to 4. We will show that the probability of the first and the second output bytes being equal is biased.

Some Other Notations: We we will denote the m 8-bit RC4 state permutations at the beginning of PRGA round r as $S_1^r, S_2^r, S_3^r, \ldots, S_m^r$. Similarly S^r denotes the the state of 256-element array S (whose i-th entry is formed by concatenating the i-th bits of these m RC4 states) at the beginning of round r. Similarly $t_1^r, \ldots, t_m^r$ and t^r denote the values of the variables $t_1, \ldots, t_m$ and t respectively in round r.

Lemma 4. *If $S_m^1[1] = 2$ and $S_m^1[2] = N - 1$, then*

a) *The value of the of the variables t^1 and t^2 are both equal,*
b) *The values of t_1^1 and t_1^2 are both equal to 1.*

Proof. We refer to the PRGA (5) of m-RC4. The index pointers i and j are incremented as $i = 0+1 = 1$ and $j = 0 + S_m^1[1] = 2$. Let us denote $S^1[1] = x$ and $S^1[2] = y$ (where x, y are distinct m-byte integers). The variable t^1 is updated in the next step as follows

$$t^1 = S^1[1] + S^1[2] \bmod 2^{8m} = x + y \bmod 2^{8m}$$

We also have

$$\begin{aligned} t_1^1 = t^1 \ \& \ \texttt{0xFF} &= (S^1[1] + S^1[2]) \ \& \ \texttt{0xFF} \\ &= (S^1[1] \ \& \ \texttt{0xFF}) + (S^1[2] \ \& \ \texttt{0xFF}) \bmod 256 \\ &= S_m^1[1] + S_m^1[2] \bmod 256 = \ 1. \end{aligned}$$

In the next operation, $S^1[1]$ and $S^1[2]$ are swapped. After the swap we have $S^2[1] = y$ and $S^2[2] = x$. Then in the next round, the index pointers i and j are incremented as $i = 1+1 = 2$ and $j = 2+S_m^2[1] = 2+S_m^1[1] = 2+N-1 \bmod 256 = 1$. The above follows, because as per the specifications of m-RC4, the array S_m is not shuffled in the first round, and so $S_m^1 = S_m^2$. The variable t^2 is updated in the next step as:

$$t^2 = S^2[2] + S^2[1] \bmod 2^{8m} = x + y \bmod 2^{8m} = t^1.$$

Again we have,

$$t_1^2 = t^2 \ \& \ \texttt{0xFF} = t^1 \ \& \ \texttt{0xFF} = t_1^1 = 1. \tag{11}$$

□

Since it was established in Lemma 4, that $t^1 = t^2$, we automatically have $t_1^1 = t_1^2$, $t_2^1 = t_2^2$,..., $t_m^1 = t_m^2$. Furthermore it has already been shown that $t_1^1 = t_1^2 = 1$. For convenience let us denote $t_k^1 = t_k^2 = b_k$ for all $2 \leq k \leq m$. Hence we can write

$$Z_1 = S^1[1] \oplus S^1[b_2] \oplus S^1[b_3] \oplus \cdots \oplus S^1[b_m]$$

$$Z_2 = S^2[1] \oplus S^2[b_2] \oplus S^2[b_3] \oplus \cdots \oplus S^2[b_m]$$

From the proof of Lemma 4, it is also clear that the array S^1 and S^2 differ in only two locations which are 1 and 2. We have $S^1[1] = S^2[2] = x$ and $S^1[2] = S^2[1] = y$. For all other k not equal to 1 or 2 we have $S^1[k] = S^2[k]$. So consider the following non-intersecting cases:

Case 1. If the number of $1's$ among $b_2, b_3, \ldots, b_m$, is odd and the number of $2's$ among $b_2, b_3, \ldots, b_m$ is even. In this case

$$Z_1 = \bigoplus_{even} S^1[1] \ \oplus \ \bigoplus_{even} S^1[2] \ \oplus \ \bigoplus_{b_k \neq 1,2} S^1[b_k] = \bigoplus_{b_k \neq 1,2} S^1[b_k]$$

$$Z_2 = \bigoplus_{even} S^2[1] \ \oplus \ \bigoplus_{even} S^2[2] \ \oplus \ \bigoplus_{b_k \neq 1,2} S^2[b_k] = \bigoplus_{b_k \neq 1,2} S^2[b_k]$$

where $\bigoplus_{even} u$ denotes the bitwise XOR of u even number of times which is obviously 0. Since $S^1[b_k] = S^2[b_k]$ if b_k is different from 1 or 2 we have $Z_1 = Z_2$.

Case 2. If the number of $1's$ among $b_2, b_3, \ldots, b_m$, is even and the number of $2's$ among $b_2, b_3, \ldots, b_m$ is odd. Let the number of $1's$ be c and the number of $2's$ be d, where c is even and d is odd. Without loss of generality, let $c + 1 \geq d$. In this case

$$\begin{aligned} Z_1 &= \bigoplus_{d} \left(S^1[1] \oplus S^1[2]\right) \ \oplus \ \bigoplus_{c+1-d} S^1[1] \ \oplus \ \bigoplus_{b_k \neq 1,2} S^1[b_k] \\ &= (x \oplus y) \ \oplus \ \bigoplus_{b_k \neq 1,2} S^1[b_k] \end{aligned}$$

$$\begin{aligned} Z_2 &= \bigoplus_{d} \left(S^2[1] \oplus S^2[2]\right) \ \oplus \ \bigoplus_{c+1-d} S^2[1] \ \oplus \ \bigoplus_{b_k \neq 1,2} S^2[b_k] \\ &= (x \oplus y) \ \oplus \ \bigoplus_{b_k \neq 1,2} S^2[b_k] \end{aligned}$$

Here $\bigoplus_n u$ denotes the bitwise XOR of u, a total of n times which is 0 if n is even and u if n is odd. Since d is odd and $c + 1 - d$ is even, the above result follows. Again since $S^1[b_k] = S^2[b_k]$ if b_k is different from 1 or 2 we have $Z_1 = Z_2$.

Lemma 5. *The event E_μ will said to have occurred if both the events occur simultaneously.*

a) $S^1_m[1] = 2$ *and* $S^1_m[2] = N - 1$,
b) *Either* **Case 1** *or* **Case 2**, *described above holds true.*

The probability that $\mathsf{Pr}[E_\mu] \approx \frac{2(m-1)}{N^3}$.

Proof. First of all, from all the previous discussion we have established that $\mathsf{Pr}[Z_1 = Z_2 | E_\mu] = 1$. Turning our attention to **Case 1**/**Case 2**, we can make the following observation:

– Both **Case 1** and **Case 2**, involve fixing an odd number of values among $b_2, b_3, \ldots, b_m$ to either 1 or 2.

Starting with **Case 1**, if the number of fixed values is 1, this implies only one of the $b'_k s$ equal 1. Hence, the number of combinations is $C_1 = \binom{m-1}{1} \cdot (N-2)^{m-2}$. The above holds since, the index to be set to 1 can be chosen in $\binom{m-1}{1}$ ways and the remaining $m-2$ indices can be set to any value other than 1 or 2. Similarly, if the number of fixed values is 3, this implies either three $1's$ or one 1 and two $2's$. The total number of combinations is therefore

$$C_3 = \binom{m-1}{3} \cdot (N-2)^{m-4} + \binom{m-1}{3} \cdot \frac{3!}{2!} \cdot (N-2)^{m-4}$$

It is clear that if m is much smaller that N then, C_1 is much larger than C_3. Similarly, C_3 would be much larger than C_5, i.e. the number of combinations

when 5 indices are fixed, and so on. Hence C_1 contributes the maximum number of combinations to **Case 1** and we have $C_1 \approx (C_1 + C_3 + C_5 + \cdots)$. So we have

$$\mathsf{Pr}[\textbf{Case 1}] \approx \frac{C_1}{N^{m-1}} = \frac{\binom{m-1}{1} \cdot (N-2)^{m-2}}{N^{m-1}} \approx \frac{m-1}{N}$$

Analogously, similar arguments can be made about **Case 2**, and since these cases are non-intersecting we have

$$\mathsf{Pr}[\textbf{Case 1} \vee \textbf{Case 2}] \approx \frac{2(m-1)}{N}.$$

And so that the events $S_m^1[1] = 2$ and $S_m^1[2] = N-1$ and **Case 1** $\vee$ **Case 2** are independently distributed, we have

$$\begin{aligned}\mathsf{Pr}[E_\mu] &= \mathsf{Pr}[(S_m^1[1] = 2) \ \wedge \ (S_m^1[2] = N-1) \ \wedge \ (\textbf{Case 1} \vee \textbf{Case 2})] \\ &= \frac{1}{N} \cdot \frac{1}{N-1} \cdot \frac{2(m-1)}{N} \approx \frac{2(m-1)}{N^3}.\end{aligned}$$

□

Theorem 5. *The probability that $Z_1 = Z_2$ in m-RC4 (for $m > 2$), is given by the equation* $\mathsf{Pr}[Z_1 = Z_2] = \frac{2(m-1)}{N^3}$.

Proof. We have already established that $\mathsf{Pr}[Z_1 = Z_2 | E_\mu] = 1$. By the results of the extensive computer experiments performed using 2^{30} keys, we have verified $\mathsf{Pr}[Z_1 = Z_2 | E_\mu^c] = \frac{1}{N^m}$. Therefore the final probability can be given as

$$\begin{aligned}\mathsf{Pr}[Z_1 = Z_2] \ &= \ \mathsf{Pr}[Z_1 = Z_2 | E_\mu] \cdot \mathsf{Pr}[E_\mu] + \mathsf{Pr}[Z_1 = Z_2 | E_\mu^c] \cdot \mathsf{Pr}[E_\mu^c] \\ &= \ 1 \cdot \frac{2(m-1)}{N^3} + \frac{1}{N^m} \cdot \left(1 - \frac{2(m-1)}{N^3}\right) \\ &\approx \ \frac{2(m-1)}{N^3}.\end{aligned} \tag{12}$$

In an ideal cipher $\mathsf{Pr}[Z_1 = Z_2]$ was required to be $\frac{1}{N^m}$. So, in m-RC4 this probability is $2(m-1) \cdot N^{m-3}$ times that of an ideal cipher. For example in 3-RC4 this figure is 4, for 7-RC4 this figure is $12 \cdot 2^{32} \approx 2^{36}$. This underscores the point that the design is vulnerable for any m.

□

3.6 The Flaws in the Design

In this subsection we will discuss the flaws in the design of Quad-RC4/m-RC4 which results in highly biased output bytes.

Simple `XOR` operation in Output Function. The output function based on simple bitwise `XOR` of $S[t_1], S[t_2], \ldots, S[t_m]$ when m is even is clearly not a

good idea. From our analysis presented in the Section 3.4 for even m, one can clearly understand that the reason for the high bias in Z_r is the simplicity of the output function is a simple `XOR`. The output function also contributes to bias in $Z_1 = Z_2$ for any general value of m. Thus the output function needs to be changed to some operation which would involve modular addition, rotation and `XOR` (ARX functions). This may result in some degradation of performance in software with respect to speed, but this is one correction that the design must make to be secure.

The "not so random" S. Referring to the PRGA of m-RC4 given in Algorithm 5, it is clear that the main array S used in the algorithm is only updated by swaps. In original RC4, swap update works because RC4 state is a permutation on $\mathbb{Z}_{256}$. Each swap makes the original RC4 state a new permutation on $\mathbb{Z}_{256}$, and therefore the total entropy in the state is $\log_2 256! \approx 1684$ bits. However, the array S used in m-RC4 is not a permutation on $\mathbb{Z}_{2^{8m}}$. Once the cipher array enters the PRGA phase, the array S is updated by only swap operations, this means that during the entire PRGA phase S contains the same elements. So for a fixed Key/IV, thee entropy of the state space comes only from the permutation of the 256 elements of S, which is again 1684 bits. Once m-RC4 states are used, thee designers would have wanted the entropy of the State Space to be as close to $1684m$ bits as possible. However due to the simplistic state update function which consists of only swaps, the entropy never increases. This implies that the for the design to be secure, a more complicated state update was necessary.

The effect on t. This effect of the reduced state entropy is directly felt on the $8m$-bit variable t. The designers had probably intended t to be a pseudorandom index with entropy close to $8m$ bits. However t is calculated as $S[i] + S[j] \bmod 2^{8m}$. Because of the simplistic nature of the state-update, the array S, contains the same N elements during the entire evolution of the PRGA. As a result $S[i], S[j]$ can take at most N values each, and hence t can take at most N^2 values. Due to this the entropy of t is only $\log_2 N^2 = 16$ bits for any value of m. Thus the probability that the values of t in two successive PRGA rounds are equal is only about $\frac{1}{N^2}$, and it can be seen that this directly contributes to the bias in the distribution of $Z_1 = Z_2$, as described in Section 3.5. This further emphasizes the point that in a design like Quad-RC4/m-RC4, which combines several RC4 states by simple concatenation, the state update can *not* be a simple swap operation. A more complicated update using modular addition/`XOR` is necessary. This would again involve decrease of software speed, but this is again a necessary correction that the design must make to be secure.

4 Experimental Results

We will now tabulate some experimental results to validate our theoretical findings. The results can be found in Table 1. All the experiments were performed

Table 1. Experimental Results

#	Cipher	Event	Theoretical Value	Experimental Result	N
1	RC4B	$\mathsf{Pr}[Z_1 = Z_2 = 0]$	$2/N^2$	$1.99/N^2$	256
2	Quad-RC4	$\mathsf{Pr}[Z_1 = 0]$	$3/N^2$	$2.94/N^2$	256
	6-RC4		$15/N^3$	$15.53/N^3$	64
3	Quad-RC4	$\mathsf{Pr}[Z_1 = Z_2]$	$6/N^3$	$6.13/N^3$	128
	5-RC4		$8/N^3$	$7.84/N^3$	128

with 2^{30} randomly chosen Keys, and in certain cases we have reported the probability values for reduced variants of the actual cipher i.e. for $N = 64, 128$ in place of 256. This was done only because the computational resources required to perform the experiments on the full ciphers were unavailable. As can be seen in Table 1 the experimental results are in accordance with our theoretical findings.

5 Conclusion

In this paper we discuss the stream ciphers which are designed by combining one or more independently produced RC4 states. We study three such stream ciphers namely RC4B, Quad-RC4 and m-RC4 (where m can be odd or even), and demonstrate distinguishing attacks on them by showing biases in the output bytes produced by each of these stream ciphers. In addition we also discuss the scenarios which leads to such vulnerabilities present in the output bytes of these stream ciphers. Combining multiple stream cipher states to produce new stream ciphers which perform better than the original stream cipher seems to be an attractive research discipline as evidenced by numerous papers in this area [4, 10, 11]. As far as combining RC4 states are concerned, much can be achieved if the problem is addressed judiciously. This does seem to be an area worth looking into.

References

1. Partition (Number Theory), `http://en.wikipedia.org/wiki/Partition_%28number_theory%29`
2. Banik, S., Sarkar, S., Kacker, R.: Security analysis of the RC4+ stream cipher. In: Paul, G., Vaudenay, S. (eds.) INDOCRYPT 2013. LNCS, vol. 8250, pp. 297–307. Springer, Heidelberg (2013)
3. Gong, G., Gupta, K.C., Hell, M., Nawaz, Y.: Towards a general RC4-like keystream generator. In: Feng, D., Lin, D., Yung, M. (eds.) CISC 2005. LNCS, vol. 3822, pp. 162–174. Springer, Heidelberg (2005)
4. Lv, J., Zhang, B., Lin, D.: Distinguishing Attacks on RC4 and A New Improvement of the Cipher. Cryptology ePrint Archive: Report 2013/176
5. Maitra, S.: Four Lines of Design to Forty Papers of Analysis: The RC4 Stream Cipher, `http://www.isical.ac.in/~indocrypt/indo12.pdf`

6. Maitra, S., Paul, G.: Analysis of RC4 and proposal of additional layers for better security margin. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 27–39. Springer, Heidelberg (2008)
7. Mantin, I., Shamir, A.: A practical attack on broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer, Heidelberg (2002)
8. Maximov, A., Khovratovich, D.: New state recovery attack on RC4. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 297–316. Springer, Heidelberg (2008)
9. Maximov, A.: Two linear distinguishing attacks on VMPC and RC4A and weakness of RC4 family of stream ciphers. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 342–358. Springer, Heidelberg (2005)
10. Paul, G., Chattopadhyay, A.: Designing stream ciphers with scalable data-widths: a case study with HC-128. J. Cryptographic Engineering 4(2), 135–143 (2014)
11. Paul, G., Maitra, S., Chattopadhyay, A.: Quad-RC4: Merging Four RC4 States towards a 32-bit Stream Cipher. IACR Cryptology eprint Archive 2013: 572 (2013)
12. Paul, S., Preneel, B.: A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 245–259. Springer, Heidelberg (2004)
13. 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)
14. Sarkar, S.: Further non-randomness in RC4, RC4A and VMPC. Cryptography and Communications 7(3), 317–330 (2015)
15. Tsunoo, Y., Saito, T., Kubo, H., Shigeri, M., Suzaki, T., Kawabata, T.: The Most Efficient Distinguishing Attack on VMPC and RC4A. In: SKEW 2005 (2005), `http://www.ecrypt.eu.org/stream/papers.html`
16. Zoltak, B.: VMPC one-way function and stream cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 210–225. Springer, Heidelberg (2004)