Cryptanalysis of $RC4(n, m)$ Stream Cipher

Mohammad Ali Orumiehchiha[1], Josef Pieprzyk[1], Elham Shakour[2] and Ron Steinfeld[3]

[1]Center for Advanced Computing, Algorithms and Cryptography, Department of Computing,
Faculty of Science, Macquarie University, Sydney, NSW 2109, Australia
(mohammad.orumiehchiha,josef.pieprzyk)@mq.edu.au
[2]Faculty of Mathematics, Amirkabir University,Tehran, Iran
elham.shakoor@gmail.com
[3]Clayton School of Information Technology
Monash University, Clayton VIC 3800, Australia
ron.steinfeld@monash.edu

Abstract. $RC4(n, m)$ is a stream cipher based on RC4 and is designed by G. Gong *et al.*. It can be seen as a generalization of the famous RC4 stream cipher designed by Ron Rivest. The authors of $RC4(n, m)$ claim that the cipher resists all the attacks that are successful against the original RC4. The paper reveals cryptographic weaknesses of the $RC4(n, m)$ stream cipher. We develop two attacks. The first one is based on non-randomness of internal state and allows to distinguish it from a truly random cipher by an algorithm that has access to $2^{4 \cdot n}$ bits of the keystream. The second attack exploits low diffusion of bits in the KSA and PRGA algorithms and recovers all bytes of the secret key. This attack works only if the initial value of the cipher can be manipulated.
Apart from the secret key, the cipher uses two other inputs, namely, initial value and initial vector. Although these inputs are fixed in the cipher specification, some applications may allow the inputs to be under the attacker control. Assuming that the attacker can control the initial value, we show a distinguisher for the cipher and a secret key recovery attack that for the L-bit secret key, is able to recover it with about $(L/n) \cdot 2^n$ steps. The attack has been implemented on a standard PC and can reconstruct the secret key of RC(8,32) in less than a second.

Keywords: $RC4(n, m)$ Stream cipher; Cryptanalysis; Key Recovery Attack; Distinguishing Attack; RC4-like cipher; Weak Keys; Weak States

1 Introduction

The well-known AES block cipher standard is widely used to protect communication. However, in many applications, where computing resources are limited, cryptographic algorithms of choice are stream ciphers. They

offer high speed and can be adapted to specific implementation requirements. The cryptographic community has assisted the business and industry alike by providing a wide range of stream ciphers. One of such ciphers is RC4 designed in 1987 by Ron Rivest. The cipher uses a large internal state that is stored in an array of words. Because of a simplicity of design and a high speed offered by software implementation, the cipher has gained popularity in many internet applications such as TLS/SSL and WEP.

In fact, RC4 is a family of stream ciphers indexed by an integer n that indicates the size of the word in bits. The internal state is an array S of 2^n words. RC4 consists of two algorithms. The first is a key scheduling algorithm (KSA) and it initialises the internal state. The second is a pseudo-random generation algorithm (PRGA). It generates the output keystream. The KSA algorithm takes an array S and a secret key K and produces the initial state or a secret permutation of $\{0, 1, 2, \ldots, 2^n - 1\}$. The PRGA algorithm accepts the initial state S and produces a sequence of words - one word per clock. A popular instantiation of RC4 is for $n = 8$. In this instantiation, words are 8-bit long and the array S contains $2^8 = 256$ entries. The security of RC4 has been extensively studied. The key schedule of RC4 is examined in [3, 13, 12, 16, 18, 21]. Distinguishing attacks are presented in [4, 5, 12, 14, 19]. The internal state recovery attacks are investigated in [11, 15].

When the parameter n is bigger, for instance $n = 32$, the implementation requires more memory and, in general, the cipher becomes slower. On the positive side, one would expect that the cipher is going to be stronger. This line of investigations resulted in several generalizations of RC4-like stream ciphers, see for example RC4A [18], VMPC [23], NGG [17] and $RC4(n, m)$ [6]. Our attention is concentrated on the Gong et al. design given in [6]. In this cipher the state array S no longer consists of 2^n entries. Consequently the state is no longer a permutation. This cipher is called $RC4(n, m)$, where the state array consists 2^n entries (words) and each word is m-bit long ($n < m$). For a $32-$bit architecture, the recommended parameter values are $n = 8$ and $m = 32$.

$RC4(n, m)$ is a fast synchronous stream cipher proposed by Guang Gong, Kishan Chand Gupta, Martin Hell and Yassir Nawaz in [6]. $RC4(n, m)$ produces m bits per clock. The main idea to design $RC4(n, m)$ is to exploit the 32-bit and 64-bit processor architectures without increasing the size of the table significantly. The internal state size of $RC4(n, m)$ is $(2^n m) + 2n + m$ bit long, since it consists of an array of 2^n entries and each entry takes m bits , one m-bit variable k and two n-bit indexes i and

j. Note that the key length is proposed up to 8192 bits but the security is provided for keys of size up to 256 bits.

1.1 Previous Works

$RC4(n, m)$ has been proposed based on a 32-bit RC4-like stream cipher [17] by Y. Nawaz, K.C. Gupta, and G. Gong called NGG Stream Cipher to improve the security of the cipher against the proposed attacks. H. Wu proposed a distinguishing attack on NGG [22] which could distinguish the keystream outputs from random sequences with about 3200 output bits. Also, In [8], A new distinguisher and a key recovery attack on the cipher has been proposed. The attack can distinguish the cipher from a random stream using only the first keystream word. Attacker also can recover the secret key by exploiting leaked information from the first few kilobytes of the keystream output. But the story about the new version called $RC4(n, m)$ or GGHN is different [1]. Note that all the above proposed attacks on NGG are not applicable on $RC4(n, m)$.

To our best knowledge, there are few attacks on $RC4(n, m)$. Paul and Preneel have proposed a distinguishing attack that needs $2^{32.89}$ output keystream words to distinguish the cipher from a random source [19]. The second attack [20] proposed by Tsunoo, Saito, Kubo, and Suzaki is a distinguishing attack, which uses the bias along with the first two words of a keystream associated with approximately 2^{30} secret keys. In the attack, the authors explore the correlation between indices and entries of the array. The third attack proposed by Kircanski and Youssef [9] is a fault attack, which extracts the internal state of the cipher by applying induced faults. The attack also needs 2 keystream words for each of 257×255 induced faults and approximately 257 non-faulted keystream words.

1.2 Our Contributions

We study security of $RC4(n, m)$. We will show several weaknesses in the initialization part and the update function of the algorithm. Two distinguishing attacks are described. The first attack takes advantage of the bias of least significant bits of the internal state. The idea of this attack is similar to [19, 20] but we apply it to the key scheduling algorithm. The second attack is based on truncated differentials and requires 256 output words only. Finally, we will present a key recovery attack, which is able

[1] NGG (or NGG(n,m)) is a previous version of $RC4(n, m)$ (or GGHN, GGHN(n,m)). In this paper, we analyse the security of $RC4(n, m)$ (GGHN).

to find 256-bit secret keys with time complexity about 2^{13} algorithm operations for $RC4(8, 32)$. The current state of analysis of $RC4(n, m)$ is summarized in Table 1.

Table 1. Comparison between the previous attacks and our proposals

	Attack type	The result	Date Complexity	Time Complexity	Comments
1	Correlation attack [19]	Distinguishing	$2^{32.82}$ output words of a single stream	$O(2^{32.82})$	Attack is applied on RC4(8,32)
2	Correlation attack [20]	Distinguishing	2^{30} first two words of keystreams	$O(2^{30})$	Attack is applied on RC4(8,32)
3	Fault attack [9]	Internal state Recovery	2 keystream words for each of 257×255 induced faults and approximately 257 non-faulted keystream words	$\approx O(2^{16})+$ negligible additional complexity to perform attack	Attack is applied on RC4(8,32)
4	our proposed Correlation attack	Distinguishing	$O(2^{4 \cdot n})$	$O(2^{4 \cdot n})$	Attack is applied on $RC4(n, m)$
5	our proposed Differential attack	Distinguishing	2^n output words corresponding to two initial vectors	$O(2^{n+1})$	Attack is applied on $RC4(n, m)$
6	our proposed Differential attack	Secret Key Recovery	$2^n \times 2^n$ output words corresponding to two initial vectors to recover each key byte	$O((L/n) \cdot 2^n)$ where L is secret key length in bit	Attack is applied on $RC4(n, m)$

In application protocols like WEP(Wired Equivalent Privacy), there is a session-dependent initial value that needs to be introduced as input to the stream cipher to produce different pseudo-random streams for different sessions. In these protocols, the attacker can often manipulate the initial value, as for example in the attack [3] on RC4 used in the WEP protocol, and exploit a chosen IV attack model to investigate the security of the scheme. Consequently, for such applications, the stream cipher needs to be designed to accept initial value inputs, and its security needs to be assessed with respect to initial value inputs chosen by the attacker. $RC4(n, m)$ also uses three input parameters (Figure 1): Secret Key, initial vector, and initial value (to initialise internal state before applying

Key scheduling Algorithm). From a practical point of view, the system designer may exploit all the features of the crypto-algorithm to enhance better efficiency. In this case, using initial value and initial vector as variable input parameters to differentiate applications and also to increase the security level may seem to be reasonable in the first glance. We study the extreme misuse of $RC4(n, m)$ when the initial value is assumed to be under the attacker control. The protocol initial value could be incorporated in two ways: either as the "initial value" input to the KSA* module, or as part of the secret key input (using a hash function) as the authors of RC4(m,n) proposed. From the implementation point of view, the first option might be tempting since it may be simpler to implement. For the sake of clarity, we assume that the attacker is able to change the initial value. In this case, we will prove the cipher is surprisingly insecure against the distinguishing and key recovery attacks. We also note that the attacks (5) and (6) in Table 1 are not applicable when the attacker is not allowed to manipulate the initial value.

This paper is organized as follows. Section 2 provides a description of initialisation and key generation part of the scheme. Section 3 is the main part of the work, which contains our distinguishing and key recovery attacks.

2 Description of $RC4(n, m)$ Stream Cipher

The $RC4(n, m)$ stream cipher uses the building blocks defined for the RC4 stream cipher. These blocks, however, are modified by the authors. We first describe the original key scheduling algorithm of RC4 and then we give the modified algorithm of $RC4(n, m)$. A general illustration of $RC4(n, m)$ is presented in Figure 1.

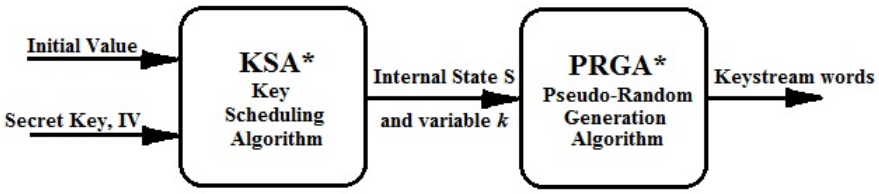

Fig. 1. $RC4(n, m)$ Stream Cipher Scheme

– KSA (key scheduling algorithm of RC4) – this algorithm takes as an input a secret key of a size between 40 and 256 bits and outputs the internal state $\langle S \rangle$, where $S = (S[0], \ldots, S[255])$ is a 256-byte sequence. The algorithm is described in Figure 2.

```
1  Input:   Secret Key Key
2  Output: Internal State ⟨S, i, j⟩
3      for i = 0   to   255
4          S[i] = i;
5      end for
6      j = 0;
7      for i = 0   to   255
8          j = (j + S[i] + Key[i  mod l])  mod 256;
9          swap(S[i], S[j]);
10     end for
```

Fig. 2. KSA Function of RC4

The algorithms of $RC4(n, m)$ are called KSA* and PRGA* to distinguish them from KSA and PRGA, respectively. To recall, the parameters of $RC4(n, m)$ are defined as follows. The size of the state array is $N = 2^n$ and each entry of the array holds m bits. Entries are going to be called words. We define a constant $M = 2^m$. For example, $RC4(8, 32)$ means that the size of the array S is 256 and each entry of S holds 32-bit words.

– KSA* (key scheduling algorithm of $RC4(n, m)$) – the algorithm takes a secret key[2] of a size between 40 and 256 bits and the state array as the input and returns an updated internal state stored in the array S and variable k. The full details are given in Figure 3.

[2] The designers suggest using a Hash Function to generate Key array from Secret Key and initial vector to prevent possible attacks on KSA*

```
1  Input:  initial values a_i, Secret Key and initial value
   Key[j] 0 ≤ i < N, 0 ≤ j < l
2  Output: Internal State ⟨S⟩, variable k
3  for i = 0  to  N − 1
4      S[i] = a_i;
5  end for
6  j = 0;
7  k = 0;
8  Repeat r times;
9      for i = 0  to  N − 1
10         j = (j + S[i] + Key[i mod l]) mod N;
11         swap(S[i], S[j]);
12         S[i] = S[i] + S[j] mod M;
13         k = k + S[i] mod M;
14     end for
```

Fig. 3. KSA*: The Key Scheduling Algorithm of $RC4(n, m)$

– PRGA* (Pseudo-Random Generation Algorithm) – it takes the pair: the internal state $\langle S \rangle$ and variable k as the input and generates output keystream words. The pseudo-code of the algorithm is given in Figure 4.

```
1  Input:  Internal State ⟨S⟩, variable k
2  Output: Output (Keystream words)
3  i = 0;
4  j = 0;
5  while ()
6      i = i + 1 mod N;
7      j = (j + S[i]) mod N;
8      k = (k + S[j]) mod M;
9      output = (S[S[i] + S[j] mod N] + k) mod M;
10     S[S[i] + S[j] mod N] = S[i] + k mod M;
```

Fig. 4. PRGA*: Pseudo-Random Generation Algorithm of $RC4(n, m)$

2.1 Notations

$[X]_0$ is the least significant bit of the word X.
$[X]_{i,\ldots,j}$ are $(j-i+1)$ consecutive bits of word X started from the position of $i-th$ to $j-th$.

3 Cryptanalysis of $RC4(n, m)$ Stream Cipher

In this section, we prove that $RC4(n, m)$ is not resistant against distinguishing and key recovery attacks. First, we are going to identify weaknesses in the KSA* algorithm. Next, we exploit these weaknesses and show how to distinguish the output stream of $RC4(n, m)$ from a random cipher. The data complexity of the attack is 256 output words. Then, we propose a key recovery attack based on truncated differentials. The time complexity attack to recover a 256-bit secret key of $RC4(8, 32)$ is about 2^{13} algorithm operations.

In some applications, one may design a cryptosystem in which the initial values are varied in different sessions. At first glance, it looks like this may increase the security level of the cipher as the attacker cannot use the results from the analysis of previous sessions generated for different initial values. In the second distinguishing attack and key recovery attack, we assume that the initial value can be modified and selected as in the chosen IV attack. We show that the cipher is susceptible to this kind of attacks.

3.1 Weaknesses of $RC4(n, m)$

Before describing our attacks, we discuss properties of the $RC4(n, m)$ that underpin our attacks.

Non-Randomness Property of Internal States: The array S has 256 elements whose lengths are 32 bits and the pointer j takes one byte. This means that if we choose two indices $i, j \in \{0, 1\}^8$ at random, then the probability $Pr(S[i] = S[j]) = Pr(i = j) = 2^{-8}$ assuming that $S[i] \neq S[j]$ for $i \neq j$, while for two random 32-bit words, this probability is 2^{-32}. Now, we can prove that for every element in the array S after applying initialization algorithm, $Pr([S[i]]_0 = 0) = 0.5 + 2^{-8}$, where $0 < i < 256$.

Weak Keys: There are several classes for secret keys that generate internal states with short cycles. The final internal states (after a run of KSA* but before an execution of PRGA*) can be computed using a certain relation among the states. For example, in the following, we show that the state $S[0]$ in the array moves and all other states swap with this state only.

Example 1. Let internal states of algorithm be equal to the values suggested in the appendix A in [6], and the secret key is 0X0101 ... 01. According to the KSA algorithm, we can write the following relations:

$i=0, j=0, k=0$	$i=1, j=1, k=S[0]$	$i=2, j=2, k=S[0]+S[1]$
$j=0+S[0]+K[0]=1$	$j=1+S[1]+K[1]=2$	$j=2+S[2]+K[2]=3$
$Swap(S[0], S[1])$	$Swap(S[1], S[2])$	$Swap(S[2], S[3])$
$S[0]=S[0]+S[1]$	$S[1]=S[1]+S[2]$	$S[2]=S[2]+S[3]$
$k=0+S[0]$	$k=S[0]+S[1]$	$k=S[0]+S[1]+S[2]$

The above relations show that in $RC4(n, m)$, there are Finney states [7] that swap $S[i]$ with $S[i+1]$ and both indices i and j are incremented by 1. Other weak keys can be found using probabilistic relations. Note that other weaknesses have been deeply investigated in [1] recently.

Weak States: We are going to find several initial states, for which the outputs will be distinguishable from a truly random source. The main weak point, which we exploit here, is a low diffusion of bits in KSA*.

1. For an arbitrary secret key, if the least significant bits of initial states are equal to zero, then the least significant bits of keystreams will be zero with the probability one. We can extend this observation for 2, 3, ..., 32-bit of LSB of initial states.
2. Assume that

$$S[i] \pmod{2^n} = 1 - K[i \pmod{l}]$$

and

$$S[0] \pmod{2^n} = -K[0]$$

then j will be equal to i in the first round. It means that after one round, all the internal states will be even (the least significant bits of internal states will always be zero).
3. Suppose that $K[0]$ is odd and $K[1] = K[0] - 2$. Also assume that $(S[0])\ mod\ 2^n = (1 - K[0])\ mod\ 2^n$, $S[1]$ is even, $(S[2])\ mod\ 2^n = (2 - K[2])\ mod\ 2^n$ and $(S[i])\ mod\ 2^n = 1 - K[i\ mod\ l]\ \ 3 < i < 255$, then the internal states after one round will be even. In other words, the least significant bit of keystreams will be always zero.

Low Diffusion Property: Clearly, the update function of $RC4(n, m)$ is like a T-function [10]. It means that the i-th bit of output depends on the i-th bit of input and all less significant bits (i.e. bits $i-1, \ldots, 0$) of the input. This is a serious weakness for $RC4(n, m)$, because if the cryptanalyst changes the most significant bits of initial values (a_i) in KSA*, then only the most significant bits of keystreams will be changed (other bits will be

unchanged). In the RC4 initialization algorithm, all bytes of initial state and secret key are involved to provide internal state as input for PRGA to generate output keystream. This means that by complementing one bit of initial state, all bits of output key streams will be changed with probability close to 1/2. In fact, this property called *the avalanche criterion* is one of the most essential properties of a secure cipher. However, this property has been confirmed only for the least significant bytes of initial value array. In other words, if we change the $I-$th bit ($32 > I \geq 8$), then more significant bits i may change ($i > I$), while less significant bits (with index $i < I$) are not going to change. This property of the KSA* algorithm of $RC4(n, m)$ is illustrated in Figure 5. Now, we are ready to describe the proposed attacks on $RC4(n, m)$.

3.2 Distinguishing Attack on $RC4(n, m)$

The first attack: This attack is based on the non-randomness property of internal states, which is described in the previous section. Consider the line 12 of Figure 3 in the $r - th$ round of the algorithm.

Proposition 1 *Assume that (1) The index j at line 10 of KSA^* is uniformly distributed in $\{0, ..., N-1\}$ and independent of i, and (2) if $i \neq j$ then S[i]+S[j] mod M is independent and uniform in $\{0, ..., N-1\}$. Then, for all elements of Array S, after performing KSA^* we have:*

$$Pr([S[i]]_0 = 0) = \frac{1}{2}(1 + \frac{1}{2^n}) \qquad 0 \leq i < 2^n$$

Proof. First, we know if $i = j$ then in line 12, we have $S[i] = 2 \cdot S[i] \bmod M$ is even since M is even. And then, $Pr([S[i]]_0 = 0) = Pr([S[i]]_0 = 0 | i \neq j) \cdot Pr(i \neq j) + Pr([S[i]]_0 = 0 | i = j) \cdot Pr(i = j) = \frac{1}{2} \cdot \frac{2^n - 1}{2^n} + 1 \cdot \frac{1}{2^n} = \frac{1}{2} \cdot (1 + \frac{1}{2^n})$. □

Thus, we can find out that all the least significant bits of the array S contents are biased. If the keystream just depended on the array S, then we could exploit the bias to mount a distinguishing attack. But, the keystream output is summation of a word from the array S and the variable k. In addition, the variable k is the sum of randomly chosen elements of the array S. It can be shown that the least significant bit of the variable k is also biased but the bias is very close to zero. So, we need to use combination of outputs to eliminate the effect of variable k and find a biased linear relation. For instance, linear combination of two consecutive outputs can reveal the expected bias. To do this, let the event

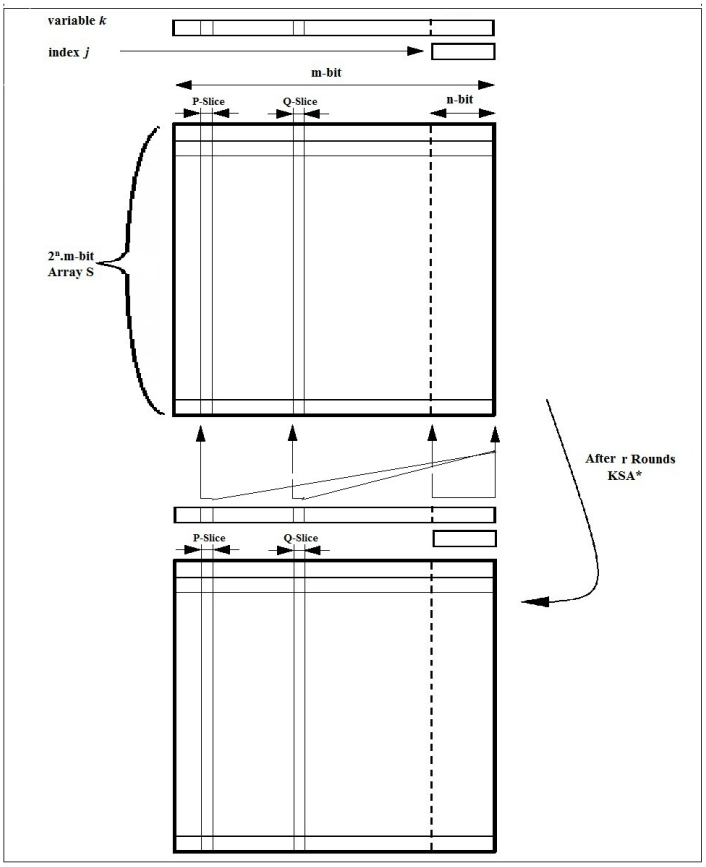

Fig. 5. $RC4(n, m)$: Another perspective of KSA*. According to Low Diffusion Property, P-th slice of internal state in r-th round of KSA* (r is an arbitrary round) depends on P-th slice of internal state and previous slices in initial state. And also, any random difference in P-th slice in initial state does not change Q-th slice in r-th round of KSA*.

E denote the condition, in which the relation $k_{t+1} = k_t + S[y]$ is satisfied as follows:

$$Output[t] = S[x] + k_t \mod M,$$
$$Output[t+1] = S[y] + k_{t+1} \mod M,$$

where x and y are randomly chosen indices and $t = 0$. Now, by adding $Output[0]$ and $Output[1]$, we get

$$[Output[1] \oplus Output[0]]_0 = [S[x]]_0.$$

We can formulate the following proposition.

Proposition 2 *In RC(n,m), the probability of* $([Output[1] \oplus Output[0]]_0 = 0)$ *is* $\frac{1}{2} \cdot (1 + \frac{1}{2^{(2 \cdot n)}})$.

Proof. $Pr([Output[1] \oplus Output[0]]_0 = 0) = Pr([Output[1] \oplus Output[0]]_0 = 0|E).Pr(E) + Pr([Output[1] \oplus Output[0]]_0 = 0|E^c) \cdot Pr(E^c) = (\frac{1}{2} \cdot (1 + \frac{1}{2^n})) \cdot \frac{1}{2^n} + \frac{1}{2} \cdot \frac{2^n - 1}{2^n} = \frac{1}{2} \cdot (1 + \frac{1}{2^{(2 \cdot n)}})$. □

For an ideal PRBG, the above probability would have been exactly $\frac{1}{2}$. We can extend our assumption for more than two consecutive output words in which $S[x]$ is not updated in time $t = 0$ (or $t = 0$ and $t = 1$). In other words,

$$\begin{aligned} Output[0] &= S[z] + k_t \mod M; \\ Output[1] &= S[x] + k_{t+1} \mod M; \\ Output[2] &= S[y] + k_{t+2} \mod M; \end{aligned}$$

The probability of $[Output[2] \oplus Output[1]]_0 = 0$ can be simply computed by applying Bayes' theorem as follows:

$$Pr([Output[2] \oplus Output[1]]_0 = 0) =$$
$$Pr([Output[2] \oplus Output[1]]_0 = 0|x \neq z) \cdot Pr(x \neq z) +$$
$$Pr([Output[2] \oplus Output[1]]_0 = 0|x = z) \cdot Pr(x = z) = \frac{1}{2} \cdot (1 + \frac{1}{2^{(2 \cdot n)}} - \frac{1}{2^{(3 \cdot n)}}).$$

The above attack is a generalisation of the attack proposed in [19, 20] with emphasis on the initialization part of algorithm. In the next section, we will present distinguishing and key recovery attacks, which exploit a low diffusion of bits property of KSA*.

Algorithm 1 Distinguishing Attack Scenario on $RC4(n, m)$

Input: The first two (four) output words corresponding $2^{4.n}$ randomly chosen secret key.

Output: To distinguish between $RC4(n, m)$' outputs and a truly random source.

1. Generate $Output^k[0]$ and $Output^k[1]$, $0 \leq k < 2^{4.n}$, $Output^k[i]$ is *i-th* output associated with *k-th* secret key.
2. $S = \frac{\sum_k (Output^k[0] \oplus Output^k[1])}{2^{4.n}}$;
3. If $S \geq \frac{1}{2}$ then the algorithm which is analysed in this test, is $RC4(n, m)$.

Note that the required amount of data to distinguish a biased sequence Z in which $Pr(Z_i = 0) = 1/2 + 1/2^n$ from a truly random sequence is determined as Chernoff bound by an exponential function in n is greater than $2^{2n} \ln \frac{1}{\sqrt{1-P_S}}$ where P_S is the expected success probability. In Table 2, the success probabilities in theory and practice are shown.

Table 2. Experimental Results and Comparison between theory and simulation

n	The required amount of data	Success Probability P_e in Theory	The founded Success Probability P_e in simulation
4	$2^{16.58}$	0.95	0.97
4	$2^{14.87}$	0.60	0.59
5	$2^{20.58}$	0.95	0.94
5	$2^{18.87}$	0.60	0.58

The second attack: The second distinguishing attack is based on differential cryptanalysis [2]. Differential attacks on stream ciphers use a chosen initial value or other public variables. This kind of attack can be launched, if the adversary has access to the cipher and can manipulate the external (public) elements. But of course, they cannot see the secret elements that are assumed to be hidden by, for example, a tamper proof hardware. We are going to use a generalisation of differential cryptanalysis called the truncated differential cryptanalysis. Whereas the standard differential cryptanalysis considers the full difference between two inputs, the truncated variant takes differences that are only partially determined. So, attacker can predict only some of the bits.

As we noted before, a modification of more significant bits will not change less significant bits. This property is rephrased below.

Remark 1. Let $X+Y = Z$, $X, Y, Z \in GF(2^m)$, $\Delta = R\underbrace{0 \ldots 0}_{m-k}$, $R \in GF(2^k)$ and R is an arbitrary differential input then for

$$\begin{cases} (X \oplus \Delta_1) \boxplus Y = Z_{\Delta_1} \\ (X \oplus \Delta_2) \boxplus Y = Z_{\Delta_2} \end{cases} \tag{1}$$

where $\Delta_1 = R_1 0 \cdots 0$, $\Delta_2 = R_2 0 \cdots 0$, and $R_1 \neq R_2$, We have:

$$[Z_{\Delta_1}]_{0\cdots(m-k)} = [Z_{\Delta_2}]_{0\cdots(m-k)}$$

In modular addition, the most significant bits do not affect the least significant bits. So, applying difference vectors to more significant bits changes just corresponding output bits and the difference for less significant bits will be zero.

Remark 2. If $Y = (X \oplus \Delta)$ and $\Delta = 1\underbrace{0\cdots000}_{m-1}$ then $X \boxplus X = Y \boxplus Y = (2 \cdot X)$ *mod* 2^m. (*i.e.* the differential value in output will be disappeared.)

Theorem 1. *Given the* $RC4(n, m)$ *cipher. Let there be two initial values* IV_1 *and* IV_2*, where* $IV_2[i] = IV_1[i] \oplus \Delta_{IV}[i]$ *and* $\Delta_{IV}[i] = 0XRR\ 00\cdots\ 00$ *is a truncated differential vector. The length of* $\Delta_{IV}[i]$ *is* m *bits and* $0 \leq i < 2^n$*. For* $\Delta_{IV}[i]$*, the byte* RR *is different from zero. Then, for all output keystream words* $Output_1$ *and* $Output_2$ *related to* IV_1 *and* IV_2*, we have:*

$$[Output_1[j]]_{0\cdots(m-8)} = [Output_2[j]]_{0\cdots(m-8)}$$

with probability one, where $[Output_k[j]]_0$ *is the least significant bit of* j*-th output keystream word,* $j \geq 0$*, and k=1,2.*

Proof. For two initial vectors IV_1 and IV_2, the least significant bytes are the same. According to the lines 9 and 10 in Figure 3 and the lines 6 and 7 in Figure 4, the indices i and j are all updated modulo 2^8. So, the index j which depends on the secret key bytes and the least significant bytes of internal state will be the same. Consequently, updating internal state and variable k is similar. However, updating array S and variable k is based on modular addition then changing MSB does not change the least significant bits (see Remark 1), then the bits with less significant bits will remain the same. □

A distinguishing attack on $RC4(8, 32)$ based on Theorem 1 is shown as Algorithm 2.

3.3 Key Recovery Attack on $RC4(n, m)$

Now, we prove that the attacker is able to recover the secret key of $RC4(n, m)$ by guessing each byte of the secret key separately. There are three phases of our key recovery attack.

1. Guess each byte of the secret key,

Algorithm 2 Distinguishing Attack Scenario on $RC4(8,32)$

Input: Two initial vectors IV_1 and IV_2 which satisfy Equation 2.
Output: To distinguish between RC4(8,32) outputs and a truly random source.

1. Select k elements to apply differential input vectors from set $\mathcal{K}$ ($|\mathcal{K}| = k$) and $1 \leq k < 2^8$.
2. Select differential vectors $\Delta_{IV}[i] = 0xRR000000$, $i \in \mathcal{K}$ and RR are non-zero and arbitrary bytes.
3. Generate 2^n output keystream words $Output_1[j]$ and $Output_2[j]$ corresponding to IV_1 and IV_2 where

$$\begin{cases} IV_1[i] = IV_2[i] \oplus \Delta_{IV}[i] & i \in \mathcal{K} \\ IV_1[i] = IV_2[i] & otherwise \end{cases} \tag{2}$$

4. Compute output differential vector as $\Delta_{Output}[j] = Output_1[j] \oplus Output_2[j]$
5. If the general form of $\Delta_{Output}[j] = 0xSS$ 00 00 00 where SS is output truncated differential bytes, then the algorithm which is analysed in this test, is RC4(8,32).

2. Generate appropriate input differential initial values by Equation 2,
3. Verify the validity of the guess

Without loss of generality, we first focus on recovering the first byte of secret key. According to Figure 3, this byte first affects $S[0]$ array. Let define $\Delta = 0X80$ 00 00 00. We consider $\mathcal{K} = \{0\}$, and $\Delta_{IV}[0] = \Delta$, and $SK[0]$ as the least significant byte of secret key. Now, the key recovery attack on $RC4(8,32)$ can be designed based on Theorem 1 and it is shown as Algorithm 3.

Algorithm 3 Key Recovery Attack Scenario on $RC4(8,32)$ (First byte of Secret Key)

Input: Two initial vectors IV_1 and IV_2 which satisfy Equation 3.
Output: Key Recovery of $SK[0]$ (the least significant byte of secret key).

1. Guess SK[0]=$\widehat{SK_0}$,
2. Compute $[IV_1[0]]_{0\cdots7} = [IV_2[0]]_{0\cdots7} = (-\widehat{SK_0})\ mod\ 2^8$,
2. Select a differential vector $\Delta_{IV}[0] = \Delta$.
3. Generate 2^8 output keystream words $Output_1[j]$ and $Output_2[j]$ corresponding to IV_1 and IV_2 where

$$\begin{cases} IV_1[0] = IV_2[0] \oplus \Delta_{IV}[0] \\ IV_1[i] = IV_2[i] & 1 \leq i < 2^8 \end{cases} \tag{3}$$

4. Compute output differential vector as $\Delta_{Output}[j] = Output_1[j] \oplus Output_2[j]$
5. If $\Delta_{Output}[j] = 0X00$ 00 00 00, Then $\widehat{SK_0}$ is the least significant byte of secret key with probability close to one. Otherwise, Go to step 1.

Remark 3. When the attacker finds a $\widehat{SK_0}$ which is confirmed in step 5, then he has to repeat the scenario with same $\widehat{SK_0}$ and different $IV_1[i]$ and $IV_2[i]$ to be sure the guessed $\widehat{SK_0}$ is the least significant byte of secret key with probability one.

Remark 4. The attack efficiency does not depend on r parameter. It means that if the designers increase r, the attack will be still applicable.

To recover all bytes of secret key, we just need to perform the above sequences. For example, to recover $k-th$ ($0 \leq k < 32$ for 256-bit secret key) byte of secret key, attacker has to find all bytes of $SK[i]$ where $0 \leq i < k$ according to Algorithm 4.

Algorithm 4 Key Recovery Attack Scenario on $RC4(8, 32)$

Input: Two initial vectors IV_1 and IV_2 which satisfy Equation 4.
Output: Key Recovery of $SK[k]$ (the k-th byte of secret key).

1. Guess $SK[k] = \widehat{SK_k}$,
2. Compute $[IV_1[k]]_{0\cdots7} = [IV_2[k]]_{0\cdots7} = (-\widehat{SK_k})\ mod\ 2^8$,
2. Select differential vector $\Delta_{IV}[k] = \Delta$.
3. Generate 2^8 output keystream words $Output_1[j]$ and $Output_2[j]$ corresponding to IV_1 and IV_2 where

$$\begin{cases} [IV_1[i]]_{0\cdots7} = [IV_2[i]]_{0\cdots7} = SK[i] & 0 \leq i < k \\ IV_1[i] = IV_2[i] \oplus \Delta_{IV}[i] & i = k \\ IV_1[i] = IV_2[i] = arbitrary\ \ 32-bit\ \ values & k < i < 2^8 \end{cases} \tag{4}$$

4. Compute output differential vector as $\Delta_{Output}[j] = Output_1[j] \oplus Output_2[j]$
5. If $\Delta_{Output}[j] = 0X00\ 00\ 00\ 00$, Then $\widehat{SK_k}$ is the the $k-th$ byte of secret key with probability close to one. Otherwise, Go to step 1.

To verify theoretical results, we implemented the key recovery attack on RC4(8,32). The attack can recover a 256-bit secret key less than one second in on a standard PC. Also, the attack complexity is not different for other members of $RC4(n, m)$ family.

3.4 Discussion

Thwarting the proposed attacks: The attacks proposed in this paper were based on two critical weak points. The first weakness is the non-randomness property of the internal state after applying KSA*. This weakness is actually a natural attribute of the cipher. The main difference between RC4

and $RC4(n, m)$ is the length of elements of internal i and j are kept fixed. The second weak point is a low diffusion property. This weak point can be removed by using some simple linear operations like bit-rotation to relocate the positions of the least and most significant bits of internal states during running of KSA* and PRGA*.

4 Conclusion

We investigated the security of $RC4(n, m)$ and particularly analysed the initialization part of the algorithm. The attacks in this paper were based on non-randomness of internal state which lead to a statistical distinguishing attack, and based on low diffusion property in KSA* and PRGA* which shows the attacker can apply a truncated differential technique to recover all bytes of Key array. These attacks are only applicable when the protocol allows manipulation of initial value. In this scenario, we have shown that the output keystream can be distinguished from a truly random sequence with having just 256 output words. By using this weak point, a practical key recovery attack which recovers $256-$bit secret key with time complexity about 2^{13} algorithm operations has been proposed.

References

1. S. Banik, S. Maitra, and S. Sarkar, *On the evolution of GGHN cipher*, in *Proceedings of the 12th international conference on Cryptology in India*, INDOCRYPT'11, Chennai, India, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 181–195.
2. E. Biham and A. Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*, in *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '90, URL http://dl.acm.org/citation.cfm?id=646755.705229, Springer-Verlag, London, UK, UK, 1991, pp. 2–21.
3. S. Fluhrer, I. Mantin, and A. Shamir, *Weaknesses in the Key Scheduling Algorithm of RC4*, in *RC4, Proceedings of the 4th Annual Workshop on Selected Areas of Cryptography*, 2001, pp. 1–24.
4. S. Fluhrer and D. McGrew, *Statistical Analysis of the Alleged RC4 Keystream Generator*, in *Fast Software Encryption, Lecture Notes in Computer Science*, vol. 1978, Springer Berlin / Heidelberg, 2001, pp. 66–71.
5. J.D. Golic, *Linear statistical weakness of alleged RC4 keystream generator*, in *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, Konstanz, Germany, Springer-Verlag, Berlin, Heidelberg, 1997, pp. 226–238.
6. G. Gong, K.C. Gupta, M. Hell, and Y. Nawaz, *Towards a General RC4-Like Keystream Generator*, in *FIRST SKLOIS CONFERENCE, CISC 2005*, Springer-Verlag, 2005, pp. 162–174.
7. F. H., *An RC4 cycle that cant happen*, in , Newsgroup post in sci. crypt, September 1994.

8. A. Kircanski, R. Al-Zaidy, and A. Youssef, *A new distinguishing and key recovery attack on ngg stream cipher*, Cryptography and Communications 1 (2009), pp. 269–282, URL http://dx.doi.org/10.1007/s12095-009-0012-4, 10.1007/s12095-009-0012-4.
9. A. Kircanski and A. Youssef, *On the structural weakness of the GGHN stream cipher*, in , vol. 2, Springer New York, 2010, pp. 1–17.
10. A. Klimov and A. Shamir, *Cryptographic applications of t-functions*, in *Selected Areas in Cryptography*, M. Matsui and R. Zuccherato, eds., *Lecture Notes in Computer Science*, vol. 3006, Springer Berlin / Heidelberg, 2004, pp. 248–261.
11. L. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege, *Analysis Methods for (Alleged) RC4*, in *Advances in Cryptology ASIACRYPT98*, *Lecture Notes in Computer Science*, vol. 1514, Springer Berlin / Heidelberg, 1998, pp. 327–341.
12. I. Mantin, *Predicting and Distinguishing Attacks on RC4 Keystream Generator.*, in *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, URL http://www.iacr.org/cryptodb/archive/2005/EUROCRYPT/2597/2597.pdf, 2005, pp. 491–506.
13. I. Mantin, , I. Mantin, and A. Shamir, *A Practical Attack on Broadcast RC4*, in *Proc. of FSE01*, Springer-Verlag, 2001, pp. 152–164.
14. A. Maximov, *Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers*, in *FSE*, 2005, pp. 342–358.
15. A. Maximov and D. Khovratovich, *New State Recovery Attack on RC4*, in *CRYPTO*, 2008, pp. 297–316.
16. I. Mironov, *Not So Random Shuffles of RC4*, in *Proc. of CRYPTO02*, Springer-Verlag, 2002, pp. 304–319.
17. Y. Nawaz, K.C. Gupta, and G. Gong, *A 32-bit rc4-like keystream generator*, IACR Cryptology ePrint Archive 2005 (2005), p. 175.
18. S. Paul and B. Preneel, *A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher*, in *FSE*, 2004, pp. 245–259.
19. ———, *On the (In)security of Stream Ciphers Based on Arrays and Modular Addition*, in *ASIACRYPT 2006*, Springer, 2006, pp. 69–83.
20. Y. Tsunoo, T. Saito, H. Kubo, and T. Suzaki, *A Distinguishing Attack on a Fast Software-Implemented RC4-Like Stream Cipher*, in *IEEE Transactions on Information Theory,*, vol. 53, September, 2007, pp. 3250 –3255.
21. S. Vaudenay and M. Vuagnoux, *PassiveOnly Key Recovery Attacks on RC4*, in *Selected Areas in Cryptography*, *Lecture Notes in Computer Science*, vol. 4876, Springer Berlin / Heidelberg, 2007, pp. 344–359.
22. H. Wu, *Cryptanalysis of a 32-bit rc4-like stream cipher*, IACR Cryptology ePrint Archive 2005 (2005), p. 219.
23. B. Zoltak, *VMPC One-Way Function and Stream Cipher*, in *of LNCS*, Springer-Verlag, 2004, pp. 210–225.