A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher*

Souradyuti Paul and Bart Preneel

Katholieke Universiteit Leuven, Dept. ESAT/COSIC,
Kasteelpark Arenberg 10,
B–3001 Leuven-Heverlee, Belgium
{Souradyuti.Paul,Bart.Preneel}@esat.kuleuven.ac.be

Abstract. The paper presents a new statistical bias in the distribution of the first two output bytes of the RC4 keystream generator. The number of outputs required to reliably distinguish RC4 outputs from random strings using this bias is only 2^{25} bytes. Most importantly, the bias does not disappear even if the initial 256 bytes are dropped. This paper also proposes a new pseudorandom bit generator, named RC4A, which is based on RC4's exchange shuffle model. It is shown that the new cipher offers increased resistance against most attacks that apply to RC4. RC4A uses fewer operations per output byte and offers the prospect of implementations that can exploit its inherent parallelism to improve its performance further.

1 Introduction

RC4 is the most widely used software based stream cipher. The cipher has been integrated into TLS/SSL and WEP implementations. The cipher was designed by Ron Rivest in 1987 and kept as a trade secret until it was leaked out in 1994. RC4 is extremely fast and its design is simple.

The RC4 stream cipher is based on a secret internal state of $N = 256$ bytes and two index-pointers of size $n = 8$ bits. In this paper we present a bias in the distribution of the first two output bytes. We observe that the first two output words are equal with probability that is significantly less than expected. Based on this bias we construct a distinguisher with non-negligible advantage that distinguishes RC4 outputs from random strings with only 2^{24} pairs of output bytes when $N = 256$. More significantly, the bias remains detectable even after discarding the initial N output bytes. This fact helps us to create another practical distinguisher with only 2^{32}

* This work was partially supported by the Concerted Research Action GOA-MEFISTO-666 of the Flemish government.

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

pairs of output bytes that works 256 rounds away from the beginning when $N = 256$.

A second contribution of the paper is a modified RC4 keystream generator, within the scope of the existing model of an exchange shuffle, in order to achieve better security. The new cipher is given the name RC4A. We compare its security to the original RC4. Most of the known attacks on RC4 are less effective on RC4A. The new cipher needs fewer instructions per byte and it is possible to exploit the inherent parallelism inherent to improve its performance.

1.1 Description of RC4

RC4 runs in two phases (description in Fig. 1). The first part is the key scheduling algorithm KSA which takes an array S or S-box to derive a permutation of $\{0, 1, 2, \ldots, N-1\}$ using a variable size key K. The second part is the output generation part PRGA which produces pseudo-random bytes using the permutation derived from KSA. Each iteration or loop or 'round' produces one output value. Plaintext bytes are *bit-wise* XORed with the output bytes to produce ciphertext. In most of the applications RC4 is used with word length $n = 8$ bits and $N = 256$. The symbol l denotes the *byte-length* of the secret key.

KSA (K, S)	PRGA(S)
for $i = 0$ to $N-1$	$i = 0$
$S[i] = i$	$j = 0$
$j = 0$	Output Generation loop
	$i = (i+1) \bmod N$
for $i = 0$ to $N-1$	$j = (j + S[i]) \bmod N$
$j = (j + S[i] + K[i \bmod l]) \bmod N$	Swap($S[i], S[j]$)
Swap($S[i], S[j]$)	Output=$S[(S[i] + S[j]) \bmod N]$

Fig. 1. The Key Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA).

1.2 Previous Attacks on RC4

RC4 came under intensive scrutiny after it was made public in 1994. Finney [1] showed a class of internal states that RC4 never enters. The class contains all the states for which $j = i+1$ and $S[j] = 1$. A fraction of N^{-2} of all possible states fall under Finney's forbidden states. It is simple

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

to show that these states are connected by a cycle of length $N(N-1)$. We know that RC4 states are also connected in a cycle (because the *next state function* of RC4 is a bijective mapping on a finite set) and the initial state, where $i = 0$ and $j = 0$, is not one of Finney's forbidden states.

Jenkins [6] detected a probabilistic correlation between the secret information (S, j) and the public information (i, output). Golić [4] showed a positive correlation between the second binary derivative of the least significant bit output sequence and 1. Using this correlation RC4 outputs can be distinguished from a perfectly random stream of bits by observing only $2^{44.7}$ output bytes. Fluhrer and McGrew [3] observed stronger correlations between consecutive bytes. Their distinguisher works using $2^{30.6}$ output bytes. Properties of the state transition graph of RC4 were analyzed by Mister and Tavares [11]. Grosul and Wallach demonstrated a related key attack that works better on very long keys [5]. Andrew Roos also discovered classes of weak keys [15].

Knudsen *et al.* have attacked versions of RC4 with $n < 8$ by their backtracking algorithm in which the adversary guesses the internal state and checks if an anomaly occurs in later stage [7]. In the case of contradiction the algorithm backtracks through the internal states and re-guesses. So far this remains the only efficient algorithm which attempts to discover the secret internal state of this cipher.

The most serious weakness in RC4 was observed by Mantin and Shamir [9] who noted that the probability of a zero output byte at the second round is twice as large as expected. In broadcast applications a practical ciphertext only attack can exploit this weakness.

Fluhrer *et al.* [2] have recently shown that if some portion of the secret key is known then RC4 can be broken completely. This is of practical importance because in the Wired Equivalence Privacy Protocol (WEP in short) a fixed secret key is concatenated with IV modifiers to encrypt different messages. In [16] it is shown that the attack is feasible.

Pudovkina [14] has attempted to detect a bias, only analytically, in the distribution of the first, second output values of RC4 and digraphs under certain uniformity assumptions.

Mironov modeled RC4 as a Markov chain and recommended to dump the initial $12 \cdot N$ bytes of the output stream (at least $3 \cdot N$) in order to obtain uniform distribution of the initial permutation of elements [10].

More recently Paul and Preneel [12] have formally proved that only a known elements of the S-box along with two index-pointers cannot predict more than a output bytes in the next N rounds. They have also designed

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

an efficient algorithm to deduce certain special RC4-states known as *Non-fortuitous Predictive States*.

1.3 Organization

The remainder of this paper is organized as follows. Section 2 introduces the new statistical bias in RC4. Section 3 reflects on the design principles of RC4. Our new variant RC4A is introduced in Sect. 4 and its security is analyzed in Sect. 5. Section 6 presents our concluding remarks.

2 The New Weakness

Our major observation is that the distribution of the first two output bytes of RC4 is not uniform. We noted that the probability that the first two output bytes are equal is $\frac{1}{N}(1 - \frac{1}{N})$. This fact is, in a sense, counter-intuitive from the results obtained by Fluhrer and McGrew [3] who showed that the first two outputs take the value (0, 0) with probability significantly larger than $1/N^2$. Pudovkina [14] analytically obtained, under certain assumptions, that the first two output bytes are equal with probability larger than $1/N$. However, experiments revealed that this result is incorrect. Our main objective is to find a reasonable explanation for this particular bias.

Throughout the paper $S_r[l]$ and O_r denote the lth element of the S-box after the swapping at round r and the output byte generated at that round respectively. Similarly, i_r and j_r should be understood. All arithmetic operations are computed modulo N.

2.1 Motivational Observation

Theorem 1. *If $S_0[1] = 2$ then the first two output bytes of RC4 are always different.*

Proof. Fig. 2 shows the execution of the first two rounds of the output generation. We note that, $O_1 = S_1[X + 2]$ and $O_2 = S_2[Z + 2]$. Clearly $X + 2$ and $Z + 2$ point to two different cells of the array. Therefore, the first two outputs are the same if ($X = 0$ and $Z = 2$) or if ($X = 2$ and $Z = 0$). But this is impossible because $X \neq Z \neq 2$ as they are permutation elements. □

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

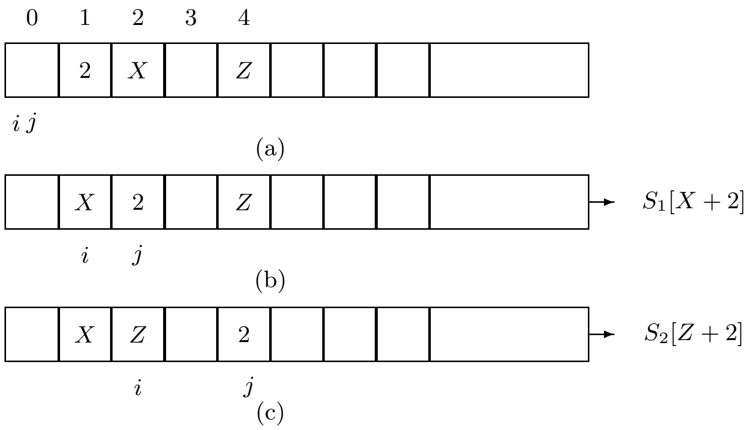

Fig. 2. (a) Just before the beginning of the output generation. (b) First output $S_1[X+2]$ is produced. (c) Second output $S_2[Z+2]$ is produced. $S_1[X+2] \neq S_2[Z+2]$.

2.2 Quantifying the Bias in the First Two Outputs

Theorem 1 immediately implies a bias in the first two outputs of the cipher that are captured in the following corollaries.

Corollary 1. *If the first two output bytes are equal then $S_0[1] \neq 2$.*

Proof. This is an obvious and important deduction from Theorem 1. This fact can be used to speed up exhaustive search by a factor of $\frac{N}{N-1}$. □

Corollary 2. *The probability that the first two output bytes are equal is $(1-1/N)/N$ (assuming that $S_0[1] = 2$ occurs with probability $1/N$ and that for the rest of the permutations for which $S_0[1] \neq 2$ the first two output bytes are equal with probability $1/N$).*[1]

Proof. If $S_0[1] = 2$ occurs with probability $1/N$ then the first two output bytes are different for a fraction of $1/N$ of the total keys (see Theorem 1). The output bytes are equal with probability $1/N$ for each of the other keys, i.e., a fraction of $(1-1/N)$ of the total keys. Combining these two,

$$\begin{aligned}P[O_1 = O_2] = P[O_1 = O_2|S_0[1] = 2] \cdot P[S_0[1] = 2] + \\ P[O_1 = O_2|S_0[1] \neq 2] \cdot [S_0[1] \neq 2] \\ = 0 \cdot \frac{1}{N} + \frac{1}{N} \cdot (1 - \frac{1}{N}) \\ = \frac{1}{N} \cdot (1 - \frac{1}{N}) \, .\end{aligned}$$

□

[1] Note that this condition is more relaxed than assuming that the initial permutation is distributed uniformly.

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

2.3 Distinguisher Based on the Weakness

A *distinguisher* is an efficient algorithm which distinguishes a stream of bits from a perfectly random stream of bits, that is, a stream of bits that has been chosen according to the uniform distribution. There are two ways an adversary can attempt to distinguish between strings, one generated by a pseudorandom generator and the other by a perfectly random source. In the first case the adversary selects *only* one key randomly and produces keystream, seeded by the chosen key, long enough to detect a bias. In this scenario the adversary is "weak" as she has a keystream produced by a single key and therefore the *distinguisher* is called a *weak distinguisher*. In the other case the adversary may use *any number of* randomly chosen keys and the respective keystreams generated by those keys. In this case the adversary is "strong" because she may collect outputs to her advantage from many keystreams to detect a bias. Therefore, the *distinguisher* so constructed is termed a *strong distinguisher*. A bias present in the output at time t in a single stream may hardly be detected by a *weak distinguisher* but a *strong distinguisher* can easily discover the anomaly with fewer bytes. This fact was wonderfully used by Mantin and Shamir [9] to identify a strong bias toward zero in the second output byte of RC4. We also construct a *strong* distinguisher for RC4 based on the non-uniformity of the first two output bytes. We use the following result (see [9] for a proof).

Theorem 2. *If an event e occurs in a distribution X with probability p and in Y with probability $p(1+q)$ then, for small p and q, $O(\frac{1}{pq^2})$ samples are required to distinguish X from Y with non-negligible probability of success.*

In our case, X, Y and e are the distribution of the first two output bytes collected from a perfectly random source, the distribution of these variables from RC4 and the occurrence of equal successive bytes respectively. Therefore, the number of required samples equals $O(N^3)$ (by Corollary 2, $p = 1/N$ and $q = -1/N$).

Experimental observations agree well with the theoretical results. For $N = 256$, with 2^{24} pairs of the first two output bytes, generated from as many randomly chosen keys, our distinguisher separates RC4 outputs from random strings with an advantage of 40% when the threshold is set at 65408. Empirical results show that the expected number of pairs, for which the bytes are equal, trails the expected number from a random source by *1.21 standard deviations* of the binomial distribution with parameters $(2^{24}, 1/256)$.

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

2.4 Bias after Dropping the First N Output Bytes

A similar but a smaller bias is also expected in the output bytes O_{t+1} and O_{t+2}, where $t = 0 \bmod N$ and $t > 0$, if we assume that $P[S_t[1] = 2 \cap j_t = 0] = 1/N^2$ and the expected probability that $O_{t+1} = O_{t+2}$ for rest of the internal states is $1/N$. Almost in a similar manner, we can compute $P[O_{t+1} = O_{t+2}]$, where $t = 0 \bmod N$ and $t > 0$.

$$\begin{aligned} P[O_{t+1} = O_{t+2}] &= P[O_{t+1} = O_{t+2} | S_t[1] = 2 \cap j_t = 0] \cdot P[S_t[1] = 2 \cap j_t = 0] \\ &+ P[O_{t+1} = O_{t+2} | S_t[1] \neq 2 \cup j_t \neq 0] \cdot P[S_t[1] \neq 2 \cup j_t \neq 0] \\ &= 0 \cdot \frac{1}{N^2} + \frac{1}{N} \cdot (1 - \frac{1}{N^2}) \\ &= \frac{1}{N} \cdot (1 - \frac{1}{N^2}) \, . \end{aligned}$$
□

Therefore, the required number of samples needed to establish the distinguisher is $O(N^5)$ according to Theorem 2 (note that in this case $p = 1/N$ and $q = -1/N^2$).

However, our experiments for $N = 256$ show that the bias can be detected much sooner: 2^{32} pairs of the output bytes (each pair is chosen from rounds $t = 257$ and $t = 258$) are sufficient for a distinguisher, where the theory predicts that this should be 2^{40}. In our experiments, the expected number of pairs, for which the bytes are equal, trails the expected number from a random source by *1.13 standard deviations* of the binomial distribution with parameters $(2^{32}, 1/256)$. A large number of experiments showed that the frequency of simultaneous occurrence of $j = 0$ and $S[1] = 2$ at the 256th round is much higher than expected. This phenomenon accounts for the optimistic behavior of our distinguisher. However, it is still unknown how to quantify the bias in $P[j_{256} = 0 \cap S_{256}[1] = 2]$ theoretically. It is worth noting at this point that Mironov, based on an idealized model of RC4, has suggested to drop the initial $3 \cdot N$ bytes (more conservatively the initial $12 \cdot N$ bytes) in order to obtain a uniform distribution of the initial permutation thereby ruling out any possibility of a strong attack [10].

2.5 Possibility of a Weak Distinguisher

The basic fact examined in the Theorem 1 can be used to characterize a general internal state which would produce unequal consecutive bytes. One can see that for an RC4 internal state, if $i_t = j_t$ and $S_t[i_t + 1] = 2$ then $O_{t+1} \neq O_{t+2}$. If we assume uniformity of RC4 internal states when the value of i is fixed, then the above observation allows for a weak distinguisher for RC4 (i.e., distinguishing RC4 using a single stream generated

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

by a randomly chosen key). However, extensive experiments show that, in case of a single stream, the bias in the output bytes due to these special states is much weaker. With a sample size of 2^{32} pairs of bytes, the expected number of pairs for which the output bytes are equal trails the expected number from a random source by *0.21 standard deviations* of the binomial distribution with parameters $(2^{32}, 1/256)$ (compared to *1.13 standard deviations* for the strong distinguisher with the same sample size). The experimentally obtained standard deviation of the distribution of the number of pairs with equal members in a sample of 2^{32} pairs equals $0.9894 \cdot \sigma$ where σ is the standard deviation of the binomial distribution with parameters $(2^{32}, 1/256)$. The closeness of these two distributions shows that such a *weak distinguisher* is less effective than the strong distinguisher with respect to the required number of outputs. Further experimental work is required to determine the effectiveness of distinguishers which require outputs for fewer keys (say 2^{20} rather than 2^{32}) but longer output streams than just a pair of consecutive bytes for each key.

It is still unclear how this correlation between the internal and external states can be used to mount a full attack on RC4. However, the observation reveals a weakness in the working principle of the cipher, even if N output bytes are dropped.

3 Analyzing RC4 Design Principles

The pseudorandom bit generation in RC4 is divided into two stages (see Sect. 1.1). The key scheduling algorithm KSA intends to turn an identity permutation S into a pseudorandom permutation of elements and the pseudorandom byte generation algorithm PRGA issues one output byte from a pseudorandom location of S in every round. At every round the secret internal state S is changed by the swapping of elements, one in a known location and another pointed to by a 'random' index. The whole idea is inspired by the principle of an exchange shuffle to obtain a 'random' distribution of a deck of cards [8]. Therefore, the security of RC4 in this model of exchange shuffle depends mainly on the following three factors.

- Uniform distribution of the initial permutation of elements (that is, S).
- Uniform distribution of the value of the index-pointer of the element to be swapped with the element contained in a known index (that is, the index-pointer j).
- Uniform distribution of the value of the index-pointer from which the output is issued in a round (i.e., $S[i] + S[j]$).

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

Note that the above three conditions are necessary conditions of the security of the cipher but by no means they can be sufficient. This fact is wonderfully demonstrated by Golić in [4] using the *Linear Sequential Circuit Approximation* (LSCA for short) methods capturing the basic flaw in the model that the arrangement of the S-box elements in two successive rounds can be approximated to be identical because of 'negligible' changes of S in two successive rounds.

In this paper we take the key scheduling algorithm of RC4 for granted and assume that the distribution of the initial permutation of elements is uniform; we focus solely on the pseudorandom output generation process. As a consequence, the adversary concentrates on deriving the secret internal state S (not the secret key K) from the known outputs exploiting correlations between the internal state and the output bytes.

Most of the known attacks on RC4 to derive part of the secret internal state are based on fixing a few elements of the S-box along with the two index pointers i and j that give information about the outputs at certain rounds with probability 1 or very close to it. This at once results in *distinguishing attacks* on the cipher and helps to derive the secret internal state with probability that is significantly larger than expected. Paul and Preneel have proved that under reasonable assumptions the maximum probability with which a part of the internal state (i.e., certain S-box elements and the value of j) can be predicted by observing known outputs is $1/N$ which is too high [12]. Note that these correlations between the internal state and the external state immediately violate the 'randomness' criteria of an ideal cipher.

The only algorithm attempting to derive the entire internal state of RC4 from the sequence of outputs is by Knudsen *et al.* which is based on a "guess on demand" strategy [7]. The expected complexity of the algorithm is much smaller than a trivial exhaustive search because it implicitly uses the weakness of RC4 that a part of internal state can be guessed with non-trivial probability by observing certain outputs.

In the following discussion we modify RC4 in an attempt to achieve tighter security than the original cipher within the scope of the existing model of exchange shuffle without degrading its speed.

4 RC4A: An Attempt to Improve RC4

4.1 RC4A Design Principles

As most of the existing known plaintext attacks on RC4 harness the stronger correlations between the internal and external states (in generic

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

term b-predictive a-state attack [9, 12]), in principle, making the output generation dependent on more random variables weakens the correlation between them, i.e., the probability to guess the internal state by observing output sequence can be reduced. The larger the number of the variables the weaker will be the correlation between them. On the other hand, intuitively, the large number of variables increases the time complexity as it involves more arithmetic operations.

4.2 RC4A Description

We take one randomly chosen key k_1. Another key k_2 is also generated from a pseudorandom bit generator (e.g. RC4) using k_1 as the seed. Applying the Key Scheduling Algorithm, as described in Fig. 1, we construct two S-boxes S_1 and S_2 using the keys k_1 and k_2 respectively. As mentioned before we assume that S_1 and S_2 are two random permutations of $\{0, 1, 2, \ldots, N-1\}$. In this new scheme the Key Scheduling Algorithm is assumed to produce a uniform distribution of permutation of $\{0, 1, 2, \ldots, N-1\}$. Therefore, our effort focuses on the security of the Pseudo-Random Generation Algorithm. In Fig. 3, we show the pseudocode of the pseudorandom byte generation algorithm of RC4A. All the arithmetic operations are computed modulo N. The transition of the internal states of the two S-boxes are based on an exchange shuffle as before. Here we introduce two variables j_1 and j_2 corresponding to S_1 and S_2 instead of one. The only modification is that the index-pointer $S_1[i] + S_1[j]$ evaluated on S_1 produces output from S_2 and vice-versa (see steps 5, 6 and 9, 10 of Fig. 3). The next round starts after each output generation.

RC4A uses fewer instructions per output byte than RC4. To produce two successive output bytes the i pointer is incremented once in case of RC4A where it is incremented twice to produce as many output words in RC4.

Parallelism in RC4A. The performance of RC4A can be further improved by extracting the parallelism latent in the algorithm. The parallel steps of the algorithm can be easily found by drawing a dependency graph of the steps shown in Fig. 3. In the following list the parallel steps of RC4A are shown within brackets.

1. (3, 7).
2. (4, 5, 9).
3. (6, 10).
4. (8, 2).

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

1. Set $i = 0, j_1 = j_2 = 0$
2. i++
3. $j_1 = j_1 + S_1[i]$
4. Swap$(S_1[i], S_1[j_1])$
5. $I_2 = S_1[i] + S_1[j_1]$
6. Output=$S_2[I_2]$
7. $j_2 = j_2 + S_2[i]$
8. Swap$(S_2[i], S_2[j_2])$
9. $I_1 = S_2[i] + S_2[j_2]$
10. Output=$S_1[I_1]$
11. Repeat from step 2.

Fig. 3. Pseudo-random Generation Algorithm of RC4A.

The existence of many parallel steps in RC4A is certainly an important aspect of this new cipher and it offers the possibility of a faster stream cipher if RC4A is implemented efficiently.

5 Security Analysis of RC4A

The RC4A pseudorandom bit generator has passed all the statistical tests listed in [13]. RC4A achieves two major gains over RC4. By making every byte depend on at least two random values (e.g. O_1 depends on $S_1[1]$, $S_1[j_1]$ and $S_2[S_1[1] + S_1[j_1]]$) of S_1 and S_2 the secret internal state of RC4A becomes $N!^2 \times N^3$. So, for $N = 256$, the number of secret internal states for RC4A is approximately 2^{3392} when the number is only 2^{1700} for RC4.

Let the events E_A and E_B denote the occurrences of an *internal state* (i.e., a known elements in the S-boxes, i, j_1 and j_2) and the corresponding b outputs when the i value of an *internal state* is known. We assume uniformity of the *internal state* and the corresponding *external state* for any fixed i value of the *internal state*. Assuming that a is much smaller than N and disregarding the small bias induced in E_B due to E_A, we apply Bayes' Rule to get

$$P[E_A|E_B] = \frac{P[E_A]}{P[E_B]} P[E_B|E_A] \approx \frac{N^{-(a+2)}}{N^{-b}} \cdot 1 = N^{b-a-2} \,. \qquad (1)$$

Paul and Preneel [12] have proved that $a \geq b$ for RC4 for small values of a. We omit the proof as it is quite rigorous and beyond the scope of this paper. Exactly the same technique can be applied here to prove that a

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

known elements of the S-boxes along with i, j_1 and j_2 cannot predict more than a elements for small values of a. Therefore, the maximum probability with which any internal state of RC4A can be predicted from a known output sequence equals $1/N^2$ compared to $1/N$ for RC4. In the following sections we describe how RC4A resists the two major attacks on it: one attempts to derive the entire internal state deterministically and another to derive a part of the internal state probabilistically.

5.1 Precluding the Backtracking Algorithm by Knudsen et al.

As mentioned earlier that the "guess on demand" backtracking algorithm by Knudsen *et al.* is so far the best algorithm to deduce the internal state of RC4 from the known plaintext [7]. Now we briefly discuss the functionality of the variant of the algorithm to be applied for RC4A.

The algorithm simulates RC4A by observing only the output bytes in recursive function calls. The values of $S[i]$ and $S[j]$ in one S-box are guessed from the permutation elements to agree with the output and its possible location in the other S-box. If they match then the algorithm calls the round function for the next round. If an anomaly occurs then it backtracks through the previous rounds and re-guesses. The number of outputs m, needed to uniquely determine the entire internal state, is bounded below by the inequality, $2^{nm} > (2^n!)^2$. Therefore, $m \geq 2N$ (note, $N = 2^n$).

Theorem 3 (RC4 vs RC4A). *If the expected computational complexity to derive the secret internal state of RC4A from known $2N$ initial output bytes with the algorithm by Knudsen et al. is C_{rc4a} and if the corresponding complexity for RC4 using N known initial output bytes is C_{rc4} then C_{rc4a} is much higher than C_{rc4} and C_{rc4a} can be approximated to C^2_{rc4} under certain assumptions.*[2]

Proof. According to the algorithm by Knudsen *et al.*, the internal state of RC4 is derived using only the first N output bytes, that is, simulating RC4 for the first N rounds. The variant of this algorithm which works on RC4A uses the initial $2N$ bytes, thereby runs for the first $2N$ rounds.

Let the algorithms A_1 and A_2 derive the secret internal states for RC4 and RC4A respectively. At every round the S-boxes are assigned either 0, 1, 2, or 3 elements and move to the next round.

Let, at the tth round, A_2 go to the next round after assigning k elements an expected number of $m_{k,t}$ times. So the number of value assign-

[2] The complexity is measured in terms of the number of value assignments.

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

ments in the tth round is $\sum_{k=0}^{3} k \cdot m_{k,t}$. Note, each of the $\sum_{k=0}^{3} m_{k,t}$ iterations gives rise to an S-box arrangement in the next round. It is possible that we reach some S-box arrangements from which no further transition to the next rounds is possible because of contradictions. In such case, we assume assignment of zero elements in the S-box till the Nth round is reached. Let the number of S-box arrangements at the tth round from which these $\sum_{k=0}^{3} m_{k,t}$ arrangements are generated is L_t. Consequently,

$$\sum_{k=0}^{3} m_{k,t} = L_{t+1}\,. \tag{2}$$

Now we set,

$$\sum_{k=0}^{3} k \cdot m_{k,t} = \tilde{k}_t \cdot \sum_{k=0}^{3} m_{k,t} = \tilde{k}_t \cdot L_{t+1}\,. \tag{3}$$

In Eqn. (3), $\tilde{k}_t$ is the expected number elements which are assigned to the S-boxes in each iteration in that particular round. If each of L_t is assumed to produce an expected $\tilde{L}_{t+1}$ number of S-box arrangements in the next round then Eqn. (3) becomes,

$$\sum_{k=0}^{3} k \cdot m_{k,t} = \tilde{k}_t \cdot (\tilde{L}_{t+1} \cdot L_t)\,. \tag{4}$$

Denoting the total number of value assignments in the tth round by $C(t)$, it is easy to note from Eqn. (4),

$$C(t) = \tilde{k}_t \cdot (\tilde{L}_{t+1} \cdot L_t)\,. \tag{5}$$

Proceeding this way it can be shown that,

$$C(t+s) = \tilde{k}_{t+s} \cdot L_t \cdot \prod_{i=t}^{t+s} \tilde{L}_{i+1}\,. \tag{6}$$

If $t = 1$ then $L_t = 1$. Setting $t + s = n$ in Eqn. (6), we get,

$$C(n) = \tilde{k}_n \cdot \prod_{i=1}^{n} \tilde{L}_{i+1}\,. \tag{7}$$

From Eqn. (2) and Eqn. (3), $C(n)$ can be evaluated $\forall n \in \{1, 2, \ldots 2N\}$ when $m_{k,t}$ is known $\forall (k,t) \in \{0,1,2,3\}\{1,2,\ldots 2N\}$.

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

It is important to note that on a random output sequence $\tilde{k}_{2f-1} \approx \tilde{k}_{2f}$ and $\tilde{L}_{2f} \approx \tilde{L}_{2f+1}\ \forall f \in \{1, 2, \ldots N\}$. The reason behind the approximation is that, with the algorithm by Knudsen *et al.*, the difference between the expected number of assignments in the S-boxes in the $(2f-1)$th and the $2f$th rounds is very small. Therefore, the overall complexity C_{rc4a} becomes,

$$\begin{aligned}
C_{rc4a} &= \sum_{n=1}^{2N} C(n) \\
&= \sum_{n=1}^{2N} (\tilde{k}_n \cdot \prod_{i=1}^{n} \tilde{L}_{i+1}) \\
&= \tilde{k}_1 \cdot \tilde{L}_2 + \sum_{i=1}^{N} (\tilde{k}_{2i} \cdot \prod_{j=1}^{i} \tilde{L}^2_{2j+1}) + \sum_{i=2}^{N} (\tilde{k}_{2i-1} \cdot \tilde{L}_{2i} \cdot \prod_{j=1}^{i-1} \tilde{L}^2_{2j}) \\
&= \tilde{k}_1 \cdot \tilde{L}_2 + \sum_{i=1}^{N} (\tilde{k}_{2i-1} \cdot \prod_{j=1}^{i} \tilde{L}^2_{2j}) + \sum_{i=2}^{N} (\tilde{k}_{2i-1} \cdot \tilde{L}_{2i} \cdot \prod_{j=1}^{i-1} \tilde{L}^2_{2j}) \,.
\end{aligned}$$

Replacing $\tilde{k}_q$ and $\tilde{L}_q$ by $x_{\frac{q+1}{2}}$ and $g_{\frac{q}{2}}$ we get,

$$C_{rc4a} = \sum_{i=1}^{N} (x_i \cdot \prod_{j=1}^{i} g_j^2) + x_1 \cdot g_1 + \sum_{i=2}^{N} (x_i \cdot g_i \cdot \prod_{j=1}^{i-1} g_j^2) \,. \tag{8}$$

Applying a similar technique as above it is easy to see that,

$$C_{rc4} = \sum_{i=1}^{N} (x_i \cdot \prod_{j=1}^{i} g_j) \,. \tag{9}$$

Again we note that the difference between the expected number of elements that are already assigned in S_1 for RC4A at round $(2t-1)$ and the expected number of elements in S for RC4 at round t is negligible. Therefore, the corresponding $\tilde{k}_t$ and $\tilde{L}_{t+1}$ for RC4 can be approximated to $\tilde{k}_{2t-1}$ and $\tilde{L}_{2t}$ for RC4A.

As the g_i's are real numbers greater than 1 and the x_i's are non-negative real numbers, from Eqn. (8) and Eqn. (9) it is easy to see that $C_{rc4a} \gg C_{rc4}$.

We observe from the algorithm that $x_i \in \{y : 0 \le y \le 3, y \in \mathbb{R}\}$. It is clear from the algorithm that x_i decreases as i increases. Intuitively, x_i is less than one in the last rounds. Therefore, assuming $C_{rc4a} \approx \prod_{i=1}^{N} g_i^2$ and $C_{rc4} \approx \prod_{i=1}^{N} g_i$, we get $C_{rc4a} \approx C_{rc4}^2$. □

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

By Theorem 3, the expected complexity to deduce the secret internal state of RC4A ($N = 256$) with the algorithm by Knudsen *et al.* is 2^{1558} when the corresponding complexity is 2^{779} for RC4.

5.2 Resisting the Fortuitous States Attack

Fluhrer and McGrew discovered certain RC4 states in which only m known consecutive S-box elements participate in producing the next m successive outputs. Those states are defined to be *Fortuitous States* (see [3, 12] for a detailed analysis). *Fortuitous States* increase the probability to guess a part of internal state in a known plaintext attack (see Eqn. (1)). The larger the probability of the occurrence of a fortuitous state, the smaller will be the number of required rounds to obtain one of them.

RC4A also weakens the fortuitous state attack by a large degree. A moment's reflection shows that RC4A does not have any fortuitous state of length 1. Now we will compare the probability of the occurrence of a fortuitous state of length $2a$ in RC4A to that of length a in RC4. It is easy to note that a fortuitous state of length $2a$ of RC4A implies and is implied by two fortuitous states of length a of RC4 appearing simultaneously in S_1 and S_2. If C denotes the number of fortuitous states of length a of RC4 then the expected number of fortuitous states of length $2a$ in RC4A is C^2/N. Let P_a denote the probability of the occurrence of a fortuitous state of length a in RC4 and P_{2a} denote the probability of the occurrence of a fortuitous state of length $2a$ in RC4A. Then, for small values of a, $P_a = \frac{C}{N^{a+2}}$ and $P_{2a} = \frac{C^2}{N^{2a+4}}$ which immediately implies $P_{2a} < P_a$.

5.3 Resisting Other Attacks

One can see that the strong positive bias of the second output byte of RC4 toward zero [9], and the bias described in the first part of this paper are also diminished in this new cipher as more random variables are required to be fixed for the biased state to occur.

5.4 Open Problems and Directions for Future Work

Although RC4A has an improved security over the original cipher against most of the known plaintext attacks, it is still as vulnerable as RC4 against the attack by Golić which uses the positive correlation between the second binary derivative of the least significant bit output sequence and 1. The weakness originates from the slow change of the S-box in successive

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

rounds that seems to be inherent in any model based on exchange shuffle. Therefore, this still remains an open problem whether it is possible to remove this weakness from the output words of the stream cipher based on an exchange shuffle while retaining all of its speed and security.

Our work leaves room for more research. It is worthwhile to note that one output byte generation in this existing model of exchange shuffle involves two random pointers; j and $S[i]+S[j]$. In RC4 both the pointers fetch values from a single S-box. We obtained better results by making $S[i]+S[j]$ fetch value from a different S-box. What if we obtain $S[j]$ from another S-box and generate output using three S-boxes?

6 Conclusions

In this paper we have described a new statistical weakness in the first two output bytes of the RC4 keystream generator. The weakness does not disappear even after dropping the initial N bytes. Based on this observation, we recommend to drop at least the initial $2N$ bytes of RC4 in all future applications of it. In the second part of the paper we attempted to improve the security of RC4 by introducing more random variables in the output generation process thereby reducing the correlation between the internal and the external states.

As a final comment we would like to mention that the security of RC4A could be further improved. For example, one could introduce key-dependent values of i and j at the beginning of the first round, and one could address the weaknesses of the Key Scheduling Algorithm. In this paper, we have assumed that the original Key Scheduling Algorithm produces a uniform distribution of the initial permutation of elements, which is certainly not correct.

Acknowledgments. We are grateful to Adi Shamir for helpful comments. The authors also thank Kenneth G. Paterson for pointing out the possibility of a *weak* distinguisher using our observation. We would like to thank Joseph Lano for providing us with experimental results. Thanks are due to Alex Biryukov, Christophe De Cannière, Jorge Nakahara Jr. and Dai Watanabe for many useful discussions. The authors also acknowledge the constructive comments of the anonymous reviewers.

References

1. H. Finney, "An RC4 cycle that can't happen," Post in `sci.crypt`, September 1994.

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.

2. S. Fluhrer, I. Mantin, A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," *SAC 2001* (S. Vaudenay, A. Youssef, eds.), vol. 2259 of *LNCS*, pp. 1-24, Springer-Verlag, 2001.
3. S. Fluhrer, D. McGrew, "Statistical Analysis of the Alleged RC4 Keystream Generator," *Fast Software Encryption 2000* (B. Schneier, ed.), vol. 1978 of *LNCS*, pp. 19-30, Springer-Verlag, 2000.
4. J. Golić, "Linear Statistical Weakness of Alleged RC4 Keystream Generator," *Eurocrypt '97* (W. Fumy, ed.), vol. 1233 of *LNCS*, pp. 226-238, Springer-Verlag, 1997.
5. A. Grosul, D. Wallach, "A related key cryptanalysis of RC4," *Department of Computer Science, Rice University, Technical Report TR-00-358,* June 2000.
6. R. Jenkins, "Isaac and RC4," Published on the Internet at `http://burtleburtle.net/bob/rand/isaac.html`.
7. L. Knudsen, W. Meier, B. Preneel, V. Rijmen, S. Verdoolaege, "Analysis Methods for (Alleged) RC4," *Asiacrypt '98* (K. Ohta, D. Pei, eds.), vol. 1514 of *LNCS*, pp. 327-341, Springer-Verlag, 1998.
8. D.E. Knuth, *"The Art of Computer Programming," vol. 2, Seminumerical Algorithms,* Addison-Wesley Publishing Company, 1981.
9. I. Mantin, A. Shamir, "A Practical Attack on Broadcast RC4," *Fast Software Encryption 2001* (M. Matsui, ed.), vol. 2355 of *LNCS*, pp. 152-164, Springer-Verlag, 2001.
10. I. Mironov, "Not (So) Random Shuffle of RC4," *Crypto 2002* (M. Yung, ed.), vol. 2442 of *LNCS*, pp. 304-319, Springer-Verlag, 2002.
11. S. Mister, S. Tavares, "Cryptanalysis of RC4-like Ciphers," *SAC '98* (S. Tavares, H. Meijer, eds.), vol. 1556 of *LNCS*, pp. 131-143, Springer-Verlag, 1999.
12. S. Paul, B. Preneel, "Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator," *Indocrypt 2003* (T. Johansson, S. Maitra, eds.), vol. 2904 of *LNCS*, pp. 52-67, Springer-Verlag, 2003.
13. B. Preneel *et al.*, "NESSIE Security Report," Version 2.0, IST-1999-12324, February 19, 2003, `http://www.cryptonessie.org`.
14. M. Pudovkina, "Statistical Weaknesses in the Alleged RC4 keystream generator," *Cryptology ePrint Archive 2002–171,* IACR, 2002.
15. A. Roos, "Class of weak keys in the RC4 stream cipher," Post in `sci.crypt`, September 1995.
16. A. Stubblefield, J. Ioannidis, A. Rubin, "Using the Fluhrer, Mantin and Shamir attack to break WEP," *Proceedings of the 2002 Network and Distributed Systems Security Symposium,* pp. 17–22, 2002.

Appeared in *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science, Springer-Verlag, pp. 245–259, 2004.