RC4 State Information at Any Stage Reveals the Secret Key

Goutam Paul*, Subhamoy Maitra†

Abstract

A theoretical analysis of the RC4 Key Scheduling Algorithm (KSA) is presented in this paper, where the nonlinear operation is swapping among the permutation bytes. Explicit formulae are provided for the probabilities with which the permutation bytes after the KSA are biased to the secret key. Theoretical proofs of these formulae have been left open since Roos's work (1995). Based on this analysis, an algorithm is devised to recover the l bytes (i.e., $8l$ bits, typically $5 \leq l \leq 16$) secret key from the final permutation after the KSA with constant probability of success. The search requires $O(2^{4l})$ many operations which is the square root of the exhaustive key search complexity 2^{8l}. Moreover, given the state information, i.e., (a) the permutation, (b) the number of bytes generated (which is related to the index i) and (c) the value of the index j, after any number of rounds in Pseudo Random Generation Algorithm (PRGA) of RC4, one can deterministically get back to the permutation after the KSA and thereby extract the keys efficiently with a constant probability of success. Finally, a generalization of the RC4 KSA is analyzed corresponding to a class of updation functions of the indices involved in the swaps. This reveals an inherent weakness of shuffle-exchange kind of key scheduling.

Keywords: Bias, Cryptanalysis, Key Recovery, Key Scheduling, Permutation, RC4, Stream Cipher.

1 Introduction

Two decades have passed since the inception of RC4. Though a variety of other stream ciphers have been discovered after RC4, it is still the most popular and most frequently used stream cipher algorithm due to its simplicity, ease of implementation, speed and efficiency. RC4 is widely used in the Secure Sockets Layer (SSL) and similar protocols to protect the internet traffic, and was integrated into Microsoft Windows, Lotus Notes,

*Department of Computer Science and Engineering, Jadavpur University, Kolkata 700 032, India, Email: goutam_paul@cse.jdvu.ac.in

†Applied Statistics Unit, Indian Statistical Institute, 203 B T Road, Kolkata 700 108, India, Email: subho@isical.ac.in

Apple AOCE, Oracle Secure SQL, etc. Though the algorithm can be stated in less than ten lines, even after many years of analysis its strengths and weaknesses are of great interest to the community. In this paper, we study the Key Scheduling Algorithm of RC4 in detail and find out results that have implications towards the security of RC4. Before getting into the contribution in this paper, we first revisit the basics of RC4.

The RC4 stream cipher has been designed by Ron Rivest for RSA Data Security in 1987, and was a propriety algorithm until 1994. It uses an S-Box $S = (S[0], \ldots, S[N-1])$ of length N, each location being of 8 bits. Typically, $N = 256$. S is initialized as the identity permutation, i.e., $S[i] = i$ for $0 \leq i \leq N-1$. A secret key of size l bytes (typically, $5 \leq l \leq 16$) is used to scramble this permutation. An array $K = (K[0], \ldots, K[N-1])$ is used to hold the secret key, where each location is of 8 bits. The key is repeated in the array K at key length boundaries. For example, if the key size is 40 bits, then $K[0], \ldots, K[4]$ are filled by the key and then this pattern is repeated to fill up the entire array K.

The RC4 cipher has two components, namely, the Key Scheduling Algorithm (KSA) and the Pseudo Random Generation Algorithm (PRGA). The KSA turns the random key K into a permutation S of $0, 1, \ldots, N-1$ and PRGA uses this permutation to generate a pseudo random keystream bytes. The keystream output byte z is XOR-ed with the message byte to generate the ciphertext byte at the sender end. Again, z is XOR-ed with the ciphertext byte to get back the message byte at the receiver end.

Any addition used related to the RC4 description is in general addition modulo N unless specified otherwise.

Algorithm KSA
Initialization:
For $i = 0, \ldots, N-1$
$S[i] = i$;
$j = 0$;
Scrambling:
For $i = 0, \ldots, N-1$
$j = (j + S[i] + K[i])$;
Swap$(S[i], S[j])$;

Algorithm PRGA
Initialization:
$i = j = 0$;
Output Keystream Generation Loop:
$i = i + 1$;
$j = j + S[i]$;
Swap($S[i]$, $S[j]$);
$t = S[i] + S[j]$;
Output $z = S[t]$;

Note that defining the array K to be of size N enables us to write $K[i]$ instead of the typical $K[i \bmod l]$ in the description of the algorithm. This is done for the sake of simplification in the subsequent analysis of the algorithm.

1.1 Outline of the contribution

In this paper, the updation of the permutation S in different rounds of the KSA is analyzed and it is theoretically proved that after the completion of the KSA, the initial bytes of the permutation will be significantly biased towards a combination of the secret key bytes. Such biases were observed by Roos in [13] for the first time. It has been noted in [13] that after the completion of the KSA, the most likely value of the i-th element of the permutation

for the first few values of i is given by $S[i] = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]$. However, the probability $P(S[i] = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x])$ could not be theoretically arrived in [13] and experimental values have been provided as in the following table.

i	$P(S[i] = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x])$
0-23	.370 .368 .362 .358 .349 .340 .330 .322 .309 .298 .285 .275 .260 .245 .229 .216 .203 .189 .173 .161 .147 .135 .124 .112
24-47	.101 .090 .082 .074 .064 .057 .051 .044 .039 .035 .030 .026 .023 .020 .017 .014 .013 .012 .010 .009 .008 .007 .006 .006

Table 1: The probabilities experimentally observed by Roos [13].

We theoretically prove for the first time with what probabilities the final permutation bytes after the KSA are correlated with the secret key bytes. Roos [13] commented that "Swapping is a nasty nonlinear process which is hard to analyze." That process is analyzed in a disciplined manner in this paper that unfolds the effect of swapping in the KSA of RC4 (see Lemma 1 and Theorem 1 in Section 2).

In Section 3, we use these biases to show that if the permutation after the KSA is available, then one can get the key bytes in time much less than the exhaustive key search. For a secret key of size $e = 8l$ bits ($48 \leq e \leq 128$), the key can be recovered in $O(2^{\frac{e}{2}})$ effort with a (good) constant probability of success. In a shuffle-exchange kind of stream cipher, for proper cryptographic security, one may expect that after the key scheduling algorithm one should not be able to get back to any information regarding the secret key bytes from the random permutation in time complexity less than the exhaustive key search. We show that the KSA of RC4 is weak in this aspect.

Next in Section 4, we point out that if the state information of RC4 during the PRGA is available, then one can deterministically get back to the permutation after the KSA. By state information we mean (a) the entire permutation S, (b) the number of keystream output bytes generated (which is related to the index i) and (c) the value of the index j. Once the final permutation after the KSA is retrieved, using the approach of Section 3 we can recover the secret key.

Finally, in Section 5, we consider the generalization of the RC4 KSA where the index j can be updated in different manners. In RC4 KSA, the updation rule is $j = (j+S[i]+K[i])$. We show that for any arbitrary secret key and for a certain class of update functions, which compute the new value of the index j in the current round as a function of "the permutation S and j in the previous round" and "the secret key K", it is always possible to construct explicit functions of the key bytes which the final permutation will be biased to. This shows that the RC4 KSA cannot be made more secure by replacing the updation rule $j = j + S[i] + K[i]$ with any rule from a large class that we present. Such bias is intrinsic to shuffle-exchange kind of paradigm, where one index (i) is updated linearly and another index (j) is modified pseudorandomly.

1.2 Background

There are two broad approaches in the study of cryptanalysis of RC4: attacks based on the weaknesses of the KSA and those based on the weaknesses of the PRGA.

Distinguishing attacks are the main motivation for PRGA-based approach [1, 3, 5, 6, 7, 11, 12]. Important results in this approach include bias in the keystream output bytes. For example, a bias in the second output byte being zero has been proved in [5] and a bias in the equality of the first two output bytes has been shown in [12]. In [9], RC4 has been analyzed using the theory of random shuffles and it has been recommended that initial 512 bytes of the keystream output should be discarded in order to be safe.

Initial empirical works based on the weaknesses of the RC4 KSA were done in [13, 14] and several classes of weak keys had been identified. Recently, a more general theoretical study has been performed in [10] which includes the observations of [13]. The work [10] shows how the bias of the "third permutation byte" (after the KSA) towards the "first three secret key bytes" propagates to the first keystream output byte (in the PRGA). Thus, it renews the interest to study how the permutation after the KSA (which acts as a bridge between the KSA and the PRGA) is biased towards the secret key, which is theoretically solved in this paper.

Some weaknesses of the KSA have been addressed in great detail in [2] and practical attacks have been mounted on RC4 in the IV mode (e.g. WEP [4]). Further, the propagation of weak key patterns to the output keystream bytes has also been discussed in [2]. In [8, Chapter 6], correlation between the permutations that are a few rounds apart have been discussed.

2 Theoretical Analysis of the Key Scheduling

Let S_0 be the initial permutation and $j_0 = 0$ be the initial value of the index j before the KSA begins. Note that in the original RC4, S_0 is the identity permutation. Let j_{i+1} be the updated value of j and S_{i+1} be the new permutation obtained after the completion of round i of the KSA, $0 \leq i \leq N-1$. Then S_N would be the final permutation after the complete KSA.

We now prove a general formula (Theorem 1) that estimates the probability with which the permutation bytes after the RC4 KSA are related to certain combinations of the secret key bytes. The result we present has two fold significance. It gives for the first time a theoretical proof explicitly showing how these probabilities change as functions of i. Further, it does not assume that the initial permutation is an identity permutation. The result holds for any arbitrary initial permutation.

The proof of Theorem 1 depends on a lemma which we prove below first.

Lemma 1 *Assume that during the KSA rounds, the index j takes its values uniformly at random. Then, $P(j_{i+1} = \sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]) = (\frac{N-1}{N})^{\frac{i(i+1)}{2}}$, for initial values of i.*

Proof: We will prove it by induction on i.

- *Base Case*: Before the beginning of the KSA, $j_0 = 0$. Now, in the first round with $i = 0$, we have $j_1 = j_0 + S_0[0] + K[0] = 0 + S_0[0] + K[0] = \sum_{x=0}^{0} S_0[x] + \sum_{x=0}^{0} K[x]$ with probability $1 = (\frac{N-1}{N})^0 = (\frac{N-1}{N})^{\frac{0(0+1)}{2}}$. Hence, the result holds for the base case.
- *Inductive Case*: Suppose, that the result holds for all rounds from 0 to $i-1$, $i \geq 1$. Now, for the i-th round, we would have $j_{i+1} = j_i + S_i[i] + K[i]$. In the right hand side of this equation, j_i can be replaced by $\sum_{x=0}^{i-1} S_0[x] + \sum_{x=0}^{i-1} K[x]$ with probability $(\frac{N-1}{N})^{\frac{(i-1)i}{2}}$ (by *inductive hypothesis*). Further, $S_i[i]$ can be replaced by $S_0[i]$, iff it has not been involved in any swap during the previous rounds, i.e., iff any of the values $j_1, j_2, \ldots, j_i$ has not hit the index i, the probability of which is $(\frac{N-1}{N})^i$. Hence, replacing both j_i and $S_i[i]$ in the right hand side of the above equation, we get $P(j_{i+1} = \sum_{x=0}^{i-1} S_0[x] + \sum_{x=0}^{i-1} K[x] + S_0[i] + K[i]) = (\frac{N-1}{N})^{\frac{(i-1)i}{2}} \cdot (\frac{N-1}{N})^i$.

  That is, $P(j_{i+1} = \sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]) = (\frac{N-1}{N})^{\frac{(i-1)i}{2}+i} = (\frac{N-1}{N})^{\frac{i(i+1)}{2}}$. ■

Corollary 1 *If the initial permutation is an identity permutation, i.e., if $S_0[i] = i$ for $0 \leq i \leq N-1$, then we have $P(j_{i+1} = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]) = (\frac{N-1}{N})^{\frac{i(i+1)}{2}}$, for initial values of i.*

Theorem 1 *Assume that during the KSA rounds, the index j takes its values uniformly at random. Then, $P\Big(S_N[i] = S_0\Big[\sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]\Big]\Big) = (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+(N-1)]} \cdot (\frac{N-i}{N})$, for initial values of i.*

Proof: During the swap in the i-th iteration, $S_{i+1}[i]$ is assigned the value of $S_i[j_{i+1}]$. This $S_i[j_{i+1}]$ can be replaced by $S_0[j_{i+1}]$ iff j_{i+1} has not been involved in any swap during the previous rounds, i.e., $j_{i+1} \notin \{0, 1, \ldots, i-1\}$, the probability of which is $(\frac{N-i}{N})$, as well as $j_{i+1} \notin \{j_1, j_2, \ldots, j_i\}$, the probability of which is $(\frac{N-1}{N})^i$. Thus, $P(S_{i+1}[i] = S_0[j_{i+1}]) = (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^i$, and using Lemma 1, we have $P\Big(S_{i+1}[i] = S_0\Big[\sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]\Big]\Big) = (\frac{N-1}{N})^{\frac{i(i+1)}{2}} \cdot (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^i = (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+i]} \cdot (\frac{N-i}{N})$.

After the completion of the KSA, the value of $S_N[i]$ will remain the same as $S_{i+1}[i]$ iff i is not involved in any swap during the subsequent $N-1-i$ rounds, i.e., iff any of the subsequent $N-1-i$ number of j values does not equal the index i, the probability of

which is $(\frac{N-1}{N})^{N-i-1}$. Hence, $P\Big(S_N[i] = S_0\Big[\sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]\Big]\Big)$
$= (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+i]} \cdot (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N-i-1} = (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+(N-1)]} \cdot (\frac{N-i}{N})$. ■

Roos's observation [13] is proved in the following corollary.

Corollary 2 *If the initial permutation is an identity permutation, i.e., if* $S_0[i] = i$ *for* $0 \leq i \leq N-1$, *then we have* $P(S_N[i] = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]) = (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+(N-1)]} \cdot (\frac{N-i}{N})$, *for initial values of* i.

In the following table we list the values of probabilities to compare with the experimental values provided in [13] and summarized in our Table 1.

i	$P(S[i] = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x])$
0-23	.369 .366 .361 .356 .349 .341 .332 .321 .310 .298 .286 .272 .259 .245 .231 .217 .203 .189 .175 .162 .149 .137 .125 .114
24-47	.103 .093 .084 .075 .067 .060 .053 .046 .041 .036 .031 .027 .023 .020 .017 .015 .013 .011 .009 .008 .006 .005 .004 .004

Table 2: The probabilities following Corollary 2.

As we have mentioned in the results above, the results are true for the initial values of i. After the index 48 and onwards, the experimental values tend to $\frac{1}{N}$ ($= 0.0039$ for $N = 256$) as is expected when we consider the equality between two randomly chosen values from a set of N elements. Our theoretical results (that converge to 0) should not be used to estimate the values for $i \geq 48$. This happens because of one approximation which we have made for simplification of the derivation. Whenever we replace $S[i]$ by i in the derivation, it is assumed that the index i was not involved in any previous swaps. Actually, the value at index i may go to some other indices and again come back to i after a sequence of swaps. Since the number of swaps increases with i, this has negligible effect on the initial values of i (as we observed, for $i < 48$) and is likely to have more prominent effect on latter values of i.

3 Recovering Secret Key Bytes from Permutation after KSA

In this section, we discuss how to get the the secret key bytes from the random looking permutation after the KSA using the equations of Corollary 2.

We explain the scenario with an example first.

Example 1 *Consider a 5 byte secret key with* $K[0] = 106, K[1] = 59, K[2] = 220, K[3] = 65$, *and* $K[4] = 34$. *We denote* $f_i = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]$. *If one runs the KSA, then the first* 16 *bytes of the final permutation will be as follows.*

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
f_i	106	166	132	200	238	93	158	129	202	245	105	175	151	229	21	142
$S_{256}[i]$	230	166	87	48	238	93	68	239	202	83	105	147	151	229	35	142

The strategy of key recovery would be to consider all possible sets of 5 equations chosen from the 16 equations $S_N[i] = f_i$, $0 \leq i \leq 15$, and then try to solve them. Whether the solution is correct or not can be checked by running the KSA and comparing the permutation obtained with the permutation in hand. Some of the choices may not be solvable at all.

The case of correct solution for this example correspond to the choices $i = 1, 4, 5, 8$ and 12, and the corresponding equations are:

$$K[0] + K[1] + (1 \cdot 2)/2 = 166 \quad (1)$$

$$K[0] + K[1] + K[2] + K[3] + K[4] + (4 \cdot 5)/2 = 238 \quad (2)$$

$$K[0] + \ldots + K[5] + (5 \cdot 6)/2 = 93 \quad (3)$$

$$K[0] + \ldots + K[8] + (8 \cdot 9)/2 = 202 \quad (4)$$

$$K[0] + \ldots + K[12] + (12 \cdot 13)/2 = 151 \quad (5)$$

In general, the correctness of the solution depends on the correctness of the selected equations. The probability that we will indeed get correct solutions is be related to the joint probability of $S_N[i] = f_i$ for the set of chosen i-values.

For a 5 byte key, if we go for an exhaustive search for the key, then the complexity would be 2^{40}. Whereas in our approach, we need to consider at the most $\binom{16}{5} = 4368 < 2^{13}$ sets of 5 equations. Assuming that solving a set of equation takes constant amount of time, the improvement is by more than a factor of 2^{27}.

From Corollary 2, we get how $S_N[i]$ is biased to different combinations of the keys. Let us denote $f_i = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]$ and $P(S_N[i] = f_i) = p_i$ for $0 \leq i \leq N-1$. We initiate the discussion for RC4 with secret key size l bytes. Suppose we want to recover exactly m out of l of secret key bytes by solving equations and the other $l - m$ bytes by exhaustive key search. For this, we consider n ($m \leq n \leq N$) many equations $S_N[i] = f_i$, $i = 0, 1, \ldots, n-1$, in l variables (the key bytes). Let EI_t denote the set of all independent systems of t equations, or, equivalently, the collection of the indices $\{i_1, i_2, \ldots, i_t\} \subseteq \{0, 1, \ldots, n-1\}$, corresponding to all sets of t independent equations (selected from the above system of n equations).

If we want to recover m key bytes by solving m equations out of the first n equations of the form $S_N[i] = f_i$, in general, we need to check whether each of the $\binom{n}{m}$ sets of m equations are independent or not. In the next Theorem, we present the criteria for checking the independence of such a set of equations and also the total number of such sets.

Theorem 2 *Let $l \geq 2$ be the RC4 key length in bytes. Suppose we want to select systems of m independent equations, $2 \leq m \leq l$, from the following n equations, $m \leq n \leq N$, involving the final permutation bytes: $S_N[i] = f_i$, where $f_i = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]$, $0 \leq i \leq n-1$.*

1. *The system* $S_N[i_q] = f_{i_q}$, $1 \leq q \leq m$, *of* m *equations selected from* $S_N[i] = f_i$, $0 \leq i \leq n-1$, *corresponding to* $i = i_1, i_2, \ldots, i_m$, *is independent if and only if any one of the following two conditions hold: either (i)* $i_q \bmod l$, $1 \leq q \leq m$, *yields* m *distinct values, or (ii)* $i_q \bmod l \neq (l-1)$, $1 \leq q \leq m$, *and there is exactly one pair* $i_x, i_y \in \{i_1, i_2, \ldots, i_m\}$ *such that* $i_x = i_y \pmod{l}$, *and all other* $i_q \bmod l, q \neq x, q \neq y$ *yields* $m-2$ *distinct values different from* $i_x, i_y \pmod{l}$.

2. *The total number of independent systems of* m (≥ 2) *equations is given by*

$$\begin{aligned}
|EI_m| &= \sum_{r=0}^{m} \binom{n \bmod l}{r} \binom{l-n \bmod l}{m-r} (\lfloor \tfrac{n}{l} \rfloor + 1)^r (\lfloor \tfrac{n}{l} \rfloor)^{m-r} \\
&+ \binom{n \bmod l}{1} \binom{\lfloor \frac{n}{l} \rfloor + 1}{2} \sum_{r=0}^{m-2} \binom{n \bmod l-1}{r} \binom{l-n \bmod l-1}{m-2-r} (\lfloor \tfrac{n}{l} \rfloor + 1)^r (\lfloor \tfrac{n}{l} \rfloor)^{m-2-r} \\
&+ \binom{l-n \bmod l-1}{1} \binom{\lfloor \frac{n}{l} \rfloor}{2} \sum_{r=0}^{m-2} \binom{n \bmod l}{r} \binom{l-n \bmod l-2}{m-2-r} (\lfloor \tfrac{n}{l} \rfloor + 1)^r (\lfloor \tfrac{n}{l} \rfloor)^{m-2-r},
\end{aligned}$$

where the binomial coefficient $\binom{u}{v}$ *has the value* 0, *if* $u < v$.

Proof: (*Part 1*) Since $f_i = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]$, each equation $S_N[i] = f_i$ involves a new variable $K[i]$ which is not present in any of the equations $S_N[x] = f_x$, $x = 0, 1, i-1$. However, the key bytes are repeated at key length boundaries, and so in any equation, $K[i]$ can be replaced by $K[i \bmod l]$. The m equations $S_N[i_q] = f_{i_q}$, $1 \leq q \leq m$ will be independent iff each equation involves a different key byte as a variable. Thus, if $i_q \bmod l$ $(1 \leq q \leq m)$ yields m distinct values, then the system is obviously independent. However, if the values are not distinct, then for some $x, y \in \{1, \ldots, m\}$, $x \neq y$, we have $i_x = i_y \bmod l$. Without loss of generality, suppose $i_x < i_y$. Then we can subtract $S_N[i_x] = f_{i_x}$ from $S_N[i_y] = f_{i_y}$ to get one equation involving some multiple of the sum $s = \sum_{x=0}^{l-1} K[x]$ of the key bytes. So we can replace exactly one equation involving either i_x or i_y by the new equation involving s, which will become a different equation with a new variable $K[l-1]$ provided $l - 1 \notin \{i_1 \bmod l, i_2 \bmod l, \ldots, i_m \bmod l\}$. Thus, the resulting system becomes independent, if no pair other than i_x, i_y are equal $(\bmod l)$.

(*Part 2*) We know that $n = (\lfloor \frac{n}{l} \rfloor)l + (n \bmod l)$. If we compute $i \bmod l$, for $i = 0, 1, \ldots n-1$, then we will have the following residue classes:

$$\begin{aligned}
[0] &= \{0, l, 2l, \ldots, (\lfloor \tfrac{n}{l} \rfloor)l\} \\
[1] &= \{1, l+1, 2l+1, \ldots, (\lfloor \tfrac{n}{l} \rfloor)l + 1\} \\
&\ldots \\
[n \bmod l - 1] &= \{n \bmod l - 1, l + (n \bmod l - 1), 2l + (n \bmod l - 1), \ldots, \\
&\quad (\lfloor \tfrac{n}{l} \rfloor)l + (n \bmod l - 1)\} \\
[n \bmod l] &= \{n \bmod l, l + (n \bmod l), 2l + (n \bmod l), \ldots, (\lfloor \tfrac{n}{l} \rfloor - 1)l + (n \bmod l)\} \\
&\ldots \\
[l-1] &= \{l-1, l + (l-1), 2l + (l-1), \ldots, (\lfloor \tfrac{n}{l} \rfloor - 1)l + (l-1)\}
\end{aligned}$$

The set of these l many residue classes can be classified into two mutually exclusive subsets, namely $A = \{[0], \ldots, [n \bmod l - 1]\}$ and $B = \{[n \bmod l], \ldots, [l-1]\}$, such that each residue class $a \in A$ has $\lfloor \frac{n}{l} \rfloor + 1$ members and each residue class $b \in B$ has $\lfloor \frac{n}{l} \rfloor$ members. Note that $|A| = n \bmod l$ and $|B| = l - (n \bmod l)$.

Now, the independent systems of m equations can be selected in three mutually exclusive and exhaustive ways. Case I corresponds to the condition (i) and Cases II & III correspond to the condition (ii) stated in the theorem.

Case I: *Select m different residue classes from $A \cup B$ and choose one i-value (the equation number) from each of these m residue classes.* Now, r of the m residue classes can be selected from the set A in $\binom{n \bmod l}{r}$ ways and the remaining $m - r$ can be selected from the set B in $\binom{l-n \bmod l}{r}$ ways. Again, corresponding to each such choice, the first r residue classes would give $\lfloor \frac{n}{l} \rfloor + 1$ choices for i (the equation number) and each of the remaining $m - r$ residue classes would give $\lfloor \frac{n}{l} \rfloor$ choices for i. Thus, the total number of independent equations in this case is given by $\sum_{r=0}^{m} \binom{n \bmod l}{r} \binom{l-n \bmod l}{m-r} (\lfloor \frac{n}{l} \rfloor + 1)^r (\lfloor \frac{n}{l} \rfloor)^{m-r}$.

Case II: *Select two i-values from any residue class in A. Then select $m-2$ other residue classes except $[l-1]$ and select one i-value from each of those $m-2$ residue classes.* We can pick one residue class $a \in A$ in $\binom{n \bmod l}{1}$ ways and subsequently two i-values from a in $\binom{\lfloor \frac{n}{l} \rfloor + 1}{2}$ ways. Of the remaining $m-2$ residue classes, r can be selected from $A \setminus \{a\}$ in $\binom{n \bmod l - 1}{r}$ ways and the remaining $m-2-r$ can be selected from $B \setminus \{[l-1]\}$ in $\binom{l-n \bmod l - 1}{m-2-r}$ ways. Again, corresponding to each such choice, the first r residue classes would give $\lfloor \frac{n}{l} \rfloor + 1$ choices for i (the equation number) and each of the remaining $m - r$ residue classes would give $\lfloor \frac{n}{l} \rfloor$ choices for i. Thus, the total number of independent equations in this case is given by $\binom{n \bmod l}{1} \binom{\lfloor \frac{n}{l} \rfloor + 1}{2} \sum_{r=0}^{m-2} \binom{n \bmod l - 1}{r} \binom{l-n \bmod l - 1}{m-2-r} (\lfloor \frac{n}{l} \rfloor + 1)^r (\lfloor \frac{n}{l} \rfloor)^{m-2-r}$.

Case III: *Select two i-values from any residue class in $B \setminus \{[l-1]\}$. Then select $m-2$ other residue classes and select one i-value from each of those $m-2$ residue classes.* This case is similar to case II, and the total number of independent equations in this case is given by $\binom{l-n \bmod l - 1}{1} \binom{\lfloor \frac{n}{l} \rfloor}{2} \sum_{r=0}^{m-2} \binom{n \bmod l}{r} \binom{l-n \bmod l - 2}{m-2-r} (\lfloor \frac{n}{l} \rfloor + 1)^r (\lfloor \frac{n}{l} \rfloor)^{m-2-r}$.

Adding the counts for the above three cases, we get the result. ∎

Let us consider an example to demonstrate the case when we have two i-values (equation numbers) from the same residue class in the selected system of m equations, but still the system is independent and hence solvable.

Example 2 *Assume that the secret key is of length 5 bytes. Let us consider* 16 *equations of the form $S_N[i] = f_i$, $0 \le i \le 15$. We would consider all possible sets of 5 equations chosen from the above 16 equations and then try to solve them. One such set would correspond to $i = 0, 1, 2, 3$ and* 13*. Let the corresponding $S_N[i]$ values be 246, 250, 47, 204 and 185*

respectively. Then we can form the following equations:

$$K[0] = 246 \quad (6)$$
$$K[0] + K[1] + (1 \cdot 2)/2 = 250 \quad (7)$$
$$K[0] + K[1] + K[2] + (2 \cdot 3)/2 = 47 \quad (8)$$
$$K[0] + K[1] + K[2] + K[3] + (3 \cdot 4)/2 = 204 \quad (9)$$
$$K[0] + \ldots + K[13] + (13 \cdot 14)/2 = 185 \quad (10)$$

From the first four equations, we readily get $K[0] = 246, K[1] = 3, K[2] = 51$ *and* $K[3] = 154$. *Since the key is 5 bytes long,* $K[5] = K[0], \ldots, K[9] = K[4], K[10] = K[0], \ldots, K[13] = K[3]$. *Denoting the sum of the key bytes* $K[0] + \ldots + K[4]$ *by* s, *we can rewrite equation (10) as:*

$$2s + K[0] + K[1] + K[2] + K[3] + 91 = 185 \quad (11)$$

Subtracting (9) from (11), and solving for s, *we get* $s = 76$, *i.e.,*

$$K[0] + K[1] + K[2] + K[3] + K[4] = 76 \quad (12)$$

Subtracting (9) from (12), we get $K[4] = 134$.

We now present the general algorithm for recovering the secret key bytes from the final permutation obtained after the completion of the KSA.

Algorithm RecoverKey

Inputs:
1. The final permutation bytes: $S_N[i]$, $0 \leq i \leq N-1$.
2. Number of key bytes: l.
3. Number of key bytes to be solved from equations: m.
4. Number of equations to be tried: n.

Output:
The recovered key bytes $K[0], K[1], \ldots, K[l-1]$, if they are found. Otherwise, the algorithm halts after trying all the $|EI_m|$ systems of m independent equations.

Steps:
1. Form n equations $S_N[i] = f_i$, $i = 0, 1, \ldots, n-1$.
2. Form the set EI_m, the set of all m-tuples of indices $\{i_1, i_2, \ldots, i_m\}$, each corresponding to one independent system of m equations (using Theorem 2).
3. For each system $\{i_q : q = 1, 2, \ldots, m\} \in EI_m$ do
    - 3.1. Arbitrarily select any m variables present in the system;
    - 3.2. For each possible assignment of the remaining $l - m$ variables do
        - 3.2.1. Solve for the m variables;
        - 3.2.2. Run the KSA with the solved key;
        - 3.2.3. If the permutation obtained after the KSA is the same as the given S_N, then the recovered key is the correct one.

If one does not use the independence criteria (Theorem 2), all $\binom{n}{m}$ sets of equations need to be checked. However, the number of independent systems is $|EI_m|$, which is

much smaller than $\binom{n}{m}$. Table 3 shows that $|EI_m| < \frac{1}{2}\binom{n}{m}$ for most values of l, n, and m. Thus, the independence criteria in step 2 reduces the number of iterations in step 3 by a substantial factor.

The following Theorem quantifies the amount of savings in the time required to recover the keys due to our algorithm over exhaustive key search. We consider that solving a set of equations takes constant time as the number of variables is at most 16 (typically RC4 keys are 5 to 16 bytes long).

Theorem 3 *Suppose we want to recover $l - m$ key bytes by exhaustive search and the remaining m key bytes by solving n equations using the RecoverKey algorithm. Then the improvement over the exhaustive key search will be by a factor of $\frac{2^{8m}}{|EI_m|}$, independent of the number of key bytes l, where $|EI_m|$ is given by Theorem 2.*

Proof: The RecoverKey algorithm exhaustively searches $l-m$ key bytes for at most $|EI_m|$ times, giving a search complexity $|EI_m| \cdot 2^{8(l-m)}$. Whereas, the brute force search for all the l key bytes would require a search complexity 2^{8l}. So we have an improvement by a factor $\frac{2^{8l}}{|EI_m| \cdot 2^{8(l-m)}} = \frac{2^{8m}}{|EI_m|}$ ■

Next, we estimate what is the probability of getting a set of independent correct equations when we run the above algorithm.

Theorem 4 *Suppose that we are given the system of equations $S_N[i] = f_i$, $i = 0, 1, \ldots, n-1$. Let c be the number of independent correct equations. Then*

$$P(c \geq m) = \sum_{t=m}^{n} \sum_{\{i_1, i_2, \ldots, i_t\} \in EI_t} p(i_1, i_2, \ldots, i_t),$$

where EI_t is the collection of the indices $\{i_1, i_2, \ldots, i_t\}$ corresponding to all sets of t independent equations, and $p(i_1, i_2, \ldots, i_t)$ is the joint probability that the t equations corresponding to the indices $\{i_1, i_2, \ldots, i_t\}$ are correct and the other $n - t$ equations corresponding to the indices $\{0, 1, \ldots, n-1\} \setminus \{i_1, i_2, \ldots, i_t\}$ are incorrect.

Proof: We need to sum $|EI_t|$ number of terms of the form $p(i_1, i_2, \ldots, i_t)$ to get the probability that exactly t equations are correct, i.e., $P(c = t) = \sum_{\{i_1, i_2, \ldots, i_t\} \in EI_t} p(i_1, i_2, \ldots, i_t)$.

Hence, $P(c \geq m) = \sum_{t=m}^{n} P(c = t) = \sum_{t=m}^{n} \sum_{\{i_1, i_2, \ldots, i_t\} \in EI_t} p(i_1, i_2, \ldots, i_t)$. ■

Note that $P(c \geq m)$ gives the success probability with which one can recover the secret key from the permutation after the KSA.

As the events $(S_N[i] = f_i)$ are not independent for different i's, theoretically presenting the formulae for the joint probability $p(i_1, i_2, \ldots, i_t)$ seems to be extremely tedious. In the following table, we provide experimental results on the probability of having at least m independent correct equations, when the first n equations $S_N[i] = f_i, 0 \leq i \leq n-1$ are considered for the RecoverKey algorithm for different values of n, m, and the key length

l, satisfying $m \leq l \leq n$. For each probability calculation, the complete KSA was repeated a million times, each time with a randomly chosen key. We also compare the values of the exhaustive search complexity and the reduction due to our algorithm. Let us denote $d = \lceil \log_2 |EI_m| \rceil$, $e = 8l$, and $g = \lceil \log_2(|EI_m| \cdot 2^{8(l-m)}) \rceil = \lceil \log_2 |EI_m| \rceil + 8(l-m) = d + 8(l-m)$. Then the time complexity of exhaustive search is $O(2^e)$ and that of the RecoverKey algorithm is $O(2^g)$. Thus, the reduction in search complexity due to our algorithm is by a factor $O(2^{e-g})$. One may note from Table 3 that by suitably choosing the parameters one can achieve the search complexity $O(2^{\frac{e}{2}}) = O(2^{4l})$, which is the square root of the exhaustive key search complexity. The results in Table 3 clearly show that the probabilities (i.e., the empirical value of $P(c \geq m)$) in most of the cases are greater than 10%. The keys, that can be recovered from the permutation after the KSA using the RecoverKey algorithm, may be considered as weak keys in RC4.

l	n	m	$\binom{n}{m}$	$\lvert EI_m \rvert$	d	e	g	$e-g$	$P(c \geq m)$
5	16	5	4368	810	10	40	10	30	0.250
5	24	5	42504	7500	13	40	13	27	0.385
8	16	6	8008	3472	12	64	28	36	0.273
8	20	7	77520	13068	14	64	22	42	0.158
8	40	8	76904685	1484375	21	64	21	43	0.092
10	16	7	11440	5840	13	80	37	43	0.166
10	24	8	735471	130248	17	80	33	47	0.162
10	48	9	1677106640	58125000	26	80	34	46	0.107
12	24	8	735471	274560	19	96	51	45	0.241
12	24	9	1307504	281600	19	96	43	53	0.116
16	24	9	1307504	721800	20	128	76	52	0.185
16	32	10	64512240	19731712	25	128	73	55	0.160
16	32	11	129024480	24321024	25	128	65	63	0.086
16	40	12	5586853480	367105284	29	128	61	67	0.050

Table 3: Running the RecoverKey algorithm with different parameters

4 From RC4 State Information to the Permutation after KSA

This section focusses on the PRGA of RC4. Consider that τ many keystream output bytes are generated in the PRGA and the current permutation is S_C. Further we take the current value of j as j_C. These values constitute the state information of RC4. Note that we only need to get the value of τ, and not the keystream output bytes themselves. From τ, we can get the current value of i which we denote as i_C. In the first round of PRGA, i starts from 1, and thereafter i is updated by $(i+1) \bmod N$ in every step. Hence

$i_C = \tau \bmod N$. Assuming j_C to be known, Algorithm PRGAreverse stated below retrieves the permutation after the KSA from the permutation after τ many rounds of the PRGA. Note that all subtractions except $r = r - 1$ in Algorithm PRGAreverse are modulo N operations.

Algorithm PRGA
Initialization:
$i = 0$;
$j = 0$;
Output Keystream Generation Loop:
$i = i + 1$;
$j = j + S[i]$;
Swap($S[i]$, $S[j]$);
$t = S[i] + S[j]$;
Output $z = S[t]$;

Algorithm PRGAreverse
Initialization:
$i = i_C; j = j_C; S = S_C$;
$r = \tau$;
Do
Swap($S[i]$, $S[j]$);
$j = j - S[i]$;
$i = i - 1$;
$r = r - 1$;
While $r > 0$;

Once the permutation after the KSA is retrieved, Algorithm RecoverKey (Section 3) can be used to efficiently get back the secret key with constant success probability.

5 Intrinsic Weakness of Shuffle-exchange Type KSA

In the KSA of RC4, i is incremented by one and j is updated pseudorandomly by the rule $j = j + S[i] + K[i]$. Here, the increment of j is a function of the permutation and the secret key. One may expect that the correlation between the secret key and the final permutation can be removed by modifying the updation rule for j. Here we show that for a certain class of rules of this type, where j across different rounds is uniformly randomly distributed, there will always exist significant bias of the final permutation after the KSA towards some combination of the secret key bytes with significant probability.

Using the notation of Section 2, we can model the updation of j in the KSA as an arbitrary function u of (a) the current values of i, j, (b) the i-th and j-th permutation bytes from the previous round, and (c) the i-th and j-th key bytes, i.e., $j_{i+1} = u(i, j_i, S_i[i], S_i[j_i], K[i], K[j_i])$. For subsequent reference, let us call the KSA with this generalized updation rule as GKSA.

Lemma 2 *Assume that during the GKSA rounds, the index j takes its values uniformly at random. Then, one can always construct functions $h_i(S_0, K)$, which depends only on i, the secret key bytes and the initial permutation, and probabilities π_i, which depends only on i and N, such that $P(j_{i+1} = h_i(S_0, K)) = \pi_i$, for initial values of i.*

Proof: We will show how to construct the recursive functions $h_i(S_0, K)$ and probabilities π_i by induction on i.

- *Base Case*: Initially, before the beginning of round 0, $j_0 = 0$. Hence, in round $i = 0$, we would have $j_1 = u(0, 0, S_0[0], S_0[0], K[0], K[0]) = h_0(S_0, K)$ (say), with probability $\pi_0 = 1$.

- *Inductive Case*: Suppose, $P(j_i = h_{i-1}(S_0, K)) = \pi_{i-1}$ (*inductive hypothesis*). We know that $j_{i+1} = u(i, j_i, S_i[i], S_i[j_i], K[i], K[j_i])$. In the right hand side of this equality, all occurrences of $S_i[i]$ can be replaced by $S_0[i]$ with probability $(\frac{N-1}{N})^i$, which is the probability of index i not being involved in any swap in the rounds 0 through i. Also, due to swap in the previous round (i.e., round $i-1$), we have $S_i[j_i] = S_{i-1}[i-1]$, which again can be replaced by $S_0[i-1]$ with probability $(\frac{N-1}{N})^{i-1}$. Finally, all occurrences of j_i can be replaced by $h_{i-1}(S_0, K)$ with probability π_{i-1} (using the inductive hypothesis). Thus, j_{i+1} equals $u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)])$ with some probability π_i which can be calculated as a function of i, N, and π_{i-1}, depending on the occurrence or non-occurrence of various terms in u. Denoting $h_i(S_0, K) = u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)])$, we get the result. ■

Theorem 5 *Assume that during the GKSA rounds, the index j takes its values uniformly at random. Then, one can always construct functions $f_i(S_0, K)$, which depends only on i, the secret key bytes and the initial permutation, such that $P(S_N[i] = f_i(S_0, K)) = (\frac{N-1}{N})^{N-1} \cdot (\frac{N-i}{N}) \cdot \pi_i$, for initial values of i.*

Proof: After the swap in round i, $S_{i+1}[i] = S_i[j_{i+1}]$, which is equal to $S_0[j_{i+1}]$ with probability $(\frac{N-i}{N}) \cdot (\frac{N-1}{N})^i$, assuming that j_{i+1} has not been involved in any swap during the previous rounds (similar to the proof of Theorem 1). From Lemma 2, we have $P(j_{i+1} = h_i(S_0, K)) = \pi_i$ and hence $P(S_{i+1}[i] = S_0[h_i(S_0, K)]) = \pi_i \cdot (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^i$. Denote $S_0[h_i(S_0, K)]$ by $f_i(S_0, K)$. Once the value of $S_{i+1}[i]$ is set to $f_i(S_0, K)$ in the i-th round, this value remains the same throughout the subsequent rounds of the GKSA iff i is not involved in any more swaps, and this happens with probability $(\frac{N-1}{N})^{N-i-1}$. Hence, after the completion of the GKSA, we have $P(S_N[i] = f_i(S_0, K)) = \pi_i \cdot (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^i \cdot (\frac{N-1}{N})^{N-i-1} = (\frac{N-1}{N})^{N-1} \cdot (\frac{N-i}{N}) \cdot \pi_i$. ■

Next, we discuss some special cases of the updation rule u as illustrative examples of how to construct the functions f_is and the probabilities π_is. In all the following cases, we assume S_0 to be an identity permutation and hence $f_i(S_0, K)$ is the same as $h_i(S_0, K)$.

Example 3 *Consider the KSA of RC4, where*

$$u(i, j_i, S_i[i], S_i[j_i], K[i], K[j_i]) = j_i + S_i[i] + K[i].$$

We have $h_0(S_0, K) = u(0, 0, S_0[0], S_0[0], K[0], K[0]) = 0 + 0 + K[0] = K[0]$. Moreover, $\pi_0 = P(j_1 = h_0(S_0, K)) = 1$. For $i \geq 1$, $h_i(S_0, K) = u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)]) = h_{i-1}(S_0, K) + S_0[i] + K[i] = h_{i-1}(S_0, K) + i + K[i]$. Solving the recurrence, we get $h_i(S_0, K) = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]$. Since $S_i[i]$ is replaced by $S_0[i]$, and j_i is replaced by $h_{i-1}(S_0, K)$, we have $\pi_i = (\frac{N-1}{N})^i \cdot \pi_{i-1}$. Solving this recurrence, we get $\pi_i = \prod_{x=0}^{i} (\frac{N-1}{N})^x = (\frac{N-1}{N})^{\frac{i(i+1)}{2}}$. These expressions coincide with those in Corollary 1 and 2.

Example 4 *Consider the updation rule*

$$u(i, j_i, S_i[i], S_i[j_i], K[i], K[j_i]) = j_i + S_i[j_i] + K[j_i].$$

Here, $h_0(S_0, K) = u(0, 0, S_0[0], S_0[0], K[0], K[0]) = 0 + 0 + K[0] = K[0]$ *and* $\pi_0 = P(j_1 = h_0(S_0, K)) = 1$. *For* $i \geq 1$,
$h_i(S_0, K) = u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)]) = h_{i-1}(S_0, K) + S_0[i-1] + K[h_{i-1}(S_0, K)] = h_{i-1}(S_0, K) + (i-1) + K[h_{i-1}(S_0, K)]$. *Since* $S_{i-1}[i-1]$ *and* j_i *are respectively replaced by* $S_0[i-1]$ *and* $h_{i-1}(S_0, K)$, *we would have* $\pi_i = (\frac{N-1}{N})^{i-1} \cdot \pi_{i-1}$. *Solving this recurrence, we get* $\pi_i = \prod_{x=1}^{i} (\frac{N-1}{N})^{x-1} = (\frac{N-1}{N})^{\frac{i(i-1)}{2}}$.

Example 5 *As another example, suppose*

$$u(i, j_i, S_i[i], S_i[j_i], K[i], K[j_i]) = j_i + i \cdot S_i[j_i] + K[j_i].$$

As before, $h_0(S_0, K) = u(0, 0, S_0[0], S_0[0], K[0], K[0]) = 0 + 0 \cdot S[0] + K[0] = 0 + 0 + K[0] = K[0]$ *and* $\pi_0 = P(j_1 = h_0(S_0, K)) = 1$. *For* $i \geq 1$, $h_i(S_0, K) = u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)]) = h_{i-1}(S_0, K)]) + i \cdot S_0[i-1] + K[h_{i-1}(S_0, K)] = h_{i-1}(S_0, K)]) + i \cdot (i-1) + K[h_{i-1}(S_0, K)]$. *As in the previous example, here also the recurrence relation for the probabilities is* $\pi_i = (\frac{N-1}{N})^{i-1} \cdot \pi_{i-1}$, *whose solution is* $\pi_i = \prod_{x=1}^{i} (\frac{N-1}{N})^{x-1} = (\frac{N-1}{N})^{\frac{i(i-1)}{2}}$.

Our results show that the design of RC4 KSA cannot achieve further security by changing the updation rule $j = j + S[i] + K[i]$ by any rule from a large class that we present. In [9, Section 4.2], an idealized model of RC4 KSA has been suggested where j has to be chosen at random from $[0, \ldots, N-1]$. Our result shows that if j is a deterministic function of the secret key, then this model cannot be termed "idealized" due to the weakness discussed above.

6 Conclusion

We theoretically prove Roos's [13] observation about the correlation between the secret key bytes and the final permutation bytes after the KSA. In addition, we show how to use this result to recover the secret key bytes from the RC4 state at any stage after the KSA (i.e., after arbitrary number of rounds of the PRGA) with constant probability of success in less than the square root of the time required for exhaustive key search. Since the state (which includes the permutation and the indices) is in general not observable, this does not immediately pose an additional threat to the security of RC4. However, for an ideal stream cipher, no information about the secret key should be revealed even if the complete state of the system is known at any instant. Our work clearly points out intrinsic structural weaknesses of RC4 and certain generalizations of it.

References

[1] S. R. Fluhrer and D. A. McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. FSE 2000, pages 19-30, vol. 1978, Lecture Notes in Computer Science, Springer-Verlag.

[2] S. R. Fluhrer, I. Mantin and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. Selected Areas in Cryptography 2001, pages 1-24, vol. 2259, Lecture Notes in Computer Science, Springer-Verlag.

[3] J. Golic. Linear statistical weakness of alleged RC4 keystream generator. EUROCRYPT 1997, pages 226-238, vol. 1233, Lecture Notes in Computer Science, Springer-Verlag.

[4] LAN/MAN Standard Committee. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1999 edition. IEEE standard 802.11, 1999.

[5] I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. FSE 2001, pages 152-164, vol. 2355, Lecture Notes in Computer Science, Springer-Verlag.

[6] I. Mantin. A Practical Attack on the Fixed RC4 in the WEP Mode. ASIACRYPT 2005, pages 395-411, volume 3788, Lecture Notes in Computer Science, Springer-Verlag.

[7] I. Mantin. Predicting and Distinguishing Attacks on RC4 Keystream Generator. EUROCRYPT 2005, pages 491-506, vol. 3494, Lecture Notes in Computer Science, Springer-Verlag.

[8] I. Mantin. Analysis of the strem cipher RC4. Master's Thesis, The Weizmann Institute of Science, Israel, 2001.

[9] I. Mironov. (Not So) Random Shuffles of RC4. CRYPTO 2002, pages 304-319, vol. 2442, Lecture Notes in Computer Science, Springer-Verlag.

[10] G. Paul, S. Rathi and S. Maitra. On Non-negligible Bias of the First Output Byte of RC4 towards the First Three Bytes of the Secret Key. Proceedings of the International Workshop on Coding and Cryptography 2007, pages 285-294.

[11] S. Paul and B. Preneel. Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator. INDOCRYPT 2003, pages 52-67, vol. 2904, Lecture Notes in Computer Science, Springer-Verlag.

[12] S. Paul and B. Preneel. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. FSE 2004, pages 245-259, vol. 3017, Lecture Notes in Computer Science, Springer-Verlag.

[13] A. Roos. A class of weak keys in the RC4 stream cipher. Two posts in sci.crypt, message-id 43u1eh$1j3@hermes.is.co.za and 44ebge$llf@hermes.is.co.za, 1995. Available at http://marcel.wanda.ch/Archive/WeakKeys.

[14] D. Wagner. My RC4 weak keys.
Post in sci.crypt, message-id 447o1l$cbj@cnn.Princeton.EDU, 26 September, 1995. Available at http://www.cs.berkeley.edu/∼daw/my-posts/my-rc4-weak-keys.