Permutation After RC4 Key Scheduling Reveals the Secret Key

Goutam Paul[1] and Subhamoy Maitra[2]

[1] Department of Computer Science and Engineering,
Jadavpur University, Kolkata 700 032, India
goutam_paul@cse.jdvu.ac.in
[2] Applied Statistics Unit, Indian Statistical Institute,
203, B T Road, Kolkata 700 108, India
subho@isical.ac.in

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}. Further, a generalization of the RC4 KSA is analyzed corresponding to a class of update functions of the indices involved in the swaps. This reveals an inherent weakness of shuffle-exchange kind of key scheduling.

Keywords: Bias, Cryptanalysis, 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, 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.

C. Adams, A. Miri, and M. Wiener (Eds.): SAC 2007, LNCS 4876, pp. 360–377, 2007.

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 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	**Algorithm PRGA**
Initialization:	*Initialization*:
For $i = 0, \ldots, N-1$	$i = j = 0$;
$S[i] = i$;	*Output Keystream Generation Loop*:
$j = 0$;	$i = i + 1$;
Scrambling:	$j = j + S[i]$;
For $i = 0, \ldots, N-1$	Swap($S[i]$, $S[j]$);
$j = (j + S[i] + K[i])$;	$t = S[i] + S[j]$;
Swap($S[i], S[j]$);	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 update 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 some combination of the secret key bytes. Such biases were observed by Roos in [15] for the first time. It has been noted in [15] 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 [15] and experimental values have been provided as in Table 1 below.

Table 1. The probabilities experimentally observed by Roos [15]

i	$P(S[i] = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x])$
0-15	.370 .368 .362 .358 .349 .340 .330 .322 .309 .298 .285 .275 .260 .245 .229 .216
16-31	.203 .189 .173 .161 .147 .135 .124 .112 .101 .090 .082 .074 .064 .057 .051 .044
32-47	.039 .035 .030 .026 .023 .020 .017 .014 .013 .012 .010 .009 .008 .007 .006 .006

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 [15] 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, Lemma 2 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 retrieve the key bytes in time much less than the exhaustive key search. For a secret key of size $8l$ bits ($40 \leq 8l \leq 128$), the key can be recovered in $O(2^{\frac{8l}{2}})$ effort with a 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 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.

Further, we consider the generalization of the RC4 KSA where the index j can be updated in different manners. In RC4 KSA, the update rule is $j = (j + S[i] + K[i])$. We show that for any arbitrary secret key and for a certain class of update functions (see Section 4) 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 update 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 pseudo-randomly.

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 [1,3,6,7,8,13,14]. 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 [6] and a bias in the equality of the first two output bytes has been shown in [14]. In [10], 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 [15,17] and several classes of weak keys had been identified. Recently, a more general theoretical study has been performed in [11] which includes the observations of [15]. The work [11] 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]. Subsequently, the work [5] improved [2]. In [9, 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 the round with i as the deterministic index (we call it round $i+1$), $0 \le i \le N-1$. Then S_N would be the final permutation after the complete KSA.

We now prove a general formula (Theorem 1) that estimates the probabilities 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. Note that though j is updated using a deterministic formula, it is a linear function of the pseudo-random secret key bytes, and is therefore itself pseudo-random. If the secret key generator produces the secret keys uniformly at random, which is a reasonable assumption, then the distribution of j will also be uniform.

The proof of Theorem 1 depends on Lemma 1 and Lemma 2 which we prove below first.

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

Proof. One contribution towards the event $E : (j_{i+1} = \sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x])$ is approximately $(\frac{N-1}{N})^{\frac{i(i+1)}{2}}$. This part is due to the association based on the recursive updates of j and can be proved 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 $1 = (\frac{N-1}{N})^{\frac{0(0+1)}{2}}$. Hence, the result holds for the base case.
- *Inductive Case*: Suppose, that the result holds for the first i rounds, when the deterministic index takes its values from 0 to $i-1$, $i \geq 1$. Now, for the $(i+1)$-th round, we would have $j_{i+1} = j_i + S_i[i] + K[i]$. Thus, j_{i+1} can equal $\sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]$, if $j_i = \sum_{x=0}^{i-1} S_0[x] + \sum_{x=0}^{i-1} K[x]$ and $S_i[i] = S_0[i]$.

  By *inductive hypothesis*, we get $P(j_i = \sum_{x=0}^{i-1} S_0[x] + \sum_{x=0}^{i-1} K[x]) \approx (\frac{N-1}{N})^{\frac{i(i-1)}{2}}$.

  Further, $S_i[i]$ remains the same as $S_0[i]$, if it has not been involved in any swap during the previous rounds, i.e., if any of the values $j_1, j_2, \ldots, j_i$ has not hit the index i, the probability of which is $(\frac{N-1}{N})^i$. Thus, the probability that the event E occurs along the above recursive path is $\approx (\frac{N-1}{N})^{\frac{i(i-1)}{2}} \cdot (\frac{N-1}{N})^i = (\frac{N-1}{N})^{\frac{i(i+1)}{2}}$.

  A second contribution towards the event E is due to random association when the above recursive path is not followed. This probability is approximately $\left(1-(\frac{N-1}{N})^{\frac{i(i+1)}{2}}\right) \cdot \frac{1}{N}$. Adding these two contributions, we get the total probability $\approx (\frac{N-1}{N})^{\frac{i(i+1)}{2}} + \left(1-(\frac{N-1}{N})^{\frac{i(i+1)}{2}}\right) \cdot \frac{1}{N} = (1-\frac{1}{N}) \cdot (\frac{N-1}{N})^{\frac{i(i+1)}{2}} + \frac{1}{N} = (\frac{N-1}{N})^{1+\frac{i(i+1)}{2}} + \frac{1}{N}$. □

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

Lemma 2. *Assume that during the KSA rounds, the index j takes its values from $\{0, 1, \ldots, N-1\}$ uniformly at random. Then,* $P(S_N[i] = S_0[j_{i+1}]) \approx (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N-1}$, $0 \le i \le N-1$.

Proof. During the swap in round $i+1$, $S_{i+1}[i]$ is assigned the value of $S_i[j_{i+1}]$. Now, the index j_{i+1} is not involved in any swap during the previous i many rounds, if it is not touched by the indices $\{0, 1, \ldots, i-1\}$, the probability of which is $(\frac{N-i}{N})$, as well as if it is not touched by the indices $\{j_1, j_2, \ldots, j_i\}$, the probability of which is $(\frac{N-1}{N})^i$. Hence, $P(S_{i+1}[i] = S_0[j_{i+1}]) \approx (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^i$. After round $i+1$, index i is not touched by any of the subsequent $N-1-i$ many j values with probability $(\frac{N-1}{N})^{N-1-i}$. Hence, $P(S_N[i] = S_0[j_{i+1}]) \approx (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^i \cdot (\frac{N-1}{N})^{N-1-i} = (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N-1}$. □

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

Proof. $S_N[i]$ can equal $S_0\Big[\sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]\Big]$ in two ways. One way is that $j_{i+1} = \sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]$ following the recursive path as in the proof of Lemma 1, and $S_N[i] = S_0[j_{i+1}]$. Combining the results of Lemma 1 and Lemma 2, we get the contribution of this part $\approx (\frac{N-1}{N})^{\frac{i(i+1)}{2}} \cdot (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N-1} = (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+(N-1)]}$. Another way is that neither of the above events happen and still $S_N[i]$ equals $S_0\Big[\sum_{x=0}^{i} S_0[x] + \sum_{x=0}^{i} K[x]\Big]$ due to random association. The contribution of this second part is approximately $\Big(1 - (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+(N-1)]}\Big) \cdot \frac{1}{N}$. Adding these two contributions, we get the total probability $\approx (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+(N-1)]} + \Big(1 - (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+(N-1)]}\Big) \cdot \frac{1}{N} = (1 - \frac{1}{N}) \cdot (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+(N-1)]} + \frac{1}{N} = (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{[\frac{i(i+1)}{2}+N]} + \frac{1}{N}$. □

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

Table 2. The probabilities following Corollary 2

i	$P(S[i] = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x])$															
0-15	.371	.368	.364	.358	.351	.343	.334	.324	.313	.301	.288	.275	.262	.248	.234	.220
16-31	.206	.192	.179	.165	.153	.140	.129	.117	.107	.097	.087	.079	.071	.063	.056	.050
32-47	.045	.039	.035	.031	.027	.024	.021	.019	.016	.015	.013	.011	.010	.009	.008	.008

In the following table (Table 2) we list the theoretical values of the probabilities to compare with the experimental values provided in [15] and summarized in our Table 1.

After the index 48 and onwards, both the theoretical as well as 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.

3 Recovering the Secret Key from the Permutation After the KSA

In this section, we discuss how to get 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 $S_N[i] = f_i$, $0 \le i \le 15$, 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 related to the joint probability of $S_N[i] = f_i$ for the set of chosen i-values. Note that we do not need the assumption that the majority of the equations are correct. Whether indeed the equations selected are correct or not can be cross-checked by running the KSA again. Moreover, empirical results show that in a significant proportion of the cases we get enough correct equations to solve for the key.

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. Since the equations are triangular in form, solving each set of 5 equations would take approximately $5^2 = 25$ (times a small constant) $< 2^5$ many additions/subtractions. Hence the improvement over exhaustive search is almost by a factor of $\frac{2^{40}}{2^{13}.2^5} = 2^{22}$.

From Corollary 2, we get how $S_N[i]$ is biased to different combinations of the key bytes. 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 of size l bytes. Suppose we want to recover exactly m out of the l 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 $\binom{n}{m}$ systems of m equations is 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, $m \leq n \leq N$, of the form $S_N[i] = f_i$ involving the final permutation bytes, 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*

$$|EI_m| = \sum_{r=0}^{m} \binom{n \bmod l}{r} \binom{l - n \bmod l}{m-r} (\lfloor \tfrac{n}{l} \rfloor + 1)^r (\lfloor \tfrac{n}{l} \rfloor)^{m-r}$$

$$+ \binom{n \bmod l}{1} \binom{\lfloor \frac{n}{l} \rfloor + 1}{2} \sum_{r=0}^{m-2} \binom{n \bmod l - 1}{r} \binom{l - n \bmod l - 1}{m-2-r} (\lfloor \tfrac{n}{l} \rfloor + 1)^r (\lfloor \tfrac{n}{l} \rfloor)^{m-2-r}$$

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

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

Proof. (*Part 1*) First, we will show that any one of the conditions (i) and (ii) is sufficient. Suppose that the condition (i) holds, i.e., $i_q \bmod l$ $(1 \leq q \leq m)$ yields m distinct values. Then each equation involves a different key byte as a variable, and hence the system is independent. Now, suppose that the condition (ii) holds. Then there exists exactly one pair $x, y \in \{1, \ldots, m\}$, $x \neq y$, where $i_x = i_y \bmod l$. 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 $K[l-1]$, since $l - 1 \notin \{i_1 \bmod l, i_2 \bmod l, \ldots, i_m \bmod l\}$. Thus, the resulting system is independent.

Next, we are going to show that the conditions are necessary. Suppose that neither condition (i) nor condition (ii) holds. Then either we will have a triplet x, y, z such that $i_x = i_y = i_z = \bmod l$, or we will have a pair x, y with $i_x = i_y \bmod l$ and $l - 1 \in \{i_1 \bmod l, i_2 \bmod l, \ldots, i_m \bmod l\}$. In the first case, subtracting two of the equations from the third one would result in two equations involving s and the same key bytes as variables. Thus the resulting system will not be independent. In the second case, subtracting one equation from the other will result in an equation which is dependent on the equation involving the key byte $K[l-1]$.

(*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:

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

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.

<u>Case I:</u> *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 $\binom{l - n \bmod l}{m-r}$ 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 $\sum_{r=0}^{m} \binom{n \bmod l}{r} \binom{l - n \bmod l}{m-r} (\lfloor \frac{n}{l} \rfloor + 1)^r (\lfloor \frac{n}{l} \rfloor)^{m-r}$.

<u>Case II</u>: *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 $m - 2 - r$ 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 $\binom{n \bmod l}{1} \binom{\lfloor \frac{n}{l} \rfloor + 1}{2} \sum_{r=0}^{m-2} \binom{n \bmod l - 1}{r} \binom{l - n \bmod l - 1}{m-2-r} (\lfloor \frac{n}{l} \rfloor + 1)^r (\lfloor \frac{n}{l} \rfloor)^{m-2-r}$.

<u>Case III</u>: *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. □

Proposition 1. *Given n and m, it takes $O(m^2 \cdot \binom{n}{m})$ time to generate the set EI_m using Theorem 2.*

Proof. We need to check a total of $\binom{n}{m}$ many m tuples $\{i_1, i_2, \ldots, i_m\}$, and using the independence criteria of Theorem 2, it takes $O(m^2)$ amount of time to determine if each tuple belongs to EI_m or not. □

Proposition 2. *Suppose we have an independent system of equations of the form $S_N[i_q] = f_{i_q}$ involving the l key bytes as variables corresponding to the tuple $\{i_1, i_2, \ldots, i_m\}$, $0 \le i_q \le n-1$, $1 \le q \le m$, where $f_i = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]$. If* *there is one equation in the system involving* $s = \sum_{x=0}^{l-1} K[x]$*, then we would have at most* $\lfloor \frac{n}{l} \rfloor$ *many solutions for the key.*

Proof. If the coefficient of s is a, then by Linear Congruence Theorem [16], we would have at most $gcd(a, N)$ many solutions for s, each of which would give a different solution for the key. To find the maximum possible number of solutions, we need to find an upper bound of $gcd(a, N)$.

Since the key is of length l, the coefficient a of s would be $\lfloor \frac{i_s}{l} \rfloor$, where i_s is the i-value $\in \{i_1, i_2, \ldots, i_m\}$ corresponding to the equation involving s. Thus, $gcd(a, N) \leq a = \lfloor \frac{i_s}{l} \rfloor \leq \lfloor \frac{n}{l} \rfloor$. □

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 $S_N[i] = f_i$, $0 \leq i \leq 15$. 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:

$$K[0] = 246 \tag{6}$$

$$K[0] + K[1] + (1 \cdot 2)/2 = 250 \tag{7}$$

$$K[0] + K[1] + K[2] + (2 \cdot 3)/2 = 47 \tag{8}$$

$$K[0] + K[1] + K[2] + K[3] + (3 \cdot 4)/2 = 204 \tag{9}$$

$$K[0] + \ldots + K[13] + (13 \cdot 14)/2 = 185 \tag{10}$$

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:

$$2s + K[0] + K[1] + K[2] + K[3] + 91 = 185 \tag{11}$$

Subtracting (9) from (11), and solving for s, we get $s = 76$ or 204. Taking the value 76, we get

$$K[0] + K[1] + K[2] + K[3] + K[4] = 76 \tag{12}$$

Subtracting (9) from (12), we get $K[4] = 134$. $s = 204$ does not give the correct key, as can be verified by running the KSA and observing the permutation obtained.

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: $S_N[i]$, $0 \le i \le N-1$.
2. Number of key bytes: l.
3. Number of key bytes to be solved from equations: m $(\le l)$.
4. Number of equations to be tried: n $(\ge m)$.

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. For each distinct tuple $\{i_1, i_2, \ldots, i_m\}$, $0 \le i_q \le n-1$, $1 \le q \le m$ do
    1.1. If the tuple belongs to EI_m then do
        1.1.1 Arbitrarily select any m variables present in the system;
        1.1.2 For each possible assignment of the remaining $l-m$ variables do
            1.1.2.1 Solve for the m variables;
            1.1.2.2 Run the KSA with the solved key;
            1.1.2.3 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 1.1 reduces the number of iterations in step 1.1.2 by a substantial factor.

The following Theorem quantifies the amount of time required to recover the key due to our algorithm.

Theorem 3. *The time complexity of the RecoverKey algorithm is given by*

$$O\Big(m^2 \cdot \tbinom{n}{m} + m^2 \cdot |EI_m| \cdot \lfloor \tfrac{n}{l} \rfloor \cdot 2^{8(l-m)}\Big),$$

where $|EI_m|$ is given by Theorem 2.

Proof. According to Proposition 1, for a complete run of the algorithm, checking the condition at step 1.1 consumes a total of $O(m^2 \cdot \binom{n}{m})$ amount of time.

Further, the loop in step 1.1.2 undergoes $|EI_m|$ many iterations, each of which exhaustively searches $l-m$ many key bytes and solves a system of m equations. By Proposition 2, each system can yield at the most $O(\lfloor \frac{n}{l} \rfloor)$ many solutions for the key. Also, finding each solution involves $O(m^2)$ many addition/subtraction operations (the equations being traingular in form). Thus, the total time consumed by step 1.1.2 for a complete run would be $O(m^2 \cdot |EI_m| \cdot \lfloor \frac{n}{l} \rfloor \cdot 2^{8(l-m)})$.

Hence, the time complexity is given by $O\Big(m^2 \cdot \binom{n}{m} + m^2 \cdot |EI_m| \cdot \lfloor \frac{n}{l} \rfloor \cdot 2^{8(l-m)}\Big)$. □

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

Proposition 3. *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 $e = \lceil \log_2(m^2 \cdot \binom{n}{m} + m^2 \cdot |EI_m| \cdot \lfloor \frac{n}{l} \rfloor \cdot 2^{8(l-m)}) \rceil$. The time complexity of exhaustive search is $O(2^{8l})$ and that of the RecoverKey algorithm, according to Theorem 3, is given by $O(2^e)$. Thus, the reduction in search complexity due to our algorithm is by a factor $O(2^{8l-e})$. One may note from Table 3 that by suitably choosing the parameters one can achieve the search complexity $O(2^{\frac{8l}{2}}) = O(2^{4l})$, 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%. However, the algorithm does not use the probabilities to recover the key. For certain keys the algorithm will be able to recover the keys and for certain other keys the algorithm will not be able to recover the keys by solving the equations. The success probability can be interpreted as the proportion of keys for which the algorithm will be able to successfully recover the key. The keys, that can be recovered from the permutation after the KSA using the RecoverKey algorithm, may be considered as weak keys in RC4.

It is important to note that the permutation is biased towards the secret key not only after the completion of the KSA, rather the bias persists in every round

Table 3. Running the RecoverKey algorithm with different parameters

l	n	m	$\binom{n}{m}$	$\lvert EI_m \rvert$	$8l$	e	$8l-e$	$P(c \geq m)$
5	16	5	4368	810	40	18	22	0.250
5	24	5	42504	7500	40	21	19	0.385
8	16	6	8008	3472	64	34	30	0.273
8	20	7	77520	13068	64	29	35	0.158
8	40	8	76904685	1484375	64	33	31	0.092
10	16	7	11440	5840	80	43	37	0.166
10	24	8	735471	130248	80	40	40	0.162
10	48	9	1677106640	58125000	80	43	37	0.107
12	24	8	735471	274560	96	58	38	0.241
12	24	9	1307504	281600	96	50	46	0.116
16	24	9	1307504	721800	128	60	68	0.185
16	32	10	64512240	19731712	128	63	65	0.160
16	32	11	129024480	24321024	128	64	64	0.086
16	40	12	5586853480	367105284	128	64	64	0.050

of the KSA. If we know the permutation at any stage of the KSA, we can use our key recovery technique to get back the secret key [12]. Moreover, the PRGA is exactly the same as the KSA with the starting value of i as 1 (instead of 0) and with $K[i]$ set to 0 for all i. Thus, if we know the RC4 state information at any round of PRGA, we can deterministically get back the permutation after the KSA [12] and thereby recover the secret key.

4 Intrinsic Weakness of Shuffle-Exchange Type KSA

In the KSA of RC4, i is incremented by one and j is updated pseudo-randomly 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 update 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. Though the proof technique is similar to that in Section 2, it may be noted that the analysis in the proofs here focus on the weakness of the particular "form" of RC4 KSA, and not on the exact quantity of the bias.

Using the notation of Section 2, we can model the update 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 update rule as GKSA.

Lemma 3. *Assume that during the GKSA rounds, the index j takes its values from $\{0, 1, \ldots, N-1\}$ 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 $P(j_{i+1} = h_i(S_0, K)) = (\frac{N-1}{N})\pi_i + \frac{1}{N}$, $0 \leq i \leq N-1$.*

Proof. By induction on i, we will show (i) how to construct the recursive functions $h_i(S_0, K)$ and probabilities π_i and (ii) that one contribution towards the event $(j_{i+1} = h_i(S_0, K))$ is π_i.

- *Base Case*: Initially, before the beginning of round 1, $j_0 = 0$. In round 1, we have $i = 0$ and hence $j_1 = u(0, 0, S_0[0], S_0[0], K[0], K[0]) = h_0(S_0, K)$ (say), with probability $\pi_0 = 1$.
- *Inductive Case*: Suppose, $P(j_i = h_{i-1}(S_0, K)) = \pi_{i-1}$, $i \geq 1$ (*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 previous i many rounds. Also, due to the swap in round i, 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 computed as a function of i, N, and π_{i-1}, depending on the occurrence or non-occurrence of various terms in u. If we denote $h_i(S_0, K) = u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)])$, then (i) and (ii) follow by induction.

  When the recursive path does not occur, then the event

  $$(j_{i+1} = u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)]))$$

  occurs due to random association with probability $(1 - \pi_i) \cdot \frac{1}{N}$. Adding the above two contributions, we get $P(j_{i+1} = h_i(S_0, K)) = \pi_i + (1 - \pi_i) \cdot \frac{1}{N} = (\frac{N-1}{N})\pi_i + \frac{1}{N}$. □

Theorem 4. *Assume that during the GKSA rounds, the index j takes its values from $\{0, 1, \ldots, N-1\}$ 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 $P(S_N[i] = f_i(S_0, K)) \approx (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^N \cdot \pi_i + \frac{1}{N}$, $0 \leq i \leq N-1$.*

Proof. We will show that $f_i(S_0, K) = S_0[h_i(S_0, K)]$ where the function h_is are given by Lemma 3.

Now, $S_N[i]$ can equal $S_0[h_i(S_0, K)]$ in two ways. One way is that $j_{i+1} = h_i(S_0, K))$ following the recursive path as in Lemma 3 and $S_N[i] = S_0[j_{i+1}]$.

Combining Lemma 3 and Lemma 2, we find the probability of this event to be approximately $(\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N-1} \cdot \pi_i$. Another way is that the above path is not followed and still $S_N[i] = S_0[h_i(S_0, K)]$ due to random association. The contribution of this part is approximately $\left(1-(\frac{N-i}{N})\cdot(\frac{N-1}{N})^{N-1}\cdot\pi_i\right)\cdot\frac{1}{N}$. Adding the above two contributions, we get the total probability $\approx (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N-1} \cdot \pi_i + \left(1 - (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N-1} \cdot \pi_i\right) \cdot \frac{1}{N} = (1 - \frac{1}{N}) \cdot (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N-1} \cdot \pi_i + \frac{1}{N} = (\frac{N-i}{N}) \cdot (\frac{N-1}{N})^{N} \cdot \pi_i + \frac{1}{N}$. □

Next, we discuss some special cases of the update rule u as illustrative examples of how to construct the functions f_is and the probabilities π_is using Lemma 3. 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 $i \geq 1$,

$$\begin{aligned} h_i(S_0, K) &= u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)]) \\ &= h_{i-1}(S_0, K) + S_0[i] + K[i] \\ &= h_{i-1}(S_0, K) + i + K[i]. \end{aligned}$$

Solving the recurrence, we get $h_i(S_0, K) = \frac{i(i+1)}{2} + \sum_{x=0}^{i} K[x]$. From the analysis in the proof of Lemma 3, we see that in the recurrence of h_i, $S_i[i]$ has been replaced by $S_0[i]$ and j_i has been replaced by $h_{i-1}(S_0, K)$. Hence, we would have $\pi_i = P(S_i[i] = S_0[i]) \cdot P(j_i = h_{i-1}(S_0, K)) = (\frac{N-1}{N})^i \cdot \pi_{i-1}$. 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.

Example 4. Consider the update 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 $i \geq 1$,

$$\begin{aligned} h_i(S_0, K) &= u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)]) \\ &= h_{i-1}(S_0, K) + S_0[i-1] + K[h_{i-1}(S_0, K)] \\ &= h_{i-1}(S_0, K) + (i-1) + K[h_{i-1}(S_0, K)]. \end{aligned}$$

From the analysis in the proof of Lemma 3, we see that in the recurrence of h_i, $S_{i-1}[i-1]$ and j_i are respectively replaced by $S_0[i-1]$ and $h_{i-1}(S_0, K)$. Thus, 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 $i \geq 1$,

$$\begin{aligned} h_i(S_0, K) &= u(i, h_{i-1}(S_0, K), S_0[i], S_0[i-1], K[i], K[h_{i-1}(S_0, K)]) \\ &= h_{i-1}(S_0, K)]) + i \cdot S_0[i-1] + K[h_{i-1}(S_0, K)] \\ &= h_{i-1}(S_0, K)]) + i \cdot (i-1) + K[h_{i-1}(S_0, K)]. \end{aligned}$$

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 update rule $j = j + S[i] + K[i]$ by any rule from a large class that we present.

5 Conclusion

We theoretically prove Roos's [15] 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 final permutation bytes with constant probability of success in less than the square root of the time required for exhaustive key search. Since the final permutation is in general not observable, this does not immediately pose an additional threat to the security of RC4. However, for a perfect key scheduling, any information about the secret key from the final permutation should not be revealed, as the final permutation may in turn leak information in the PRGA. Our work clearly points out an intrinsic structural weaknesses of RC4 KSA and its certain generalizations.

References

1. Fluhrer, S.R., McGrew, D.A.: Statistical Analysis of the Alleged RC4 Keystream Generator. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 19–30. Springer, Heidelberg (2001)
2. Fluhrer, S.R., Mantin, I., Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 1–24. Springer, Heidelberg (2001)
3. Golic, J.: Linear statistical weakness of alleged RC4 keystream generator. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 226–238. Springer, Heidelberg (1997)
4. LAN/MAN Standard Committee. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1999 edn., IEEE standard 802.11 (1999)
5. Klein, A.: Attacks on the RC4 stream cipher (February 27, 2006) (last accessed on June 27, 2007), available at `http://cage.ugent.be/~klein/RC4/`

6. Mantin, I., Shamir, A.: A Practical Attack on Broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer, Heidelberg (2002)
7. Mantin, I.: A Practical Attack on the Fixed RC4 in the WEP Mode. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 395–411. Springer, Heidelberg (2005)
8. Mantin, I.: Predicting and Distinguishing Attacks on RC4 Keystream Generator. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 491–506. Springer, Heidelberg (2005)
9. Mantin, I.: Analysis of the stream cipher RC4. Master's Thesis, The Weizmann Institute of Science, Israel (2001)
10. Mironov, I.: (Not So) Random Shuffles of RC4. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 304–319. Springer, Heidelberg (2002)
11. Paul, G., Rathi, S., Maitra, S.: On Non-negligible Bias of the First Output Byte of RC4 towards the First Three Bytes of the Secret Key. In: Proceedings of the International Workshop on Coding and Cryptography, pp. 285–294 (2007)
12. Paul, G., Maitra, S.: RC4 State Information at Any Stage Reveals the Secret Key. IACR Eprint Server, eprint.iacr.org, number 2007/208 (June 1, 2007)
13. Paul, S., Preneel, B.: Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 52–67. Springer, Heidelberg (2003)
14. Paul, S., Preneel, B.: A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 245–259. Springer, Heidelberg (2004)
15. Roos, A.: 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`
16. Silverman., J., Friendly, A.: Introduction to Number Theory, 2nd edn., p. 56. Prentice Hall, Englewood Cliffs (2001)
17. Wagner, D.: My RC4 weak keys. Post in sci.crypt, message-id 447o1l$cbj@cnn.Princeton.EDU (September 26, 1995), available at `http://www.cs.berkeley.edu/~daw/my-posts/my-rc4-weak-keys`