RC4 State Information at Any Stage Reveals the Secret Key*

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

Subhamoy Maitra
Applied Statistics Unit, Indian Statistical Institute,
Kolkata 700 108, India,
Email: 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 at any stage of the KSA are biased to the secret key. Theoretical proofs of these formulae have been left open since Roos' work (1995). Next, 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. We additionally show that each byte of S_N actually reveals secret key information. Looking at all the elements of the final permutation S_N and its inverse S_N^{-1}, the value of the hidden index j in each round of the KSA can be estimated from a "pair of values" in $0, \ldots, N-1$ with a constant probability of success $\pi = \frac{N-2}{N} \cdot (\frac{N-1}{N})^{N-1} + \frac{2}{N}$ (we get $\pi \approx 0.37$, for $N = 256$), which is significantly higher than the random association. Using the values of two consecutive j's, we estimate the y-th key byte from at most a "quadruple of values" in $0, \ldots, N-1$ with a probability > 0.12. As a secret key of l bytes is repeated at least $\lfloor \frac{N}{l} \rfloor$ times in RC4, these many quadruples can be accumulated to get each byte of the secret key with very high probability (e.g., 0.8 to close to 1) from a small set of values. Based on our

*This is a revised and substantially extended version of the paper [20] "Permutation after RC4 Key Scheduling Reveals the Secret Key", presented in the 14th Annual Workshop on Selected Areas in Cryptography, SAC 2007, August 16-17, Ottawa, Canada, LNCS (Springer) vol. 4876, pages 360-377. Sections 2.1, 2.2, 3.3 are similar to [20] with revision in Section 3.3.1. Rest of the technical contents in this version are new.

analysis of the key scheduling, we show that the secret key of RC4 can be recovered from the state information in a time much less than the exhaustive search with good probability.

Keywords: Bias, Cryptanalysis, Keystream, Key Leakage, Permutation, RC4, Stream Cipher.

1 Introduction

RC4 is one of the most widely used software stream ciphers. Apart from being used in network protocols such as SSL, TLS, WEP and WPA, the cipher finds applications in Microsoft Windows, Lotus Notes, Apple AOCE, Oracle Secure SQL etc. Though a variety of other stream ciphers have been proposed after RC4, it is still the most popular stream cipher algorithm due to its simplicity, ease of implementation, speed and efficiency. The algorithm can be stated in less than ten lines, yet after two decades of analysis its strengths and weaknesses are of great interest to the community.

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 storing one byte. Typically, $N = 256$. S is initialized as the identity permutation, i.e., $S[y] = y$ for $0 \leq y \leq N-1$. A secret key k 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 $K[y] = k[y \bmod l]$ for any y, $0 \leq y \leq N-1$, i.e., the key is repeated in the array K at key length boundaries.

The RC4 cipher has two components, namely, the Key Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA). The KSA turns the secret key K into a random 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.

```
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($S[i], S[j]$);
```

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

In this paper, we study both the KSA and the PRGA of RC4 in detail and find out results that have implications towards the security of RC4. In the proofs, whenever we replace the probability of a joint event by the product of the probabilities of the individual events, independence of the underlying events is implicitly assumed.

1.1 Outline of the Contribution

In Section 2.1 of this paper, the update of the permutation S in different rounds of the KSA is analyzed and it is theoretically proved that at any stage 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 [24] for the first time. The experimental values reported in [24] is provided in Table 1 below.

y	$P(S_N[y] = f_y)$															
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

Table 1: The probabilities experimentally observed by Roos [24].

The most likely value of the y-th element of the permutation after the KSA (denoted by S_N) for the first few values of y is given by $S_N[y] = f_y$, where

$$f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x].$$

Proof sketches of the above relation appears in [24, Section 2] and later in [17, Section 3.4.1], but the exact analytical form of $P(S_r[y] = f_y)$ has not been presented. Related analysis in this area has been performed independently in [27, Section 4] and [29, Section 3] around the same time of our work [20]. These efforts are mostly tuned to exploit the WEP scenario, where the secret key is used with IV. In our Theorem 1, we derive the expressions for the probabilities $P(S_r[y] = f_y)$ for all values of the index y in $[0, N-1]$ and for all rounds r, $1 \leq r \leq N$. We believe that this probability expression has not been explicitly presented earlier.

Note that Roos' observation [24] was about the final permutation after the KSA. We here theoretically prove with what probabilities the permutation bytes *at any stage of the KSA* are correlated with the secret key bytes. Thus, our results include Roos' observation as a special case. Roos [24] 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 2.1).

In Section 2.2, 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 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 permutation at every stage of the KSA 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.

In Section 2.3, we exploit all entries of both the permutations S_N and S_N^{-1} to gather information about the index j (we narrow it down to only two values in the range of $0, \ldots, N-1$ instead of N options) in each round of the KSA with a very good and constant probability of success (> 0.37). The estimates of the two consecutive pairs of j's give four possible values of a key byte with good probability. These results (Theorems 3 and 4) are the theoretical foundations of this section. Since each key is repeated at least $\lfloor \frac{N}{l} \rfloor$ times, using the above idea we can form a frequency table for each secret key byte (see Theorems 5, 6).

In Section 3, we present our work on recovering the secret key from RC4 permutation. *The cryptanalytic motivation for this work is discussed elaborately in the beginning of Section 3.* In Section 3.3, we use the biases of Section 2.1 to show that if the permutation at any stage of 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. Based on the theoretical results of Section 2.3, in Section 3.4 we present further ideas in this direction, which produce better complexities and success probabilities than those in Section 3.3.

Subsequent to our work in [20], other researchers have shown interest in the problem of recovering the secret key from RC4 permutation as evident from [2, 1]. In the similar direction, a bit-by-bit key recovery approach is discussed in [7]. We have also added to our initial work of [20] in Sections 2.3, 3.4 of the current paper. Part of the Sections 2.3, 3.4 of our current work has some common features with [1] which has been independently studied at the same time and this is acknowledged in [1] too.

One may note 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.

We like to mention here that several intermediate steps of our proofs (such as Lemma 2) involve expressions of the form $(\frac{N-m}{N})^r$, which denotes the probability of a random variable (for example, index j) distributed uniformly in $[0, \ldots, N-1]$ not taking a fixed set of m values in r independent trials. Such expressions have appeared previously in [15, Chapter

6] and in the Evolution Lemma of [13, 29].

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 [3, 5, 12, 13, 14, 22, 23]. 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 [12] and a bias in the equality of the first two output bytes has been shown in [23]. In [18], 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 [24, 30] and several classes of weak keys had been identified. Recently, a more general theoretical study has been performed in [19] which includes the observations of [24]. The work [19] 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).

Some weaknesses of the KSA have been addressed in great detail in [4] and practical attacks have been mounted on RC4 in the IV mode (e.g. WEP [10]). Further, the propagation of weak key patterns to the output keystream bytes has also been discussed in [4]. Subsequently, the work [8] improved [4]. In [15, Chapter 6], correlation between the permutations that are a few rounds apart have been discussed.

Reconstruction of the permutation looking at the keystream output bytes is another approach to attack RC4. In [9, Table 2], it has been estimated that this kind of attack would require around 2^{779} to 2^{797} complexity. Later in [28, Table 7], an improved idea has been presented that estimates a complexity of 2^{731}. A much improved result [16] in this area shows that the permutation can be recovered in around 2^{241} complexity. This shows that RC4 is not secure when the key length is more than 30 bytes. Fortunately, this result does not affect RC4 for the typical secret key size of 5 to 16 bytes.

2 Theoretical Analysis of the Key Scheduling

Let S_{y+1} be the permutation and j_{y+1} be the value of the pseudo-random index j after $y+1$ many rounds of the RC4 KSA, when the deterministic index i has taken the value y, $0 \leq y \leq N-1$. According to this notation, S_N is the permutation after the complete key scheduling. We denote the initial permutation by S_0 and the initial value 0 of the index j by j_0. We use the notation S^{-1} for the inverse of the permutation S, i.e., if $S[y] = v$, then $S^{-1}[v] = y$. Further, let Z_N denote the set $\{0, 1, \ldots, N-1\}$.

2.1 Probability Expression for Roos' Bias [24]

We now prove a general formula (Theorem 1) that estimates the probabilities with which the permutation bytes after each round of the RC4 KSA are related to certain combinations of the secret key bytes. The result we present has two-fold significance. It gives 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 the index j takes its value from Z_N independently and uniformly at random at each round of the KSA. Then,* $P\Big(j_{y+1} = \sum_{x=0}^{y} S_0[x] + \sum_{x=0}^{y} K[x]\Big) \approx (\frac{N-1}{N})^{1+\frac{y(y+1)}{2}} + \frac{1}{N}$, $0 \leq y \leq N-1$.

Proof: For $y \geq 0$, let E_y denote the event that $j_{y+1} = \sum_{x=0}^{y} S_0[x] + \sum_{x=0}^{y} K[x]$ and let A_y denote the event that $S_x[x] = S_0[x]$ for all $x \in [0, y]$. Let $\bar{A}_y$ stand for the complementary event of A_y. We have $P(E_y) = P(E_y|A_y) \cdot P(A_y) + P(E_y|\bar{A}_y) \cdot P(\bar{A}_y)$. We also have $P(E_y|A_y) = 1$ and $P(E_y|\bar{A}_y)$ approximately equals $\frac{1}{N}$ due to random association. We show by induction that $P(A_y) = (\frac{N-1}{N})^{\frac{y(y+1)}{2}}$. The base case $P(A_0) = 1$ is trivial. For $y \geq 1$ we have $P(A_y) = P\big(A_{y-1} \wedge (S_y[y] = S_0[y])\big) \approx P(A_{y-1}) \cdot P(S_y[y] = S_0[y]) = (\frac{N-1}{N})^{\frac{(y-1)y}{2}} \cdot P(S_y[y] = S_0[y])$. Conditioned on the event that all the values $j_1, j_2, \ldots, j_y$ are different from y, which happens with probability $(\frac{N-1}{N})^y$, $S_y[y]$ remains the same as $S_0[y]$. Note that if at least one of the values $j_1, j_2, \ldots, j_y$ hits the index y, the probability that $S_y[y]$ remains the same as $S_0[y]$ is too small. Hence $P(S_y[y] = S_0[y]) \approx (\frac{N-1}{N})^y$. This completes the proof of the induction as well as the proof of the Lemma. ■

Lemma 2 *Assume that the index j takes its value from Z_N independently and uniformly at random at each round of the KSA. Then, $P(S_r[y] = S_0[j_{y+1}]) \approx (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{r-1}$, $0 \leq y \leq r-1$, $1 \leq r \leq N$.*

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

Theorem 1 *Assume that the index j takes its value from Z_N independently and uniformly at random at each round of the KSA. Then,* $P(S_r[y] = f_y) \approx (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{[\frac{y(y+1)}{2}+r]} + \frac{1}{N}$, *where* $f_y = S_0\Big[\sum_{x=0}^{y} S_0[x] + \sum_{x=0}^{y} K[x]\Big]$, $0 \leq y \leq r-1$, $1 \leq r \leq N$.

Proof: Let A_y denote the event as defined in the proof of Lemma 1. Now, $S_r[y]$ can equal f_y in two ways. One way is that the event A_y and the event $S_r[y] = S_0[j_{y+1}]$ occur together (recall that $P(E_y|A_y) = 1$)). Combining the results of Lemma 1 and Lemma 2, we get the contribution of this part $\approx P(A_y) \cdot P(S_r[y] = S_0[j_{y+1}]) = (\frac{N-1}{N})^{\frac{y(y+1)}{2}} \cdot (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{r-1}$ $= (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{[\frac{y(y+1)}{2}+(r-1)]}$. Another way is that neither of the above events happen and still $S_r[y]$ equals $S_0\Big[\sum_{x=0}^{y} S_0[x] + \sum_{x=0}^{y} K[x]\Big]$ due to random association. The contribution of this second part is approximately $\Big(1 - (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{[\frac{y(y+1)}{2}+(r-1)]}\Big) \cdot \frac{1}{N}$. Adding these two contributions, we get the total probability $\approx (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{[\frac{y(y+1)}{2}+(r-1)]} + \Big(1 - (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{[\frac{y(y+1)}{2}+(r-1)]}\Big) \cdot \frac{1}{N} = (1 - \frac{1}{N}) \cdot (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{[\frac{y(y+1)}{2}+(r-1)]} + \frac{1}{N} = (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{[\frac{y(y+1)}{2}+r]} + \frac{1}{N}$.
■

Corollary 1 *The bias of the final permutation after the KSA towards the secret key is given by* $P(S_N[y] = f_y) \approx (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{[\frac{y(y+1)}{2}+N]} + \frac{1}{N}$, $0 \leq y \leq N-1$.

Proof: Substitute $r = N$ in the statement of the theorem. ■

Note that if the initial permutation S_0 is identity, then we have $f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]$. Table 2 shows the theoretical values of the probability $P(S_N[y] = f_y)$ in this case which is consistent with the experimental values provided in [24].

y	$P(S_N[y] = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x])$
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

Table 2: The probabilities following Corollary 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.

2.2 Intrinsic Weakness of Shuffle-exchange Type KSA

In the KSA of RC4, the index i is incremented by one and j is updated pseudo-randomly by the rule $j = j + S[i] + K[i]$. In the notations of Section 2, we may write, for $0 \leq y \leq N-1$,

$$j_{y+1} = j_y + S_y[y] + K[y].$$

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 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 permutation at any stage of 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.

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. Using the notations of Section 2, we may write

$$j_{y+1} = u\big(y, j_y, S_y[y], S_y[j_y], K[y], K[j_y]\big).$$

For subsequent reference, let us call the KSA with this generalized update rule as GKSA.

Lemma 3 *Assume that the index j takes its value from Z_N independently and uniformly at random at each round of the GKSA. Then, one can always construct functions $h_y(S_0, K)$, which depends only on y, the secret key bytes and the initial permutation, and probabilities π_y, which depends only on y and N, such that $P\big(j_{y+1} = h_y(S_0, K)\big) = (\frac{N-1}{N})\pi_y + \frac{1}{N}$, $0 \leq y \leq N-1$.*

Proof: We have $j_{y+1} = u\big(y, j_y, S_y[y], S_y[j_y], K[y], K[j_y]\big)$. Due to the swap in round y, we have $S_y[j_y] = S_{y-1}[y-1]$, so $j_{y+1} = u\big(y, j_y, S_y[y], S_{y-1}[y-1], K[y], K[j_y]\big)$. Let $h_y(S_0, K)$ have the following recursive formulation.

$$h_0(S_0, K) = u\big(0, 0, S_0[0], S_0[0], K[0], K[0]\big)$$

and for $1 \leq y \leq N-1$,

$$h_y(S_0, K) = u\big(y, h_{y-1}(S_0, K), S_0[y], S_0[y-1], K[y], K[h_{y-1}(S_0, K)]\big).$$

For $y \geq 0$, let E_y denote the event that $j_{y+1} = h_y(S_0, K)$. For $y \geq 1$, let A_y denote the event that $u\big(x, j_x, S_x[x], S_{x-1}[x-1], K[x], K[j_x]\big) = u\big(x, j_x, S_0[x], S_0[x-1], K[x], K[j_x]\big)$ for all $x \in [1, y]$. Let A_0 denote the trivial event $S_0[0] = S_0[0]$ or $u\big(0, 0, S_0[0], S_0[0], K[0], K[0]\big) = u\big(0, 0, S_0[0], S_0[0], K[0], K[0]\big)$, which occurs with probability $\pi_0 = P(A_0) = 1$. Let $\bar{A}_y$ stand for the complementary event of A_y. We have $P(E_y) = P(E_y|A_y) \cdot P(A_y) + P(E_y|\bar{A}_y) \cdot P(\bar{A}_y)$. We also have $P(E_y|A_y) = 1$ and $P(E_y|\bar{A}_y)$ approximately equals $\frac{1}{N}$ due to random

association. By induction on y, we will show how to construct the the probabilities π_y recursively such that $P(A_y) = \pi_y$. For $y \geq 1$, suppose $P(A_{y-1}) = \pi_{y-1}$. Now the event A_y occurs, if the following two holds:
(1) the event A_{y-1} (with probability π_{y-1}) occurs, and
(2) one or both of the events $S_y[y] = S_0[y]$ (with probability $(\frac{N-1}{N})^y$) and $S_{y-1}[y-1] = S_0[y-1]$ (with probability $(\frac{N-1}{N})^{y-1}$) occurs, depending on whether the form of u contains one or both of these terms respectively.
Thus, $\pi_y = P(A_y)$ can be computed as a function of y, N, and π_{y-1}, depending on the occurrence or non-occurrence of various terms in u. This completes the proof of the induction as well as the proof of the Lemma. ■

Theorem 2 *Assume that the index j takes its value from Z_N independently and uniformly at random at each round of the GKSA. Then, one can always construct functions $f_y(S_0, K)$, which depends only on y, the secret key bytes and the initial permutation, such that $P(S_r[y] = f_y(S_0, K)) \approx (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^r \cdot \pi_y + \frac{1}{N}$, $0 \leq y \leq r-1$, $1 \leq r \leq N$.*

Proof: We will show that $f_y(S_0, K) = S_0[h_y(S_0, K)]$ where the function h_y's are given by Lemma 3. Let A_y denote the event as defined in the proof of Lemma 3. Now, $S_r[y]$ can equal $S_0[h_y(S_0, K)]$ in two ways. One way is that the event A_y and the event $S_r[y] = S_0[j_{y+1}]$ occur together (recall that $P(E_y|A_y) = 1)$). Combining Lemma 3 and Lemma 2, we find the probability of this event to be approximately $(\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{r-1} \cdot \pi_y$. Another way is that neither of the above events happen and still $S_r[y] = S_0[h_y(S_0, K)]$ due to random association. The contribution of this part is approximately $\left(1 - (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{r-1} \cdot \pi_y\right) \cdot \frac{1}{N}$. Adding the above two contributions, we get the total probability $\approx (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{r-1} \cdot \pi_y + \left(1 - (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{r-1} \cdot \pi_y\right) \cdot \frac{1}{N} = (1 - \frac{1}{N}) \cdot (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^{r-1} \cdot \pi_y + \frac{1}{N} = (\frac{N-y}{N}) \cdot (\frac{N-1}{N})^r \cdot \pi_y + \frac{1}{N}$.
■

Next, we discuss some special cases of the update rule u as illustrative examples of how to construct the functions f_y's and the probabilities π_y's for small values of y using Lemma 3. In all the following cases, we assume S_0 to be an identity permutation and hence $f_y(S_0, K)$ is the same as $h_y(S_0, K)$.

Example 1 *Consider the KSA of RC4, where*

$$u(y, j_y, S_y[y], S_y[j_y], K[y], K[j_y]) = j_y + S_y[y] + K[y].$$

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* $y \geq 1$,

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

Solving the recurrence, we get $h_y(S_0, K) = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]$. *From the analysis in the proof* *of Lemma 3, we see that in the recurrence of h_y, $S_y[y]$ has been replaced by $S_0[y]$ and j_y has been replaced by $h_{y-1}(S_0, K)$. Hence, we would have $\pi_y = P(S_y[y] = S_0[y]) \cdot P(j_y = h_{y-1}(S_0, K)) = (\frac{N-1}{N})^y \cdot \pi_{y-1}$. Solving this recurrence, we get $\pi_y = \prod_{x=0}^{y} (\frac{N-1}{N})^x = (\frac{N-1}{N})^{\frac{y(y+1)}{2}}$. These expressions coincide with those in Lemma 1.*

Example 2 *Consider the update rule*

$$u(y, j_y, S_y[y], S_y[j_y], K[y], K[j_y]) = j_y + S_y[j_y] + K[j_y].$$

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 $y \geq 1$,

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

From the analysis in the proof of Lemma 3, we see that in the recurrence of h_y, $S_{y-1}[y-1]$ and j_y are respectively replaced by $S_0[y-1]$ and $h_{y-1}(S_0, K)$. Thus, we would have $\pi_y = (\frac{N-1}{N})^{y-1} \cdot \pi_{y-1}$. Solving this recurrence, we get $\pi_y = \prod_{x=1}^{y} (\frac{N-1}{N})^{x-1} = (\frac{N-1}{N})^{\frac{y(y-1)}{2}}$.

Example 3 *As another example, suppose*

$$u(y, j_y, S_y[y], S_y[j_y], K[y], K[j_y]) = j_y + y \cdot S_y[j_y] + K[j_y].$$

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 $y \geq 1$,

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

As in the previous example, here also the recurrence relation for the probabilities is $\pi_y = (\frac{N-1}{N})^{y-1} \cdot \pi_{y-1}$, whose solution is $\pi_y = \prod_{x=1}^{y} (\frac{N-1}{N})^{x-1} = (\frac{N-1}{N})^{\frac{y(y-1)}{2}}$.

Our results show that the design of RC4 KSA cannot achieve further security by changing the update rule by any rule from a large class that we present.

2.3 Getting Individual Key Bytes using All Bytes of S_N

First note that, every value y in the permutation is touched at least once during the KSA by the indices i, j, $0 \leq y \leq N-1$. Initially, y is located at index y in the permutation. In round $y+1$, when i reaches index y, either y is still in index y, or it has been moved due to swaps in one of the previous y rounds. In the former case, i will touch it in round $y+1$. In the latter case, one of $\{j_1, j_2, \ldots, j_y\}$ has touched it already.

Theorem 3 $P(j_{y+1} = S_N^{-1}[y] \text{ or } j_{y+1} = S_N[y]) = \frac{N-2}{N} \cdot (\frac{N-1}{N})^{N-1} + \frac{2}{N}$, $0 \le y \le N-1$.

Proof: The event $(j_{y+1} = S_N^{-1}[y])$, or, equivalently, the event $(S_N[j_{y+1}] = y)$ occurs, if $E_1(y)$, which is a combination of the following two events, holds.

1. y is not touched by any of $\{j_1, j_2, \ldots, j_y\}$ in the first y rounds. This happens with probability $(\frac{N-1}{N})^y$.

2. In round $y+1$, when i becomes y, j_{y+1} moves y to one of the indices in $\{0, \ldots, y\}$ due to the swap and y remains there until the end of KSA. This happens with probability $P(j_{y+1} \in \{0, \ldots, y\}) \cdot P(j_t \neq j_{y+1}, y+2 \le t \le N) = \frac{y+1}{N} \cdot (\frac{N-1}{N})^{N-y-1}$.

Thus, $P(E_1(y)) = (\frac{N-1}{N})^y \cdot \frac{y+1}{N} \cdot (\frac{N-1}{N})^{N-y-1} = \frac{y+1}{N} \cdot (\frac{N-1}{N})^{N-1}$.

Again, the event $(j_{y+1} = S_N[y])$ occurs, if $E_2(y)$, which is a combination of the following two events, holds. (Note that the event $E_2(y)$ is taken from Lemma 2. We here outline the proof of the probability of $E_2(y)$ for easy reference.)

1. $S_y[j_{y+1}] = j_{y+1}$ and therefore after the swap in round $y+1$, $S_{y+1}[y] = j_{y+1}$. This happens if $j_{y+1} \in \{y, \ldots, N-1\}$ and had not been touched by any of $\{j_1, j_2, \ldots, j_y\}$ in the first y rounds. The probability of this is $\frac{N-y}{N} \cdot (\frac{N-1}{N})^y$.

2. Once j_{y+1} sits in index y due to the above, it is not touched by any of the remaining $N-y-1$ many j values until the end of the KSA. The probability of this is $(\frac{N-1}{N})^{N-y-1}$.

Thus, $P(E_2(y)) = \frac{N-y}{N} \cdot (\frac{N-1}{N})^y \cdot (\frac{N-1}{N})^{N-y-1} = \frac{N-y}{N} \cdot (\frac{N-1}{N})^{N-1}$.

Now, both $E_1(y)$ and $E_2(y)$ hold if y is not touched by any of $\{j_1, j_2, \ldots, j_y\}$ in the first y rounds, and then $j_{y+1} = y$ so that y is not moved due to the swap, and subsequently y is not touched by any of the remaining $N-y-1$ many j values until the end of the KSA. Thus, $P(E_1(y) \cap E_2(y)) = (\frac{N-1}{N})^y \cdot \frac{1}{N} \cdot (\frac{N-1}{N})^{N-y-1} = \frac{1}{N}(\frac{N-1}{N})^{N-1}$.

Hence, $P(E_1(y) \cup E_2(y)) = P(E_1(y)) + P(E_2(y)) - P(E_1(y) \cap E_2(y)) = \frac{y+1}{N} \cdot (\frac{N-1}{N})^{N-1} + \frac{N-y}{N} \cdot (\frac{N-1}{N})^{N-1} - \frac{1}{N}(\frac{N-1}{N})^{N-1} = (\frac{N-1}{N})^{N-1}$.

One way the event $(j_{y+1} = S_N^{-1}[y] \text{ or } j_{y+1} = S_N[y])$ occurs is through $E_1(y) \cup E_2(y)$. Another way is that neither $E_1(y)$ nor $E_2(y)$ holds, yet $j_{y+1} \in \{S_N^{-1}[y], S_N[y]\}$ due to random association, whose probability contribution is $(1 - (\frac{N-1}{N})^{N-1}) \cdot \frac{2}{N}$. Adding these two contributions, we get the result. ■

In the above proof, we have introduced two events $E_1(y)$ and $E_2(y)$. These events would be referred further in the proof of Theorem 4 later.

The bias of $S_N^{-1}[y]$ is also analyzed in [1] indepedently and at the same time as the current work. In this regard, we like to point out that the ideas of our Theorem 3 and [1, Theorem 5] are related. However, the rest of the analysis in the current section have no intersection with [1].

For $N = 256$, the value turns out to be > 0.37, which conforms to experimental observation. The result of Theorem 3 identifies that the permutation S_N and its inverse

S_N^{-1} reveal information about the secret index j in each byte. This theorem can be used to reveal the secret key in the following manner.

Let $G_0 = \{S_N[0], S_N^{-1}[0]\}$ and for $1 \le y \le N-1$, let $G_y = \{u - v - y | u \in \{S_N[y]\} \cup \{S_N^{-1}[y]\}, v \in \{S_N[y-1]\} \cup \{S_N^{-1}[y-1]\}\}$. Once more we like to remind that in $(u-v-y)$, the operations are modulo N. For $0 \le y \le N-1$, each set G_y contains the possible values that the key byte $K[y]$ can take.

Remark 1 *It is highly likely that $S_N[y] \neq S_N^{-1}[y]$ and $S_N[y-1] \neq S_N^{-1}[y-1]$. So we consider $|G_0| = 2$ and $|G_y| = 4$, $1 \le y \le N-1$.*

We write $G_0 = \{g_{01}, g_{02}\}$, where $g_{01} = S_N^{-1}[0]$, and $g_{02} = S_N[0]$; and for $1 \le y \le N-1$, $G_y = \{g_{y1}, g_{y2}, g_{y3}, g_{y4}\}$, where $g_{y1} = S_N^{-1}[y] - S_N^{-1}[y-1] - y$, $g_{y2} = S_N[y] - S_N[y-1] - y$, $g_{y3} = S_N^{-1}[y] - S_N[y-1] - y$, and $g_{y4} = S_N[y] - S_N^{-1}[y-1] - y$. Further, let $p_{0x} = P(K[0] = g_{0x})$, $1 \le x \le 2$, and for $1 \le y \le N-1$, let $p_{yx} = P(K[y] = g_{yx})$, $1 \le x \le 4$. We have the following result.

Theorem 4
(1) $p_{01} = \frac{1}{N} \cdot (\frac{N-1}{N})^N + \frac{1}{N}$ *and* $p_{02} = (\frac{N-1}{N})^N + \frac{1}{N}$.
(2) *For* $1 \le y \le N-1$,
$p_{y1} = \frac{y(y+1)}{N^2} \cdot (\frac{N-1}{N})^{2N-1} + \frac{1}{N}$, $p_{y2} = \frac{(N-y)(N-y+1)}{N^2} \cdot (\frac{N-1}{N})^{2N-1+y} + \frac{1}{N}$,
$p_{y3} = \frac{(y+1)(N-y+1)}{N^2} \cdot (\frac{N-1}{N})^{2N-1+y} + \frac{1}{N}$ *and* $p_{y4} = \frac{y(N-y)}{N^2} \cdot (\frac{N-1}{N})^{2N-1+y} + \frac{1}{N}$.

Proof: We would be referring to the events $E_1(y)$ and $E_2(y)$ in the proof of Theorem 3. From Theorem 3, we have $P(E_1(y)) = \frac{y+1}{N} \cdot (\frac{N-1}{N})^{N-1}$ and $P(E_2(y)) = \frac{N-y}{N} \cdot (\frac{N-1}{N})^{N-1}$.

For each probability p_{yx} in items (1) and (2), we would consider two components. The component which comes due to the contributions of the events $E_1(y), E_2(y)$ etc, would be called α_{yx}. The other component is due to random association and is given by $(1-\alpha_{yx}) \cdot \frac{1}{N}$. So for each probability p_{yx}, deriving the part α_{yx} suffices, as the total probability can be computed as $\alpha_{yx} + (1-\alpha_{yx}) \cdot \frac{1}{N} = \frac{N-1}{N} \cdot \alpha_{yx} + \frac{1}{N}$.

Consider the update rule in the KSA: $j_{y+1} = j_y + S_y[y] + K[y]$, $0 \le y \le N-1$, where $j_0 = 0$.

First, we prove item (1). Since $S_0[0] = 0$, we can write $j_1 = K[0]$. Considering $j_1 = S_N^{-1}[0]$, we have $\alpha_{01} = P(E_1(0))$ and considering $j_1 = S_N[0]$, we have $\alpha_{02} = P(E_2(0))$. Substituting 0 for y in the expressions for $P(E_1(y))$ and $P(E_2(y))$, we get the results.

Now, we come to item (2). In the update rule, $S_y[y]$ can be replaced by y, assuming that it has not been touched by any one of $j_1, j_2, \ldots, j_y$ in the first y rounds of the KSA. This happens with a probability $(\frac{N-1}{N})^y$, $0 \le y \le N-1$. Assuming $S_y[y] = y$, we can write $K[y] = j_{y+1} - j_y - y$. When considering the contribution of $E_1(y)$ to j_{y+1}, the factor $(\frac{N-1}{N})^y$ need not be taken into account, as the event $(S_y[y] = y)$ is already contained in $E_1(y)$. Thus, the components α_{yx}'s for the probabilities p_{y1}, p_{y2}, p_{y3} and p_{y4} are respectively given by
$\alpha_{y1} = P(E_1(y)) \cdot P(E_1(y-1))$, $\alpha_{y2} = P(E_2(y)) \cdot P(E_2(y-1)) \cdot (\frac{N-1}{N})^y$,
$\alpha_{y3} = P(E_1(y)) \cdot P(E_2(y-1))$, and $\alpha_{y4} = P(E_2(y)) \cdot P(E_1(y-1)) \cdot (\frac{N-1}{N})^y$.

Substituting the probability expressions for $E_1(y), E_1(y-1), E_2(y)$ and $E_2(y-1)$, we get the results. ■

Corollary 2
(1) $P(K[0] \in G_0) = 1 - (1-p_{01})(1-p_{02})$.
(2) *For* $1 \leq y \leq N-1$, $P(K[y] \in G_y) = 1 - (1-p_{y1})(1-p_{y2})(1-p_{y3})(1-p_{y4})$.

Substituting values for y, we find that $P(K[0] \in G_0) \approx 0.37$ and For $1 \leq y \leq N-1$, $P(K[y] \in G_y)$ varies between 0.12 and 0.15. Experimental results also confirm these theoretical estimates.

Theorems 3, 4 are the foundations of this paper. Next, we present additional theoretical results based on the above two theorems to build the framework of retrieving individual key bytes.

The RC4 key k of l bytes gets repeated to fill the N bytes of the key array K. The number of places in K where the same key byte $k[w]$ is repeated is given by

$$n_w = \begin{cases} \lfloor \frac{N}{l} \rfloor + 1 & \text{for } 0 \leq w < N \bmod l; \\ \lfloor \frac{N}{l} \rfloor & \text{for } N \bmod l \leq w < l. \end{cases}$$

Thus, when considering a key byte k_w, we are interested in the set

$$T_w = \cup_{y \in [0,N-1], y \bmod l = w} G_y,$$

which is a union of n_w many G_y's. Let us denote these G_y's by $G_{w_1}, G_{w_2}, \ldots, G_{w_{n_w}}$ and the corresponding p_{yx}'s by $p_{w_1x}, p_{w_2x}, \ldots, p_{w_{n_w}x}$. Also, for notational convenience in representing the formulas, we denote the size of G_y by M_y. According to Remark 1, $M_0 = 2$ and $M_y = 4$, $1 \leq y \leq N-1$. In the results below, $E(X)$ denotes the expectation of a random variable X.

Theorem 5 *For* $0 \leq w \leq l-1$, *let* $freq_w$ *be the frequency (i.e., no. of occurrences) of* $k[w]$ *in the set* T_w. *Then for* $0 \leq w \leq l-1$, *we have the following.*
(1) $P(k[w] \in T_w) = 1 - \displaystyle\prod_{t=1}^{n_w} \prod_{x=1}^{M_{w_t}} (1 - p_{w_tx})$.
(2) $P(freq_w = c) =$

$$\sum_{\substack{\{t_1,t_2,\ldots,t_c\} \\ \subseteq \{1,2,\ldots,n_w\}}} \left(\prod_{r=1}^{c} \sum_{x=1}^{M_{w_{t_r}}} p_{w_{t_r}x} \prod_{x' \neq x} (1 - p_{w_{t_r}x'}) \right) \left(\prod_{\substack{r \in \{1,2,\ldots,n_w\} \setminus \\ \{t_1,t_2,\ldots,t_c\}}} \prod_{x=1}^{M_{w_r}} (1 - p_{w_rx}) \right).$$

(3) $E(freq_w) = \displaystyle\sum_{t=1}^{n_w} \sum_{x=1}^{M_{w_t}} p_{w_tx}$.

Proof: First, we prove item (1). We know that $k_w \notin T_w$ iff $k_w \notin G_{w_t}$ for all $t \in \{1, \ldots, n_w\}$. Again, $k_w \notin G_{w_t}$, iff for each $x \in \{1, \ldots, M_{w_t}\}$, $k_w \neq g_{w_tx}$, the probability of which is $(1 - p_{w_tx})$. Hence the result follows.

Next, we prove item (2). Item (2) is a generalization of item (1), since $P(k_w \in T_w) = 1 - P(freq_w = 0)$. For arbitrary c, $0 \leq c \leq n_w$, k_w occurs exactly c times in T_w, iff it occurs once in exactly c out of n_w many G_w's, say once in each of $G_{w_{t_1}}, G_{w_{t_2}}, \ldots, G_{w_{t_c}}$, and it does not occur in any of the remaining $n_w - c$ many G_w's. We call such a division of the G_w's a *c-division*. Again, $k[w]$ occurs exactly once in G_{w_t} iff $k[w]$ equals exactly one of the M_{w_t} members of G_{w_t} and it does not equal any of the remaining $(M_{w_t} - 1)$ members of G_{w_t}. Thus, the probability that $k[w]$ occurs exactly once in G_{w_t} is given by $\sum_{x=1}^{M_{w_{t_r}}} p_{w_{t_r}x} \prod_{x' \neq x} (1 - p_{w_{t_r}x'})$. Also, for any $r \in \{1, 2, \ldots, n_w\} \setminus \{t_1, t_2, \ldots, t_c\}$, $k[w]$ does not occur in G_{w_r} with probability $\prod_{x=1}^{M_{w_r}} (1 - p_{w_r x})$. Adding the contributions of all $\binom{n_w}{c}$ many *c-division*'s, we get the result.

Finally, we come to item (3). For $1 \leq t \leq n_w$, $1 \leq x \leq M_{w_t}$ let $u_{t,x} = 1$, if $k[w] = g_{w_t x}$; otherwise, let $u_{t,x} = 0$. Thus, the number of occurrences of $k[w]$ in T_w is $freq_w = \sum_{t=1}^{n_w} \sum_{x=1}^{M_{w_t}} u_{t,x}$. Then $E(freq_w)$ is given by $\sum_{t=1}^{n_w} \sum_{x=1}^{M_{w_t}} E(u_{t,x})$, where $E(u_{t,x}) = P(u_{t,x} = 1) = p_{w_t x}$. ■

Corollary 3 *For $0 \leq w \leq l - 1$, given a threshold TH, $P(freq_w > TH)$ can be estimated as $1 - \sum_{c=0}^{TH} P(freq_w = c)$, where $P(freq_w = c)$'s are as given in Theorem 5, item 2.*

Theorem 6 *Let $q_{yx} = \frac{1 - p_{yx}}{N-1}$, $0 \leq x \leq M_y$, $0 \leq y \leq N - 1$. Then for $0 \leq w \leq l - 1$, we have the following.*
(1) The expected number of distinct values $\in [0, N-1]$ occurring in T_w is given by
$E_{dist} = N - \prod_{t=1}^{n_w} \prod_{x=1}^{M_{w_t}} (1 - p_{w_t x}) - (N-1) \prod_{t=1}^{n_w} \prod_{x=1}^{M_{w_t}} (1 - q_{w_t x})$.
(2) The expected number of distinct values $\in [0, N-1]$, each occurring exactly c times in T_w, is given by

$$E_c = \sum_{\substack{\{t_1, t_2, \ldots, t_c\} \\ \subseteq \{1,2,\ldots,n_w\}}} \left(\prod_{r=1}^{c} \sum_{x=1}^{M_{w_{t_r}}} p_{w_{t_r}x} \prod_{x' \neq x} (1 - p_{w_{t_r}x'}) \right) \left(\prod_{\substack{r \in \{1,2,\ldots,n_w\} \setminus \\ \{t_1, t_2, \ldots, t_c\}}} \prod_{x=1}^{M_{w_r}} (1 - p_{w_r x}) \right) +$$
$$(N-1) \sum_{\substack{\{t_1, t_2, \ldots, t_c\} \\ \subseteq \{1,2,\ldots,n_w\}}} \left(\prod_{r=1}^{c} \sum_{x=1}^{M_{w_{t_r}}} q_{w_{t_r}x} \prod_{x' \neq x} (1 - q_{w_{t_r}x'}) \right) \left(\prod_{\substack{r \in \{1,2,\ldots,n_w\} \setminus \\ \{t_1, t_2, \ldots, t_c\}}} \prod_{x=1}^{M_{w_r}} (1 - q_{w_r x}) \right).$$

Proof: First, we prove item (1). For $0 \leq u \leq N - 1$, let $x_u = 1$, if u does not occur in T_w; otherwise, let $x_u = 0$. Hence, the number of values from $[0, N-1]$ that do not occur at all in T_w is given by $X = \sum_{u=0}^{N-1} x_u$. A value u does not occur in T_w iff it does not occur in any G_{w_t}. For $u = k[w]$, according to item 1 of Theorem 5, $P(x_u = 1) = \prod_{t=1}^{n_w} \prod_{x=1}^{M_{w_t}} (1 - p_{w_t x})$. Assuming that each $g_{w_t x}$, $1 \leq x \leq M_{w_t}$, takes each value $u \in [0, N] \setminus \{k[w]\}$ with equal probabilities, we have $P(g_{w_t x} = u) = \frac{1 - p_{w_t x}}{N-1} = q_{w_t x}$. So, for $u \in [0, N] \setminus \{k[w]\}$, $P(x_u = 1) = \prod_{t=1}^{n_w} \prod_{x=1}^{M_{w_t}} (1 - q_{w_t x})$. We compute $E(X) = \sum_{u=0}^{N-1} E(x_u) = E(x_{k[w]}) + \sum_{u \in [0,N] \setminus \{k[w]\}} E(x_u)$, where $E(x_u) = P(x_u = 1)$. The expected number of distinct values $\in [0, N-1]$ occurring in T_w is then given by $N - E(X)$.

Next, we come to item (2). Here, let $x'_u = 1$, if u occurs exactly c times in T_w; otherwise, let $x'_u = 0$. Hence, the number of values from $[0, N-1]$ that occurs exactly c times in T_w is given by $X' = \sum_{u=0}^{N-1} x'_u$. Now, u occurs exactly c times in T_w, iff it occurs once in exactly c out of n_w many G_w's and it does not occur in any of the remaining $n_w - c$ many G_w's. For $u = k[w]$, $P(x'_u = 1)$ is given by item (2) of Theorem 5. In the same expression, $p_{w_{t_r} x}$ would be replaced by $q_{w_{t_r} x}$, when $u \neq k[w]$. Since $E(x'_u) = P(x'_u = 1)$, and $E(X') = \sum_{u=0}^{N-1} E(x'_u)$, we get the result by adding the individual expectations. ∎

Corollary 4 *For $0 \leq w \leq l-1$, given a threshold TH, the expected number of distinct values $\in [0, N-1]$, each occurring $> TH$ times in T_w, can be estimated as $N - \sum_{c=0}^{TH} E_c$, where E_c is the expected number of distinct values $\in [0, N-1]$, each occurring exactly c times in T_w, as given in Theorem 6, item 2.*

We can use the above results to devise an algorithm *BuildKeyTable* for building a frequency table for each key byte and use the table to extract important information about the key.

```
BuildKeyTable($S_N$)
Data Structures:
    Arrays $jarr1[N+1], jarr2[N+1], karr[l][N]$.
    1. $jarr1[0] = jarr2[0] = 0$;
    2. For $y = 0$ to $N-1$ do
            $jarr1[y+1] = S[y]$ and $jarr2[y+1] = S_N^{-1}[y]$;
    3. For $w = 0$ to $l-1$ and for $y = 0$ to $N-1$ do
            $karr[w][y] = 0$;
    4 $karr\big[0\big]\big[jarr1[1]\big]$ += 1 and $karr\big[0\big]\big[jarr2[1]\big]$ += 1;
    5. For $y = 1$ to $N-1$ do
          5.1 $karr\big[y \bmod l\big]\big[jarr1[y+1] - jarr1[y] - y\big]$ += 1;
          5.2 $karr\big[y \bmod l\big]\big[jarr1[y+1] - jarr2[y] - y\big]$ += 1;
          5.3 $karr\big[y \bmod l\big]\big[jarr2[y+1] - jarr1[y] - y\big]$ += 1;
          5.4 $karr\big[y \bmod l\big]\big[jarr2[y+1] - jarr2[y] - y\big]$ += 1;
```

The arrays $jarr1$ and $jarr2$ contain the values of j_y's as estimated from S_N and S_N^{-1} respectively. For each key byte $k[w]$, $0 \le w \le l-1$, the frequency of a value $y \in [0, N-1]$ is stored in $karr[w][y]$. The notation 'x += 1' is used to denote that x is incremented by 1.

When guessing a specific key byte $k[w]$, one would generally consider all values $\in [0, N-1]$ that have occurred at least once. However, one may try other alternatives such as considering only those values $\in [0, N-1]$ that have frequency above a certain threshold c.

2.3.1 Experimental Evidences

To compare how close the theoretical estimates are to the experimental ones, we present some results here. All the experiments are carried out with 1 million randomly chosen keys and the results presented are average of each run.

First we present experimental results corresponding to Theorems 5, 6 in Table 3. For all key lengths, the estimates are given for $k[3]$ (i.e. for $w = 3$ and $c = 2$) only as a representative. However, we have verified the results for all key bytes $k[w]$, $0 \le w \le l-1$, for each key length $l = 5, 8, 10, 12$ and 16, and for different values of c.

In few cases, the theoretical and the empirical values are not close. These cases arise because the idealistic assumption of independence of events does not always hold in practice. However, we find that in general the theoretical formula fits the empirical data to a very good approximation.

Next we present some experiments related to the *BuildKeyTable* algorithm. We show that each individual key byte can be recovered with a very high probability. Table 4 shows data for the first two bytes of secret keys with key lengths 5, 8, 10, 12 and 16. The results are obtained by considering 1 million randomly chosen secret keys of different key lengths. In Table 4, 'Succ.' denotes the success probability and 'Search' denotes the number of

	l	5	8	10	12	16
$P(k[w] \in T_w)$	Theory	0.9991	0.9881	0.9729	0.9532	0.8909
	Exp.	0.9994	0.9902	0.9764	0.9578	0.8970
$P(freq_w = c)$	Theory	0.0225	0.1210	0.1814	0.2243	0.2682
	Exp.	0.0186	0.1204	0.1873	0.2353	0.2872
$E(freq_w)$	Theory	6.8	4.3	3.5	3.0	2.1
	Exp.	6.8	4.3	3.5	2.9	2.1
E_{dist}	Theory	138.5	99.2	84.2	73.4	55.9
	Exp.	138.2	98.9	84.0	73.2	55.8
E_c	Theory	34.7	18.1	13.1	10.0	5.8
	Exp.	35.2	18.6	13.6	10.4	6.2

Table 3: Theoretical vs. empirical estimates with $w = 3$ and $c = 2$ for Theorems 5, 6.

values $\in [0, N-1]$ that have frequency above the threshold c. Theoretical estimates of these values are given in Theorem 4 and Theorem 6 respectively.

		Threshold c = 0		Threshold c = 1		Threshold c = 2	
l	Key byte	Succ.	Search	Succ.	Search	Succ.	Search
5	k[0]	0.9997	138.9	0.9967	47.9	0.9836	12.4
	k[1]	0.9995	138.2	0.9950	47.4	0.9763	12.2
8	k[0]	0.9927	97.6	0.9526	22.2	0.8477	4.1
	k[1]	0.9902	98.9	0.9400	22.9	0.8190	4.2
10	k[0]	0.9827	82.5	0.9041	15.7	0.7364	2.6
	k[1]	0.9761	84.0	0.8797	16.3	0.6927	2.7
12	k[0]	0.9686	71.6	0.8482	11.8	0.6315	1.8
	k[1]	0.9577	73.2	0.8118	12.3	0.5763	1.8
16	k[0]	0.9241	54.1	0.7072	6.7	0.4232	0.9
	k[1]	0.8969	55.8	0.6451	7.1	0.3586	0.9

Table 4: Experimental results for first two key bytes using different thresholds.

As an example, each secret key byte for $l = 5$ can be guessed with a probability > 0.97 amongst only around 12 values each of which has frequency at least 3; whereas, for random guess, one need to consider at least 248 ($\approx 256 \times 0.97$) values to achieve the same probability.

For any key length, the success probability and search complexity of other bytes from $k[2]$ onwards are almost same as those of $k[1]$. So the data for $k[1]$ is a representative of the other key bytes. Observe that the success probability for $K[0]$ is always little more than that for the other key bytes. This happens because $K[0]$ is estimated as $j_1 - j_0 - 0$, where $j_0 = 0$ with probability 1. This is also consistent with item (1) of Theorem 4.

For the same threshold c, the probability as well as the search complexity for each key byte decreases as the key length l increases. This happens because the number of repetitions corresponding to each key byte decreases with the increase in l.

This is an independent novel work for separately retrieving the individual secret key bytes from S_N as opposed to the earlier works of solving simultaneous equations. This reveals a new kind of weakness in the key scheduling of RC4. How this result can be combined with the earlier techniques to improve upon them for complete key recovery is not our immediate goal. However, we present some simple strategies in the next section to demonstrate possible applications of our results towards the complete key recovery. In certain cases, our results are currently the best known.

3 Recovering the Secret Key from the Permutation

In this section, we discuss how to get back the secret key, if one knows the RC4 permutation. State recovery attacks [9, 28, 16] are an important class of attacks on RC4 and the key recovery attack from the internal state which covers a major part of the current work is useful to turn a state recovery attack into a key recovery attack. If the complexity of "recovering the secret key from the permutation" is less than that of "recovering RC4 permutation from the keystream output bytes in PRGA", then by cascading the techniques of the latter [9, 28, 16] with those of the former, "recovering the secret key from the keystream output bytes" is possible at the same complexity as the latter.

In many cryptographic applications, a secret key is combined with a known IV to form a session key. For a single session, recovering the permutation is enough for cryptanalysis. However, there are many applications (such as WEP [10]), where the key and the IV are combined (to form the session key) in such a way that the secret key can be easily extracted from the session key. For these applications, if one can recover the session key from the permutation then it is possible to get back the secret key. In that case, for subsequent sessions where the same secret key would be used with different known IV's, the RC4 encryption would be completely insecure.

If we know the RC4 internal state after τ rounds of PRGA, we can get back the permutation S_N after the KSA using the *PRGAreverse* algorithm described below and thereby recover the secret key. Even if the value j_C of the index j is not known, one can make exhaustive search over all N possible values for j_C and for each value perform the state-reversing loop in PRGAreverse. If for a value of j_C the final value of j after the completion of the loop equals zero, the resulting final permutation can be a candidate for S_N. On average one expects to get one such candidates.

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 $i_C = \tau \bmod N$. Note that all subtractions except $r = r - 1$ in Algorithm *PRGAreverse* are modulo N operations.

```
**PRGAreverse**
Inputs:
1. The permutation $S_C$.
2. Number of rounds $\tau$.
Output:
Few candidates for $S_N$.

1. $i_C = \tau \bmod N$;
2. For $j_C = 0$ to $N - 1$
    2.1. $i = i_C$; $j = j_C$; $S = S_C$; $r = \tau$;
    2.2. Do
        2.2.1. Swap($S[i]$, $S[j]$);
        2.2.2. $j = j - S[i]$; $i = i - 1$; $r = r - 1$;
      While $r > 0$;
    2.3. If $j = 0$
        2.3.1. Report $S$ as a candidate for $S_N$;
```

The technique for recovering secret key from the permutation after the KSA or at any stage during the KSA is the main theme of this section.

3.1 Other Related Works

Mantin mentioned in [15, Appendix A.2] how to recover the first A bytes of the secret key from an "Early Permutation State" after round A. We introduced the idea of recovering the complete key from S_N in [20]. Subsequent to our work of [20], Biham and Carmeli [2] refined the idea of [20] and also pointed out some minor errors in the tables related to complexity calculations. In Section 3.3.1 of this paper, we revise the complexity analysis of *RecoverKey* algorithm and related experimental results of our earlier version presented in [20, Section 3]. In particular, refer to Table 5 in Section 3.3.1 of this paper for a corrected and revised version of [20, Table 3]. Very recently, further improvements in this direction have been presented in [1].

3.2 Issues Related to Time Complexity Estimates

Given the permutation S_r after the r-th round of the KSA, whatever may be the underlying algorithm for guessing the key, after each guess an additional complexity of around r is required to run the KSA for the first r rounds and compare the permutation obtained with S_r to verify correctness of the key. If we start with the permutation S_N after the KSA, then (with the typical $N = 256$) this would require an additional complexity of 2^8 for verifying each key. Thus, the search space for exhaustive search of an l byte secret key is 2^{8l}, but the actual complexity including the verification time is around 2^{8l+8}. In the complexity analysis we do not include the key verification time, but only consider the number of keys to be searched. The reason for this is two-fold. First, the additional complexity overhead of 2^8 per key verification is constant for any algorithm. So all the algorithms can be compared under a common model, if only the key search complexity is considered, excluding the key verification time. Secondly, when a wrong key is verified, in practice one may not need to run N complete rounds of the KSA. If the initial bytes of the permutation after a few (such as 32) rounds of the KSA do not match at any position with the corresponding permutation bytes in hand, then one may discard the current guess and try with the next one. Since almost all the keys in the search space is wrong, on average key verification can be assumed to take a constant amount of time.

Remark 2 *Recent works [2, 1] on key recovery use time estimates instead of exact complexity. We have checked that on a 2.8 GHz CPU, verifying 2^{20} many secret keys can be completed in a second by a simple C code. Thus, the values of our time complexities can be divided by 2^{20} to get the estimate in seconds.*

3.3 Complete Key Recovery by Solving Simultaneous Equations

We explain the scenario with an example first. In all the examples in this section, we consider, without loss of generality, only the final permutation after the KSA, i.e., we consider the case $r = N$ only.

Example 4 *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 $f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]$. If one runs the KSA, then the first 16 bytes of the final permutation will be as follows.*

y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
f_y	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 $S_N[y] = f_y$, $0 \leq y \leq 15$, 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 $y = 1, 4, 5, 8$ and 12, and the corresponding equations are:

$$\begin{aligned} K[0] + K[1] + (1 \cdot 2)/2 &= 166 & (1)\\ K[0] + K[1] + K[2] + K[3] + K[4] + (4 \cdot 5)/2 &= 238 & (2)\\ K[0] + \ldots + K[5] + (5 \cdot 6)/2 &= 93 & (3)\\ K[0] + \ldots + K[8] + (8 \cdot 9)/2 &= 202 & (4)\\ K[0] + \ldots + K[12] + (12 \cdot 13)/2 &= 151 & (5) \end{aligned}$$

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 $S_r[y] = f_y$ for the set of chosen y-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 $\frac{2^{40}}{2^{13} \cdot 2^5} = 2^{22}$.

From Theorem 1, we get how $S_r[y]$ is biased to different combinations of the keys, namely, with $f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]$. Let us denote $P(S_r[y] = f_y) = p_{r,y}$ for $0 \le y \le r-1$, $1 \le r \le N$. 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 ($m \le n \le r$) many equations $S_r[y] = f_y$, $y = 0, 1, \ldots, n-1$, 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 $\{y_1, y_2, \ldots, y_t\} \subseteq \{0, 1, \ldots, n-1\}$, 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 $S_r[y] = f_y$, 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 7 *Let $l \ge 2$ be the RC4 key length in bytes. Suppose we want to select systems of m independent equations, $2 \le m \le l$, from the following n equations of the form $S_r[y] = f_y$ involving the permutation bytes after round r of the KSA, $m \le n \le r \le N$, where $f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]$, $0 \le y \le n-1$.*

1. *The system $S_r[y_q] = f_{y_q}$, $1 \le q \le m$, of m equations selected from $S_r[y] = f_y$, $0 \le y \le n-1$, corresponding to $y = y_1, y_2, \ldots, y_m$, is independent if and only if any one of the following two conditions hold: either (i) $y_q \bmod l$, $1 \le q \le m$, yields m distinct values, or (ii) $y_q \bmod l \ne (l-1)$, $1 \le q \le m$, and there is exactly one pair $y_a, y_b \in \{y_1, y_2, \ldots, y_m\}$ such that $y_a = y_b \pmod{l}$, and all other $y_q \bmod l, q \ne a, q \ne b$ yields $m-2$ distinct values different from $y_a, y_b \pmod{l}$.*

2. *The total number of independent systems of m (≥ 2) equations is given by*
$$|EI_m| = \sum_{x=0}^{m} \binom{n \bmod l}{x} \binom{l - n \bmod l}{m-x} (\lfloor \tfrac{n}{l} \rfloor + 1)^x (\lfloor \tfrac{n}{l} \rfloor)^{m-x}$$
$$+ \binom{n \bmod l}{1} \binom{\lfloor \frac{n}{l} \rfloor + 1}{2} \sum_{x=0}^{m-2} \binom{n \bmod l - 1}{x} \binom{l - n \bmod l - 1}{m-2-x} (\lfloor \tfrac{n}{l} \rfloor + 1)^x (\lfloor \tfrac{n}{l} \rfloor)^{m-2-x}$$

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

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., $y_q \bmod l$ $(1 \le q \le m)$ 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 $a, b \in \{1, \ldots, m\}$, $a \neq b$, where $y_a = y_b \bmod l$. Without loss of generality, suppose $y_a < y_b$. Then we can subtract $S_r[y_a] = f_{y_a}$ from $S_r[y_b] = f_{y_b}$ 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 y_a or y_b by the new equation involving s, which will become a different equation with a new variable $K[l-1]$, since $l - 1 \notin \{y_1 \bmod l, y_2 \bmod l, \ldots, y_m \bmod l\}$. 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 a, b, c such that $y_a = y_b = y_c = \bmod l$, or we will have a pair a, b with $y_a = y_b \bmod l$ and $l - 1 \in \{y_1 \bmod l, y_2 \bmod l, \ldots, y_m \bmod l\}$. 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 $y \bmod l$, for $y = 0, 1, \ldots n-1$, then we will have the following residue classes:

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

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

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 7.*

Proof: We need to check a total of $\binom{n}{m}$ many m tuples $\{y_1, y_2, \ldots, y_m\}$, and using the independence criteria of Theorem 7, 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 $S_r[y_q] = f_{y_q}$ involving the l key bytes as variables corresponding to the tuple $\{y_1, y_2, \ldots, y_m\}$, $0 \leq y_q \leq n-1$, $1 \leq q \leq m$, where $f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]$. 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 [25], 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 $\lfloor \frac{y_s}{l} \rfloor$, where y_s is the y-value $\in \{y_1, y_2, \ldots, y_m\}$ corresponding to the equation involving s. Thus, $gcd(a, N) \leq a = \lfloor \frac{y_s}{l} \rfloor \leq \lfloor \frac{n}{l} \rfloor$. ∎

Let us consider an example to demonstrate the case when we have two y-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 5 *Assume that the secret key is of length 5 bytes. Let us consider* 16 *equations of the form* $S_N[y] = f_y$, $0 \leq y \leq 15$. *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* $y = 0, 1, 2, 3$ *and* 13. *Let the corresponding* $S_N[y]$ *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* $s = 76$ *or* 204. *Taking the value* 76, *we get*

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

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.*

3.3.1 Algorithm and Complexity Analysis

[1] We now present the general algorithm for recovering the secret key bytes from the permutation at any stage of the KSA.

[1]Algorithm *RecoverKey*, Theorem 8 and Table 5 of Section 3.3.1 are completely revised and updated over the corresponding material in [20, Section 3] and give improved results.

```
RecoverKey
Inputs:
1. Number of key bytes: $l$.
2. Number of key bytes to be solved from equations: $m$ ($\leq l$).
3. Number of equations to be tried: $n$ ($\geq m$).
4. The permutation bytes: $S_r[y]$, $0 \leq y \leq r-1$ and the stage $r$, $n \leq r \leq N$.
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.
1. For each distinct tuple $\{y_1, y_2, \ldots, y_m\}$, $0 \leq y_q \leq n-1$, $1 \leq q \leq m$ 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. Solve for the $m$ variables in terms of the remaining $l-m$ variables;
      1.1.3. For each possible assignment of the $l-m$ variables do
         1.1.3.1. Find values of the other $m$ key bytes;
         1.1.3.2. If the correct key is found, return it.
```

If one does not use the independence criteria (Theorem 7), 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 5 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 8 *The time complexity of the RecoverKey algorithm is given by*

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

where $|EI_m|$ is given by Theorem 7.

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 Steps 1.1.1, 1.1.2 and 1.1.3 are executed $|EI_m|$ times. Among them, finding the solution in Step 1.1.2 involves $O(m^2)$ many addition/subtraction operations (the equations being triangular in form). By Proposition 2, each system can yield at the most $O(\lfloor \frac{n}{l} \rfloor)$ many solutions for the key. After the solution is found, Step 1.1.3 involves $2^{8(l-m)}$ many trials. Thus, the total time consumed by Steps 1.1.1, 1.1.2 and 1.1.3 for a complete run would be $O\Big(|EI_m| \cdot \big(m^2 + \lfloor \frac{n}{l} \rfloor \cdot 2^{8(l-m)}\big)\Big)$.

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

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 $S_r[y] = f_y$, $y = 0, 1, \ldots, n-1$, $m \leq n \leq r \leq N$. Let $c_{r,n}$ be the number of independent correct equations. Then*

$$P(c_{r,n} \geq m) = \sum_{t=m}^{n} \sum_{\{y_1, y_2, \ldots, y_t\} \in EI_t} p_r(y_1, y_2, \ldots, y_t),$$

where EI_t is the collection of the indices $\{y_1, y_2, \ldots, y_t\}$ corresponding to all sets of t independent equations, and $p_r(y_1, y_2, \ldots, y_t)$ is the joint probability that the t equations corresponding to the indices $\{y_1, y_2, \ldots, y_t\}$ are correct and the other $n - t$ equations corresponding to the indices $\{0, 1, \ldots, n-1\} \setminus \{y_1, y_2, \ldots, y_t\}$ are incorrect.

Proof: We need to sum $|EI_t|$ number of terms of the form $p_r(y_1, y_2, \ldots, y_t)$ to get the probability that exactly t equations are correct, i.e.,

$$P(c_{r,n} = t) = \sum_{\{y_1, y_2, \ldots, y_t\} \in EI_t} p_r(y_1, y_2, \ldots, y_t).$$

Hence, $P(c_{r,n} \geq m) = \sum_{t=m}^{n} P(c_{r,n} = t) = \sum_{t=m}^{n} \sum_{\{y_1, y_2, \ldots, y_t\} \in EI_t} p_r(y_1, y_2, \ldots, y_t)$. ∎

Note that $P(c_{r,n} \geq m)$ gives the success probability with which one can recover the secret key from the permutation after the r-th round of the KSA.

In Theorem 1, we observed that as the number r of rounds increase, the probabilities $P(S_r[y] = f_y)$ decrease. Finally, after the KSA, when $r = N$, (see Corollary 1) the probabilities settle to the values as given in Table 2. However, as the events $(S_r[y] = f_y)$ are not independent for different y's, theoretically presenting the formulae for the joint probability $p_r(y_1, y_2, \ldots, y_t)$ seems to be extremely tedious.

In Table 5, we provide experimental results on the probability of having at least m independent correct equations, when the first n equations $S_N[y] = f_y, 0 \leq y \leq n-1$, after the complete KSA (i.e., $r = N$), are considered for the *RecoverKey* algorithm for different values of n, m, and the key length l, satisfying $m \leq l \leq n$. Table 5 is a corrected and revised version of [20, Table 3].

For each probability calculation, the complete KSA (with $N = 256$ rounds) is 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 $e = \log_2 \left(m^2 \cdot \binom{n}{m} + |EI_m| \cdot \left(m^2 + \lfloor \frac{n}{l} \rfloor \cdot 2^{8(l-m)} \right) \right)$. The time complexity of exhaustive search is $O(2^{8l})$ and that of the *RecoverKey* algorithm, according to Theorem 8, 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 5 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 5 clearly show that the probabilities (i.e., the empirical values of $P(c_{N,n} \geq m)$) in most of the cases are greater than 10%. However, the algorithm does

l	n	m	$\binom{n}{m}$	$\lvert EI_m \rvert$	$8l$	e	$P(c_{N,n} \geq m)$
5	48	5	1712304	238500	40	25.6	0.431
5	24	5	42504	7500	40	20.3	0.385
5	16	5	4368	810	40	17.0	0.250
8	22	6	74613	29646	64	31.9	0.414
8	16	6	8008	3472	64	28.8	0.273
8	20	7	77520	13068	64	23.4	0.158
10	16	7	11440	5840	80	36.5	0.166
10	24	8	735471	130248	80	34.0	0.162
12	24	8	735471	274560	96	51.1	0.241
12	24	9	1307504	281600	96	43.1	0.116
12	21	10	352716	49920	96	31.6	0.026
16	24	9	1307504	721800	128	75.5	0.185
16	32	10	64512240	19731712	128	73.2	0.160
16	32	11	129024480	24321024	128	65.5	0.086
16	40	12	5586853480	367105284	128	61.5	0.050
16	27	12	17383860	2478464	128	53.2	0.022
16	26	12	9657700	1422080	128	52.4	0.019
16	44	14	114955808528	847648395	128	46.9	0.006
16	24	14	1961256	69120	128	32.2	0.0006

Table 5: Running the *RecoverKey* algorithm using different parameters for the final permutation after the complete KSA (with $N = 256$ rounds).

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.

The last 8 entries corresponding to 16 bytes key length in Table 5 give a flavour about the relationship between the complexity and success probability. For example, when the success probability increases from 0.0006 to 0.006 (i.e. increases by 10 times), the complexity increases by a factor of $2^{14.7}$.

3.4 Complete Key Recovery from Frequency Table

In this section, we discuss how the complete key can be recovered using our technique described in Section 2.3. We first like to refer to the *BuildKeyTable* algorithm from Section 2.3. Corresponding to this algorithm, we have already explained the probabilities of getting an individual key byte from a selected set of values. Now we estimate the probability and time complexity when all the key bytes are identified at the same time. By *Method* 1, we refer to this simple strategy of guessing the complete key using different thresholds on the basic table obtained from *BuildKeyTable* algorithm.

Towards improving *Method* 1, we present *Method* 1A below. *Method* 1A updates the basic frequency table obtained from the *BuildKeyTable* algorithm by considering the values obtained from the $S[S[y]]$ type of biases. In [11, Section 2], it was shown that biases towards $f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[y]$ exist at the permutation bytes $S_N[y]$, $S_N[S_N[y]]$, $S_N[S_N[S_N[y]]]$, $S_N[S_N[S_N[S_N[y]]]]$, and so on. In *Method* 1A, we guess $K[0]$ first. Then, we increase y one by one starting from 1 and given the values of $K[0], K[1], \ldots, K[y-1]$, we compute the value of $K[y]$ from the equations $S_N^d[y] = f_y$, where d is the level of indirections considered. Here, for frequency updates of $k[0]$, the first four levels of indirections (i.e., $d = 1, 2, 3$ and 4) are used and for frequency updates of other four key bytes, only the first two levels of indirections are used. For frequency updates of $k[1]$, only the values of $k[0]$ with frequency > 2 are considered. Similarly, for $k[2]$, the values $k[0]$ with frequency > 3 and the values of $k[1]$ with frequency > 4 respectively are considered. These values (e.g. 3, 4 here) are called *threshold frequencies*. For $k[3]$, the threshold frequencies of $k[0], k[1]$ and $k[2]$ are 4 and 5 and 6 respectively. And finally, for $k[4]$, the threshold frequencies of $k[0], k[1], k[2]$ and $k[3]$ are 4, 5, 6 and 7 respectively. Observe that while updating the table for the key byte $k[w]$ for a fixed w, we increase the thresholds for $k[0], \ldots, k[w-1]$, as w increases. This is due to the reason that we want to selectively consider those cases which are highly probable. Low thresholds for $k[0], \ldots, k[w-1]$ substantially increase the number of choices for $k[w]$ without significantly increasing the probability for correct $k[w]$. The thresholds we have presented here are tuned empirically.

3.4.1 Experimental Results for 5 Byte Keys

In Table 6, we present the complete data related to all the bytes for 5 bytes secret key using both *Method* 1 and *Method* 1A. 'Succ.' denotes the success probability and 'comp.' denotes the search complexity. The complexity of retrieving the entire key is computed by multiplying the average number of values that need to be searched for individual key bytes. The success probability of recovering the entire key is calculated empirically. This value is typically more than the product of the individual probabilities, implying that the values of the key bytes are not independent given the final permutation bytes. We see that without using any heuristic, just applying our simple *Method* 1 on the basic table obtained from *BuildKeyTable* algorithm helps to achieve 89.46% success rate in a complexity $12.4 \times (12.2)^4 \approx 2^{18.1}$.

Next, we try enhancements of the basic technique for 5 byte key to achieve better results. For each key byte, we first try the value with the maximum frequency, then the second maximum and so on. The search is done in an *iterative deepening* manner so that if d_w is the depth (starting from the most frequent guess) of the correct value for $k[w]$,

		Threshold c = 0		Threshold c = 1		Threshold c = 2	
Key byte	Method	Succ.	Comp.	Succ.	Comp.	Succ.	Comp.
$k[0]$	1	0.9997	138.9	0.9967	47.9	0.9836	12.4
	1A	0.9998	140.2	0.9980	49.2	0.9900	13.0
$k[1]$	1	0.9995	138.2	0.9950	47.4	0.9763	12.2
	1A	0.9997	149.2	0.9971	56.6	0.9857	16.1
$k[2]$	1	0.9995	138.2	0.9949	47.4	0.9764	12.2
	1A	0.9996	142.2	0.9965	50.7	0.9834	13.6
$k[3]$	1	0.9995	138.2	0.9950	47.4	0.9761	12.2
	1A	0.9996	138.7	0.9958	47.8	0.9796	12.4
$k[4]$	1	0.9995	138.2	0.9950	47.4	0.9766	12.2
	1A	0.9996	138.7	0.9958	47.8	0.9796	12.4
Entire Key	1	0.9976	$2^{35.6}$	0.9768	$2^{27.8}$	0.8946	$2^{18.1}$
	1A	0.9983	$2^{35.7}$	0.9830	$2^{28.3}$	0.9203	$2^{18.7}$

Table 6: Experimental results for all key bytes using different thresholds for $l = 5$.

then the search never go beyond depth $d_{max} = max\{d_w, 0 \leq w \leq 4\}$ for any key byte. The complexity is calculated by finding the average of d_{max}^5 over 1 million trials, each with a different key. We also set a *depth limit* G, which denotes at most how many different values is to be tried for each key in descending order of their frequencies, starting from the most frequent value. We denote this strategy as *Method* 2. If we update the frequency table as in *Method* 1A before performing the search in descending order of frequencies, then we name it as *Method* 2A.

The experimental results for the above two enhancements are presented in Table 7. As before, 'succ.' denotes the success probability and 'comp.' denotes the time complexity.

Method	G	10	16	32	48	64	80	96	160
2	Succ.	0.8434	0.9008	0.9383	0.9719	0.9800	0.9841	0.9870	0.9980
	Comp.	$2^{14.3}$	$2^{17.0}$	$2^{21.3}$	$2^{23.5}$	$2^{24.8}$	$2^{26.0}$	$2^{27.1}$	$2^{29.0}$
2A	Succ.	0.8678	0.9196	0.9503	0.9772	0.9855	0.9876	0.9906	0.9985
	Comp.	$2^{14.0}$	$2^{16.7}$	$2^{20.9}$	$2^{23.1}$	$2^{24.5}$	$2^{25.7}$	$2^{26.6}$	$2^{28.7}$

Table 7: Experimental results for $l = 5$ using the most frequent guesses of the key bytes.

Note that for 5 byte key, [2] reports a success probability of 0.8640 in 0.02 seconds and [1] reports a success probability of 0.998 in 0.008 seconds. We achieve a success probability of 0.9985 in complexity $2^{28.7}$.

Our further results related to $l = 8, 10, 12, 16$ are discussed later in Table 8 in the next section.

3.4.2 Experimental Results for Other Key Lengths

For key lengths $l = 8, 10, 12$ and 16, we performed simple experiments using *Method* 1A where the technique of updating the frequency table is applied for the first 5 key bytes only (as we have already implemented this for our experiments on 5 byte secret keys). Unlike [2, 1], where time estimates are presented in seconds, we present time complexity estimates of searching the number of keys.

In Table 8, we present three sets of results, namely, selected entries from Table 5 and data generated from Exp. I and II which are explained below. In Exp. I, we report the exact time complexities for our experiments in which we just exceed the probabilities given in [2, Table 4]. In Exp. II, we report the success probabilities for each key length that is achievable in complexity around 2^{40}, which should take less than a day using a few state of the art machines (see Remark 2). In Exp. I, for $l = 8, 10$ and 12, we use a threshold of 2 for each of the first 4 bytes and for $l = 16$, we use a threshold of 2 for each of the first 2 bytes. A threshold of 1 is used for the remaining bytes for each key length. In Exp. II, for $l = 16$, the thresholds are same as those in Exp. I. For other key lengths, the thresholds are as follow. For $l = 8$, we use a threshold of 1 for each of the first 6 bytes and a threshold of 0 for each of the last two bytes. A threshold of 1 is used for each key byte for the case $l = 10$. For $l = 12$, a threshold of 2 is used for the first byte and a threshold of 1 is used for each of the rest.

	l	8	10	12	16
Data from [2]	Probability	0.4058	0.1290	0.0212	0.0005
	Time in Seconds	0.60	3.93	7.43	278
Our Results from Table 5	Probability	0.414	0.162	0.026	0.0006
	Complexity	$2^{31.9}$	$2^{34.0}$	$2^{31.6}$	$2^{32.2}$
Our Exp. I	Probability	0.4362	0.1421	0.0275	0.0007
	Complexity	$2^{26.8}$	$2^{29.9}$	$2^{32.0}$	$2^{40.0}$
Our Exp. II	Probability	0.7250	0.2921	0.0659	0.0007
	Complexity	$2^{40.6}$	$2^{40.1}$	$2^{40.1}$	$2^{40.0}$
Data from [1]	Probability	0.931	–	0.506	0.0745
	Time in Seconds	8.602	–	54.390	1572

Table 8: Experimental results for $l = 8, 10, 12, 16$.

Here we only study the difference between two consecutive j's to extract a single byte of the key. However, the difference between any two j's can be considered to get equations involving sum of more than one key bytes (in the same line as the idea of [20] has been extended in [2]).

In the first and last row of Table 8, we present the best probability values as reported in [2, Table 4] and [1, Table 5] respectively. The works [2, 1] improved upon our initial results in [20]. The study in [2] considered the equations of [20] as well as their differences. The approach in [1] created new equations over [20, 2] based on the biases in [11, Section

2] and new biases of the inverse permutation S_N^{-1} and their differences which lead them to obtain higher success rate than [20, 2]. On the other hand, we have also revised and improved upon our earlier work [20] independently and at the same time as [1] and this is presented in Section 3.4 of the current paper. In Table 8, we do not intend to present a comparative study on the performance of the different algorithms for key recovery, rather we use the data available from [2, 1] as benchmark. In fact, the last row of Table 8 clearly shows the superiority of [1]. Also slightly improving the probability values (as given in the first row of Table 8) of [2], we do not claim that our strategy is more efficient. This is to show with what complexity we can improve the probability values of [2].

We would like to emphasize here that our immediate goal in this paper is not to study all possible choices of parameters and heuristics through empirical trial and error for faster recovery of the secret key. Our main aim is to build a theoretical framework for recovering the key based on formal analysis of the correlation between the permutation and the secret key. We believe that our basic technique, when combined with the works of [2, 1] is likely to provide further improvements in success probability as well as time complexity.

4 Conclusion

In this paper we present a detailed study on weaknesses of RC4 KSA and PRGA. We theoretically prove with what probability the permutation bytes $S[y]$ are biased to some linear combinations secret key bytes. Further, we analyze a generalization of the RC4 KSA corresponding to a class of update functions and our results show that it is possible to find biases of the permutation bytes towards the secret key in this generalized framework.

These results are used to recover the secret key bytes from the RC4 state at any stage. Our algorithm has a constant probability of success in order of square root of the time required for the exhaustive key search. This has generated significant interests in the community in inverting the RC4 KSA (i.e. getting back the secret key from the permutation), leading to a series of subsequent publications by other researchers.

We also show that significant amount of information about the hidden index j at every round can be obtained by exploiting all the bytes of S_N, S_N^{-1}. This can be used to retrieve each of the key bytes with very high probability. Moreover, in recovering the complete key, our simple experiments, which are straightforward applications of our theoretical results, sometimes outperform other works such as [2] in terms of success probabilities (see Table 8).

Since the state (which includes the permutation and the indices) is in general not observable, getting the secret key from the permutation does not immediately pose a 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 KSA and certain generalizations of it. Added to this, the biases in the RC4 permutation bytes towards the secret key get transferred to the keystream output bytes [19, 11]. Such key leakage is not desirable for a stream cipher. Our present work provides a direction in designing an improved version of RC4 with better security. More generally, these ideas may be used to explore a new design paradigm for shuffle exchange kind of stream ciphers.

References

[1] M. Akgün, P. Kavak and H. Demirci. New Results on the Key Scheduling Algorithm of RC4. INDOCRYPT 2008, pages 40-52, vol. 5365, Lecture Notes in Computer Science, Springer.

[2] E. Biham and Y. Carmeli. Efficient Reconstruction of RC4 Keys from Internal States. FSE 2008, pages 270-288, vol. 5086, Lecture Notes in Computer Science, Springer.

[3] 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.

[4] 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.

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

[6] R. J. Jenkins. ISAAC and RC4. 1996. Available at `http://burtleburtle.net/bob/rand/isaac.html`.

[7] S. Khazaei and W. Meier. On Reconstruction of RC4 Keys from Internal States. Accepted in Mathematical Methods in Computer Science (MMICS), December 17-19, 2008, Karlsruhe, Germany.

[8] A. Klein. Attacks on the RC4 stream cipher. *Designs, Codes and Cryptography*, pages 269-286, vol. 48, no. 3, September, 2008]. A draft dated February 27, 2006 is available at `cage.ugent.be/~ klein/RC4/RC4-en.ps`.

[9] L. R. Knudsen, W. Meier, B. Preneel, V. Rijmen and S. Verdoolaege. Analysis Methods for (Alleged) RCA. ASIACRYPT 1998, pages 327-341, vol. 1514, Lecture Notes in Computer Science, Springer.

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

[11] S. Maitra and G. Paul. New Form of Permutation Bias and Secret Key Leakage in Keystream Bytes of RC4. FSE 2008, pages 253-269, vol. 5086, Lecture Notes in Computer Science, Springer. A revised and extended version with the same title is available at the IACR Eprint Server, eprint.iacr.org, number 2007/261, July 3, 2007.

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

[13] 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.

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

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

[16] A. Maximov and D. Khovratovich. New State Recovering Attack on RC4. CRYPTO 2008, pages 297-316, vol. 5157, Lecture Notes in Computer Science, Springer.

[17] M. E. McKague. Design and Analysis of RC4-like Stream Ciphers. Master's Thesis, University of Waterloo, Canada, 2005.

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

[19] 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. An extended version appears in *Designs, Codes and Cryptography*, pages 123-134, vol. 49, no. 1-3, December, 2008.

[20] G. Paul and S. Maitra. Permutation after RC4 Key Scheduling Reveals the Secret Key. SAC 2007, pages 360-377, vol. 4876, Lecture Notes in Computer Science, Springer.

[21] G. Paul, S. Maitra and R. Srivastava. On Non-Randomness of the Permutation after RC4 Key Scheduling. Applied Algebra, Algebraic Algorithms, and Error Correcting Codes (AAECC-17), pages 100-109, vol. 4851, Lecture Notes in Computer Science, Springer.

[22] 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.

[23] 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.

[24] 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`.

[25] J. Silverman. A Friendly Introduction to Number Theory. Prentice Hall, NJ. Page 56, Second Edition, 2001.

[26] E. Tews. Email Communications, September, 2007.

[27] E. Tews, R. P. Weinmann and A. Pyshkin. Breaking 104 bit WEP in less than 60 seconds. IACR Eprint Server, eprint.iacr.org, number 2007/120, April 1, 2007.

[28] V. Tomasevic, S. Bojanic and O. Nieto-Taladriz. Finding an internal state of RC4 stream cipher. *Information Sciences*, pages 1715-1727, vol. 177, 2007.

[29] S. Vaudenay and M. Vuagnoux. Passive-Only Key Recovery Attacks on RC4. SAC 2007, pages 344-359, vol. 4876, Lecture Notes in Computer Science, Springer.

[30] 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`.