Analysis of RC4 and Proposal of Additional Layers for Better Security Margin*

Subhamoy Maitra[1] and Goutam Paul[2]

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

Abstract. In this paper, the RC4 Key Scheduling Algorithm (KSA) is theoretically studied to reveal non-uniformity in the expected number of times each value of the permutation is touched by the indices i, j. Based on our analysis and the results available in literature regarding the existing weaknesses of RC4, few additional layers over the RC4 KSA and RC4 Pseudo-Random Generation Algorithm (PRGA) are proposed. Analysis of the modified cipher (we call it RC4$^+$) shows that this new strategy avoids existing weaknesses of RC4.

Keywords: Bias, Cryptography, Keystream, KSA, PRGA, RC4, Secret Key, Stream Cipher.

1 Introduction

RC4 is one of the most popular and efficient stream ciphers. The basic idea of RC4 is to start with the identity permutation and then use the secret key to produce a random looking permutation. This is the Key Scheduling Algorithm (KSA). Based on this (secret key dependent) pseudo-random permutation, the next stage of Pseudo-Random Generation Algorithm (PRGA) generates keystream bytes which get XOR-ed with the plaintext bytes to generate ciphertexts.

The KSA and the PRGA of RC4 are presented below. The data structure consists of an array S of size N (typically, 256), which contains a permutation of the integers $\{0, \ldots, N-1\}$, two indices i (deterministic) and j (pseudo-random) and a secret key array K. Given a secret key key of l bytes (typically 5 to 32), the array K of size N is such that $K[y] = key[y \bmod l]$ for any y, $0 \leq y \leq N-1$. All additions in both the KSA and the PRGA are additions modulo N.

* This is an extended version of the paper with the same title which has been accepted for presentation in the 9th International Conference on Cryptology in India (INDOCRYPT 2008), to be held at Indian Institute of Technology, Kharagpur, during December 14-17, 2008.

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 $z = S[t]$;

Before proceeding further, we like to point out that there are standard procedures to generate pseudo-random permutations, but the RC4 KSA is different from them.

1.1 Random Permutation vs RC4 KSA

Let us present one well known result from [13]. Given a permutation of N elements $\prod = (\pi_0, \ldots, \pi_{N-1})$, a series of $N-1$ transpositions can produce random permutation as follows:

RandPerm
From $i = N-1$ down to 1
$\pi_i \leftrightarrow \pi_{rand(0,i)}$;

Here, $rand(0, i)$ produces uniformly distributed random integers in the range 0 through i. One may refer to [30, Page 171] for the proof that after $N-1$ many transpositions, the permutation is indeed random. One may note the following issues where the algorithms KSA and *RandPerm* are not similar (this has also been discussed in [19, Chapter 6]).

1. The index i increases from 0 to $N-1$ in RC4 KSA (number of transpositions N), but the index i decreases from $N-1$ to 1 in *RandPerm* (number of transpositions $N-1$).
2. The index j can take any value between 0 to $N-1$ at any step and the pseudo-random value of j is based on the secret key bytes in KSA. On the other hand, in *RandPerm*, $j = rand(0, i)$, that implies that it can take values only in the range 0 to i at the i-th step.

It has been pointed out in [19, Chapter 6] that the RC4 KSA does not produce permutations uniformly at random. On the other hand, the *RandPerm* algorithm cannot be used for key scheduling using the secret key bytes instead of $rand(0, i)$. In that case, the permutation after the key scheduling algorithm would have revealed the secret key completely (e.g., π_{N-1} will contain the first accessed secret key byte, π_{N-2} will contain the second accessed secret key byte, and so on). Thus there is a need to design a key scheduling algorithm that will produce a random looking permutation with certain cryptographic properties.

1.2 Further Motivation for a New Design

Reconstruction of the permutation looking at the keystream output bytes is an approach to attack RC4. In [12, Table 2], it has been estimated that this kind of attack would require

around 2^{779} to 2^{797} complexity. Later, in [35, Table 7], an improved idea has been presented that estimates a complexity of 2^{731}. A much improved result [21] 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 (240 bits).

If the RC4 state information (i.e., the permutation S; the number of keystream bytes generated after the KSA or the value of i; and the value of j) is given, then one can come back (by running the PRGA in reverse direction) to the permutation after the KSA. Then the problem is how to get back the secret key from the final permutation after the KSA. Some biases of the permutation after the KSA towards the secret key have been exploited first in [25], and subsequently in [2, 1], to recover the complete secret key from the RC4 permutation. 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 [12, 35, 21] 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 [14]), 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. However, if the weaknesses of the KSA are removed then the above methods of recovering the secret key or the session key from the keystream bytes or from permutation would fail.

In addition to the permutation recovery attacks [12, 35, 21], there exist several other works [7, 5, 24, 15–18, 28, 29] on the weaknesses of the RC4 PRGA. However, all of these exploit the initial keystream bytes only. According to [22], if the first 512 keystream bytes are thrown away, then RC4 is quite safe to use. Moreover, it is argued in [15, 27] that many biases in the PRGA are due to the propagation of the biases in the KSA via Glimpse Theorem [10, 17]. These biases in the keystream would disappear, if one could remove the corresponding biases in the permutation during the KSA.

In this paper, we discuss several weaknesses of RC4 and suggest remedies to overcome them. During last few years, there have been many efforts [40, 29, 8] on the modification of RC4 towards further improvement and there also exist distinguishing attacks on them [20, 36, 37]. This shows that there is significant interest in the cryptographic community for analysis and design of RC4 and its modifications. However, in all of these ciphers (VMPC [40], RC4A [29], RC4(n, m) [8]), the design is modified to a great extent relative to RC4. We keep the RC4 structure as it is and add a few more operations to strengthen the cipher. Thus, we attempt to exploit the good points of RC4 and then provide some additional features for a better security margin.

One may argue that concentrating on the *eSTREAM* candidates [4] is more practical than modifying RC4. However, one should also note that RC4 is the most widely used stream cipher 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 *eSTREAM* candidates have much complicated structure in general and they work on word (32 bit) oriented manner. Our goal is not to compete with those ciphers, but to keep the simple structure of RC4 and just add a few steps to it to have a byte oriented stream cipher with further strength. The existing literature on RC4 reveals that in spite of having a very simple description, the cipher possesses nice combinatorial structures in the shuffle-exchange paradigm. Our design retains this elegant property of RC4 and at the same time removes the existing weaknesses of RC4.

1.3 Paper Outline

Let us now present the outline of the paper. In Section 2, we investigate into the roots of the weaknesses of RC4 KSA and present new theoretical results on the non-uniformity in the expected number of times each value of S is touched by the indices i, j during the KSA. Section 3 focus on the design of the new key scheduling. We summarise the existing weaknesses of the RC4 KSA in Section 3.1 and describe the modified KSA in Section 3.2. In Section 3.3, we argue how the new algorithm avoids the weaknesses of the standard RC4 KSA. Section 4 works on the modification of the RC4 PRGA, identifying the existing weaknesses and justifying why the modified PRGA has better security margin. We present logical arguments as well as empirical evidence to support our security conjectures of the new design. Finally, in Section 5, we present some performance results comparing the speed of RC4 and our new design, followed by concluding remarks in Section 6.

2 Movement Frequency of Permutation Values

Before we go into the technicalities, let us introduce a few notations. We denote the initial identity permutation by S_0 and the permutation at the end of the r-th round of the KSA by S_r, $1 \leq r \leq N$. Note that $r = y + 1$, when the deterministic index i takes the value y, $0 \leq y \leq N - 1$. Thus, the permutation after the KSA will be denoted by S_N. By j_r, we denote the value of the index j after it is updated in round r. Also, let $f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[y]$, that would be referred frequently in the subsequent discussions.

We observe that many values in the permutation are touched once with a very high probability by the indices i, j during the KSA.

Theorem 1. *The probability that a value v in the permutation is touched exactly once during the KSA by the indices i, j, is given by $\frac{2v}{N} \cdot (\frac{N-1}{N})^{N-1}$, $0 \leq v \leq N - 1$.*

Proof. Initially, v is located at index v in the permutation. It is touched exactly once in one of the following two ways.

1. v is not touched by any of $\{j_1, j_2, \ldots, j_v\}$ in the first v rounds. This happens with probability $(\frac{N-1}{N})^v$. In round $v+1$, when i becomes v, v is moved to the left by j_{v+1} due to the swap and remains there until the end of KSA. This happens with probability $P(j_{v+1} \in \{0, \ldots, v-1\}) \cdot P(j_t \neq j_{v+1}, v+1 \leq t \leq N) = \frac{v}{N} \cdot (\frac{N-1}{N})^{N-v-1}$. Thus, the probability contribution of this part is $(\frac{N-1}{N})^v \cdot \frac{v}{N} \cdot (\frac{N-1}{N})^{N-v-1} = \frac{v}{N} \cdot (\frac{N-1}{N})^{N-1}$.
2. For some t, $1 \leq t \leq v$, it is not touched by any of $\{j_1, j_2, \ldots, j_{t-1}\}$; then it is touched for the first time by $j_t = v$ in round t and hence is moved to index $t-1$; and it is not touched by any one of the subsequent $(N-t)$ many j values. The probability contribution of this part is $\sum_{t=1}^{v} (\frac{N-1}{N})^{t-1} \cdot \frac{1}{N} \cdot (\frac{N-1}{N})^{N-t} = \frac{v}{N} \cdot (\frac{N-1}{N})^{N-1}$.

Adding the above two contributions, we get the result. □

Using similar arguments one could compute the probability that a value is touched exactly twice, thrice and in general x times, during the KSA. However, the computation would be tedious and complicated for $x > 1$. A more natural measure of this asymmetric behaviour would be the expected number of times each value in the permutation is touched during the KSA. This is computed in the next theorem.

Theorem 2. *The expected number of times a value v in the permutation is touched by the indices i, j during the KSA is given by $E_v = 1 + (\frac{2N-v}{N}) \cdot (\frac{N-1}{N})^v$, $0 \leq v \leq N-1$.*

Proof. Let $x_{v,y} = 1$, if the value v is touched by the indices i, j in round $y+1$ of the KSA (i.e., when $i = y$); otherwise, let $x_{v,y} = 0$, $0 \leq v \leq N-1$, $0 \leq y \leq N-1$. Then the number of times v is touched by i, j during the KSA is given by $X_v = \sum_{y=0}^{N-1} x_{v,y}$. In any round $y+1$, any value v is touched by j with a probability $\frac{1}{N}$. To this, we need to add the probability of v being touched by i, in order to find $P(x_{v,y} = 1)$. Now, v is touched by the index i in round $y+1$, iff $S_y[y] = v$. We consider three possible cases.

1. Case $y < v$. Initially, the value v was situated in index v. In order for v to move from index v to index $y < v$, either v has to be touched by i and y has to be touched by j, or vice versa, during the first y rounds. But this is not possible, since neither the index y nor the index v has been touched by the index i so far. Thus, $P(S_y[y] = v) = 0$.
2. Case $y = v$. We would have $S_v[v] = v$, if v is not touched by any of $\{j_1, j_2, \ldots, j_v\}$ in the first v rounds, the probability of which is $(\frac{N-1}{N})^v$.
3. Case $y > v$. Once $S_v[v] = v$, the swap in the next round moves the value v to a random location j_{v+1}. Thus, $P(S_{v+1}[y] = v) = P(S_v[v] = v) \cdot P(j_{v+1} = y) = (\frac{N-1}{N})^v \cdot \frac{1}{N}$. For all $y > v$, until y is touched by the deterministic index i, i.e., until round $y+1$, v will remain randomly distributed. Hence, for all $y > v$, $P(S_y[y] = v) = P(S_{v+1}[y] = v) = \frac{1}{N}(\frac{N-1}{N})^v$.

Noting that $E(x_{v,y}) = P(x_{v,y} = 1) = \frac{1}{N} + P(S_y[y] = v)$, we have $E_v = E(X_v) = \sum_{y=0}^{N-1} E(x_{v,y})$

$= 1 + \sum_{y=0}^{N-1} P(S_y[y] = v) = 1 + \sum_{y=0}^{v-1} P(S_y[y] = v) + P(S_v[v] = v) + \sum_{y=v+1}^{N-1} P(S_y[y] = v)$

$= 1 + 0 + (\frac{N-1}{N})^v + (N - v) \cdot \frac{1}{N}(\frac{N-1}{N})^v = 1 + (\frac{2N-v}{N}) \cdot (\frac{N-1}{N})^v$. □

We find that E_v decreases from 3.0 to 1.37, as v increases from 0 to 255. To demonstrate how close the experimental values of the expectations match with our theoretical values, we perform 100 million runs the KSA, with random key of 16 bytes in each run. The experimental results correspond to the theoretical formula, as summarised in the first two rows of Table 1 in Section 3.3. One may also refer to Figure 1 for graphical representations.

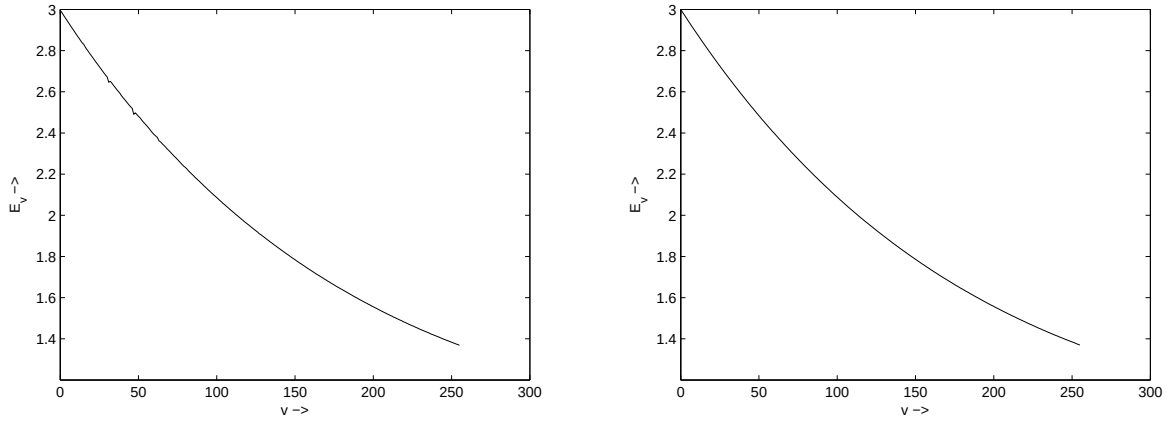

Fig. 1. E_v versus v, $0 \leq v \leq 255$. Left: Experimental data, Right: Theoretical data.

In [26, 1], it is shown that the probabilities $P(j_{y+1} = S_N^{-1}[y])$ increase with increasing y. This is connected to the above decreasing pattern in the expectations. In the first half of the KSA, i.e., when y is small, the values $v = S[y]$ are thrown more to the right with high probability by the index j_{y+1} due to the swap and hence are touched again either by the deterministic index i or by the pseudo-random index j in the subsequent rounds. On the other hand, in the second half of the KSA, i.e., when $y \geq 128$, the values $v = S[y]$ are thrown more to the left by the index j_{y+1} due to the swap and hence are never touched by i in the subsequent rounds, and may be touched by j with a small probability.

Towards designing a key scheduling algorithm in shuffle-exchange paradigm, it is important that each value in the permutation is touched (and therefore moved with probability almost one) sufficient number of times. In such a case, it will be harder to guess the values of j for which a permutation byte is swapped. In RC4 KSA, there are many permutation bytes which are swapped only once with a high probability, leading to information leakage

from S_N regarding the secret key bytes. We keep this in mind while designing the modified KSA in the next section.

3 Removing the Weaknesses of KSA

In this section, we first look into what are the existing weaknesses of the RC4 KSA, followed by suggestions to remove them. We propose a new version of the KSA and study its security issues.

3.1 Existing Weaknesses

Many works have explored the RC4 KSA and discovered different weaknesses. Here we present an overview of these results.

1. In [31], it was empirically shown that the probabilities $P(S_N[y] = f_y)$ decrease from 0.37 for $y = 0$ to 0.006 for $y = 48$ (with $N = 256$) and beyond that settle down to 0.0039 ($\approx \frac{1}{256}$). Later, in [25], explicit formula for these probabilities for all $y \in [0, \ldots, N-1]$ were theoretically derived. This result was further used in [25] to recover the secret key from the final permutation S_N after the KSA. Subsequent to the work in [25], other researchers also have shown interest in the problem of recovering the secret key from RC4 permutation and achieved better efficiency [2, 1].
2. In [25], a generalization of the RC4 KSA is also considered where the index j can be updated in different manners. In RC4 KSA, the update rule is $j = (j + S[i] + K[i])$. The work [25] showed that for a certain class of update functions which update j 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.
3. It has been shown in [15] that the bytes $S_N[y]$, $S_N[S_N[y]]$, $S_N[S_N[S_N[y]]]$, and so on, are biased to f_y. In particular, they showed that $P(S_N[S_N[y]] = f_y)$ decreases from 0.137 for $y = 0$ to 0.018 for $y = 31$ and then slowly settles down to 0.0039 (beyond $y = 48$).
4. Analysis in [26, 1] shows that the the inverse permutations $S_N^{-1}[y]$, $S_N^{-1}[S_N^{-1}[y]]$, and so on are biased to j_{y+1}, and in turn, to f_y.
5. The work [19, Chapter 6] showed for the first time that each permutation byte after the KSA is significantly biased (either positive or negative) towards many values in the range $0, \ldots, N-1$. These biases are independent of the secret key. This problem was further studied in [22, 27]. For each y, $0 \leq y \leq N-2$, $P(S_N[y] = v)$ is maximum at $v = y + 1$ and this maximum probability ranges approximately between $\frac{1}{N}(1 + \frac{1}{3})$ and $\frac{1}{N}(1 + \frac{1}{5})$ for different values of y, with $N = 256$.
6. The work [6] showed for the first time that RC4 can be attacked when used in the IV mode (e.g. WEP [14]). Subsequently, there have been series of improvements [17, 11, 34, 38] in this direction, exploiting the propagation of weak key patterns to the keystream output bytes.

3.2 Proposal for KSA$^+$: A Revised KSA

In this section, we present a modified design (called KSA$^+$) that removes the weaknesses of RC4 KSA discussed in Section 3.1. The evaluation for such a design is presented in Section 3.3. In this case, we will name the permutation after the KSA$^+$ as S_{N^+}. We have the following motivations in our mind in the new design.

1. Removing the existing weaknesses.
2. Producing a Random-looking S_{N^+} after the key scheduling so that it will be hard to identify any non-uniformity in the permutation after the KSA$^+$.
3. Secret key extraction from S_{N^+} should be of the same order as exhaustive search.
4. It should be hard to get two secret keys k, k' and two initialization vectors iv, iv' (given that at least one of the events $k \neq k'$ and $iv \neq iv'$ should hold) that can produce the same S_{N^+} after KSA$^+$.

We propose a three-layer key scheduling followed by the initialization. The initialization and basic scrambling in the first layer are the same as the original RC4 KSA.

Initialization
For $i = 0, \ldots, N-1$
$\quad S[i] = i;$
$j = 0;$

Layer 1: Basic Scrambling
For $i = 0, \ldots, N-1$
$\quad j = (j + S[i] + K[i]);$
$\quad \text{Swap}(S[i], S[j]);$

In the second layer, we scramble the permutation further using IV's. According to [9], for stream ciphers using IV's, if the IV is shorter than the key, then the algorithm may be vulnerable against the Time Memory Trade-Off attack. Thus, in this effort, we choose the IV size as the same as the secret key length. The deterministic index i moves first from the middle down to the left end and then from the middle upto the right end. In our scheme, an l-byte IV, denoted by an array $iv[0, \ldots, l-1]$, is used from index $\frac{N}{2} - 1$ down to $\frac{N}{2} - l$ during the left-ward movement and the same IV is repeated from index $\frac{N}{2}$ up to $\frac{N}{2} + l - 1$ during the right-ward movement. Here, we assume that N is even, which is usually the case in standard RC4. For ease of description, we use an array IV of length N with $IV[y] = 0$ for those indices which are not used with IV's.

$$IV[y] = \begin{cases} iv[\frac{N}{2} - 1 - y] & \text{for } \frac{N}{2} - l \leq y \leq \frac{N}{2} - 1; \\ iv[y - \frac{N}{2}] & \text{for } \frac{N}{2} \leq y \leq \frac{N}{2} + l - 1; \\ 0 & \text{otherwise.} \end{cases}$$

For $N = 256$ and $l = 16$, this gives a placement of $16 \times 2 = 32$ many bytes in the middle of the IV array spanning from index 112 to 143. This is to note that repeating the IV bytes will create a dependency so that one cannot choose all the 32 bytes freely to find some weakness in the system as one byte at the left corresponds to one byte at the right (when viewed symmetrically from the middle of an N-byte array). Further, in two different directions, the key bytes are added with the IV bytes in an opposite order. Apart from the $2l$ many operations involving the IV, the rest of $N - 2l$ many operations are without the

involvement of IV in Layer 2. This helps in covering the IV values and chosen IV kind of attacks will be hard to mount.

Layer 2: Scrambling with IV

For $i = \frac{N}{2} - 1$ down to 0
    $j = (j + S[i]) \oplus (K[i] + IV[i])$;
    Swap$(S[i], S[j])$;

For $i = \frac{N}{2}, \ldots, N - 1$
    $j = (j + S[i]) \oplus (K[i] + IV[i])$;
    Swap$(S[i], S[j])$;

Layer 3: Zigzag Scrambling

For $y = 0, \ldots, N - 1$
    If $y \equiv 0 \bmod 2$ then
        $i = \frac{y}{2}$;
    Else
        $i = N - \frac{y+1}{2}$;
    $j = (j + S[i] + K[i])$;
    Swap$(S[i], S[j])$;

In the third and final layer, we perform more scrambling in a zig-zag fashion, where the deterministic index i takes values in the following order: 0, 255, 1, 254, 2, 253, ..., 125, 130, 126, 129, 127, 128. In general, if y varies from 0 to $N - 1$ in steps of 1, then

$$i = \begin{cases} \frac{y}{2} & \text{if } y \equiv 0 \bmod 2; \\ N - \frac{y+1}{2} & \text{if } y \equiv 1 \bmod 2. \end{cases}$$

Introducing more scrambling steps definitely increases the cost of the cipher. The running time of the KSA$^+$ is around three times that of RC4 KSA, because there are three similar scrambling layers instead of one, each having N iterations. As the key scheduling is run only once, this will not affect the performance of the cipher much.

3.3 How KSA$^+$ removes the Weaknesses of RC4 KSA

In this section, we discuss how the new design avoids many weaknesses of the original RC4 KSA. We performed extensive experiments to verify that KSA$^+$ is indeed free from the weaknesses of the RC4 KSA. In all our experiments that are presented in this section, we use null IV, i.e., $iv[y] = 0$ for all y. We could not find any weakness with such null IV as well as with randomly chosen IV's.

Removal of secret key correlation with the permutation bytes Let us first discuss on Layer 2 of the KSA+. The deterministic index i is moved from the middle to the left end so that the values in the first quarter of the permutation, which were biased to linear combination of the secret key bytes, are swapped. This helps in removing the biases in the initial values of Item (1) described in Section 3.1. This is followed by a similar operation in the second half of the permutation to get rid of the biases of the inverse permutation as described in Item (4). Next, the XOR operation helps further to wipe out these biases. The biases considering the nested indexing mentioned in Item (3) and Item (4) arise due to the biases of direct indexing. So, the removal of the biases at the direct indices of S_N and S_N^{-1} gets rid of those at the nested indices also.

The bias of Item (2), which is a generalization of the bias of Item (1), originates from the incremental update of j which helps to form a recursive equation connecting the key bytes.

In the new design, the bit-by-bit XOR operation as well as the zig-zag scrambling in Layer 3 prevents in forming such recursive equations connecting the key bytes and the permutation bytes.

We could not find any correlation between $S_{N^+}[y]$ (also, $S_{N^+}[S_{N^+}[y]]$, $S_{N^+}[S_{N^+}[S_{N^+}[y]]]$, ...) with f_y. We believe that with our design, it is not possible to get correlation of the permutation bytes with any function combining the secret key bytes.

In Section 2, the relation between the biases of the inverse permutation and the movement frequency of the permutation values has been discussed in detail. The following experimental results show that, such weaknesses of RC4 KSA are absent in our design. Averaging over 100 million runs of KSA$^+$ with 16 bytes key in each run, we find that as v increases from 0 to 255, E_v decreases from 4.99 to 3.31 after the end of Layer 2 and from 6.99 to 5.31 after the end of Layer 3. Table 1 shows the individual as well as the incremental effect of each of Layer 2 and Layer 3, when they act upon the identity permutation S_0 and the permutation S_N obtained after Layer 1. The data illustrate that the effect of Layer 2 or Layer 3 over identity permutation S_0 is similar as Layer 1. However, after Layer 1 is over (when we have somewhat random permutation S_N coming out of RC4 KSA), each of Layer 2 and Layer 3 individually enforces each value in the permutation to be touched uniformly (approximately twice) when the average is considered over many runs. Thus, each layer incrementally shifts the graph of E_v versus v in the positive Y-direction approximately by an amount of 2, as is illustrated in Figure 2.

		avg	*sd*	*max*	*min*
RC4 KSA (KSA$^+$ L1)	Theory	2.0025	0.4664	3.0000	1.3700
	Experiment	2.0000	0.4655	2.9959	1.3686
KSA$^+$ L2 (Experiment)	L2 on S_0	2.0000	0.4658	2.9965	1.3683
	L2 on S_N	2.0000	0.0231	2.0401	1.9418
	L1 + L2	4.0000	0.4716	4.9962	3.3103
KSA$^+$ L3 (Experiment)	L3 on S_0	2.0000	0.4660	3.0000	1.3676
	L3 on S_N	2.0000	0.0006	2.0016	1.9988
	L1 + L2 + L3	6.0000	0.4715	6.9962	5.3116

Table 1. Average, Standard Deviation, Maximum and Minimum of the expectations E_v over all v between 0 and 255. Here Lr means Layer r, $r = 1, 2, 3$.

To arrive at uniform values of the expectations can be done easily with normal RC4, by keeping a count of how many times each element is touched and performing additional swaps involving the elements that have been touched less number of times. However, this will require additional space and time. In normal RC4, many permutation elements are touched only once (especially those towards the right end of the permutation), leaking information on j in the inverse permutation. Our target is to prevent this by increasing the number of times each element is touched, without keeping any additional space such as a counter. The data in Table 1 as well as Figure 2 show that this purpose is served using our strategy.

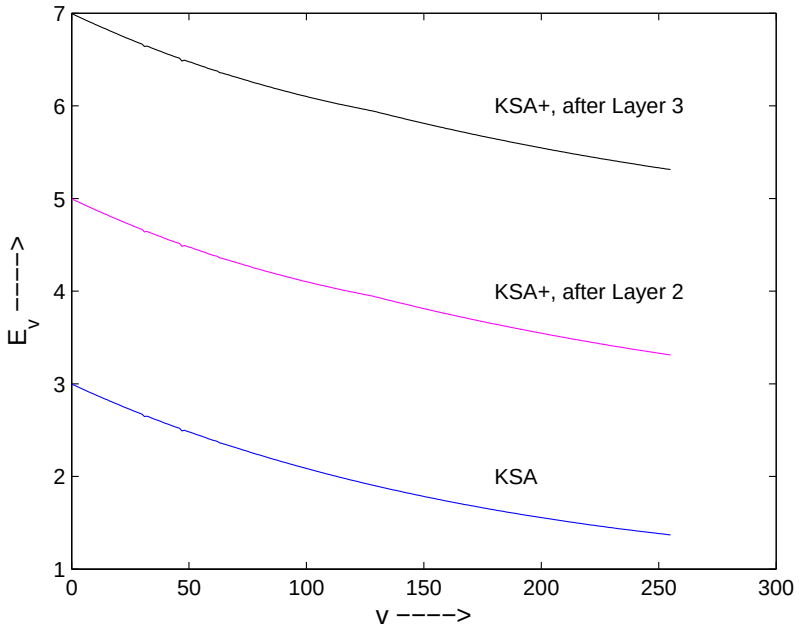

Fig. 2. KSA$^+$: E_v versus v, $0 \leq v \leq 255$.

How random is S_{N+} Now we present experimental evidences to show how the biases of Item (5) in RC4 KSA are removed. We compare the probabilities $P(S[u] = v)$ for $0 \leq u, v \leq 255$ from standard KSA and our KSA$^+$. All the experiments are performed with 100 million runs, each with a randomly chosen secret key of length 16 bytes and null IV.

One may refer to the graphs in Figure 3 to note that $P(S[u] = v)$ is flattened after Layer 2 of KSA$^+$. However, some non-uniformities are still there. After Layer 3, the graph becomes completely flat, indicating that there is no bias in the probabilities $P(S[u] = v)$. The maximum and minimum values of the probabilities as well as the standard deviations summarised in Table 2 elaborate this fact further.

		avg	*sd*	*max*	*min*
RC4 KSA	Theory [27, Theorem 1]	0.003901	0.000445	0.005325	0.002878
	Experiment	0.003906	0.000448	0.005347	0.002444
KSA$^+$ (Experiment)	After Layer 2	0.003906	0.000023	0.003983	0.003803
	After Layer 3	0.003906	0.000006	0.003934	0.003879

Table 2. Average, Standard Deviation, Maximum and Minimum of the Probabilities $P(S[u] = v)$ over all u and v between 0 and 255. Note that $\frac{1}{N} = 0.003906$ for $N = 256$.

In [19, Page 67], it was mentioned that the RC4 KSA need to be executed approximately 6 times in order to get rid of these biases. Whereas, in our case, we need to run KSA effectively 3 times.

On introducing the IV's The IV-mode attacks, mentioned in Item (6) of Section 3.1, succeed because in the original RC4, IV's are either prepended or appended with the secret key. As the Layer 2 shows, in KSA$^+$, we use the IV's in the middle and also the corresponding key bytes are added in the updation of j. In Layer 2, $2l$ many operations involve IV values, but $N - 2l$ many operations do not. Moreover, after the use of IV, we perform a third layer of zig-zag scrambling where no use of IV is made. This almost eliminates the possibility of chosen IV attack once the key scheduling is complete.

SSL protocol bypasses the WEP attack [6, 33] by generating the encryption keys used for RC4 by hashing (using both MD5 and SHA-1) the secret key and the IV together, so that different sessions have unrelated keys [32]. Since our KSA$^+$ is believed to be free from the IV-weaknesses, it can be used without employing hashing. Thus, the cost of hashing can be utilized in the extra operations in Layer 2 and Layer 3. This conforms to our design motivation to keep the basic structure of RC4 KSA and still avoid the weaknesses.

On retaining the standard KSA in Layer 1 One may argue that Layer 1 is not necessary and Layer 2, 3 would have taken care of all the existing weaknesses of RC4. While this may be true, these two layers, when operated on identity permutation, might introduce some new weaknesses not yet known. It is a fact that RC4 KSA has some weaknesses, but it also reduces the key correlation with the permutation bytes and other biases at least to some extent compared to the beginning of the KSA. In the process, it randomizes the permutation to a certain extent. The structure of RC4 KSA is simple and elegant and easy to analyze. We first let this KSA run over the identity permutation, so that we can target the exact biases that are to be removed in the subsequent layers. In summary, we wanted to keep the good features of RC4 KSA, and remove only the bad ones.

4 PRGA$^+$: Modifications to RC4 PRGA

There are a number of important works related to the analysis of the RC4 PRGA. The main directions of cryptanalysis in this area are

1. finding correlations between the keystream output bytes and the secret key [31, 39, 24, 15] and key recovery in the IV mode [6, 17, 11, 34, 38] (these exploit the weaknesses of both the KSA and the PRGA),
2. recovering the RC4 permutation from the keystream output bytes [12, 35, 21] and
3. identifying distinguishers [16, 29, 18].

In Section 3.2, we proposed KSA$^+$ in such a manner that one cannot get secret key correlations from the permutation bytes. This guarantees that the keystream output bytes, which are some combination of the permutation bytes, cannot have any correlation with the secret

key. As argued in Section 3.3, IV's are used in such a way, that they cannot be easily exploited to mount an attack. So we target the other two weaknesses, enlisted in Item (2) and (3) above, in our design of PRGA$^+$.

As we have discussed earlier, a very recent and improved result [21] shows that the permutation can be recovered in around 2^{241} complexity. This shows that RC4 is not secured for secret key size greater than 30 bytes. Given this situation, use of RC4 with secret key size of 16 bytes seems quite secure, but at the same time we need to revisit the RC4 PRGA for a better design with minimum changes and keeping the structure more or less same.

For any byte b, b_R^n (respectively b_L^n) denotes the byte after right (respectively left) shifting b by n bits. For $r \geq 1$, we denote the permutation, the indices i, j and the keystream output byte after round r of the PRGA (or PRGA$^+$) by S_r^G, i_r^G, j_r^G and z_r respectively.

The main idea behind this design of PRGA$^+$ is masking the output byte such that it is not directly coming out from any permutation byte. Two bytes from the permutation are added modulo 256 (a nonlinear operation) and then the outcome is XOR-ed with a third byte (for masking non-uniformity). Introducing additional $S[t'], S[t'']$, over the existing $S[t]$ in RC4, makes the running time of PRGA$^+$ little more than that of RC4 PRGA (see Section 5 for details). Note that the evolution of the permutation S in PRGA$^+$ stays exactly the same as in RC4 PRGA. We introduce a constant value $0xAA$ (equivalent to 10101010 in binary) in t', as without this, if j^G becomes 0 in rounds 256, 512, ... (i.e., when $i^G = 0$), then t and t' in such a round become equal with probability 1, giving an internal bias.

RC4 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 $z = S[t]$;

PRGA$^+$

Initialization:
$i = j = 0$;
Keystream Generation Loop:
$i = i + 1$;
$j = j + S[i]$;
Swap($S[i]$, $S[j]$);
$t = S[i] + S[j]$;
$t' = (S[i_R^3 \oplus j_L^5] + S[i_L^5 \oplus j_R^3]) \oplus 0xAA$;
$t'' = j + S[j]$;
Output $z = (S[t] + S[t']) \oplus S[t'']$;

4.1 Resisting permutation recovery attacks

The basic idea of cryptanalysis in [21] is as follows. Corresponding to a window of $w + 1$ keystream output bytes, one may assume that all the j's are known, i.e., $j_r^G, j_{r+1}^G, \ldots, j_{r+w}^G$ are known. Thus w many $S_r^G[i_r^G]$ will be available from $j_{r+1}^G - j_r^G$. Then w many equations of the form $S_r^{G^{-1}}[z_r] = S_r^G[i_r^G] + S_r^G[j_r^G]$ will be found where each equation contains only two unknowns. The idea of [12] (having complexity around 2^{779} to 2^{797}) actually considered four unknowns $j^G, S^G[i^G], S^G[j^G], S^{G^{-1}}[z]$.

Our design does not allow the strategy of [21] as $S^G[S^G[i^G] + S^G[j^G]]$ is not exposed directly, but it is masked by several other quantities. To form the equations as given in [21],

one first needs to guess $S^G[t], S^G[t'], S^G[t'']$ and looking at the value of z, there is no other option than to go for all the possible choices. The same permutation structure of S in RC4$^+$ can be similarly exploited to get the good patterns [21, Section 3], but introducing additional t', t'', we ensure the non-detectability of such a pattern in the keystream and thus the idea of [21, Section 4] will not work.

Information on permutation bytes is also leaked in the keystream via the Glimpse Main Theorem [10, 17], which states that during any PRGA round, $P(S[j] = i - z) = P(S[i] = j - z) \approx \frac{2}{N}$. The assumption $i = S[i] + S[j]$ holds with a probability $\frac{1}{N}$, leading to the bias $P(S[j] = i - z) = \frac{1}{N} \cdot 1 + (1 - \frac{1}{N}) \cdot \frac{1}{N} = \frac{2}{N} - \frac{1}{N^2} \approx \frac{2}{N}$. To obtain such biases in PRGA$^+$, one need to have more assumptions of the above form. Thus, Glimpse like biases of PRGA$^+$, if at all exist, would be much weaker.

4.2 Resisting distinguishing attacks

In [16], it was proved that $P(z_2 = 0) = \frac{2}{N}$ instead of the uniformly random case of $\frac{1}{N}$. This originates from the fact that when $S_N[2] = 0$ and $S_N[1] \neq 2$ after the KSA, the second keystream output byte z_2 takes the value 0. Based on this, they showed a distinguishing attack and a ciphertext-only attack in broadcast mode. We avoid this kind of situation in our design. As a passing remark, we like to present an experimental result. Hundred million secret keys of length 16 byte are generated and 1024 rounds of PRGA are executed for each such key. The empirical evidences indicate that $P(z_r = v) = \frac{1}{N}, 1 \leq r \leq 1024, 0 \leq v \leq N-1$.

In the work [29], it was observed that $P(z_1 = z_2) = \frac{1}{N} - \frac{1}{N^2}$, which leads to a distinguishing attack. Even after extensive experimentation, we could not observe such bias in the keystream. The same experiment described above supported that $P(z_r = z_{r+1})$ is uniformly distributed for $1 \leq r \leq 1023$.

In [18], it has been shown that getting strings of pattern $ABTAB$ (A, B are bytes and T is a string of bytes of small length G, say $G \leq 16$) are more probable in RC4 keystream than in random stream. In uniformly random keystream, the probability of getting such pattern irrespective of the length of T is $\frac{1}{N^2}$. It has been shown in [18, Theorem 1] that for RC4, the probability of such an event is $\frac{1}{N^2}(1 + \frac{e^{\frac{-4-8G}{N}}}{N})$, which is above $\frac{1}{N^2}$, but less than $\frac{1}{N^2} + \frac{1}{N^3}$. This result is based on the fact that the permutation values in locations that affect the swaps and the selection of output bytes in both pairs of rounds that are G-round apart, remain unchanged with high probability during the intermediate rounds. The permutation in PRGA$^+$ evolves in the same way as RC4 PRGA, but the keystream output generation in PRGA$^+$ is different, which does not allow the pattern AB to propagate down the keystream with higher probability for smaller interval lengths (G). In [18], 2^{16} keystreams of size 2^{24} each were used to observe these biases effectively. The simulation on PRGA$^+$ reveals that it is free from these biases.

5 Performance Evaluation

We evaluated the performance of our new design using the *eSTREAM testing framework* [3]. The C-implementation of the testing framework was installed in a machine with Intel(R)

Pentium(R) 4 CPU, 2.8 GHz Processor Clock, 512 MB DDR RAM on Ubuntu 7.10 (Linux 2.6.22-15-generic) OS. A benchmark implementation of RC4 is available within the test suite. We implemented our modified RC4, which we call RC4$^+$, that incorporates both KSA$^+$ and PRGA$^+$, maintaining the API compliance of the suite. Test vectors were generated in the NESSIE [23] format.

The results presented below correspond to tests with 16 bytes secret key and null IV using the *gcc_default_O3-ual-ofp* compiler. As per the test, RC4 KSA took 16944.70 cycles/setup, whereas the KSA$^+$ of RC4$^+$ took 49823.69 cycles/setup. The stream encryption speed for RC4 and RC4$^+$ turned out to be 14.39 cycles/byte and 24.51 cycles/byte respectively[1]. Thus, we can claim that the running time of our KSA$^+$ is approximately $\frac{49823.69}{16944.70} = 2.94$ times than that of RC4 KSA and the running time of one round of our PRGA$^+$ is approximately $\frac{24.51}{14.39} = 1.70$ times than that of RC4 PRGA.

6 Conclusion

Though RC4 can be stated in less than ten lines, newer weaknesses are being discovered every now and then even after twenty years of its discovery. This raises the need for a new design of a stream cipher, which would be as simple as the description of RC4, yet devoid of the existing weaknesses of RC4. This is the target of this paper. We present a three-layer architecture of the scrambling phase after the initialization, which removes many weaknesses of the KSA. We also add a few extra steps in the PRGA to strengthen the cipher. Experimental results also supports our claim.

Even after our arguments and empirical evidences, the security claim of RC4$^+$ is a conjecture, as is the case with many of the existing stream ciphers. We could not observe any immediate weakness of the new design and the cipher is subject to further analysis.

Acknowledgments: The authors like to thank Dr. Alexander Maximov for discussion and valuable comments and Mr. Snehasis Mukherjee, Indian Statistical Institute, Kolkata for his support in the preparation of the graphs.

References

1. M. Akgun, P. Kavak and H. Demirci. New Results on the Key Scheduling Algorithm of RC4. Indocrypt 2008. A sketch of this work has been presented in Eurocrypt 2008 Rump Session, available at `http://www.iacr.org/conferences/eurocrypt2008v/index.html` [last accessed on July 18, 2008].
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 Berlin / Heidelberg.
3. C. D. Cannire. eSTREAM testing framework. Available at `http://www.ecrypt.eu.org/stream/perf` [last accessed on September 19, 2008].
4. eSTREAM, the ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream` [last accessed on July 18, 2008].

[1] In the testing framework, the cycles are measured using the RDTSC (Read Time Stamp Counter) instruction of X86 assembly language. This instruction returns a 64-bit value in registers EDX:EAX that represents the count of ticks from processor reset.

5. S. R. Fluhrer and D. A. McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. FSE 2000, pages 19-30, vol. 1978, Lecture Notes in Computer Science, Springer-Verlag.
6. S. R. Fluhrer, I. Mantin and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. SAC 2001, pages 1-24, vol. 2259, Lecture Notes in Computer Science, Springer-Verlag.
7. J. Golic. Linear statistical weakness of alleged RC4 keystream generator. EUROCRYPT 1997, pages 226-238, vol. 1233, Lecture Notes in Computer Science, Springer-Verlag.
8. G. Gong, K. C. Gupta, M. Hell and Y. Nawaz. Towards a General RC4-Like Keystream Generator. CISC 2005, pages 162-174, vol. 3822, Lecture Notes in Computer Science, Springer-Verlag.
9. J. Hong and P. Sarkar. New Applications of Time Memory Data Tradeoffs. ASIACRYPT 2005, pages 353-372, vol. 3788, Lecture Notes in Computer Science, Springer-Verlag.
10. R. J. Jenkins. ISAAC and RC4. 1996.
Available at `http://burtleburtle.net/bob/rand/isaac.html` [last accessed on July 18, 2008].
11. A. Klein. Attacks on the RC4 stream cipher. *Designs, Codes and Cryptography*, pages 269-286, vol. 48, no. 3, September, 2008.
12. 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-Verlag.
13. D. E. Knuth. *The Art of Computer Programming*, vol. 3, Addison-Wesley Publishing Company, 1973.
14. LAN/MAN Standard Committee. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1999 edition. IEEE standard 802.11, 1999.
15. 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 Berlin / Heidelberg.
16. I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. FSE 2001, pages 152-164, vol. 2355, Lecture Notes in Computer Science, Springer-Verlag.
17. I. Mantin. A Practical Attack on the Fixed RC4 in the WEP Mode. ASIACRYPT 2005, pages 395-411, volume 3788, Lecture Notes in Computer Science, Springer-Verlag.
18. I. Mantin. Predicting and Distinguishing Attacks on RC4 Keystream Generator. EUROCRYPT 2005, pages 491-506, vol. 3494, Lecture Notes in Computer Science, Springer-Verlag.
19. I. Mantin. Analysis of the stream cipher RC4. Master's Thesis, The Weizmann Institute of Science, Israel, 2001.
20. A. Maximov. Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers. FSE 2005, pages 342-358, vol. 3557, Lecture Notes in Computer Science, Springer-Verlag.
21. A. Maximov and D. Khovratovich. New State Recovering Attack on RC4 (Full Version). IACR Eprint Server, eprint.iacr.org, number 2008/017, Jan 10, 2008. (To Appear in Crypto 2008 under the title "New State Recovery Attack on RC4").
22. I. Mironov. (Not So) Random Shuffles of RC4. CRYPTO 2002, pages 304-319, vol. 2442, Lecture Notes in Computer Science, Springer-Verlag.
23. New European Schemes for Signatures, Integrity, and Encryption.
Available at `https://www.cosic.esat.kuleuven.be/nessie` [last accessed on September 19, 2008].
24. 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 (WCC) 2007, pages 285-294. To appear in *Designs, Codes and Cryptography*, special issue of in memory of Hans Dobbertin, DOI 10.1007/s10623-008-9177-7.
25. 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 Berlin / Heidelberg.
26. G. Paul and S. Maitra. RC4 State Information at Any Stage Reveals the Secret Key. IACR Eprint Server, eprint.iacr.org, number 2007/208, June 1, 2007. This is an extended version of [25].
27. 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) 2007, pages 100-109, vol. 4851, Lecture Notes in Computer Science, Springer Berlin / Heidelberg.
28. S. Paul and B. Preneel. Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator. INDOCRYPT 2003, pages 52-67, vol. 2904, Lecture Notes in Computer Science, Springer-Verlag.
29. S. Paul and B. Preneel. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. FSE 2004, pages 245-259, vol. 3017, Lecture Notes in Computer Science, Springer-Verlag.
30. E. M. Reingold, J. Nievergelt and N. Deo. *Combinatorial Algorithms: Theorey and Practice*. Prentice Hall, 1977.
31. 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://groups.google.com/group/sci.crypt.research/msg/078aa9249d76eacc?dmode=source [last accessed on July 18, 2008].
32. RSA Lab Report. RSA Security Response to Weaknesses in Key Scheduling Algorithm of RC4. Available at http://www.rsa.com/rsalabs/node.asp?id=2009 [last accessed on July 18, 2008].
33. A. Stubblefield, J. Ioannidis and A. D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. AT&T Labs Technical Report TD-4ZCPZZ, August 6, 2001.
34. 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 [last accessed on July 18, 2008].
35. V. Tomasevic, S. Bojanic and O. Nieto-Taladriz. Finding an internal state of RC4 stream cipher. *Information Sciences*, pages 1715-1727, vol. 177, 2007.
36. Y. Tsunoo, T. Saito, H. Kubo, M. Shigeri, T. Suzaki and T. Kawabata. The Most Efficient Distinguishing Attack on VMPC and RC4A. SKEW 2005. Available at http://www.ecrypt.eu.org/stream/papers.html [last accessed on July 18, 2008].
37. Y. Tsunoo, T. Saito, H. Kubo and T. Suzaki. A Distinguishing Attack on a Fast Software-Implemented RC4-Like Stream Cipher. *IEEE Transactions on Information Theory*, page 3250-3255, vol. 53, issue 9, September 2007.
38. 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 Berlin / Heidelberg.
39. 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 [last accessed on July 18, 2008].
40. B. Zoltak. VMPC One-Way Function and Stream Cipher. FSE 2004, pages 210-225, vol. 3017, Lecture Notes in Computer Science, Springer-Verlag.

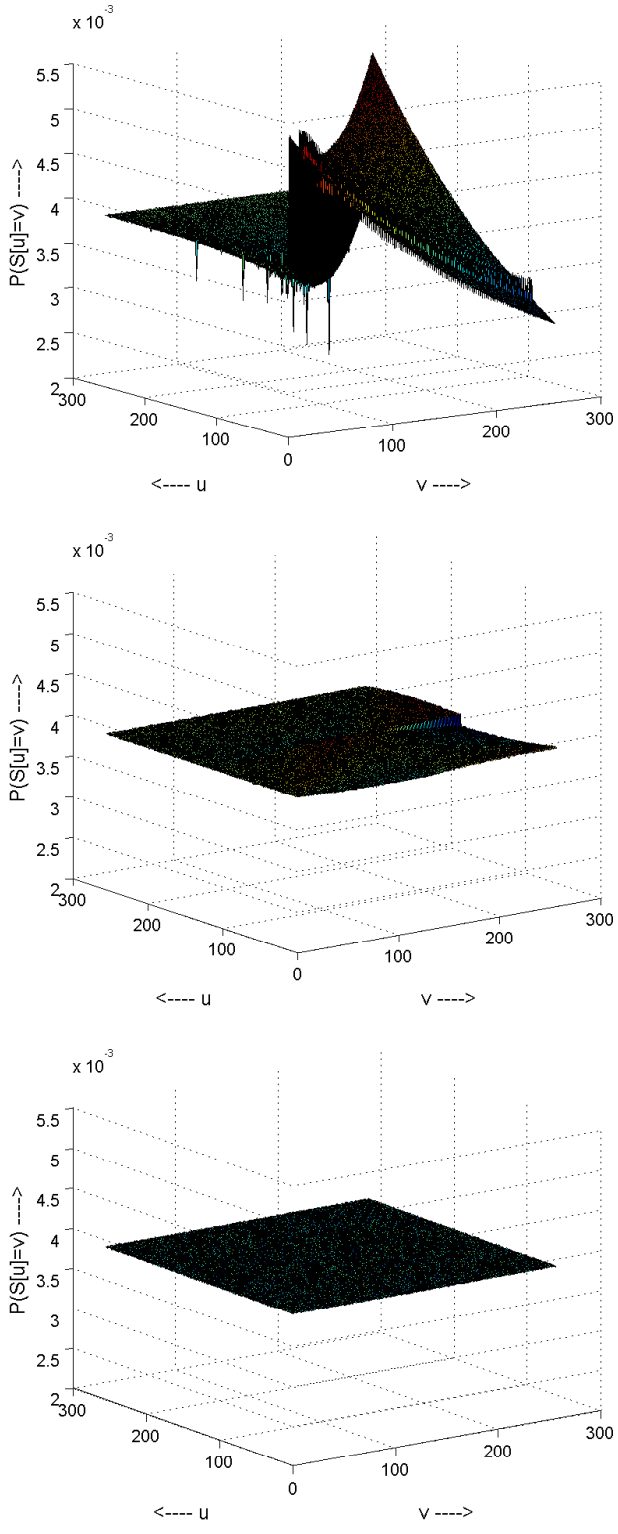

Fig. 3. $P(S[u] = v)$ versus $0 \leq u, v \leq 255$. Top: after RC4 KSA; Middle: KSA$^+$, after Layer 2; Bottom: KSA$^+$, after Layer 3.