Similar State Tables and Related Keys in RC4

Hüseyin Demirci[1], Eyüp Serdar Ayaz[2], and Ali Aydın Selçuk[2]

[1] Tübitak UEKAE, 41470 Gebze, Kocaeli, Turkey
[2] Department of Computer Engineering
Bilkent University, 06800, Ankara, Turkey
`huseyind@uekae.tubitak.gov.tr, {serdara,selcuk}@cs.bilkent.edu.tr`

Abstract. In this paper we analyze the key scheduling algorithm of RC4, and investigate the relations between bytes of the encryption key and the entries of the state table. We give a method to derive the state table partially from a partially known encryption key. We also analyze a class of related keys which yield similar state tables and produce key streams that are similar at the initial bytes.

1 Introduction

RC4 is probably the simplest and the most widely-used software stream cipher in the world today. It has only a few lines of code, but it is also quite secure—to a large extent, despite some weaknesses in its key schedule—due to the highly non-linear structure of the swap function in the algorithm.

In this paper we focus on the key scheduling algorithm of RC4 and discuss certain avenues of attack related to its key schedule.

The rest of this paper is organized as follows: In Section 2 we briefly review the RC4 algorithm. In Section 3 we summarize the existing work on RC4. In Section 4, we explore several features of the key schedule and discuss the possibilities of translating these features into attacks on RC4.

2 RC4 Stream Cipher

The operation of RC4 is defined on n-bit words where n is a variable integer parameter. The state of the cipher is defined by a 2^n-word table S that consists of a permutation of integers from 0 to $2^n - 1$. The state table S is initialized by the key scheduling algorithm (KSA) which shuffles S using the encryption key K. The key is l-word long, which is also a variable parameter. Once the key schedule is complete, the pseudo-random generation algorithm (PRGA) is run which produces a pseudo-random running key stream from the S table while simultaneously shuffling the table. The KSA and PRGA algorithms are given below. All additions in the algorithms are modulo 2^n.

The most common version of RC4 uses $n = 8$ with $l = 16$, and the table S consists of 256 elements. In this paper we will also assume these values unless otherwise is stated.

Algorithm 1 Key Scheduling Algorithm (KSA)

```
for $i = 0$ to $2^n - 1$ do
    $S[i] \leftarrow i$
end for
$i \leftarrow 0$
$j \leftarrow 0$
for $i = 0$ to $2^n - 1$ do
    $j \leftarrow j + S[i] + K[i \bmod l]$
    swap($S[i]$, $S[j]$)
end for
```

Algorithm 2 Pseudo-Random Generation Algorithm (PRGA)

```
$i \leftarrow 0$
$j \leftarrow 0$
loop
    $i \leftarrow i + 1$
    $j \leftarrow j + S[i]$
    swap($S[i]$, $S[j]$)
    $t \leftarrow S[i] + S[j]$
    output $S[t]$
end loop
```

3 Previous Work on RC4

Different kinds of attacks have been applied to RC4, including attacks both on the key schedule and on the pseudo-random generation parts of the cipher.

Distinguishing attacks try to observe a fingerprint of RC4 output by distinguishing RC4 from a random source. Golic first showed a weak distinguishing linear property on long RC4 key streams [4].

Later stronger statistical properties were observed by Fluhrer and McGrew [3]. They observed biases in the digraph distribution with respect to the increment index i. This not only leads to a distinguisher of the cipher output, but also for some "fortuitous states", it reveals information about the secret table.

Mister and Tavares [12] analyzed the cycle structure of RC4 and suggested a tracking algorithm to fill the S table given a portion of the running key.

Finney [1] observed a small cycle in the running key generation mechanism. If RC4 was not designed carefully, one of every 2^{16} keys would fall into a cycle of length $255 \cdot 256$.

Mironov [11] analyzed the shuffling structure and observed that it has a non-uniform distribution. His conclusion was that the first 512 bytes of the running key should be avoided to come to a uniform stage.

Puduvkino [13] worked the number of initial states of RC4 cipher with the same cycle structure and has shown that this number is more than expected.

The best results are due to Mantin. He observed some biases in the first and second bytes of the running key [10]. He also observed several weaknesses regarding certain uses of an IV in RC4. With these observations, practical attacks were designed and applied on the WEP protocol [2]. His thesis [8] is a collection of his work up to its date of writing.

Roos [15] and Wagner [17] observed some weak key classes for which the first entries of the table depend only on a few bytes of the key. Therefore, for these classes of keys, the key can be derived faster than exhaustive search. Roos observed that with a significant probability the first n entries of the table are determined by the first n elements of the key, which we will use in our research.

Grosul and Wallach [5] observed that, if the key size is as large as the table, a pair of keys which are identical until the last two bytes produce running key streams that are very similar in the first 256 bytes.

Knudsen et al. [7] developed a backtracking algorithm which fills the initial S table given a small part the running key stream. The complexity of this attack seems to be less than the square root of all the possible states.

Recently Mantin [9] presented a new distinguishing attack using the bias in the digraph distribution of the cipher. By this bias, one can predict the next bit or byte of the running key if 2^{45} or 2^{50} output words are known respectively.

4 Analysis of the RC4 Key Schedule

In this study we mainly focus on two questions:

1. Is there a way to guess a non-trivial portion of the table after key schedule?
2. Given two initial tables of RC4 which are identical on a certain number of entries, can we observe a significant correlation between the running key sequences they produce?

If we can answer both of the questions positively, this means that a divide and conquer attack may be possible by observing the output key stream. If only the answer to the first question is positive, this points to a weakness in the key schedule of RC4. In the following section, we answer the first question positively. We also observe that similar S tables may lead to a bias in the running key.

In what follows, we analyze the similarity of (i) the j sequences, (ii) the S tables, and (iii) the running key sequences, given a certain similarity relation in the encryption key.

4.1 Generating Similar Tables

In this section, we investigate the question that whether some useful information can be derived on the initial S table from some partial knowledge on the encryption key K. In particular, assume that we know the first m bytes of the key for some $m < l$. ¿From Roos [15] we know that the first m elements of the table can be determined with a non-trivial probability:

Theorem 1 (Roos [15]). *For $m < l$, the probability that the m'th entry of the initial S table is determined by the first m words of the key $K[0 \ldots m-1]$ is greater than or equal to 36.8%.*

Proof. During the key schedule, there are two ways that an entry of the S table can be swapped: It must be indexed by either i or j. After the mth entry is swapped by $i = m - 1$, there is a probability of $(255/256)^{l-m} \geq (255/256)^{255} = 0.3686$ that j will not address $S[m-1]$ again. Also note that the first m swaps are completely determined by $K[0 \ldots m-1]$. $\square$

This theorem gives some probabilistic information on the table entries in terms of the key bytes. But when only a small portion of the key is known, information provided will be limited to the first few entries of the table. After that point, knowledge of the j sequence will be disrupted and we cannot continue to trace the swaps further.

Nevertheless, note that in the key schedule the bytes of the encryption key are reused periodically. Hence if we can guess the j value after the lth iteration, we will be able to conduct another m swaps using the first m bytes of K. By this way we may obtain an S table that is close to the actual case. The following algorithm describes this procedure:

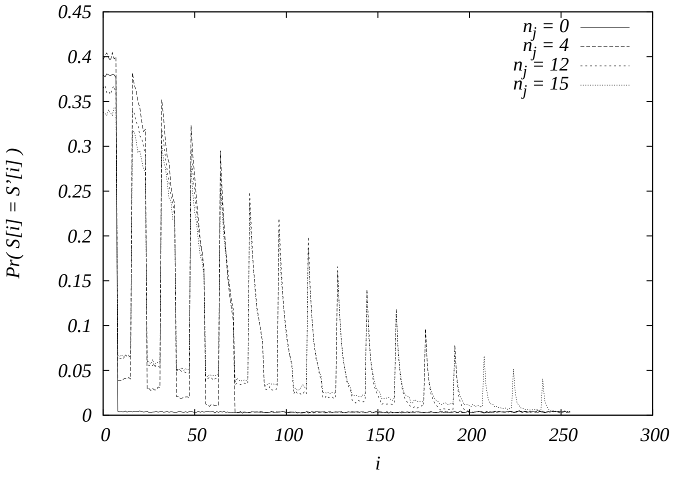
1. For the first m steps, swap the S table according to the known key bytes.
2. Skip the next $l - m$ iterations without any swaps.
3. Guess a value for j at this point. Continue m iterations with the known key bytes. Skip another $l - m$ iterations without any swaps.
4. Repeat Step 3 for a specified number of times, say n_j, or until the key schedule is complete.

The procedure above can be translated into a divide-and-conquer attack on K where the first several bytes of the key can be recovered independent from the rest: For some specified values of m and n_j, search m bytes of K and n_j values of j exhaustively and compute the S table as described above. When the right value of $K[0 \ldots m-1]$ is encountered, a significantly higher similarity with the actual S table will be observed. Hence, the right key can be distinguished accordingly.

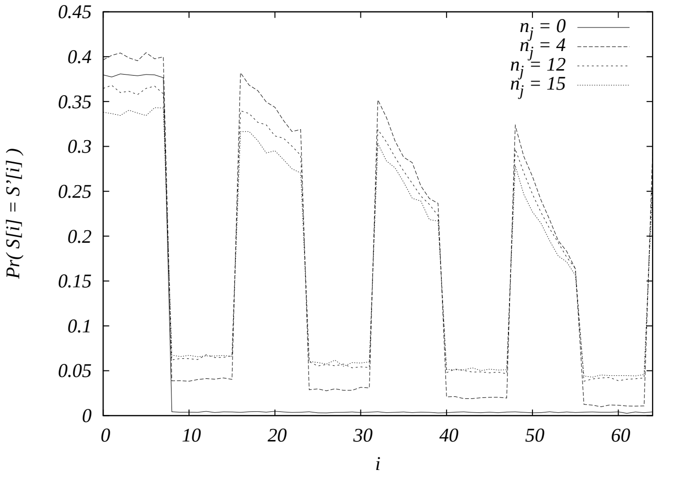
Of course, to complete this attack one would also need to translate the similarity between two S tables into some observable correlation between their output key streams.

We carried out various tests to measure the average similarity between two S tables produced as described above with $8 \leq m \leq 15$ and $0 \leq n_j \leq 15$ for $l = 16$. Here it may at first appear unnecessary or even useless to test the values where $m + n_j \geq l$ since this would mean a search more costly than an exhaustive search on K. In fact it does not have to be so: Each correctly guessed value of j carries some implicit information on the sum of the key bytes and enables possibly more informed guesses on the succeeding j values. Hence we chose to include the cases with $m + n_j \geq l$ in the tests as well.

In the tests conducted, the similarity probabilities for each value of m and n_j are calculated over 2^{14} random keys. The average probability of a match is calculated for each entry of the S tables. Figures 1–3 summarize the results of these experiments for some selected values of m and n_j.

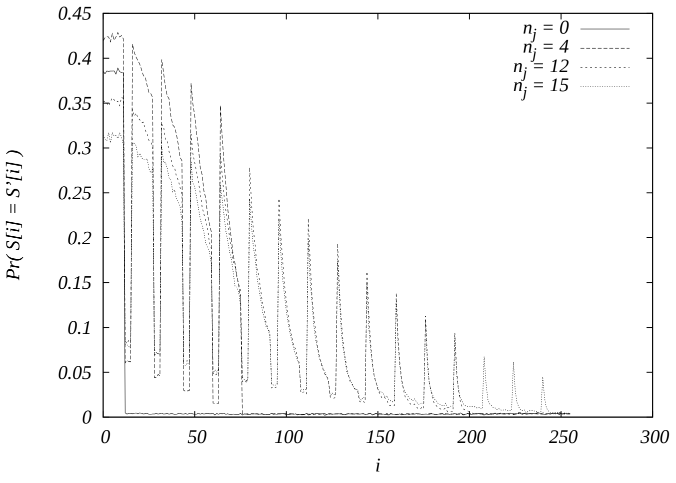


(a) The overall S table

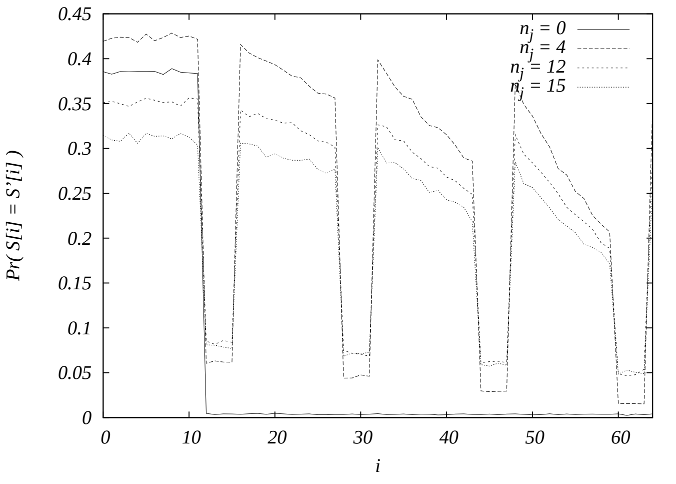


(b) Focus on $0 \leq i \leq 64$

Fig. 1. Probability of obtaining the right value at $S[i]$ with $m = 8$ known key bytes.

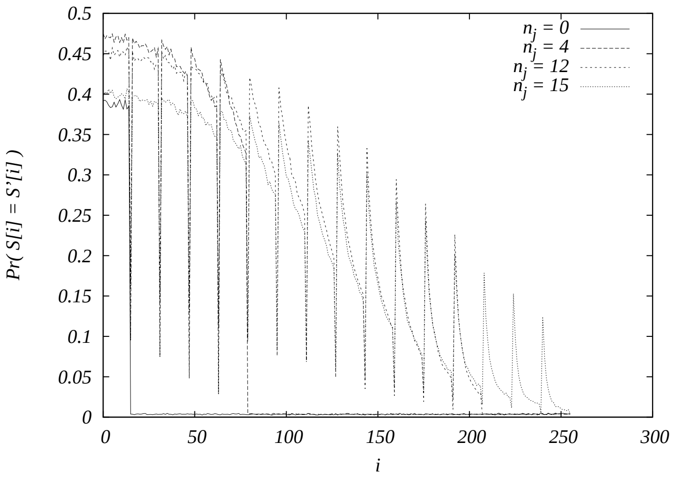


(a) The overall S table

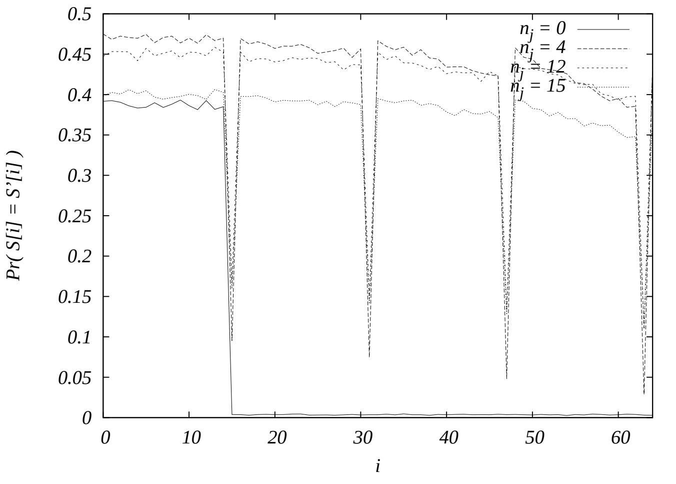


(b) Focus on $0 \leq i \leq 64$

Fig. 2. Probability of obtaining the right value at $S[i]$ with $m = 12$ known key bytes.



(a) The overall S table



(b) Focus on $0 \leq i \leq 64$

Fig. 3. Probability of obtaining the right value at $S[i]$ with $m = 15$ known key bytes.

Discussion of the Algorithm The table construction algorithm above utilizes several key ideas:

– The best way to approximate the swap function is by the swap function itself.
– Repeated use of the same key bytes in the KSA can be exploited coupled with a periodic guess of the j value.
– When the involved key byte is unknown, making no swap gives a better approximation than making a wrong swap.

The results of the experiments indicate that some partial information can be obtained on the state table from a partial knowledge of the key. For instance, for $m = 12$ and $n_j = 4$ in Figure 2, $m \times (n_j + 1) = 60$ entries of the table can be guessed with probabilities between 0.36 and 0.20, and hence we can expect to guess on average 13–15 entries of the table correctly. This observation can be used as a beginning state of the algorithm in [7] which fills the table using the running key stream from a partially known state table.

An important consequence of these results is that the KSA of RC4 is not a random permutation. In particular, the ith table entry $S[i]$ depends on $K[0 \ldots i']$, for $i' = i \bmod 16$, more heavily than on $K[i' + 1 \ldots 15]$.

As a final point, we can note that two tables with r identical entries can be expected to produce an identical output at a particular ith iteration of the PRGA with a probability about $(r/256)^3$—i.e., the probability of having identical $S[i]$, $S[j]$, $S[t]$. Furthermore, if i is known to be in the range where the two tables are mostly identical, this probability can be expected to be as high as $(r/256)^2$.

4.2 The Related Key Similarity

Grosul and Wallach [5] observed that, if the encryption key is as large as the S table, then two related keys that differ only in the last few bytes produce key streams that are very similar at the beginning. They left the construction of related keys in practical versions of RC4 where the key length l is typically much less than the table size 2^n as an open problem. In this section we investigate the related key scenarios in the commercial RC4 model where $l = 16$ and $n = 8$, and analyze the similarity of the j sequences, the initial S tables, and the running key streams. We show that related keys still exist, but in a probabilistic manner.

Lemma 1. *Let* $(K_0, \ldots, K_{14}, K_{15})$ *and* $(K_0, \ldots, K_{14} - \delta, K_{15} + \delta)$ *be two 16-byte RC4 keys for some* $1 \le \delta \le 255$. *Consider the* j *sequences of length 256 produced by the two keys during the key schedule. Then,*

i. The first 14 elements of the j *sequences are the same.*
ii. The 15th elements are always different.
iii. The 16th elements have a probability of $254/256$ *of being equal.*
iv. As for the next 16 elements; the probability of having the same two values is also very high for the 17th element; it slightly lessens from the 18th to the 30th; it is small for the 31st; and it again increases for the 32nd.

v. The succeeding elements follow the same pattern in 16-byte periods, but with a decreasing probability.

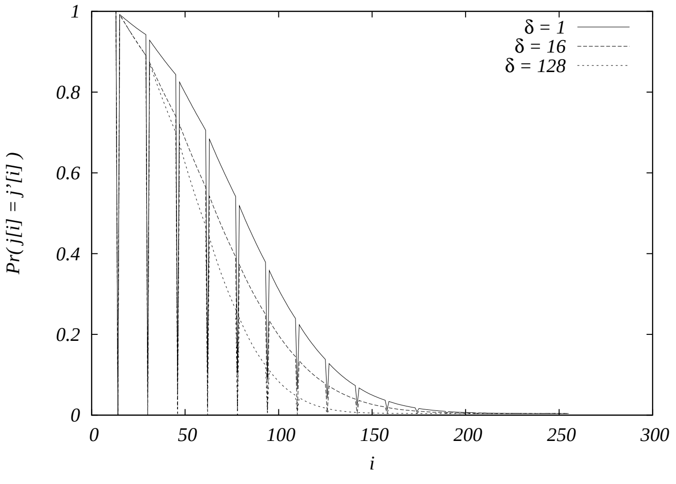
Proof. Since the first bytes are equal, this will produce the same result for the first 14 j values. Since $\delta \neq 0$, the 15th elements of the j sequences are different. But this change gets cancelled with $+\delta$ in the 16th element, if it is not swapped to a different position in the 15th step (which may happen with probability 2/256.) The next 16 elements follow the pattern in item (*iv*) given that the two j sequences re-synchronized at the 16th iteration. Then the rest follows by induction. □

The similarity of the j sequences in the key schedule results in a similarity between the corresponding S tables. This in turn may result in a similarity between the running key sequences produced.

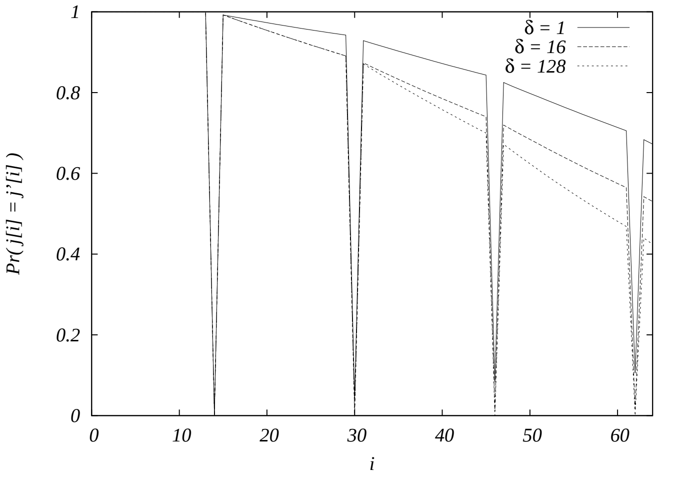
Of course, other key relations which produce a similar effect exist and can be found easily, such as a difference of $(-\delta, -\delta, 2\delta)$ on the last three key bytes. The $(-\delta, \delta)$ difference we investigated is the one that gives the most significant similarities among its kind.

We tested the similarity effects in a series of experiments where two 16-byte RC4 keys are taken which differ only on the last two bytes by a specified $(-\delta, \delta)$ value. The experiments are repeated for 2^{24} random runs for each value of $1 \leq \delta \leq 255$. The results are presented in Figure 4–6 for $\delta = 1, 16, 128$.

The results of the experiments indicate that, as in [5], the maximum similarity between the S tables is obtained for $\delta = 1$ and 255, and the similarity decreases while δ approaches 127 and 128. The related keys also produce related key streams, but this correlation is mostly limited to the first few bytes of the running key. We must mention that these values are on average. For the maximum values, we observed instances where more than the first 20 bytes of the two running key streams agreed.

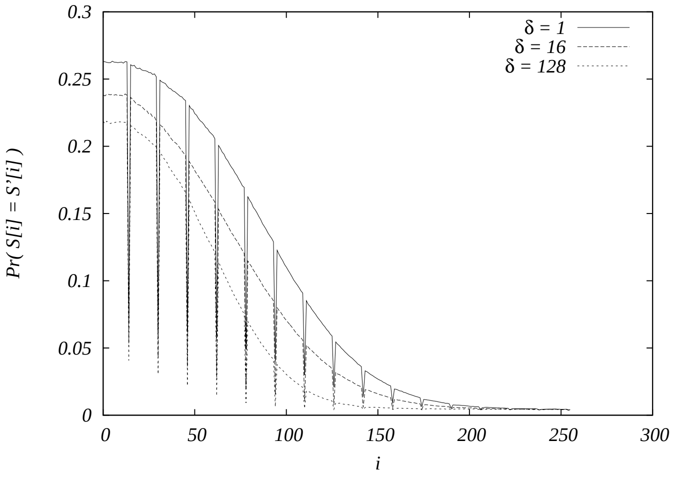


(a) $0 \leq i \leq 255$

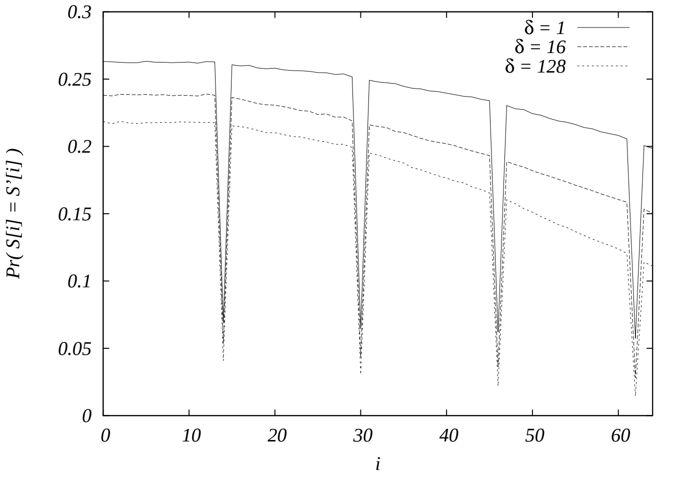


(b) $0 \leq i \leq 64$

Fig. 4. Probability of a match between the j values during the key schedule.

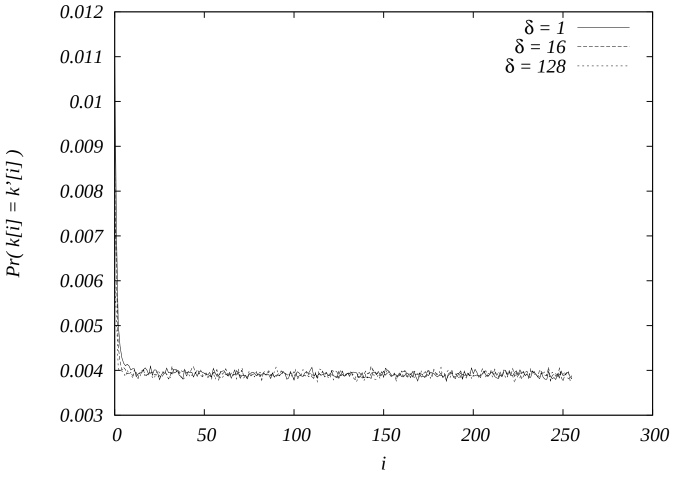


(a) $0 \leq i \leq 255$

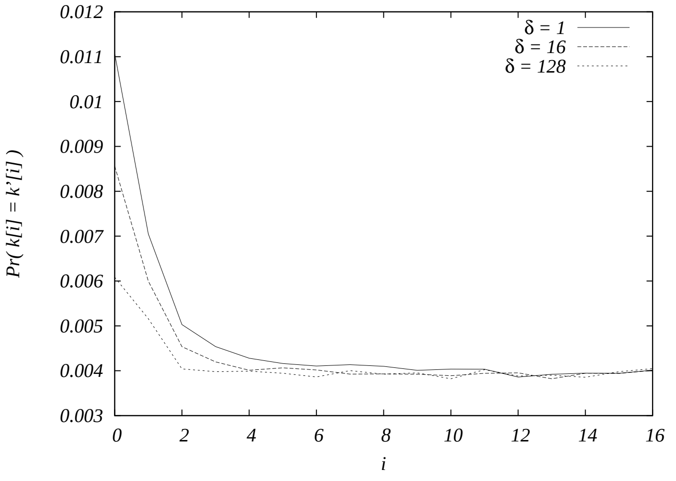


(b) $0 \leq i \leq 64$

Fig. 5. Probability of a match between the S table entries at the end of the key schedule.



(a) $0 \leq i \leq 255$



(b) $0 \leq i \leq 16$

Fig. 6. Probability of a match between the running key sequences generated.

5 Conclusion

In this paper, we studied the key scheduling algorithm of RC4 in several aspects. We proposed an algorithm to generate similar S tables in RC4 with a partial knowledge of the encryption key. These similar tables also produce similar running key sequences in the first few iterations of the PRGA algorithm. We also studied the related key properties of the algorithm, and showed that the related keys Grosul and Wallach [5] observed on a simplified version of the algorithm exist in practical versions of RC4 as well.

References

1. H. Finney, *An RC4 Cycle That Can't Happen*, sci.crypt posting, September 1994.
2. S. R. Fluhrer, I. Mantin and A. Shamir, *Weaknesses in the key scheduling algorithm of RC4*, LNCS 2259, SAC 2001, pp. 1-24, Springer-Verlag, 2001.
3. S. R. Fluhrer and D. A. McGrew, *Statistical Analysis of the Alleged RC4 Keystream Generator*, LNCS 1978, FSE 2000, pp. 19-30, Springer-Verlag, 2000.
4. J. D. Golic, *Linear Statistical Weakness Alleged RC4 Key-stream Generator*, LNCS 1233, EUROCRYPT '97, pp. 226-238, Springer-Verlag, 1997.
5. A. L. Grosul and D. S. Wallach, *A Related-key Cryptanalysis of RC4*, Technical Report-00-358, Department of Computer Science, Rice University, October 2000.
6. R. J. Jenkins, *ISAAC and RC4*, http://burtleburtle.net /bob/rand/isaac.html.
7. L. R. Knudsen, W. Meier, B. Prenel, V. Rijmen and S. Verdoolaege *Analysis Methods for (Alleged) RC4*, LNCS 1514, ASIACRYPT'98, pp. 327-341, Springer-Verlag, 1998.
8. I. Mantin *Analysis of the Stream Cipher RC4*, M. Sc. Thesis, The Weizmann Institute of Science, 2001.
9. I. Mantin, *Predicting and Distinguishing Attacks on RC4 Keystream Generator*, LNCS 3494, EUROCRYPT'2005, pp. 491-506, Springer-Verlag, 2005.
10. I. Mantin and A. Shamir, *A Practical Attack on Broadcast RC4*, LNCS 2355, FSE'2001, pp. 152-164, Springer-Verlag, 2001.
11. I. Mironov, *(Not So) Random Shuffles of RC4*, LNCS 2442 , CRYPTO'2002,pp. 304-319 , Springer-Verlag, 2002.
12. S. Mister and S. A. Tavares, *Cryptanalysis of RC4-like Ciphers*, LNCS 1556, SAC'98, pp. 131-143, Springer-Verlag, 1999.
13. M. Pudovkino, *The Number of Initial States of the RC4 Cipher with the Same Cycle Structure*, Cryptology ePrint Archive, 2002-171, IACR 2002.
14. R. Rivest, *RSA Security Response to Weaknesses in the Key Scheduling Algorithm of RC4*, Technical Note, RSA Data Security Inc, October, 2001.
15. A. Roos, *A Class of Weak Keys in the RC4 Stream Cipher*, sci.crypt posting, September, 1995.
16. A. Stubblefield, J. Ioannidis and A. D. Rubin, *Using the Fluhrer, Mantin and Shamir Attack to Break WEP*, Technical Report TD4ZCPZZ, AT&T Labs, August 2001.
17. D. Wagner, *Weak Keys in RC4*, sci.crypt posting, September, 1995.