Analysis Methods for (Alleged) RC4

Lars R. Knudsen[1], Willi Meier[2], Bart Preneel[3⋆], Vincent Rijmen[3], and Sven Verdoolaege[3]

[1] Department of Informatics, University of Bergen, N-5020 Bergen
[2] HTL Brugg-Windisch, CH-5210 Windisch
[3] SISTA/COSIC Lab, Dept. ESAT, K.U.Leuven, K. Mercierlaan 94, B-3001 Leuven

Abstract. The security of the alleged RC4 stream cipher and some variants is investigated. Cryptanalytic algorithms are developed for a known plaintext attack where only a small segment of plaintext is assumed to be known. The analysis methods reveal intrinsic properties of alleged RC4 which are independent of the key scheduling and the key size. The complexity of one of the attacks is estimated to be less than the time of searching through the square root of all possible initial states. However, this still poses no threat to alleged RC4 in practical applications.

Keywords. Cryptanalysis. Stream Cipher. RC4.

1 Introduction

Many key stream generators proposed in the literature consist of a number of possibly clocked linear feedback shift registers (LFSRs) that are combined by a function with or without memory. LFSR-based generators are often hardware oriented and for a variety of them it is known how to achieve desired cryptographic properties [3]. For software implementation, a few key stream generators have been designed which are not based on shift registers. One of these generators, known as (alleged) RC4, has been publicized and described in [1]. RC4 is widely used in commercial products and standards (one example is the Secure Sockets Layer standard SSL 3.0).

RC4 takes an interesting design approach which is quite different from that of LFSR-based stream ciphers. This implies that many of the analysis methods known for such ciphers cannot be applied. The internal state of RC4 consists of a table of 2^n n-bit words and two n-bit pointers, where n is a parameter (for the nominal version, $n = 8$). The table varies slowly in time under the control of itself. As discussed by Golić in [2], for such a generator a few general statistical properties of the key stream sequence can be measured by standard statistical tests, but these criteria are hard to establish theoretically. A noticeable exception are the results in [2], which show a (slight) statistical deviation of the output stream of RC4. These results are mainly of theoretical interest, as a large amount

[⋆] F.W.O. postdoctoral researcher, sponsored by the Fund for Scientific Research, Flanders (Belgium).

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in
Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.

of output stream is necessary before this deviation can be detected. It remains an open problem whether these results can be used to cryptanalyze RC4.

The aim of this paper is to derive some cryptanalytic algorithms that find the correct initial state of the RC4 stream cipher using only a small segment of output stream, and to give precise estimates for the complexity of the attacks where possible. The cryptanalytic algorithms in this paper exploit the combinatorial nature of RC4 and allow to find the initial table, i.e., the state at time $t = 0$. Knowledge of this table enables to compute the complete output sequence without knowing the secret key.

If the first portion of about 2^n output words are known, our basic algorithm allows to find the initial table in a reduced search with complexity much lower than exhaustive search over all possible initial states. A careful analysis, which is confirmed by numerous experiments for different values of the word length n, shows that the complexity of the best attack is lower than the square root of all possible initial states. Our algorithms become infeasible for $n > 5$ and thus pose no threat to RC4 with $n = 8$ as used in practice. However, our attacks give new insight into the design principles of RC4 and the estimates of the complexity should give some realistic parameters for the security of RC4. Our results are intrinsic to the design principles of RC4 and are independent of the key scheduling and the size of the key.

This paper is organized as follows. In Sect. 2 we give a description of RC4. In Sect. 3 we discuss an attack on a simplified version of RC4. Section 4 describes attacks on the full RC4, and Sect. 5 presents a possible optimization. We conclude in Sect. 6.

2 Description of RC4

We follow the description of RC4 as given in [1, 2]. RC4 is a family of algorithms indexed by a positive integer n (in practice $n = 8$). The internal state of RC4 at time t consists of a permutation table $S_t = (S_t[l])_{l=0}^{2^n-1}$ of 2^n n-bit words and of two pointer n-bit words i_t and j_t. Thus the internal memory size is $M = \log(2^n!) + 2n$, where log denotes logarithm to the base 2. The pointers i_0 and j_0 are initialized to zero. Let Z_t denote the output n-bit word of RC4 at time t. Then the next-state and output functions of RC4 for every $t \geq 1$ are defined by

$$i_t = i_{t-1} + 1 \tag{1}$$

$$j_t = j_{t-1} + S_{t-1}[i_t] \tag{2}$$

$$S_t[i_t] = S_{t-1}[j_t], \quad S_t[j_t] = S_{t-1}[i_t] \tag{3}$$

$$Z_t = S_t[S_t[i_t] + S_t[j_t]] \tag{4}$$

where all additions are modulo 2^n. In one update, all the words in the table except the swapped ones remain the same (and swapping is only effective if $i_t \neq j_t$). The output n-bit word sequence is $Z = (Z_t)_{t=1}^{\infty}$. Every word Z_t is XORed with a piece of plaintext of length n bits to produce ciphertext, or

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in
Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.

XORed with ciphertext to produce plaintext. The initial table S_0 is derived from the secret key. The details of this derivation are not important for our attacks.

3 Attacking Simplified RC4

The swap operation in (3) makes the recovery of the table S very difficult. In this section we develop an attack on simplified versions of RC4, where the swap operation occurs less often.

3.1 No Swap Operation

RC4 without the swap operation (3) is useless as a key stream generator. The following theorem illustrates this.

Theorem 1. *If the swap operation in the state update is omitted, the key stream of RC4 becomes cyclic with a period of* 2^{n+1}.

Proof: Equation (4) gives: $Z_{t+2^n} = S[S[i_{t+2^n}]+S[j_{t+2^n}]]$. Because of the modular addition $i_{t+2^n} = i_t$. Since S is constant now, (2) can be applied repeatedly on j_{t+2^n}. We get: $Z_{t+2^n} = S[S[i_t] + S[j_t + \sum_{u=0}^{2^n-1} S[u]]]$. Because S is a permutation, we can evaluate the summation, and $Z_{t+2^n} = S[S[i_t] + S[j_t + 2^{n-1}]]$. In a completely analogous way, we can derive $Z_{t+2^{n+1}} = S[S[i_t] + S[j_t]] = Z_t$. ■

The algorithm to recover S works as follows. Initially, we guess a small subset of the entries of S. We derive the other entries from the observed key stream and (4). If we get a contradiction at some point, we know that we guessed one of the initial values wrongly.

There are four possibly unknown variables in (4): $j_t, S[i_t], S[j_t]$ and $S^{-1}[Z_t]$. If all four variables are known and a contradiction arises, we guessed one of the initial values wrongly. If three variables are known, we can determine the fourth.

- If $S^{-1}[Z_t], j_t$ and $S[i_t]$ are known, we can determine $S[j_t]$ as follows:

$$S[j_t] = S^{-1}[Z_t] - S[i_t] . \tag{5}$$

  ($S^{-1}[Z_t]$ is known if the value Z_t is already filled in somewhere in S.)
- If $S[i_t], S[j_t]$ and thus also j_t are known, then

$$S[S[i_t] + S[j_t]] = Z_t . \tag{6}$$

- If $S[i_t], S[j_t]$ and $S^{-1}[Z_t]$ are known, then

$$j_t = S^{-1}[S^{-1}[Z_t] - S[i_t]] . \tag{7}$$

- If $S^{-1}[Z_t], j_t$ and $S[j_t]$ are known, then

$$S[i_t] = S^{-1}[Z_t] - S[j_t] . \tag{8}$$

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.

The initial value of j is known. If we guess the values of v entries at the beginning of S, we know the value of the j-pointer for the first v steps. In these steps we use (5) and (6) to determine new values of S. If we have not determined $S[v + 1]$ after v steps, we “lose” knowledge of the j-pointer. We discard the following Z_t-values until we can use (7) to recover the value of j. Once j is recovered we can use (5) and (6) again, but we can also work backwards and use (8) to determine more entries of S. If v is too small, we will lose the value of j too fast and we will not be able to recover the table in this way.

3.2 Reduced Swap Frequency

In this version of RC4 we swap two entries after every s iterations. We start by applying the same algorithm as above, until the first swap occurs. If we do not know the value of j at this moment, we do not know with what value $S_t[i_t]$ gets swapped. At this point we can only remove $S_t[i_t]$ from our (incomplete) table. If the unknown j actually points at a table entry that we have already filled in, this entry will change in the RC4 table, but not in our partial solution. In this way, errors are introduced in our S_t table. After a while we will observe contradictions; however, it is not possible to determine which element is responsible for the contradiction. A naive solution is to remove the three entries involved when we encounter a contradiction. However, in this way we will destroy more good values than we are able to produce, and we will end up with an empty table. For a good solution strategy it is important that the number of removed correct values is minimal. We have developed a number of heuristics to solve this problem; the details are omitted because of space restrictions. The resulting algorithm converges very fast.

If we increase the swap frequency $1/s$ towards 1, the algorithm needs a larger number of correctly guessed table entries before it can deduce the remainder of the table. Figure 1 shows the experimentally determined success probability as a function of the number of correctly guessed entries at the start, for swapping frequencies increasing from 1/128 to 1/2 (actual RC4 has swapping frequency 1). For a success ratio of 50% we need 40 correctly guessed entries at the start if the swapping frequency equals 1/128. If the swapping frequency increases to 1/2, we need about 240 correct entries. For a success ratio of 5%, we need 30, respectively 210 values. The complexity of this attack is proportional to the average number of trials required to guess the initial values correctly; e.g., there are approximately 2^{315} possible ways to assign 40 8-bit values of the permutation table.

Fig. 1. Success ratio for various simplified versions of RC4 for which the swap frequency is reduced to $1/s$.

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in
Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.
©1998 Springer-Verlag

4 Attacking the Full RC4

This section presents cryptanalytic attacks on RC4 which allow to find the initial table $S = S_0$, without guessing values initially. Instead, values are only guessed when they are needed. First the attacks are described and their efficiency is analyzed. Then some special cases are discussed and experimental results are presented.

4.1 Description

The idea of the algorithm may informally be described as follows. For times $t = 1, 2, \ldots, m$, if $S_{t-1}[i_t]$ or $S_{t-1}[j_t]$ have not already been assigned values in a previous time, choose a value v for $S_{t-1}[i_t]$, $0 \leq v < 2^n$, compute j_t and then choose $S_{t-1}[j_t]$. This is in order to be able to follow up the next update of the RC4 algorithm, i.e., in order that steps (1) to (4) are defined. We proceed so that at each time t an output word $\bar{Z}_t$ is produced with the property that $\bar{Z}_t$ has the correct value $\bar{Z}_t = Z_t$. This imposes several restrictions on the possible choices for $S_{t-1}[i_t]$, $S_{t-1}[j_t]$:

i) As S is a permutation table, every new value $S_{t-1}[i_t]$ or $S_{t-1}[j_t]$ to be assigned has to be different from a value already chosen as a word in the table.

The next two conditions represent two alternatives and are specific consequences of the design of RC4.

ii) If the known output word Z_t differs from all words which have previously been fixed in the S table, the sum $i_s = S_t[i_t] + S_t[j_t]$ occurring in step 4 has to differ from all index positions which have already values assigned. If this is satisfied, set $S_t[i_s] = Z_t$. Otherwise we have a contradiction in our search.
iii) If Z_t is equal to a word previously assigned in the S table then $i_s = S_t[i_t] + S_t[j_t]$ equals the index position of this assigned value. This either uniquely determines $S_t[j_t]$ or again leads to a contradiction.

Although conditions i), ii) and iii) follow directly from the description of RC4, it is not obvious how to implement an efficient algorithm that exploits these restrictions and how to obtain practically meaningful estimates for the complexity of such an algorithm.

We implemented this attack by means of a recursive function *guess*(t). In the most elementary version, at each parameter t one update following steps 1 to 4 is effected. Thereby, three entries in the S table are affected or suitably chosen, one entry determined by i_t, one by j_t and one by Z_t, so that the update at time t can be carried out and so that conditions i) to iii) are satisfied.

For a given output word sequence of length m the programs start by calling *guess*(1). In the recursive calls for increasing t most branches end up by contradictions. If one branch has reached depth $t = \mathit{size}+1$ in the recursive algorithm, we compute backwards the (correct) actual state to state $t = 0$, in order to get the initial table S_0. Experiments have shown that for the basic version of the

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.

attack as sketched, $m = size = 2^n$ known output words are sufficient to uniquely determine the correct state. Note that for RC4 with n-bit words, there are a total of $2^n!$ different initial states. Thus, the required number of output words m can be estimated as the smallest integer such that $2^{nm} > 2^n!$. Clearly, 2^n upper bounds m for any value of n. (For $n = 8$, $m \simeq 211$.)

We investigated several variants of the attack. In order to accelerate the attack in simulations, we pre-assigned the first few words in the S table at the beginning of the program execution. This has motivated a modification of the function *guess*(t) which is based on the following observation: if $S_{t-1}[i_t]$ has a value assigned one can compute j_t according to step 2. Thus one can swap $S_{t-1}[i_t]$ and $S_{t-1}[j_t]$ even if $S_{t-1}[j_t]$ was not assigned a value before swapping. After swapping, $S_t[j_t]$ is assigned but $S_t[i_t]$ is not.

As a consequence, suppose $S_{t-1}[i_t]$ has a value assigned but $S_{t-1}[j_t]$ has not. Assume now that the value Z_t is different from all previously assigned values in the S table. Then instead of guessing the value of $S_{t-1}[j_t]$ one can check whether $S_{t-1}[i_{t+1}]$ has already been assigned a value and whether the value of Z_{t+1} equals a value previously assigned in the S table. Under this condition it may pay off not to check all possible values for $S_{t-1}[j_t]$ because a check can be done at time $t + 1$ without guessing any additional values. This variant has in experiments shown to be particularly attractive for parameter values $n = 7$ and 8. Moreover note that for this variant the known output segment has to be slightly longer than for the basic attack.

There are even further refinements of the variant which we will not describe here. In another direction, computer experiments have lead to the following observation: suppose two initial tables S_0 and $\bar{S}_0$ are given with the property that $S_0[i] = \bar{S}_0[i]$ for $i = 1, 2, \ldots, k$. Then for k sufficiently large, suitable segments of the corresponding output sequences Z and $\bar{Z}$ of the RC4 algorithm are correlated. This correlation is illustrated in Fig. 2. We have built this statistical property into our attack in order to make a preliminary test at a suitable time t whether a choice of values $S_0[i]$, $i = 1, 2, \ldots, t - 1$, is correct. It turned out that this in fact can lead to an acceleration of the attack but at the cost of a decreased success probability, as often a correct choice is excluded erroneously.

Fig. 2. Correlation between key streams as a function of the number of equal table entries. Three measures for the correlation are shown: (1) the number of equal outputs until the first difference occurs, (2) the number of equal outputs in the first 250 values and (3) the number of equal outputs in the first 250 values, added with a weighting function that emphasizes the first outputs of the row. It is clear that the last two functions are better measures.

4.2 Efficiency of the Attack

The complexity of the attacks is measured in terms of the total number of assignments made for all entries in the initial table. It is necessary at this point to explain some further details of our search algorithm. The algorithm uses recursive function calls with the time variable t as parameter. Assume we are at some given time t, and let a_t denote the number of entries in the initial table, which were assigned a value at time t.

1. It is checked whether $S_{t-1}[i_t]$ has been assigned a value:
   (a) if it has, proceed to step 2.
   (b) if it has not, then assign, one after one, the $2^n - a_t$ remaining values to $S_{t-1}[i_t]$, increment a_t and go to 2.
2. It is checked whether Z_t has a value which has been used in an assignment:
   (a) if it has, we can calculate the expected value of $S_t[j_t]$ from (4) of the RC4 description. If this does not lead to a contradiction, proceed to time $t+1$ and go to step 1.
   (b) if it has not, go to 3.
3. It is checked whether $S_{t-1}[j_t]$ has already been assigned a value:
   (a) if it has not, then assign, one after one, the $2^n - a_t$ remaining values to $S_{t-1}[j_t]$ and update a_t. Subsequently, it can be checked whether the given values of i_t, j_t and Z_t lead to a contradiction. If they do not, proceed to time $t+1$ and go to step 1.

It follows that the search algorithm can be split into 8 cases, depending on whether i_t and j_t have been assigned a value or not and whether Z_t has a value already assigned to an entry in the table. It is possible to simulate the behavior of the search algorithm by assigning probabilities to the different cases in the above informal description. As an example, the case "$S_{t-1}[i_t]$ has been assigned a value" has an average probability of $a_t/2^n$ of being true and an average probability of $1 - a_t/2^n$ of being wrong. We define a function *complex*$(\cdot)$, which takes as input a, the number of assigned values in the table. The function has the following form:

$$\mathit{complex}(a) = \sum_{i=0}^{3} p_i \cdot \text{no-assignments}_i \cdot \mathit{complex}(a+i)\,. \tag{9}$$

Our approximation reduces the 8 above cases to 4 cases, each one with a recursive call of the function *complex*. The four recursive calls are explained as follows: p_i denotes the probability of the particular case, no-assignments$_i$ denotes the total number of assignments we do for $S_{t-1}[i_t]$ and $S_{t-1}[j_t]$.

By definition, *complex*$(255) = 1$ and *complex*$(a) = 0$ for all $a \geq 256$. Given the values for *complex*$(a+3)$, *complex*$(a+2)$, *complex*$(a+1)$ and expressions for p_i and no-assignments$_i$, (9) can be solved for *complex*(a). In this way *complex*(0) can be determined.

Solving the Recurrence: Instead of determining p_i and no-assignments$_i$ directly, we will rewrite (9). We define three new functions $c_1(\cdot), c_2(\cdot)$ and $c_3(\cdot)$, representing the complexity of each individual step in our algorithm. We start with the equation for $c_1(a)$. The first test of step 1 will succeed on average $a/2^n$ times. If it succeeds, we go to step 2 without assigning a value. If it does not succeed (probability $1-a/2^n$), we will do for every possible value of $S_{t-1}[i_t]$ one assignment and call step 2. Thus we have:

$$c_1(a) = \frac{a}{2^n} c_2(a) + (1 - \frac{a}{2^n})(2^n - a) c_2(a+1)\,. \tag{10}$$

In a similar way, we can derive the expressions for $c_2(a)$ and $c_3(a)$:

$$c_2(a) = \frac{a}{2^n}\left((1-\frac{a}{2^n})^2(1+c_1(a+1)) + \frac{1}{2^n}c_1(a)\right) + (1-\frac{a}{2^n})c_3(a) \tag{11}$$

$$c_3(a) = (1-\frac{a}{2^n})\left(f(a) + \frac{2a+1}{2^n}c_1(a+1) + (2^n - a)e(a)c_1(a+2)\right), \tag{12}$$

where $e(a) = (1-(a+1)/2^n)(1-1/(2^n-a))$ and $f(a) = (2^n-a)(1+e(a))+a/2^n$. Again we start with the known values $c_i(2^n)$ and work downwards. The maximal number of assignments in our algorithm is given by *complex*$(0) = c_1(0)$. The results of the calculation are presented in Table 2, where they are compared with some experimental results.

4.3 Special Streams

There are streams of output words for which our attack has an increased performance. Consider the above description of our algorithm. In step 2 of the algorithm we check whether Z_t has a value which has previously been used in an assignment. If this is the case we can calculate an expected value for the entry $S_t[j_t]$. This either leads to a contradiction or it gives an assignment of an additional entry in the (unknown) table. If this is not the case we try and assign values to $S_t[j_t]$ and proceed from there. Assume now that Z_t equals Z_{t+1}. Then in our algorithm at time $t+1$ the condition in step 2 is satisfied, since the value of Z_t was used in an assignment in a previous step without reaching a contradiction, since we assume we are at time $t+1$. Thus, the performance of the algorithm can be improved if many of the given words are equal. We have incorporated this in the above approximations, but we leave out the exact details. Table 1 lists the results of our tests for versions of RC4 with $n = 4, 5$. It follows that the performance of our algorithm for RC4 with $n = 5$ increases with more than a factor of two if the first two words of the given stream are equal, and that the improvement is a factor of about 2^{k-1} if the first k words are equal. Clearly, a similar phenomenon can be expected if the number of different values in the first k words of the stream is greater than 1, but small.

4.4 Experimental Results

The first interesting value for n is $n = 4$, where the number of entries in S_0 is 16 and the number of possible initial tables is $16! = 2.09 \cdot 10^{13} \approx 2^{44}$. It turns

Table 1. Approximations of the complexities of the attack on RC4, when the first k words in the target stream are equal.

n	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$\sqrt{2^n!}$
4	2^{21}	$2^{20.5}$	$2^{19.9}$	$2^{19.4}$	$2^{18.9}$	2^{22}
5	2^{53}	$2^{51.6}$	$2^{50.5}$	$2^{49.4}$	$2^{48.4}$	2^{58}

out that the basic algorithm for our attack always finds the correct initial table in a few seconds, which represents a considerable improvement over exhaustive search. It is interesting to compare our result for $n = 4$ with a result in [2]: the method developed in [2] needs about 2^{6n-8} output words of the RC4 stream cipher to detect a statistical deviation. This is about 2^{16} output words for RC4 with $n = 4$, whereas we need 16 or 17 output words and about 2^{20} computations to find the correct initial table.

As measure of complexity we take the total number of calls of the function *guess*(t) that are necessary to find the initial table. For $n = 4$ the average number of function calls turns out to be about 2^{20}. For $n = 5$ the complexity of the attack is too high for the computing power we have available. Therefore, in simulations for $n \geq 5$ we accelerate the programs by giving the correct values of the first few entries of the S table. Experiments show that the amount of computing time can differ some orders of magnitude depending on the initial table to be found.

In Table 2 we give the results of our experiments for parameter values $n = 4, \ldots, 8$. Hereby k denotes the number of preassigned entries $S_0[i]$, $1 \leq i \leq k$. Complexity means the average number of calls of the function *guess*(t) in the program with given parameter k in 1000 test cases. We should mention however, that the figures for the complexity are only rough estimates as the distribution for these numbers has a large variance. When the k preassigned entries have wrong values, the search terminates rather quickly with a contradiction in most cases. For $k > 0$ the total complexity is computed as the number N of all possible choices of the first k entries multiplied by the average complexity. Note that N is computed as $2^n!/(2^n - k)!$. It can be seen that our test results for the cases $n = 4$ and 5 correspond well to the estimated complexity given in Sect. 4.2. Furthermore, for $n = 5$, $k = 3$ one can apply a program variant using the statistical property as described in Sect. 4.1. It turns out that the complexity in this case is about 2^{30}, thus the total complexity is about 2^{45}. However the algorithm often terminates unsuccessfully. The average success rate may be below 50%. For comparison, in the last column of Table 2 the magnitude of square root of $2^n!$ is shown. It follows that the estimated total complexity is slightly below the square root of $2^n!$.

We already mentioned that our search algorithm works better if the first words of the output stream are equal. We close this section by listing the results for RC4 with $n = 4$ in Table 3 and leave it as an open question how large the improvement is for RC4 with $n > 4$ in these cases.

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in
Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.
©1998 Springer-Verlag

Table 2. Complexities of attacks on n-bit RC4. One column gives estimates based on the analytical calculations of Sect. 4.2. Other values are based on extrapolations of experimental results on simplified versions (preassigning k values). It follows that the (total) complexities are close to $\sqrt{2^n!}$.

	calculated		experimental			
n	k	complexity	k	complexity	total complexity	$\sqrt{2^n!}$
3	0	2^{8}	0	2^{8}	2^{8}	2^{8}
4	0	2^{21}	0	2^{20}	2^{20}	2^{22}
5	0	2^{53}	7	2^{21}	2^{55}	2^{58}
6	0	2^{132}	20	2^{23}	2^{138}	2^{148}
7	0	2^{324}	45	2^{26}	2^{302}	2^{358}
8	0	2^{779}	100	2^{30}	2^{797}	2^{842}

Table 3. Complexities of the attack on RC4 with $n = 4$, when the first k words in the target stream are equal, averaged over 1000 tests.

n	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$\sqrt{2^n!}$
4	$2^{20.5}$	$2^{19.5}$	$2^{18.4}$	$2^{17.6}$	2^{22}

5 A Possible Improvement

In this section we explain a technique that can be used to improve the efficiency of the RC4 attack of Sect. 4.

5.1 Description

The basic principle of the technique is the following. The initial state of the permutation table S depends on the cipher key and is unknown. We assume that all $2^n!$ possibilities are equally likely, or that the a priori probability distribution of S_0 is uniform. We observe the generated values Z_t and try to calculate an a posteriori probability distribution for S_0. The method can easily be extended to deal with a non-uniform a priori probability distribution.

We represent our information about the value of j and the state of S by means of probability distributions. We define the functions f_t as $f_t(a) = \Pr(j_t = a)$ and the array of functions g_t as $g_t[x](a) = \Pr(S_t[x] = a)$. Since we know that $j_0 = 0$, the function f_0 is 1 at the origin, and zero elsewhere. Also, because S is a permutation at all times, we know that for all values of t and for $a \in [0, 255]$: $\sum_{x=0}^{2^n-1} g_t[x](a) = 1$.

During the attack we observe the generated key stream $Z_t, t = 0, 1, \ldots$, and we try to extract information about the value of j and the state of S after iteration t, by using (4) and Bayes' rule. The extracted information is manifest in the functions f_t and $g_t[x]$: the closer these functions are to a delta-function, the less uncertainty we have about the values of j_t and $S_t[x]$.

In order to calculate the updated probability distributions, we have to take into account two effects: observation of Z_t, which gives us more information, or "narrows" the probability distributions, and the change of state for j and

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in
Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.

two elements of S, which tends to "flatten" the probability distributions. The derivation of the rules for updating the probability distributions is given in Appendix A. We assume that the different entries of S_t are independent from each other, except that there cannot be two equal values because S_t is a permutation. This assumption is only an approximation.

5.2 Implementation

The algorithm reads one word of the key stream and calculates the values for f_1 and $g_1[x]$. The complexity is determined by the determination of g_1: for each of the 2^n x-entries there are 2^n probabilities to calculate and every probability is the sum of $(2^n)^3$ terms (cf. (24)). This gives a total complexity of 2^{5n} steps for each value Z_t that is analyzed. In theory, we need less than 2^n values in order to determine the initial table uniquely.

Since the complexity of this algorithm is too high to test it on the full version of RC4, we tested it with a table that is partially filled in correctly, adapting the probability distributions accordingly. A partially filled table can result in a unique determination of $j_1, j_2, \ldots$ As long as j_t is known, there is no "flattening effect" and the Bayes method works as predicted. Experimental results suggest that it is difficult to get convergence when the uncertainty on j_t grows. A possible explanation for the convergence problems is that the dependence of the different entries of S_t on one another is too high to be neglected. If 160 entries or more of S_0 are given, the algorithm always succeeds in completing the table, the complexity being less than 2^{30}. If 150 entries are given, the success ratio is 70%, and it is expected to drop very quickly from here.

Figure 3 shows some experimental results for a simplified algorithm. The input of the algorithm consists of the values for k entries of S_0. The algorithm performs the attack, until knowledge of j_t is lost. The algorithm restarts and processes the key stream again with the updated information on S_0 until no new information is obtained anymore. Since j_t is known, the complexity of the algorithm is reduced; it is now about $k(2^n - k)^3$. The figure shows how many table entries can be successfully recovered as a function of k. One can deduce that the algorithm is most successful when $k \approx 120$. Since the algorithm does not output a complete table, we can use its output table as input for the attack of Sect. 4. Experiments suggest that for values of k between 100 and 200, the prior application of the simplified Bayes algorithm before starting the attack of Sect. 4 increases the efficiency. However, the problem of determining the first k values remains. Since the latter attack also works without predetermined entries of S_0, it could be used to generate a guess for these first k values. Estimating the complexity of attacks based on combining the Bayes technique with the attack of Sect. 4 is a rather involved task. We leave it as an open problem to which extent this combination will improve the attacks on RC4.

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in
Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.

Fig. 3. Average number of entries successfully recovered by the Bayes method as function of the number of known entries on beforehand.

6 Conclusions

We have demonstrated several cryptanalytic algorithms on the alleged RC4 stream cipher. The algorithms try to deduce the initial state in a known plaintext attack. First we demonstrated the importance of the swapping operation in RC4. Our results show that a less frequent use of the swapping operation enables stronger cryptanalytic attacks.

The second algorithm has the best overall performance. It finds the correct initial state using only a small segment of known plaintext. The complexity of the attack was estimated by analytical calculations and verified by extensive testing. The complexity was approximated to be less than the time of searching through the square root of all possible initial states. We have also identified certain streams of words of RC4 for which the search algorithm has an increased performance. The third algorithm is based on probability theory. It involves no guessing, but it only works if a certain number of table entries is already known. Although our attacks are by far not practical for the specified word size of RC4, they give new intrinsic insight into the algorithm. It is our hope that our results will stimulate further research on RC4.

References

1. B. Schneier, *Applied Cryptography*, Wiley, New York, 1996.
2. J. Dj. Golić, “Linear Statistical Weakness of Alleged RC4 Keystream Generator,” *Advances in Cryptology - EUROCRYPT’97, Lecture Notes in Computer Science, Vol. 1233,* Walter Fumy (Ed.), Springer-Verlag, pp. 226-238.
3. R. A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, Berlin, 1986.

A Calculating the a Posteriori Probability Distributions

There are two effects: observation of Z_t, which gives us more information, or “narrows” the probability distributions, and the change of state for j and two elements of S, which tends to “flatten” the probability distributions.

A.1 The Change of the State

The “flattening effect” is described by the following equations, denoting the new probability distribution functions with $f', g'[x]$:

$$f'_t(a) = \sum_b f_{t-1}(a-b) g_{t-1}[i_t](b) \tag{13}$$

$$g'_t[i_t](y) = \sum_b f'_t(b) g_{t-1}[b](y) \tag{14}$$

$$g'_t[x](y) = (1 - f'_t(x)) g_{t-1}[x](y) + f'_t(x) g_{t-1}[i_t](y)\,. \tag{15}$$

Equation (13) corresponds to a convolution.

A.2 Observation of Z_t

The information of the known Z_t value can be used to calculate the functions f_t and $g_t[x]$. Bayes' rule gives the following equations:

$$\Pr(j_t = a \mid Z_t = d) = \frac{\Pr(j_t = a) \Pr(Z_t = d \mid j_t = a)}{\Pr(Z_t = d)} \tag{16}$$

$$\Pr(S_t[x] = y \mid Z_t = d) = \frac{\Pr(S_t[x] = y) \Pr(Z_t = d \mid S_t[x] = y)}{\Pr(Z_t = d)}\,. \tag{17}$$

In terms of f_t and $g_t[x]$ this becomes

$$f_t(a) = \frac{f'_t(a) \Pr(Z_t = d \mid j_t = a)}{\Pr(Z_t = d)} \tag{18}$$

$$g_t[x](y) = \frac{g'_t[x](y) \Pr(Z_t = d \mid S_t[x] = y)}{\Pr(Z_t = d)}\,. \tag{19}$$

The remaining probabilities can be expressed as functions of f'_t and $g'_t[x]$.

Equation (4) gives for the probability distribution of Z_t:

$$\Pr(Z_t = d) = \sum_a \sum_b \sum_c \Pr(j_t = a, S_t[a] = b, S_t[i_t] = c, S_t[b+c] = d)\,. \tag{20}$$

We do not need to calculate the probability $\Pr(Z_t = d)$ explicitly, because it can be determined from the renormalization requirements:

$$\sum_a \Pr(j_t = a \mid Z_t = d) = 1$$

$$\sum_y \Pr(S_t[x] = y \mid Z_t = d) = 1\,.$$

The value $\Pr(Z_t = d \mid j_t = a)$ can be calculated as

$$\begin{aligned}\Pr(Z_t = d \mid j_t = a) &= \sum_b \sum_c \Pr(S_t[a] = b, S_t[i_t] = c, S_t[b+c] = d) && (21)\\ &= \sum_b \sum_c \Pr(S_t[a] = b) \Pr(S_t[i_t] = c \mid S_t[a] = b)\\ &\qquad \Pr(S_t[b+c] = d \mid S_t[a] = b, S_t[i_t] = c)\,. && (22)\end{aligned}$$

In order to rewrite this in terms of f'_t and $g'_t[x]$, we assume that for two different values x_1, x_2, the values of $S_t[x_1]$ and $S_t[x_2]$ are independent.

- $\Pr(S_t[a] = b) = g'_t[a](b)$.
- $\Pr(S_t[i_t] = c \mid S_t[a] = b)$: if both $i_t = a$ and $c = b$, the probability is one; if only one of the equalities holds, the probability is zero; else it is $\Pr(S_t[i_t] = c)/(1 - \Pr(S_t[i_t] = b)) = g'_t[i_t](c)/(1 - g'_t[i_t](b))$.
- $\Pr(S_t[b+c] = d \mid S_t[a] = b, S_t[i_t] = c)$: In the generic case, the probability is $\Pr(S_t[b+c] = d)/(1 - \Pr(S_t[b+c] = b) - \Pr(S_t[b+c] = c)) = g'_t[b+c](d)/(1 - g'_t[b+c](b) - g'_t[b+c](c))$. Special cases occur when $a = i_t$, $b = c$, $a = b + c$, $d = b$, $i_t = b + c$, and/or $c = d$.

Similarly, the value of $\Pr(Z_t = d \mid S_t[x] = y)$ can be calculated as

$$\begin{aligned}&\Pr(Z_t = d \mid S_t[x] = y)\\ &= \sum_a \sum_b \sum_c \Pr(j_t = a, S_t[a] = b, S_t[i_t] = c, S_t[b+c] = d \mid S_t[x] = y) && (23)\\ &= \sum_a \sum_b \sum_c \Pr(j_t = a \mid S_t[x] = y) \Pr(S_t[a] = b \mid S_t[x] = y, j_t = a)\\ &\qquad \Pr(S_t[i_t] = c \mid S_t[x] = y, j_t = a, S_t[a] = b)\\ &\qquad \Pr(S_t[b+c] = d \mid S_t[x] = y, j_t = a, S_t[a] = b, S_t[i_t] = c)\,. && (24)\end{aligned}$$

These equations can also be reworked in terms of f'_t and $g'_t[x]$.

Appeared in *Advances in Cryptology – ASIACRYPT 1998*, Lecture Notes in
Computer Science 1514, K. Ohta, and D. Pei (eds.), Springer-Verlag, pp. 327–341.