Fast and Efficient Key Recovery from RC4 Permutation after KSA

Research Problem Proposal for Japanese-Indian Cooperative Programme:
Security Evaluations and Design of Components and Cryptographic Primitives for RFID and Sensor Networks

Presented at the Indo-Japan meeting, ISI, Kolkata, April 13, 2009 by

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

1 Introduction

The RC4 stream cipher has been designed by Ron Rivest for RSA Data Security in 1987, and was a propriety algorithm until 1994. Currently, RC4 is extremely popular in commercial domain and widely used in network protocols such as Secure Sockets Layer (SSL), Transport Layer Security (TLS), Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA) etc.

RC4 uses an S-Box $S = (S[0], \ldots, S[N-1])$ of N bytes, initialized as the identity permutation. Typically, $N = 256$. A secret key k of size l bytes (typically, $5 \leq l \leq 16$) is used to scramble this permutation. An array $K = (K[0], \ldots, K[N-1])$ is used to hold the secret key, the key is repeated in the array K at key length boundaries. where $K[y] = k[y \bmod l]$ for any y, $0 \leq y \leq N-1$, i.e.,

The RC4 cipher has two components, namely, the Key Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA). The KSA turns the random key K into a random looking permutation S of $0, 1, \ldots, N-1$ and the PRGA uses this permutation to generate the pseudo-random keystream bytes z. Both the KSA and the PRGA uses a deterministic index i and a secret pseudo-random index j to scramble the permutation by transposition.

Any addition used related to the RC4 description is in general addition modulo N unless specified otherwise.

KSA	**PRGA**
Initialization:	*Initialization*:
For $i = 0, \ldots, N-1$	$i = j = 0$;
$S[i] = i$;	*Keystream Generation Loop*:
$j = 0$;	$i = i + 1$;
Scrambling:	$j = j + S[i]$;
For $i = 0, \ldots, N-1$	Swap($S[i]$, $S[j]$);
$j = (j + S[i] + K[i])$;	$t = S[i] + S[j]$;
Swap($S[i], S[j]$);	Output $z = S[t]$;

Let S_r be the permutation and j_r be the value of the pseudo-random index j after r many rounds of the RC4 KSA, $1 \leq r \leq N$. Thus, S_N is the permutation after the complete key scheduling. We denote the initial permutation by S_0 and the initial value 0 of the index j by j_0. We use the notation S^{-1} for the inverse of the permutation S, i.e., if $S[y] = v$, then $S^{-1}[v] = y$. Further, let Z_N denote the set $\{0, 1, \ldots, N-1\}$. We also use f_y to denote the expression $\frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]$ that is used frequently throughout this document.

2 Motivation

State recovery attacks [5, 12, 8] are an important class of attacks on RC4 and the key recovery attack from the internal state which covers a major part of the current work is useful to turn a state recovery attack into a key recovery attack. If the complexity of "recovering the secret key from the permutation" is less than that of "recovering RC4 permutation from the keystream output bytes in PRGA", then by cascading the techniques of the latter [5, 12, 8] 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 [6]), 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.

Since the PRGA update is deterministic, if we know the RC4 internal state at any stage of the PRGA, we can get back the permutation S_N after the KSA using the *PRGAreverse* algorithm presented in [10]. Thus, without loss of generality, the key recovery from any RC4 state can be reduced to key recovery from the final permutation S_N after the KSA.

3 Key Permutation Correlation

Roos [11] for the first time observed that after the KSA, the most likely value of $S_N[y]$ is f_y for initial values of y. The empirical values of $P(S_N[y] = f_y)$ is provided in Table 1 below.

y	$P(S_N[y] = f_y)$
0-15	.370 .368 .362 .358 .349 .340 .330 .322 .309 .298 .285 .275 .260 .245 .229 .216
16-31	.203 .189 .173 .161 .147 .135 .124 .112 .101 .090 .082 .074 .064 .057 .051 .044
32-47	.039 .035 .030 .026 .023 .020 .017 .014 .013 .012 .010 .009 .008 .007 .006 .006

Table 1: The probabilities experimentally observed by Roos [11].

The exact analytical form of $P(S_r[y] = f_y)$ for any round r was derived theoretically in [9] and is presented in Theorem 1 below.

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

It has been shown in [7] 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 $y = 48$). Similar nested dependencies for the inverse permutation have been reported in [1].

4 Existing Works on Key Recovery from Permutation

In [9, Section 3], for the first time an algorithm is presented to recover the complete key from the final permutation after the KSA using the Roos' biases, without any assumption on the key or IV. The algorithm recovers some secret key bytes by solving sets of independent equations of the form $S_N[y] = f_y$ and the remaining key bytes by exhaustive search.

Subsequently, the work [3] additionally considered differences of the above equations and reported better results. Recently, [1] has accumulated the ideas in the earlier works [9, 3, 7] along with some additional new results to devise a more efficient algorithm for key recovery.

After the publication of [9, 3], another work [10] which has been performed independently and around the same time as [1] shows that each byte of S_N actually reveals secret key information. The key recovery algorithm of [10] sometimes outperform that of [3]. A recent work [4] starts with the equations of [3] and considers a bit-by-bit approach to key recovery.

	l	5	8	10	12	16
Data from [9]	Probability	0.431	0.414	0.162	0.026	0.0006
	Complexity	$2^{25.6}$	$2^{31.9}$	$2^{34.0}$	$2^{31.6}$	$2^{32.2}$
Data from [3]	Probability	0.8640	0.4058	0.1290	0.0212	0.0005
	Time in Seconds	0.02	0.60	3.93	7.43	278
Data from [10]	Probability	0.9985	0.4362	0.1421	0.0275	0.0007
	Complexity	$2^{28.7}$	$2^{26.8}$	$2^{29.9}$	$2^{32.0}$	$2^{40.0}$
Data from [1]	Probability	0.998	0.931	–	0.506	0.0745
	Time in Seconds	0.008	8.602	–	54.390	1572

Table 2: Comparison of key recovery algorithms for different key lengths.

Data in the first row is from the revised table of [9] that appears in [10] and demonstrates the complexity required by the approach in [9] to have the success probability of the same order as in [3].

Very recently, a bidirectional key search algorithm has been presented in [2] that can recover a 16 bytes secret key with a success probability of 0.1409. However, this suffers from high time complexity (of the order of 2^{53}).

5 A New Formulation of the Key Recovery Problem

We focus on the instance of RC4 with 16 bytes secret key.

First, we build a frequency table for all the key bytes following the same approach as in [10, 1]. We guess one j_{y+1} from the 8 values $S_N[y]$, $S_N^{-1}[y]$, $S_N[S_N[[y]]$, $S_N^{-1}[S_N^{-1}[y]]$, $S_N[S_N[S_N[y]]]$, $S_N^{-1}[S_N^{-1}[S_N^{-1}[y]]]$, $S_N[S_N[S_N[S_N[y]]]]$ and $S_N^{-1}[S_N^{-1}[S_N^{-1}[S_N^{-1}[y]]]]$. From two successive j values j_y and j_{y+1}, $8 \times 8 = 64$ candidates for the key byte $K[y]$ are obtained.

We sort the candidate values for each key byte in decreasing order of frequencies and select the top 4 values. We find that the correct value for each key byte belongs to this quadruple with a probability > 0.4. From this selection, if we want to guess 8 key bytes exhaustively, then we need a search complexity of $\binom{16}{8}4^8 \approx 2^{30}$. The success probability is equal to the probability that at least 8 key bytes are correct and is given by $\sum_{m=8}^{16} \binom{16}{m}(0.4)^m(0.6)^{16-m} \approx 0.28$.

At this point, we would like to address the following two problems.

1. Given the values of any 8 key bytes, find the probability that this set of values lead to the knwon RC4 state S_N.

2. Given the *correct* values of any 8 key bytes for the known RC4 state S_N, derive the remaining key bytes.

For the second problem, one possibility could be to assign 4 of the 8 remaining key bytes exhaustively with a complexity 2^{32} (thereby maintaining the success probability 0.28) and then guess the remaining 4 bytes efficiently.

References

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

[2] R. Basu, S. Maitra, G. Paul and T. Talukdar. On Some Sequences of the Secret Pseudo-random Index j in RC4 Key Scheduling. To appear in the Proceedings of the 18th International Symposium on Applied Algebra, Algebraic Algorithms and Error Correcting Codes (AAECC), June 8-12, 2009, Tarragona, Spain, Lecture Notes in Computer Science, Springer [Acceptance date: February 23, 2009].

[3] 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.

[4] S. Khazaei and W. Meier. On Reconstruction of RC4 Keys from Internal States. Mathematical Methods in Computer Science (MMICS), December 17-19, 2008, Karlsruhe, Germany, pages 179-189, vol 5393, Lecture Notes in Computer Science, Springer.

[5] 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.

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

[7] S. Maitra and G. Paul. New Form of Permutation Bias and Secret Key Leakage in Keystream Bytes of RC4. FSE 2008, pages 253-269, vol. 5086, Lecture Notes in Computer Science, Springer. A revised and extended version with the same title is available at the IACR Eprint Server, eprint.iacr.org, number 2007/261, Jan 9, 2009.

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

[9] 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.

[10] G. Paul and S. Maitra. RC4 State Information at Any Stage Reveals the Secret Key. IACR Eprint Server, eprint.iacr.org, number 2007/208, Jan 9, 2009. This is an extended version of [9].

[11] 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.

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