Cryptogr. Commun. (2009) 1:269–282
DOI 10.1007/s12095-009-0012-4

A new distinguishing and key recovery attack on NGG stream cipher

Aleksandar Kircanski · Rabeah Al-Zaidy · Amr M. Youssef

Received: 11 August 2008 / Accepted: 21 April 2009 / Published online: 6 May 2009

Abstract NGG is an RC4-like stream cipher designed to make use of today's common 32-bit processors. It is 3–5 times faster than RC4. In this paper, we show that the NGG stream can be distinguished, with success probability $\approx 97\%$, from a random stream using only the first keystream word. We also show that the first few kilobytes of the keystream may leak information about the secret key which allows the cryptanalyst to recover the secret key in a very efficient way.

1 Introduction

Because of its simplicity and speed, RC4 [15] is one of the most widely used stream ciphers in software applications. It is implemented in many protocols and applications such as Secure Socket Layer (SSL), and Wired Equivalent Privacy (WEP).

Typically, RC4 operates with 8-bit values both on output and in the internal state. Ciphertext is obtained by XOR-ing keystream bytes with the plaintext. From the perspective of modern processors with 32/64-bit word size, this is inefficient. A stream cipher that produces 32/64-bit keystream words and requires similar number of operations per step would be around 4–8 times faster, since an 8-bit operation on these processors takes equal time as a 32/64-bit operation. To address this problem, several generalizations of RC4-like stream ciphers have been proposed (e.g., RC4A [19], VMPC [27], NGG [17] and GGHN [6]).

A. Kircanski · R. Al-Zaidy · A. M. Youssef (✉)
Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Quebec, H3G 1M8, Canada
e-mail: youssef@ciise.concordia.ca

A. Kircanski
e-mail: a_kircan@ciise.concordia.ca

R. Al-Zaidy
e-mail: r_alzaid@ciise.concordia.ca

The NGG cipher was proposed by Nawaz et al. [17, 24]. Originally, NGG was named RC4(n,m) where m denotes the bit-length of the keystream output word and the size of the internal state table is 2^n. Later on, the name NGG was adopted for this cipher. Another version of this cipher, called GGHN, was introduced in [6], but our focus in this paper is the original version of the cipher.

In this paper, we show that key schedule algorithm (KSA) of NGG is flawed. This allows the cryptanalyst to distinguish NGG from a random stream using only the first keystream word. Furthermore, we show that the resulting statistical bias in the key scheduling process allows the cryptanalyst to recover information about the secret key from the first few kilobytes of the keystream output.

The last decade has witnessed an extensive cryptanalytic literature on RC4 including distinguishing attacks (e.g., [3, 5, 11]), internal state recovery attacks (e.g., [9, 11, 14, 23]), and attacks on the key scheduling algorithm (e.g., [2, 4, 10, 12, 16, 20–22, 25]). Recently, analyzing the security of generalized RC4-like ciphers has also gained some momentum (e.g., [13, 18, 24, 26]). The cryptanalytic results presented in this paper resembles the work in [2, 20], in which it is shown that internal state permutation table of RC4, right after KSA, can leak secret key bytes. On the other hand, apart from [8] which studies weaknesses associated with concatenating IVs to the key, this is the first time that the security of NGG key schedule algorithm is addressed. Also, according to our knowledge, no key recovery algorithms on NGG have been proposed until now. Best previously published NGG distinguisher [26] exploited a problem in NGG pseudorandom number generation algorithm and requires around 100 consecutive keystream words. The distinguisher presented in this paper requires only the first key stream word generated right after the the KSA. The attack presented in [24] focuses on GGHN and uses the first two keystream words associated with about 2^{30} secret keys to build a distinguisher. While this attack may also be applicable to NGG, the high frequency of key changing required by this attack (2^{30} keys) questions its practical significance against both ciphers.

Among cryptographers, there exist different stances in the debate on whether distinguishing attacks represent a threat to the security of stream ciphers or not. For example, Rose [7] argues that, unlike the block ciphers case, most of the distinguishing attacks on stream ciphers do not represent a real threat to the practical security of the cipher and hence one should make a distinction between powerful distinguishing attacks and weak ones based on whether the attack may lead to deriving useful cryptanalytic information such as key bits or not. Conversely, according to Bernstein [1], even distinguishing attacks that do not yield key or other cryptanalytic information are more than mere certificational weaknesses. An attacker, at least for some plaintext distributions, might be able to detect change of entropy in the plaintext only by looking at the ciphertext. If for example, a dummy message with high entropy is sent among sender and receiver from time to time to foil the attacker's analysis, the attacker might be able to use a distinguisher to isolate and discard these dummy messages.

The statistical bias weakness presented in this paper is such that it allows distinguishing the cipher from a random stream and, at the same time, recovering some secret key information. It should be noted that the exhibited weakness is in key scheduling phase of the algorithm and not in keystream generation part of the algorithm. In particular, the NGG keystream generation procedure creates biased internal state and this is detectable in the first few kilobytes of the cipher. Thus, in a

way, by observing this bias, it is natural to expect that information about the secret key might be revealed.

The rest of the paper is organized as follows. In Section 2, the relevant details of the NGG cipher are given and previous attacks are described. Non-randomness of the S table after the NGG KSA is proved in Section 3. In Section 4, a distinguisher utilizing this weakness is constructed, and the success probability estimates are given. In Section 5, we show how it is possible to recover information about the secret key by looking at the first few kilobytes of the keystream output. We conclude in Section 6.

2 The NGG stream cipher

NGG(n, m) denotes a parameterized family of ciphers. For some fixed n and m, the NGG internal state consists of a public n-bit counter i, secret pseudorandom n-bit counter j and S table consisting of $N = 2^n$ m-bit values. The cipher consists of two separate algorithms (see Fig. 1 where $M = 2^m$):

- Key Scheduling Algorithm (KSA): In the *initialization* step of the KSA, the table S is initialized with prespecified publicly known random array **a**. Then, in the *scrambling* step of the KSA, these values are mixed depending on key bytes in a pseudorandom value as follows: the i-th and the j-th value are swapped and the sum of these two values is assigned to the i-th element. This is repeated for $i = 0, \ldots N-1$, where j is incremented pseudorandomly depending on the key.
- Pseudo Random Number Generation (PRNG): First, counters i and j are updated. Then, $S[i]$ and $S[j]$ values are swapped. Value $S[(S[i] + S[j]) \bmod N]$ is sent to the output and then changed to $S[i] + S[j]$.

Before using the cipher, it needs to be initialized by the KSA supplied with the secret key. The output of this process is a randomized secret internal state of NGG. During the encryption process, m-bit plaintext words are XORed with m-bit keystream

KSA
Initialization:
For $i = 0, ..N-1$
$S[i] = a_i$
$j = 0$
Scrambling:
For $i = 0, ..N-1$
$j = (j + S[i] + K[i \bmod l]) \bmod N$
Swap$(S[i], S[j])$
$S[i] = (S[i] + S[j]) \bmod M$

PRGA
Initialization:
$i = j = 0$
Loop:
$i = (i + 1) \bmod N$
$j = (j + S[i]) \bmod N$
Swap$(S[i], S[j])$
$t = (S[i] + S[j] \bmod M) \bmod N$
Output=$S[t]$
$S[t] = (S[i] + S[j]) \bmod M$

Fig. 1 NGG(n, m) specifications

words produced by the PRGA. Similarly, during the decryption process, the m-bit ciphertext words are XORed with the corresponding keystream words.

In [26] it was shown that NGG is distinguishable from a random sequence with about 100 keystream words. The attack relies on the fact that for three random S table entries, the relation $S[X] = S[Y] + S[Z]$ holds with biased probability, due to the update step of NGG. In [18], it was found that the least significant bit of NGG keystream word is biased, due to "bias inducing state", which occurs with probability 2^{-16}. Distinguisher based on this can be built by using $2^{32.89}$ keystream words. In Klein's work [8], in which RC4 used in WEP mode is shown to be weak, NGG is shown to be prone to a similar attack, which makes it insecure when IVs are concatenated to keys [8].

Unlike previous distinguishing attacks, our attack can naturally be extended to a key recovery attack. Throughout the rest of this paper, we consider the popular case of $n = 8$ and $m = 32$. It should be noted, however, that the attacks described in this paper becomes more sever as m increases (e.g., for $m = 64$).

3 Weakness in NGG KSA

In the initialization step of NGG KSA, $S[k]$ element is assigned $\mathbf{a}_k$, $k = 0 \ldots 255$, where array $\mathbf{a}$ is publicly known. One choice for this array is given in [17]. Then, in the scrambling step, for $i = 0, 1, 2, \ldots 255$, the i-th value is swapped with pseudorandom element of S and place i is assigned the sum of these two elements.

In this section, we show that the scrambling step does not randomize the S table sufficiently. Namely, after the KSA, most of the elements of S can be represented as a sum of one, two, three or four elements of $\mathbf{a}$ values. In other words, there is a high probability that, for random index k, one of the following four relations will hold:

- $S[k] = \mathbf{a}_{x_1}$ for some index x_1,
- $S[k] = (\mathbf{a}_{x_1} + \mathbf{a}_{x_2}) \bmod M$, for some indices x_1, x_2,
- $S[k] = (\mathbf{a}_{x_1} + \mathbf{a}_{x_2} + \mathbf{a}_{x_3}) \bmod M$, for some indices x_1, x_2, x_3,
- $S[k] = (\mathbf{a}_{x_1} + \mathbf{a}_{x_2} + \mathbf{a}_{x_3} + \mathbf{a}_{x_4}) \bmod M$, for some indices x_1, x_2, x_3, x_4,

where $0 \le x_i \le 255$. As will be shown, this is highly improbable for a randomly chosen 32-bit word. In the following we prove this observation by modelling the KSA procedure.

3.1 Definitions and assumptions

We say that the KSA is at step i, $i = 0 \ldots 256$, if i scrambling steps were executed. For example, KSA is at step 0 right after initialization step of KSA, and before the first scrambling step and at step 256 after it is finished. Accordingly, by j_i and S_i we denote j and S at step i of the KSA.

Let $W_n = \{\sum_{i=1}^{n} a_{x_i} |\ x_i \in \{0..255\}\}$. Unless otherwise specified, we always use the set of $\mathbf{a}$ values proposed in [17]. It should be noted, however, that changing the set $\mathbf{a}$ cannot prevent the attacks described in this paper. In general, $W_i, i = 1, 2, \cdots$ sets are not disjoint. However, for the choice of $\mathbf{a}$ in [17], $W_1 \cap W_2 = \emptyset$, $W_1 \cap W_3 = \emptyset$ and $W_2 \cap W_3 = \emptyset$. This is also very likely to be the case if $\mathbf{a}$ is chosen at random.

We approximate j value at each step of KSA by a pseudorandom number and make the usual independence assumptions throughout the proofs.

3.2 Deriving probabilities for S table entries after KSA

The KSA changes $S[k]$ values in a a structured way. This is shown by the following Lemma.

Lemma 1 *Let* $0 \le k \le 255$ *be an index in S. Then*

(a) $S_0[k], S_1[k], \ldots S_k[k] \in W_1$
(b) $S_{k+1}[k] \in W_{n+1}$, *for each n such that* $S_k[j_{k+1}] \in W_n$
(c) *If there exists* $k+2 \le t \le 256$ *such that* $j_t = k$, *let* t_0 *denote the smallest such number. Then,* $S_{k+2}[k], \ldots S_{t_0-1}[k] \in W_{n+1}$, $S_{t_0}[k], \ldots S_{256}[k] \in W_1$. *If not, then* $S_{k+2}[k], \ldots S_{256}[k] \in W_{n+1}$

Proof Statement (a) is proved by induction on k. For $k = 0$, the statement takes the form $S_0[0] \in W_1$, which holds since $S_0[0] = a_0 \in W_1$. Suppose statement (a) holds for some $0 \le k \le 254$, we prove that (a) also holds for $k+1$. Due to the KSA procedure specification, table S_{k+1} differs from S_k only in values at indices j_{k+1} and k. As for index j_{k+1}, according to the swap step and due to induction hypothesis, we have $S_{k+1}[j_{k+1}] = S_k[k] \in W_1$. As for index k, content of $S_{k+1}[k]$ has no relevancy for the statement. Thus, (a) holds for each $k = 0 \ldots 255$.

To prove (b), note that $S_{k+1}[k] = S_k[k] + S_k[j_{k+1}]$. According to (a), $S_k[k] \in W_1$. According to (b) assumption, for some n, $S_k[j_{k+1}] \in W_n$. Thus, $S_{k+1}[k] \in W_{n+1}$.

As for statement (c), suppose first that there does not exist t from the assumption of the statement. Due to (b), we have $S_{k+1}[k] \in W_{n+1}$. According to the above assumption no upcoming j_t will take value k. This value remains unchanged until the end of KSA, i.e., $S_{k+2}[k], \ldots S_{256}[k] \in W_{n+1}$. Now assume that there exists t from the assumption of (c) and let t_0 be the smallest such number. At steps $k+2, \ldots t_0 - 1$, S value on index k will not be changed and thus $S_{k+2}[k], \ldots S_{t_0-1}[k] \in W_{n+1}$ holds. However, in step t_0, value at index k is already overwritten by $S_{t_0-1}[t_0 - 1]$. According to (a), this value is a member of W_1 and thus $S_{t_0}[k], \ldots S_{256}[k] \in W_1$ also holds. □

Our goal is to show that probability of $S_{256}[k] \in W_1 \cup W_2 \cup W_3 \cup W_4$ is high. First, we estimate lower bounds for the probabilities $P[S_{256}[k] \in W_n]$, $n = 1, 2, 3, 4$ separately. To do this, we note that each KSA execution in our model uniquely corresponds to a tuple $(j_0, j_1, \ldots j_{255})$ and thus can be identified with it. Lower bounds are obtained by counting tuples that, according to Lemma 1, certainly yield a value $\in W_n$ at some index k.

In the Lemmas below, we also compute $P[S_i[k] \in W_n]$, $n = 1, 2, 3, 4$ at some of the steps $i \ne 256$. This is necessary because, as can be seen from the obtained formulas, $P[S_{256}[k] \in W_n]$, $n = 2, 3, 4$ depend on some of the $P[S_i[k] \in W_{n-1}]$.

Lemma 2 *Let i and k be the KSA step and S table index such that* $i \ge k + 1$. *Then,*

$$P[S_i[k] \in W_1] \ge 1 - (255/256)^{i-k-1}$$

Proof For $i = k + 1$, the statement holds trivially since the right side of the inequality is equal to 0. Let $i \geq k + 2$. Part (c) of Lemma 1 provides a sufficient condition for event $S_i[k] \in W_1$ to hold. Namely, if for some of the $t = k + 2, \ldots i$, we have $j_t = k$, $S_i[k] \in W_1$ will hold true. Thus, we can derive a lower bound for the probability of event in question as follows:

$$\begin{aligned} P[S_i[k] \in W_1] \geq \\ &= P[j_{k+2} = k \text{ or } j_{k+3} = k \text{ or } \ldots \text{ or } j_i = k] \\ &= 1 - P[j_{k+2} \neq k, j_{k+3} \neq k, \ldots j_i \neq k] \\ &= 1 - P[j_{k+2} \neq k] \times P[j_{k+3} \neq k] \times \ldots \times P[j_i \neq k] \\ &= 1 - (255/256)^{i-k-1} \end{aligned}$$

□

Lemma 3 *Let i and k be the KSA step and S table index such that $i \geq k + 1$. Then,*

$$P[S_i[k] \in W_2] \geq \left(\frac{\sum_{l=0}^{k-1} P[S_k[l] \in W_1]}{256} + \frac{256 - k}{256} \right) \times \left(\frac{255}{256} \right)^{i-k-1}$$

where we take that $\sum_{l=0}^{k-1} P[S_k[l] \in W_1] = 0$ when $k = 0$.

Proof Similar to the proof of previous Lemma, we find a lower bound for $P[S_i[k] \in W_2]$ by noting that a combination of parts (b) and (c) of Lemma 1 provide a sufficient condition for the event in question. Namely, according to part (b) of Lemma 1, if $S_k[j_{k+1}] \in W_1$, $S_{k+1}[k] \in W_2$ will hold true. According to (c), this will remain so until step i if by then none of j indices takes value k. Thus,

$$\begin{aligned} P[S_i[k] \in W_2] \geq \\ &= P(S_k[j_{k+1}] \in W_1, j_{k+2} \neq k, \ldots, j_i \neq k) \\ &= P(S_k[j_{k+1}] \in W_1) \times P(j_{k+2} \neq k), \times \ldots \times P(j_i \neq k) \\ &= (P(j_{k+1} < k) \times P(S_k[j_{k+1}] \in W_1 | j_{k+1} < k) \\ &\quad + P(j_{k+1} \geq k) \times P(S_k[j_{k+1}] \in W_1 | j_{k+1} \geq k)) \times (255/256)^{i-k-1} \\ &= \left(\frac{k}{256} \left(\frac{1}{k} \times P[S_k[0] \in W_1] + \ldots + \frac{1}{k} \times P[S_k[k-1] \in W_1] \right) \right. \\ &\quad \left. + \frac{256 - k}{256} \right) \times (255/256)^{i-k-1} \\ &= \left(\frac{\sum_{l=0}^{k-1} P[S_k[l] \in W_1]}{256} + \frac{256 - k}{256} \right) \times \left(\frac{255}{256} \right)^{i-k-1} \end{aligned}$$

If $k = 0$, taking $\sum_{l=0}^{k-1} P[S_k[l] \in W_1] = 0$ is justified by $P[j_{k+1} < k] = 0$. In that case, we have $P[S_i[0] \in W_2] = \left(\frac{255}{256}\right)^{i-1}$ for every $i \geq 1$. □

Lemma 4 *Let i and k be the KSA step and S table index such that $i \geq k + 1$. Then,*

$$P[S_i[k] \in W_3] \geq \frac{\sum_{t=0}^{k-1} P[S_k[t] \in W_2]}{256} \times \left(\frac{255}{256} \right)^{i-k-1}$$

where we take that $\sum_{i=0}^{k-1} P[S_i[k] \in W_2] = 0$ *when* $k = 0$.

Proof As in the proofs of the previous two Lemmas, the event in question can be lower bounded by events for which the probability can be easily calculated. Namely, according to parts (b) and (c) of Lemma 1, if $S_k[j_{k+1}] \in W_2$, $S_{k+1}[k] \in W_3$ will hold true and this will remain so until step i if by then none of j indices take value k.

$$\begin{aligned}
&P[S_i[k] \in W_3] \\
&\quad = P(S_k[j_{k+1}] \in W_2,\ j_{k+2} \neq k, \ldots, j_i \neq k) \\
&\quad = P(S_k[j_{k+1}] \in W_2) \times P(j_{k+2} \neq k) \times \ldots \times P(j_i \neq k) \\
&\quad = (P(j_{k+1} < k) \times P(S_k[j_{k+1}] \in W_2 | j_{k+1} < k) \\
&\qquad + P(j_{k+1} \geq k) \times P(S_k[j_{k+1}] \in W_2 | j_{k+1} \geq k)) \times \left(\frac{255}{256}\right)^{i-k-1} \\
&\quad = \frac{k}{256}\left(\frac{1}{k} \times P[S_k[0] \in W_2] + \ldots + \frac{1}{k} \times P[S_k[k-1] \in W_2]\right) \\
&\qquad \times (255/256)^{i-k-1} = \frac{\sum_{i=0}^{k-1} P[S_k[i] \in W_2]}{256} \times \left(\frac{255}{256}\right)^{i-k-1}
\end{aligned}$$

When $k = 0$, taking $\sum_{l=0}^{k-1} P[S_k[l] \in W_2] = 0$ is justified by $P[j_{k+1} < k] = 0$ and $P[S_k[j_{k+1}] \in W_2 | j_{k+1} \geq k] = 0$. In that case, we have $P[S_i[0] \in W_3] = 0$ for every $i \geq 1$. □

Lemma 5 *Let i and k be the KSA step and S table index such that* $i \geq k + 1$. *Then,*

$$P[S_i[k] \in W_4] \geq \frac{\sum_{t=0}^{k-1} P[S_k[t] \in W_3]}{256} \times \left(\frac{255}{256}\right)^{i-k-1}$$

where we take that $\sum_{i=0}^{k-1} P[S_i[k] \in W_2] = 0$ *when* $k = 0$.

Proof Analogous to proof of previous Lemma. □

Figure 2 illustrates the lower bounds obtained by Lemmas 2–5 for S values at indices $k = 0..255$ after the KSA has finished. Since the probability that a random number is an element of W_1, W_2, W_3 and W_4 is $\frac{|W_1|}{2^{32}} = \frac{256}{2^{32}} \approx 5.96 \times 10^{-8}$, $\frac{|W_2|}{2^{32}} = \frac{32892}{2^{32}} \approx 7.66 \times 10^{-6}$, $\frac{|W_3|}{2^{32}} = \frac{2792954}{2^{32}} \approx 0.00065$ and $\frac{|W_4|}{2^{32}} = \frac{118090021}{2^{32}} \approx 0.0275$ respectively, it is obvious that S table is biased. Sets W_n, $n \geq 5$ were not considered since the probability that a random number is contained in these sets is relatively high and hence the corresponding distinguishers would be useless.

Note that even though in Lemmas 2–5 we only estimated lower bounds, obtained values are very close to exact ones. This is because the events by which we approximated $S_i[k] \in W_n$ in the proofs of the Lemmas exhaust most of the event space. For example, in Lemma 2, event $S_i[k] \in W_1$ was lower bounded by the event that KSA operations will write one of the **a** values at place k at some point in time. The only other way $S_i[k] \in W_1$ event can hold is that KSA leaves an addition of **a** values at place k and this addition turns out to be an element of W_1 too, which is possible since W_i, $i = 1, 2 \ldots$ are not generally disjoint. However, the probability of

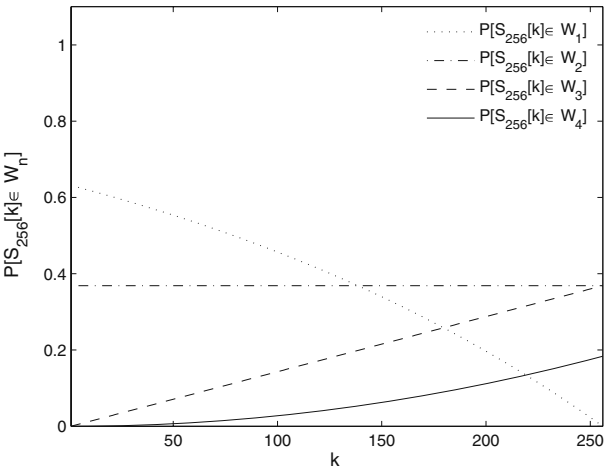

Fig. 2 Probabilities that, after KSA, $S[k]$ will be representable as a sum of 1, 2, 3 or 4 values from **a** set

this is very small and thus our estimated lower bound is actually a good estimate for exact $P[S_i[k] \in W_1]$ value. Similar reasoning applies for Lemmas 3, 4 and 5. Using 10, 000 randomly generated 128-bit keys, the experimentally calculated values of $P[S_i[k] \in W_n], n = 1, 2, 3$ confirmed the above claim.

4 Distinguishing NGG from a random stream

As noted in previous section, the probability that $S_{256}[k] \in W_n,\ n = 1, 2, 3, 4$ is significantly greater than the probability that some random number will be an element of these sets. However, to maximize this bias, we consider the union of these sets. Figure 3 illustrates the probability that after the KSA, the S values will be contained in different unions of W sets. Depending of which sets are included in

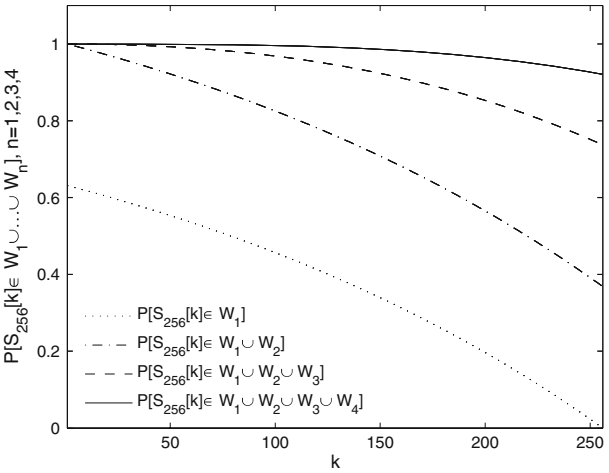

Fig. 3 Probability that after KSA S[k] will be an element of different unions of W sets

the union, different distinguishing criteria can be formulated. As will be shown, the probability that a random number will be in any of these unions is still small.

In the following theorem, we determine a lower bound on the success probabilities of the distinguishers using criteria based on different unions of the $W_i, i = 1, 2, 3, 4$ sets.

Theorem 1 *Using the first NGG PRGA keystream output word, k_1, lower bound of success probability of distinguishing k_1 from a random stream are given in Table 1 below.*

Proof First, we prove the statement for the distinguisher 3, in which the distinguisher is based on $k_1 \in W_1 \cup W_2 \cup W_3$ criterion. Proofs for distinguisher 1 and 2 are similar. Suppose that the given word, o, is the first NGG PRGA output word, i.e., $o = k_1 = S_{256}[k]$ for some k. Then, lower bound of the probability that it will be an element of $W_1 \cup W_2 \cup W_3$ can be calculated by applying the Bayes' formula over all possible values k, using the fact that sets W_1, W_2 and W_3 are mutually disjoint and substituting according values to lower bounds from Lemmas 2–4.

$$P[S_{256}[k] \in W_1 \cup W_2 \cup W_3] \tag{1}$$

$$= \sum_{i=0}^{255} P[k = i] P[S_{256}[k] \in W_1 \cup W_2 \cup W_3 | k = i] \tag{2}$$

$$= \frac{1}{256} \sum_{i=0}^{255} (P[S_{256}[i] \in W_1] + P[S_{256}[i] \in W_2] + P[S_{256}[i] \in W_3]) \tag{3}$$

$$\geq 0.92$$

Thus, in the case that the word o is in fact an NGG first keystream word, probability that the distinguisher will make a right decision is greater than 0.92. In the case of a random word, the probability that it will not be representable as a sum of one, two or three elements of **a** is given by

$$1 - \frac{|W_1 \cup W_2 \cup W_3|}{2^{32}} \approx 0.9993.$$

Thus, the success probability of distinguisher based on $k_1 \in W_1 \cup W_2 \cup W_3$ criterion will be greater than $0.5 \times 0.92 + 0.5 \times 0.9993 = 0.9597$.

As for distinguisher 4, event $S_{256}[k] \in W_1 \cup W_2 \cup W_3 \cup W_4$ is not a disjoint union of events $S_{256}[k] \in W_1, n = 1, 2, 3, 4$ and thus going from (2) to (3) in the corresponding proof would not be justified. However, the lower bound events established in the proofs of Lemmas 2–5 are mutually disjoint. Let L_1 be the lower bound event used in Lemma 2 for $i = 256$ $L_1 = \{j_{k+1} = k$ or $j_{k+2} = k$ or $\ldots$ or $j_{255} = k\}$ and let

Table 1 Lower bounds on the success rate of different distinguishers

#	Distinguishing criterion	P[Correct decision]
1	$k_1 \in W_1$	0.6836
2	$k_1 \in W_1 \cup W_2$	0.8679
3	$k_1 \in W_1 \cup W_2 \cup W_3$	0.9597
4	$k_1 \in W_1 \cup W_2 \cup W_3 \cup W_4$	0.9769

Table 2 Probability of error for distinguishers based on the m-th keystream word

m	16	32	64	128	256	512	1024
P[Error]	0.050	0.066	0.077	0.150	0.288	0.407	0.484

events L_2, L_3 and L_4 be the corresponding events from Lemmas 3–5. Then, we have $P[S_{256}[k] \in W_1 \cup W_2 \cup W_3 \cup W_4] \geq P[L_1 \cup L_2 \cup L_3 \cup L_4] = P[L_1] + P[L_2] + P[L_3] + P[L_4]$. The proofs of Lemmas 2–5 provide exact values for $P[L_1]$, $P[L_2]$, $P[L_3]$, $P[L_4]$. The rest of the proof is analogous to previous ones. □

4.1 Distinguishers based on the m-th keystream word

In our reasoning above, we only considered the first keystream word. Calculated probabilities apply only to this word since the S table is updated at each step. However, it is not hard to see that the same distinguisher is applicable to the m-th keystream word k_m too, provided that m is small, but with decreasing success rate as m increases.

To assess success rate of the distinguisher $k_m \in W_1 \cup W_2 \cup W_3$, we conducted the following experiment. For 10000 times, the NGG cipher was initialized by KSA for randomly generated key. The percentage of k_m values which were not representable as a sum of one, two or three $\mathbf{a}$ numbers was taken to be the probability of false negatives for distinguisher based on k-th element. Table 2 presents results for some m values. It can be seen that distinguisher guess still differs from random guess at $m = 1024$. This implies that the traditional solution to resist distinguishing attacks against RC4-like cipher by discarding some of the output stream words may not practicably work in this case. Naturally, the success of the distinguisher can be arbitrarily increased by looking at several keystream words at once.

5 Key information recovery

In this section, we show how to use the observations above to reduce the entropy of the the secret key based on the first few kilobytes of the keystream.

From the algorithm specifications, it is clear that knowing $j_0, j_1, \ldots j_{16}$, i.e., the values assumed by the pseudorandom counter j in the first 16 rounds of the scrambling step of KSA, is equivalent to knowing all key bytes $K_0, \ldots K_{15}$. Thus, it suffices to focus on reducing the entropy of the vector $(j_0, j_1, \ldots j_{16})$.

5.1 Recovering the first secret key byte

Among the first 1024 keystream words, all entries of table S_{256} will be present with high probability. The reason for this is that the NGG PRGA sends the S table entries to the output unmasked and changes them only once they are outputted.

As mentioned above, to determine K_0, it suffices to find j_1 since $K_0 = j_1 - 0 - \mathbf{a}_0$. At the first step of the KSA scrambling step, $S[0] + S[j_1] = \mathbf{a}_0 + \mathbf{a}_{j_1}$ is written to $S[0]$. The only way $S[0]$ can be changed again is that that for some $i \geq 2$, $j_i = 0$. The probability that this will not happen, i.e., $S[0]$ will remain $\mathbf{a}_0 + \mathbf{a}_{j_1}$ is $(255/256)^{255} = 0.3686$. By trying to decompose each of the first 1024 keystream words in the form

of $\mathbf{a}_0 + \mathbf{a}_x$, with probability 0.3686, one of the obtained x values will be equal to j_1. However, there might be more than one $\mathbf{a}_0 + \mathbf{a}_x$ element in S table. Namely, at the first KSA scrambling step, the value $S[0] = \mathbf{a}_0$ is also written to $S[j_1]$. At the j_1-th scrambling step, $S[j_1]$ will be added another element from the table. If $S[j_1]$ is not altered again by any of remaining steps, there will be two $\mathbf{a}_0 + \mathbf{a}_x$ elements in the table and there would be two candidates for j_1.

Similar reasoning applies for the case when, after first scrambling step of KSA, the value $S[0]$ is altered again by some j_i. Then, a value of the form $\mathbf{a}_0 + \mathbf{a}_{x_1} + \mathbf{a}_{x_2}$ will be present in the table after the KSA finishes, provided that it is not altered again. The algorithm can search for values of the form $\mathbf{a}_0 + \mathbf{a}_{x_1}$, $\mathbf{a}_0 + \mathbf{a}_{x_1} + \mathbf{a}_{x_2}$, $\mathbf{a}_0 + \mathbf{a}_{x_1} + \mathbf{a}_{x_2} + \mathbf{a}_{x_3}$ and $\mathbf{a}_0 + \mathbf{a}_{x_1} + \mathbf{a}_{x_2} + \mathbf{a}_{x_3} + \mathbf{a}_{x_4}$ among the first 1024 keystream words. There is a tradeoff between number of candidates and the corresponding success probability. In the following section, we experimentally evaluate a full key recovery algorithm based on first two forms.

5.2 Full key recovery algorithm

To derive a candidate for j_t, $t = 2, \ldots 16$, we use the same idea, only now we have smaller probability of success since $S_{t-1}[t-1]$ and $S_{t-1}[j_t]$ might not be equal to $\mathbf{a}_{t-1}$ and $\mathbf{a}_{j_t}$, respectively. The probability that this will hold decreases as t increases.

Let $cand(J_i)$ denote the set of candidates for value j_i. Let $cand(J) = cand(J_0) \times \ldots \times cand(J_{16})$ denote the set of candidates for the vector $j = (j_0, \ldots j_{16})$. Let $f((j_0, \ldots j_{16})) = (k_0, \ldots k_{15})$ where $k_i = (j_{i+1} - j_i - \mathbf{a}[i]) \bmod 256$. The set of candidates for key K can be obtained as $cand(K) = f(cand(J))$.

Figure 4 shows the algorithm which recovers $cand(J)$, and consequently the set of secret key candidates, based on the first 1024 keystream words.

It should be noted that computing $\mathbf{a}_{i-1} + \mathbf{a}_{x_1}$, $\mathbf{a}_{i-1} + \mathbf{a}_{x_1} + \mathbf{a}_{x_2}$ for each i, x_1 and x_2 is repeated in each loop pass. Thus, the algorithm can be efficiently executed in negligible time if these values are precomputed offline and sorted. However, as will be seen, the number of candidates for secret key K that the algorithm yields is close to

---

INPUT: First 1024 NGG keystream words

OUTPUT: A set of secret key candidates (cand(K))

1: Let $cand(J_0) = \{0\}$ and $cand(J_i) = \emptyset$ for $1 \leq i \leq 16$

2: For each word w of first 1024 keystream words do

3: For $i = 1, \ldots 16$, do

- Check if w can be written as $\mathbf{a}_{i-1} + \mathbf{a}_{x_1}$, where $x_1 \in \{0, 1, \ldots 255\}$. If yes, add x_1 to $cand(J_i)$
- Check whether w can be written as $\mathbf{a}_{i-1} + \mathbf{a}_{x_1} + \mathbf{a}_{x_2}$, where $x_1, x_2 \in \{0, 1, \ldots 255\}$. If yes, add x_1 and x_2 to $cand(J_i)$

4: $cand(J) = cand(J_0) \times cand(J_1) \times \ldots \times cand(J_{16})$

5: Return $cand(K) = f(cand(J))$

Fig. 4 Key recovery algorithm

Table 3 Success rate and average number of candidates for j_i, $1 \leq i \leq 16$

i	1	2	3	4	5	6	7	8
$P[j_i \in cand(J_i)]$	0.7223	0.7373	0.6895	0.7242	0.6848	0.7221	0.6885	0.6913
$\|cand(J_i)\|$	3.9362	4.0994	4.0225	4.0629	3.9268	3.9740	3.9510	3.8712
i	9	10	11	12	13	14	15	16
$P[j_i \in cand(J_i)]$	0.7000	0.6577	0.6558	0.6538	0.6702	0.6462	0.6375	0.6578
$\|cand(J_i)\|$	4.0837	3.7904	3.9808	3.8288	3.8221	3.8365	3.8144	3.9250

2^{32} and to discard wrong candidates, a computation of around 2^{32} operations cannot be avoided.

Since the function f is 1–1, only the correct j vector candidate will be mapped to the correct key. Thus, the effectiveness of proposed algorithm can be measured as follows:

- $P[(k_0, \ldots k_{15}) \in cand(K)] = P[(j_0, \ldots j_{16}) \in cand(J)]$, i.e., the probability that correct key, or equivalently, j values, will be found. For some particular $i \in \{1, \ldots 16\}$, $j_i \in cand(J_i)$ will hold if at most 2 **a** values are added to $\mathbf{a}_i$ which is placed in $S_0[i]$ at the beginning of KSA scrambling.
- $|cand(K)| = |cand(J)|$, i.e., the number of key candidates. For some particular $i \in \{1, \ldots 16\}$, $|cand(J_i)|$ depends on the number of interactions between $S_0[i] = \mathbf{a}_i$ and other **a** values throughout the KSA. Since the additions continue during the PRGA, wrong candidates not present in table at moment $i = 256$ might emerge during 1024 PRGA steps performed during the algorithm.

Deriving an analytical expression for $P[j_i \in cand(J_i)]$ and $|cand(J_i)|$ seems to be a relatively hard combinatorial problem. Thus, in order to test the success rate of the algorithm, the following experiment was conducted. For 1000 randomly generated keys, sets $cand(J_i)$, $i = 1.\ldots.16$ were calculated according to the algorithm above. Percentage of times $j_i \in J_i$ was true and average $|J_i|$ values are shown in Table 3. Value j_0 is omitted because it is always equal to 0.

For example, according to the experiment above, the correct j_1 value will be among on average 3.9362 candidates proposed by the algorithm with probability 0.7223. Since $j_0 = 0$ by algorithm specification, we then have K_0 will be among 3.9362 candidates with probability 0.7223. The probability that both $j_1 \in J_1$ and $j_2 \in J_2$ is 0.7223×0.7373 and in that case, both K_0 and K_1 will be among proposed candidates. To generalize, $K \in cand(K)$ with probability $\approx 2^{-8.8}$ (obtained by multiplying all the probabilities in Table 3) and $|cand(K)| \approx 2^{31.6}$ (obtained by multiplying the number of candidates in Table 3). In other words, the above result can be restated as follows: for approximately $2^{119.2}$ of the 128-bit keys, the secret key can be recovered with around 2^{32}, instead of 2^{128}, steps of computation.

6 Conclusion

The NGG key scheduling algorithm does not randomize its initial internal state table sufficiently. In particular, after the KSA, the cipher internal state is highly biased which allows us to distinguish NGG from a random stream based only on the first keystream word. Furthermore, the bias left by the KSA reveals information

about the secret key. Since NGG PRGA sends the internal state elements to output without masking, information about the secret key can be recovered by examining the first few kilobytes of the keystream. Our experimental results show that for approximately $2^{119.2}$ of the 128-bit keys, the secret key can be recovered with around 2^{32} steps. Based on the analysis above, it is clear that NGG is insecure. Designing an efficient and secure generalization of RC4 for 32/64-bit processors remains a challenging research problem.

Acknowledgements This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant N00930. The authors would like to thank the anonymous reviewers for their comments that helped improve the presentation of the paper.

References

1. Bernstein, D.: Which eStream cipher have been broken? ECRYPT report. http://www.ecrypt.eu.org/stream/papersdir/2008/010.pdf (2008)
2. Biham, E., Carmeli, Y.: Efficient reconstruction of RC4 keys from internal states. In: Proc. of Fast Software Encryption, FSE 2008. LNCS, vol. 5086, pp. 270–288. Springer, New York (2008)
3. Fluhrer, S.R., McGrew, D.A.: Statistical analysis of the alleged RC4 keystream generator. In: Proc. of Fast Software Encryption, FSE 2000. LNCS, vol. 1978, pp. 19–30. Springer, New York (2000)
4. Fluhrer, S.R., Mantin, I., Shamir, A.: Weaknesses in the key scheduling algorithm of RC4. In: Proc. of Selected Areas in Cryptography, SAC 2001. LNCS, vol. 2259, pp. 1–24. Springer, New York (2001)
5. Golić, J.D.: Linear statistical weakness of alleged RC4 keystream generator. In: Proc. of EUROCRYPT '97. LNCS, vol. 1233, pp. 226–238. Springer, New York (1997)
6. Gong, G., Chand, K., Hell, M., Nawaz, Y.: Towards a general RC4-like keystream generator. In: Proc. of CISC 2005. LNCS, vol. 3822, pp. 162–174. Springer, New York (2005)
7. Hawkes, P., Rose, G.G.: On the applicability of distinguishing attacks against stream ciphers. In: Proc. of the Third NESSIE Workshop (2002)
8. Klein, A.: Attacks on the RC4 stream cipher. http://cage.ugent.be/~klein/RC4/RC4-en.ps
9. Knudsen, L.R., Meier, W., Prenel, B., Rijmen, V., Verdoolaege, S.: Analysis methods for (alleged) RC4. In: Proc. of ASIACRYPT'98. LNCS, vol. 1514, pp. 327–341. Springer, New York (1998)
10. Mantin, I., Shamir, A.: A practical attack on broadcast RC4. In: Proc. of fast software encryption, FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer, New York (2001)
11. Mantin, I.: Predicting and distinguishing attacks on RC4 keystream generator. In: Proc. of EUROCRYPT' 2005. LNCS vol. 3494, pp. 491–506. Springer, New York (2005)
12. Mantin, I.: A practical attack on the fixed RC4 in the WEP mode. In: Proc. of ASIACRYPT 2005. LNCS, vol. 3788, pp. 395–411. Springer, New York (2005)
13. Maximov, A.: Two linear distinguishing attacks on VMPC and RC4A and weakness of RC4 family of stream ciphers. In: Proc. of Fast Software Encryption, FSE 2005. LNCS, vol. 3357, pp. 342–358. Springer, New York (2005)
14. Maximov, A., Khovratovich, D.: New state recovery attack on RC4. In: Proc. of CRYPTO 2008. LNCS, vol. 5157, pp. 297–316. Springer, New York (2008)
15. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptographic Research. CRC, Boca Raton (1996)
16. Mironov, I.: Not (So) Random shuffle RC4. In: Proc. of CRYPTO' 2002. LNCS, vol. 2442, pp. 304–319. Springer, New York (2002)
17. Nawaz, Y., Gupta, K.C., Gong, G.: A 32-bit RC4-like keystream generator. Technical report CACR 2005–21, Center for Applied Cryptographic Research, University of Waterloo. http://www.cacr.math.uwaterloo.ca/tech_reports.html. Also available at Cryptology ePrint Archive, 2005–175, http://eprint.iacr.org/2005/175 (2005)
18. Paul, S., Preenel, B.: On the (In)security of stream ciphers based on arrays and modular addition. In: Proc. of ASIACRYPT 2006. LNCS, vol. 4284, pp. 69–83. Springer, New York (2006)

19. Paul, S., Preenel, B.: A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In: Proc. of Fast Software Encryption, FSE 2004. LNCS, vol. 3017, pp. 245–259. Springer, New York (2004)
20. Paul, G., Maitra, S.: Permutation after RC4 key scheduling reveals the secret key. In: Proc. of Selected Areas in Cryptography, SAC 2007. LNCS, vol. 4876, pp. 360–377. Springer, New York (2007)
21. Paul, S., Preneel, B.: A new weakness in RC4 keystream generator and an approach to improve the security of the cipher. In: Proc. of Fast Software Encryption, FSE 2004. LNCS, vol. 3017, pp. 245–259. Springer, New York (2004)
22. Tews, E., Weinmann, R.P., Pyshkin, A.: Breaking of 104 bit WEP in less than 60 seconds. http://eprint.iacr.org/2007/120.pdf (2007)
23. Tomašević , V., Bojanić, S., Nieto-Taladriz, O.: Finding an internal state of RC4 stream cipher. Inf. Sci. Int. J. **177**(7), 1715–1727 (2007)
24. Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T.: A distinguishing attack on a fast software-implemented RC4-like stream cipher. IEEE Trans. Inf. Theory **53**(9) (2007)
25. Vaudenay, S., Vuagnoux, M.: Passive-only key recovery attacks on RC4. In: Proc. of Selected Areas in Cryptography, SAC 2007. LNCS, vol. 4876, pp. 344–359. Springer, New York (2007)
26. Wu, H.: Cryptanalysis of a 32-bit RC4-like stream cipher, Cryptology ePrint archive, 2005–219, IACR. eprint.iacr.org/2005/219.pdf (2005)
27. Zoltak, B.: VMPC one-way function and stream cipher. In: Proc. of Fast Software Encryption, FSE 2004. LNCS, vol. 3017, pp. 210–225. Springer, New York (2004)