Attack the Dragon

Håkan Englund and Alexander Maximov

Dept. of Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden

Abstract. Dragon is a word oriented stream cipher submitted to the ECRYPT project, it operates on key sizes of 128 and 256 bits. The original idea of the design is to use a nonlinear feedback shift register (NLFSR) and a linear part (counter), combined by a filter function to generate a new state of the NLFSR and produce the keystream. The internal state of the cipher is 1088 bits, i.e., any kinds of TMD attacks are not applicable. In this paper we present two statistical distinguishers that distinguish Dragon from a random source both requiring around $O(2^{155})$ words of the keystream. In the first scenario the time complexity is around $O(2^{155+32})$ with the memory complexity $O(2^{32})$, whereas the second scenario needs only $O(2^{155})$ of time, but $O(2^{96})$ of memory. The attack is based on a statistical weakness introduced into the keystream by the filter function F. This is the first paper presenting an attack on Dragon, and it shows that the cipher does not provide full security when the key of size 256 bits is used.

1 Introduction

A stream cipher is a cryptographic primitive used to ensure privacy over a communication channel. A common way to build a stream cipher is to use a keystream generator (KSG) and add the plaintext and the output from the keystream generator, resembling a one-time pad. A block cipher is another cryptographic primitive, which could be considered as a one-to-one function, mapping a block of the plaintext to a block of the ciphertext. Although block ciphers are well studied, stream ciphers can offer certain advantages compared to block ciphers. Stream ciphers can offer much higher speed, and can be constructed to be much smaller in hardware, and thus they are of great interest to the industry. To mention a few of the most recent proposals of such word-oriented KSGs are, e.g., VMPC [1], RC4A [2], RC4 [3], SEAL [4], SOBER [5], SNOW [6, 7], PANAMA [8], Scream [9], MUGI [10], Helix [11], Rabbit [12], Turing [13], etc.

The NESSIE project [14] was funded by the European Unions Fifth Framework Program and was launched in 2000. The main objective was to collect a portfolio of strong cryptographic primitives from different fields of cryptography, one of those fields was stream ciphers. During those three years of NESSIE new techniques for cryptanalysis on stream ciphers were found, and many new proposals were broken. After a few rounds of the project evaluation, all of the stream cipher proposals were found to contain some weaknesses. At the end, no stream cipher was included in the final portfolio.

The situation clearly requires the cryptographic community devote greater attention to design and analysis of stream ciphers. Due to this reason, the European project ECRYPT announced a call for stream cipher primitives. 35 proposals were submitted to the project by April 2005, and most of them were presented at the workshop SKEW 2005 [15] in May.

Cryptanalysis techniques discovered during the NESSIE project have allowed to strengthen new designs greatly, and to break new algorithms has become more difficult. However, there are many good submissions to ECRYPT, and the stream cipher Dragon [16] is one of them.

Dragon, designed by a group of researches, Ed Dawson et. al., is a word oriented stream cipher based on a linear block (counter) and a *nonlinear feedback shift register* (NLFSR) with a very large internal state of 1088 bits, which is updated by a nonlinear function denoted by F. This function is also used as a filter function producing the keystream. The idea to use a NLFSR is quite modern, and there are not many cryptanalysis techniques on NLFSRs yet found.

This is the first work which propose an attack on Dragon. In a distinguishing attack one has to decide whether a given sequence (keystream) is the product of a cipher, or a truly random generator. In this paper we show how statistical weaknesses in the F function can be used to create a distinguisher for Dragon. Our distinguishing attack requires around $O(2^{155})$ words of keystream from Dragon, it has time complexity $O(2^{155+32})$ and needs $O(2^{32})$ of memory, an alternative method is also presented that only requires time complexity $O(2^{155})$ but needs $O(2^{96})$ of memory. This is an academic attack which shows that Dragon does not provide full security when a key of size 256 bits is used, i.e., it can be broken faster than exhaustive search. This kind of analysis is, perhaps, the most powerful attack on stream ciphers, and, in some cases, it can be turned into a key recovery attack.

The outline of the paper is the following. In Section 2 a short description of the stream cipher Dragon is given. Afterward, in Section 3, we derive linear relations and build our distinguisher. In Section 4 we summarize different attack scenarios on Dragon, and finally, in Section 5 we present our results, make conclusions and discuss possible ways to overcome the attack.

1.1 Notations and Assumptions

For notation purposes we use $\oplus$ and $\boxplus$ to denote 32 bit parallel XOR and arithmetical addition modulo 2^{32}, respectively. By $x \gg n$ we denote a binary shift of x by n bits to the right. We write $x^{(t)}$ to refer the value of a variable x at the time instance t. By $P_{\texttt{Expr}}$ we denote a distribution of a random variable or an expression "`Expr`".

To build the distinguisher, we first make two reasonable assumptions common for linear cryptanalysis:

(a) Assume that at any time t the internal state of NLFSR is from the uniform distribution, i.e., the words B_i are considered independent and uniformly distributed;

(b) Assume that the keystream words are independent.

2 A Short Description of Dragon

Dragon is a stream cipher constructed using a large nonlinear feedback shift register, an update function denoted by F, and a memory denoted by M [1] . It is a word oriented cipher operating on 32 bit words, the NLFSR is 1024 bits long, i.e., 32 words long. The words in the internal state are denoted by $B_i,\ 0 \leq i \leq 31$. The memory M (counter) contains 64 bits, which is used as a linear part with the period of 2^{64}. The cipher handles two key sizes, namely 128 bits keys and 256 bit keys, in our attack we disregard the initialization procedure and just assume that the initial state of the NLFSR is truly random.

Each round the F function takes six words as input and produces six words of output, as shown in Figure 1. These six words, denoted by a, b, c, d, e, f, are

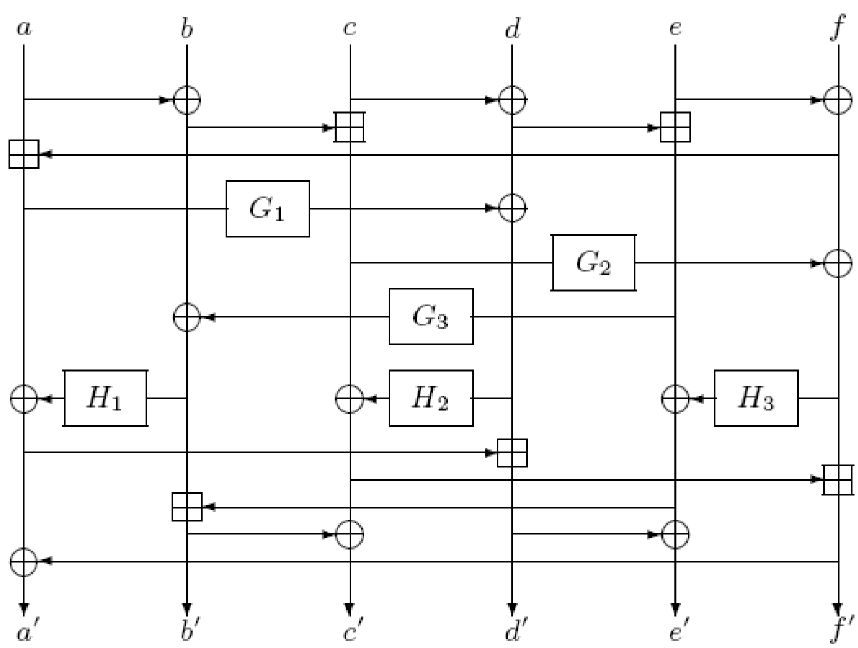

Fig. 1. F-function.

formed from words of the NLFSR and the memory register M, as explained in

[1] This is rather a new way to design stream ciphers, when two independent linear and nonlinear parts are combined by a filter function. A similar idea is used in other proposals to ECRYPT, e.g., stream cipher Grain and others.

(1), where $M = (M_L||M_R)$.

$$\begin{array}{lll} a = B_0 & b = B_9 & c = B_{16} \\ d = B_{19} & e = B_{30} \oplus M_L & f = B_{31} \oplus M_R \end{array} \tag{1}$$

The F function uses six $\mathbb{Z}_{2^{32}} \rightarrow \mathbb{Z}_{2^{32}}$ S-boxes $G_1,\ G_2,\ G_3,\ H_1,\ H_2$ and H_3, the purpose of which is to provide high algebraic immunity and non-linearity. These S-boxes are constructed from two other fixed $\mathbb{Z}_{2^8} \rightarrow \mathbb{Z}_{2^{32}}$ S-boxes, S_1 and S_2, as shown below.

$$\begin{aligned} G_1(x) &= S_1(x_0) \oplus S_1(x_1) \oplus S_1(x_2) \oplus S_2(x_3), \\ G_2(x) &= S_1(x_0) \oplus S_1(x_1) \oplus S_2(x_2) \oplus S_1(x_3), \\ G_3(x) &= S_1(x_0) \oplus S_2(x_1) \oplus S_1(x_2) \oplus S_1(x_3), \\ H_1(x) &= S_2(x_0) \oplus S_2(x_1) \oplus S_2(x_2) \oplus S_1(x_3), \\ H_2(x) &= S_2(x_0) \oplus S_2(x_1) \oplus S_1(x_2) \oplus S_2(x_3), \\ H_3(x) &= S_2(x_0) \oplus S_1(x_1) \oplus S_2(x_2) \oplus S_2(x_3), \end{aligned}$$

where 32 bits of input, x, is represented by its four bytes as $x = x_0||x_1||x_2||x_3$.

The exact specification of the S-boxes can be found in [16]. The output of the function F is denoted as (a', b', c', d', e', f'), from which the two words a' and e' forms 64 bits of keystream as $k = a'||e'$. Two other output words from the filter function are used to update the NLFSR as follows $B_0 = b'$, $B_1 = c'$, the rest of the state is updated as $B_i = B_{i-2},\ 2 \leq i \leq 31$. A short description of the keystream generation function is summarized in Figure 2.

Input = $\{B_0|| \ldots ||B_{31}, M\}$
1. $(M_L||M_R) = M$.
2. $a = B_0,\ b = B_9,\ c = B_{16},\ d = B_{19},\ e = B_{30} \oplus M_L,\ f = B_{31} \oplus M_R$.
3. $(a', b', c', d', e', f') = F(a, b, c, d, e, f)$.
4. $B_0 = b',\ B_1 = c'$
5. $B_i = B_{i-2},\ 2 \leq i \leq 31$.
6. $M++$
7. $k = a'||e'$

Output = $\{k, B_0, \ldots, B_{31}, M\}$

Fig. 2. Dragons's Keystream Generation Function.

3 A Linear Distinguishing Attack on Dragon

3.1 Linear Approximation of the Function *F*

Recall, at time t the input to the function F is a vector of six words $(a, b, c, d, e, f) = (B_0, B_9, B_{16}, B_{19}, B_{30} \oplus M_L, B_{31} \oplus M_R)$. The output from the function is $(a', b', c', d',$

$e', f')$. To simplify further expressions let us introduce new variables.

$$\begin{cases} b'' &= b \oplus a = B_9 \oplus B_0 \\ c'' &= c \boxplus (a \oplus b) = B_{16} \boxplus (B_9 \oplus B_0) \\ d'' &= d \oplus c = B_{19} \oplus B_{16} \\ f'' &= f \oplus e = B_{30} \oplus B_{31} \oplus M_L \oplus M_R \end{cases} \quad (2)$$

If the words denoted by Bs are independent, then these new variables will also be independent (since B_{19} is independent of B_{16} and random, then d'' is independent and random as well; similarly, independence of B_{16} lead to the independence of c'', etc.).

The output from F can be expressed via $(a, b'', c'', d'', e, f'')$ as follows.

$$\begin{cases} a' &= (a \boxplus f'') \oplus H_1(b'' \oplus G_3(e \boxplus d''))\oplus \\ & \quad \Big((f'' \oplus G_2(c'')) \boxplus \Big(c'' \oplus H_2(d'' \oplus G_1(a \boxplus f''))\Big)\Big) \\ e' &= (e \boxplus d'') \oplus H_3(f'' \oplus G_2(c''))\oplus \\ & \quad \Big((d'' \oplus G_1(a \boxplus f'')) \boxplus \Big((a \boxplus f'') \oplus H_1(b'' \oplus G_3(e \boxplus d''))\Big)\Big) \end{cases} \quad (3)$$

Let us now analyze the expression for a'. The variable b'' appears only once (in the input of H_1), which means that this input is independent from other terms of the expression, i.e., the term $H_1(\ldots)$ can be substituted by $H_1(r_1)$, where r_1 is some independent and uniformly distributed random variable. Then, the same will happen with the input for H_2.

We would like to approximate the expression for a' as

$$a' = a \oplus N_a, \quad (4)$$

where N_a is some non uniformly distributed noise variable. If we XOR both sides with a and then substitute a' with the expression from (3), we derive

$$N_a = a \oplus (a \boxplus f'') \oplus H_1(r_1) \oplus \Big((f'' \oplus G_2(c'')) \boxplus (c'' \oplus H_2(r_2))\Big). \quad (5)$$

Despite the fact that G and H are $\mathbb{Z}_{2^{32}} \to \mathbb{Z}_{2^{32}}$ functions, they are not likely to be one-to-one mappings, consider the way the S-boxes are used as $\mathbb{Z}_{2^8} \to \mathbb{Z}_{2^{32}}$ functions [2] . It means that even if the input to a G or a H function is completely random, then the output will still be biased. Moreover, the output from the expressions $(x \oplus G_i(x)$ and similarly $x \oplus H_i(x))$ is also biased, since x in these expressions plays a role of an approximation of the G_i and the H_i functions. These observations mean that the noise variable N_a, is also biased if the input variables are independent and uniformly distributed.

[2] The cipher Turing uses similar $\mathbb{Z}_{2^{32}} \to \mathbb{Z}_{2^{32}}$ functions based on $\mathbb{Z}_{2^8} \to \mathbb{Z}_{2^{32}}$ S-boxes, which can be regarded as a source of weakness. However, no attack was found on Turing so far.

By a similar observation, the expression for e' can also be approximated as follows.

$$e' = e \oplus N_e, \tag{6}$$

where N_e is the noise variable. The expression for N_e can similarly be derived as

$$N_e = e \oplus (e \boxplus d'') \oplus H_3(r_3) \oplus \Big((d'' \oplus G_1(a'')) \boxplus (a'' \oplus H_1(r_4)\Big), \tag{7}$$

where $a'' = a \boxplus f''$ is a new random variable, which is also independent since it has f'' as its component, and f'' does not appear anywhere else in the expression (7). The two new variables r_3 and r_4 are also independent and uniformly distributed random variables by a similar reason.

3.2 Building the Distinguisher

The key observation for our distinguisher, is that one of the input words to the filter function F, at time t is partially repeated as input to F at time $t + 15$, i.e.,

$$e^{(t+15)} = a^{(t)} \oplus M_L^{(t+15)}. \tag{8}$$

Let us consider the following sum of two words from the keystream.

$$\begin{aligned} s^{(t)} &= a'^{(t)} \oplus e'^{(t+15)} = (a^{(t)} \oplus N_a^{(t)}) \oplus (a^{(t)} \oplus M_L^{(t+15)} \oplus N_e^{(t+15)}) \\ &= \underbrace{N_a^{(t)} \oplus N_e^{(t+15)}}_{N_{tot}^{(t)}} \oplus M_L^{(t+15)} \end{aligned} \tag{9}$$

By this formula we show how to sample from a given keystream, so that the samples $s^{(t)}$ are from some nonuniform distribution $P_{\texttt{Dragon}}$ of the noise variable $N_{tot}^{(t)}$ (later also referred as $P_{N_{tot}^{(t)}}$). Collected samples $s^{(t)}$ form a so-called *type* $P_{\texttt{Type}}$, or an *empirical distribution*. Then, we have two hypothesis:

$$\begin{cases} H_1 : & P_{\texttt{Type}} \text{ is drawn according to } P_{\texttt{Dragon}} \\ H_2 : & P_{\texttt{Type}} \text{ is drawn according to } P_{\texttt{Random}} \end{cases}. \tag{10}$$

To distinguish between them with negligible probability of error (whether the samples are drawn from the noise distribution $P_{\texttt{Dragon}}$ or from the uniform distribution $P_{\texttt{Random}}$), the type should be constructed from the following number of samples

$$N \approx 1/\epsilon^2, \tag{11}$$

where ϵ is the bias, calculated as

$$\epsilon = |P_{\texttt{Dragon}} - P_{\texttt{Random}}| = \sum_{x=0}^{2^{32}-1} |P_{\texttt{Dragon}}(x) - P_{\texttt{Random}}(x)|. \tag{12}$$

When the type $P_{\texttt{Type}}$ is constructed, a common tool in statistical analysis is the log-likelihood test. The ratio I is calculated as

$$I = D(P_{\texttt{Type}}||P_{\texttt{Random}}) - D(P_{\texttt{Type}}||P_{\texttt{Dragon}}) \\ = \sum_{x=0}^{2^{32}-1} P_{\texttt{Type}}(x) \log_2 \frac{P_{\texttt{Dragon}}(x)}{P_{\texttt{Random}}(x)} \quad , \tag{13}$$

where $D(\cdot)$ is the *relative entropy* defined for any two distributions P_1 and P_2 as

$$D(P_1||P_2) = \sum_{x \in \Omega} P_1(x) \log_2 \frac{P_1(x)}{P_2(x)}, \tag{14}$$

where Ω is the probability space.

Finally, the decision rule $\delta(P_{\texttt{Type}})$ is the following

$$\delta(P_{\texttt{Type}}) = \begin{cases} H_1, & \text{if } I \geq 0 \\ H_2, & \text{if } I < 0 \end{cases}. \tag{15}$$

For more on statistical analysis and hypothesis testing we refer to, e.g., [17, 18].

The remaining question is how to deal with the counter value M_L. Below we present a set of possible solutions that one could consider.

(1) One possible solution would be to guess the initial state of the counter $M^{(0)}$ (in total 2^{64} combinations), and then construct 2^{64} types from the given keystream, assuming the value $M_L^{(t)}$ in correspondence to the guessed initial value of $M^{(0)}$. However, it will increase the time complexity of the distinguisher by 2^{64} times;

(2) One more possibility is to guess the first 32 bits $M_R^{(0)}$ of the initial value of the counter $M^{(0)}$, i.e., 2^{32} values. If we do so, then we always know when the upper 32 bits M_L are increased, i.e., at any time t we can express $M_L^{(t)}$ as follows.

$$M_L^{(t)} = M_L^{(0)} \boxplus \Delta^{(t)}, \tag{16}$$

where $\Delta^{(t)}$ is known at each time, since $M_R^{(t)}$ is known. Recall from (9), the noise variable $N_{tot}^{(t)}$ is expressed as $s^{(t)} \oplus M_L^{(t+15)}$. However, this expression can also be approximated as

$$s^{(t)} \oplus (M_L^{(0)} \boxplus \Delta^{(t+15)}) \rightarrow s^{(t)} \oplus (M_L^{(0)} \oplus \Delta^{(t+15)}) \oplus N_2, \tag{17}$$

where N_2 is a new noise variable due to the approximation of the kind "$\boxplus \Rightarrow \oplus$". Since $M_L^{(0)}$ can be regarded as a constant for every sample $s^{(t)}$, it only "shifts" the distribution, but will not change the bias. Consider that a shift of the uniform distribution is again the uniform distribution, so, the distance between the noise and the uniform distributions will remain the same. This solution requires $O(2^{32})$ guesses, and also introduce a new noise variable N_2;

(3) Another possible solution could be to consider the sum of two consecutive samples $s^{(t)} \oplus s^{(t+1)}$. Since M_L changes slowly, then with probability $(1 - 2^{-32})$ we have $M_L^{(t)} = M_L^{(t+1)}$, and this term will be eliminated from the expression for that new sample. Unfortunately, this method will decrease the bias significantly, and then the number of required samples N will be much larger than in the previous cases.

In our attack we tried different solutions, and based on simulations we decided to choose solution (2) for our attack, as it has the lowest attack complexity.

3.3 Calculation of the Noise Distribution

Consider the expression for the noise variable $s^{(t)} \oplus M_L^{(t+15)} = N_a^{(t)} \oplus N_e^{(t+15)}$. For simplicity in the formula, we omit time instances for variables.

$$\begin{aligned} N_{tot}^{(t)} = N_a^{(t)} \oplus N_e^{(t+15)} = (a \boxplus f'') \oplus (a \boxplus d'') \oplus H_1(r_1) \oplus H_3(r_3) \oplus \\ \oplus \Big(\big(f'' \oplus G_2(c'')\big) \boxplus \big(c'' \oplus H_2(r_2)\big) \Big) \oplus \Big(\big(d'' \oplus G_1(a'')\big) \boxplus \big(a'' \oplus H_1(r_4)\big) \Big) \end{aligned} \tag{18}$$

We propose two ways to calculate the distribution of the total noise random variable $N_{tot}^{(t)}$. Lets truncate the word size by n bits (when we consider the expression modulo 2^n), then in the first case the computational complexity is $O(2^{4n})$. This complexity is too high and, therefore, requires the noise variable to be truncated by some number of bits $n \ll 32$, much less than 32 bits. The second solution has a better complexity $O(n2^n)$, but introduces two additional approximations into the expression, which makes the calculated bias smaller than the real value, i.e., by this solution we can verify the lower bound for the bias of the noise variable. Below we describe two methods and give our simulation results on the bias of the noise variable $N_{tot}^{(t)}$.

(I) Consider the expression (18) taken by modulo 2^n, for some $n = 1 \ldots 32$. Then the distribution of the noise variable can be calculated by the following steps.
   a) Construct three distributions, two of them are conditioned

   $$P_{(G_2(c'') \mod 2^n | c'')}, \quad P_{(G_1(a'') \mod 2^n | a'')}, \quad P_{(H_1(x) \mod 2^n)}{}^{3}.$$

   The algorithm requires one loop for c'' (a'' and x) of size 2^{32}. The time required is $O(3 \cdot 2^{32})$;
   b) Afterwards, construct two more conditioned distributions

   $$P_{(d'' \oplus G_1(a''))\boxplus(a'' \oplus H_1(r_4)) \mod 2^n | d'')}$$

   and

   $$P_{(f'' \oplus G_1(c''))\boxplus(c'' \oplus H_2(r_2)) \mod 2^n | f'')}.$$

[3] If the inputs to the H_i functions is random, their distributions are the same, i.e., $P_{H_1} = P_{H_2} = P_{H_3}$.

This requires four loops for $d'', a'', x(= G_1(a'') \mod 2^n)$, and $y(= H_1(r_4) \mod 2^n)$, which takes time $O(2^{4n})$ (and similar for the second distribution);

c) Then, calculate another two conditioned distributions

$$P_{(\texttt{Expr}_1|a)} = P_{((a\boxplus f'')\oplus(f''\oplus G_1(c''))\boxplus(c''\oplus H_2(r_2)) \mod 2^n|a)},$$

$$P_{(\texttt{Expr}_2|a)} = P_{((a\boxplus d'')\oplus(d''\oplus G_1(a''))\boxplus(a''\oplus H_1(r_4)) \mod 2^n|a)}.$$

Each takes time $O(2^{3n})$;

d) Finally, combine the results, partially using FHT, and then calculate the bias of the noise:

$$P_{N_{tot}} = P_{(\texttt{Expr}_1|a)} \oplus P_{(\texttt{Expr}_2|a)} \oplus P_{H_1} \oplus P_{H_3}.$$

This will take time $O(2^{3n} + 3n \cdot 2^n)$.

This algorithm calculates the exact distribution of the noise variable taken modulo 2^n, and has the complexity $O(2^{4n})$. Due to such a high computational complexity we could only manage to calculate the bias of the noise when $n = 8$ and $n = 10$:

$$\begin{aligned} \epsilon_I\big|_{n=8} &= 2^{-80.59} \\ \epsilon_I\big|_{n=10} &= 2^{-80.57}. \end{aligned} \tag{19}$$

(II) Consider two additional approximations of the second $\boxplus$ to $\oplus$ in (18). Then, the total noise can be expressed as

$$\begin{aligned} N_{tot}^{(t)} =& H_1(r_1) \oplus H_2(r_2) \oplus H_3(r_3) \oplus H_1(r_4) \oplus \big(G_2(c'') \oplus c''\big) \\ &\oplus \big(G_1(a'') \oplus a''\big) \oplus N_3 \oplus N_{2,a} \oplus N_{2,e}, \end{aligned} \tag{20}$$

where

$$N_3 = (a \boxplus f'') \oplus (a \boxplus d'') \oplus f'' \oplus d'',$$

and $N_{2,a}$ and $N_{2,e}$ are two new noise variables due to the approximation $\boxplus \Rightarrow \oplus$, i.e., $N_{2,a} = (x \boxplus y) \oplus (x \oplus y)$, for some random inputs x and y, and similar for $N_{2,e}$. Introduction of two new noise variables will statistically make the bias of the total noise variable smaller, but it can give us a lower bound of the bias, and also allow us to operate with distributions of size 2^{32}.

First, calculate the distributions $P_{(H_i)}$, $P_{(G_1(a'')\oplus a'')}$ and $P_{(G_1(c'')\oplus c'')}$, each taking time $O(2^{32})$. Afterward, note that the expressions for $N_{2,a}$, $N_{2,e}$ and N_3 belong to the class of *pseudo-linear functions modulo* 2^n (PLFM), which were introduced in [19]. In the same paper, algorithms for construction of their distributions were also provided, which take time around $O(\delta \cdot 2^n)$, for some small δ. The last step is to perform the convolution of precomputed distribution tables via FHT in time $O(n2^n)$. Algorithms (PLFM distribution construction and computation of convolutions) and data structures for operating on large distributions are given in [19]. If we consider $n = 32$, then the total time complexity to calculate the distribution table for N_{tot} will be around $O(2^{38})$ operations, which is feasible for a common PC. It took us a few days to accomplish such calculations on a usual PC with memory 2Gb and 2×200Gb of HDD, and the received bias of N_{tot} was

$$\epsilon_{II}|_{n=32} = 2^{-74.515}. \tag{21}$$

If we also approximate $(M_L^{(0)} \boxplus \Delta^{(t)}) \to (M_L^{(0)} \oplus \Delta^{(t)}) \oplus N_2$, and add the noise N_2 to N_{tot}, we receive the bias

$$\epsilon_{II}^{\Delta}|_{n=32} = 2^{-77.5}, \tag{22}$$

which is the lower bound meaning that our distinguisher requires approximately $O(2^{155})$ words of the keystream, according to (12).

4 Attack Scenarios

In the previous section we have shown how to sample from the given keystream, where 32 bit samples are drawn from the noise distribution with the bias $\epsilon_{II}^{\Delta}|_{n=32} = 2^{-77.5}$. I.e., our distinguisher needs around $O(2^{155})$ words of the keystream to successfully distinguish the cipher from random. Unfortunately, to construct the type correctly we have to guess the initial value of the linear part of the cipher, the lower 32 bits $M_R^{(0)}$ of the counter M. This guess increases the time complexity of our attack to $O(2^{187})$, and requires memory $O(2^{32})$. The algorithm of our distinguisher for Dragon is given in Table 1.

```
for 0 ≤ M_R^(0) < 2^32
    P_Type(x) = 0,   ∀x ∈ Z_{2^32}
    Δ = 0  (or = −1, if M_R^(0) = 0)
    for t = 0, 1, ..., 2^155
        if  (M_R^(0) ⊞ t) = 0  then  Δ = Δ ⊞ 1
        s^(t) = a'^(t) ⊕ e'^(t+15) ⊕ Δ
        P_Type(s^(t)) = P_Type(s^(t)) + 1
    I = Σ_{x∈Z_{2^32}} P_Type(x) · log_2(P_Dragon(x)/2^−32)
    If I ≥ 0 break and output : Dragon
output : Random source
```

Table 1. The distinguisher for Dragon (Scenario I).

We, however, can also show that time complexity can easily be reduced downto $O(2^{155})$, if memory of size $O(2^{96})$ is available. Assume we first construct a special table $T[\Delta][s] = \#\{t \equiv \Delta \mod 2^{64}, s^{(t)} = s\}$, where the samples are taken as $s^{(t)} = a'^{(t)} \oplus e'^{(t+15)}$. Afterwards, for each guess of $M_L^{(0)}$ the type $P_{\mathtt{Type}}(\cdot)$ is then constructed from the table T in time $O(2^{96})$. Hence, the total time complexity will be $O(2^{155} + 2^{32} \cdot 2^{96}) \approx O(2^{155})$. This scenario is given in Table 2.

for $0 \leq t < 2^{155}$
$\quad T[t \mod 2^{64}][a'^{(t)} \oplus e'^{(t+15)}] + +$
for $M_R^{(0)} = 0, \ldots, 2^{32} - 1$
$\quad$ **for** $\Delta = 0, \ldots, 2^{64} - 1$
$\quad\quad$ **for** $x = 0, \ldots, 2^{32} - 1$
$\quad\quad\quad P_{\mathtt{Type}}\Big(x \oplus \big((\Delta \boxplus M_R^{(0)}) \gg 32\big)\Big) + = T[\Delta][x]$
$\quad I = \sum_{x \in \mathbb{Z}_{2^{32}}} P_{\mathtt{Type}}(x) \cdot \log_2(P_{\mathtt{Dragon}}(x)/2^{-32})$
$\quad$ **If** $I \geq 0$ **break** and **output** : Dragon
output : Random source

Table 2. Distinguisher for Dragon with lower time complexity (Scenario II).

5 Results and Conclusions

Two versions of a distinguishing attack on Dragon were found. The first scenarios requires a computational complexity of $O(2^{187})$ and needs memory only $O(2^{32})$. However, the second scenario has a lower time complexity around $O(2^{155})$, but requires a larger amount of memory $O(2^{96})$. These attacks show that Dragon does not provide full security and can successfully be broken much faster than the exhaustive search, when a key of 256 bits is used.

From the specification of Dragon we also note that the amount of the keystream for an unique pair of the IV and the key is limited to 2^{64}. However, our attack works when the same key and IV are used to produce 2^{155} words of the keystream. This is an academic attack which shows a statistical weakness of the keystream sequence, and reveals the leakage of the design.

Actually, our distinguisher consists of 2^{32} subdistinguishers. If one of them says "this is Dragon", then it is taken as the result of the final distinguisher. If all subdistinguishers output "Random source", then the overall result is "Random" as well [4].

Below we give a few suggestions how to prevent Dragon from this kind of attack:

[4] The idea to use many subdistinguishers was first proposed in the attack on Scream [20].

1) The linear part M changes predictably, when the initial state is known. It might be more difficult to mount the attack if the update of M would depend on some state of the NLFSR;
2) Another leakage is that two words $a'||e'$ are accessible to the attacker. If we would have an access only to a', or, may be, some other combination of the output from F (like, the output $a'||d'$, instead), then it might also protect the cipher from this attack. However, both these suggestions have weaknesses for different reasons;
3) One more weakness are poor G_i and H_i S-boxes. May be they can be constructed in a different way, closer to some one-to-one mapping function.

Several new stream cipher proposals are based on NLFSRs and this topic has been poorly investigated so far. We believe that it is important to study such primitives, since it could be an interesting replacement for widely used LFSR based stream ciphers.

References

1. B. Zoltak. VMPC one-way function and stream cipher. In B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 210–225. Springer-Verlag, 2004.
2. S. Paul and B. Preneel. A new weakness in the rc4 keystream generator and an approach to improve the security of the cipher. In B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, 2004.
3. P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm. *Journal of Cryptology*, 11(4):273–287, 1998.
4. P. Rogaway and D. Coppersmith. A software-optimised encryption algorithm. In R.J. Anderson, editor, *Fast Software Encryption'93*, volume 809 of *Lecture Notes in Computer Science*, pages 56–63. Springer-Verlag, 1994.
5. P. Hawkes and G.G. Rose. Primitive specification and supporting documentation for SOBER-t16 submission to NESSIE. In *Proceedings of First Open NESSIE Workshop*, 2000. Available at *http://www.cryptonessie.org*, Accessed October 5, 2003.
6. P. Ekdahl and T. Johansson. SNOW - a new stream cipher. In *Proceedings of First Open NESSIE Workshop*, 2000.
7. P. Ekdahl and T. Johansson. A new version of the stream cipher SNOW. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography—SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 2002.
8. J. Daemen and C. Clapp. Fast hashing and stream encryption with PANAMA. In *Fast Software Encryption'98*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer-Verlag, 1998.
9. S. Halevi, D. Coppersmith, and C.S. Jutla. Scream: A software-efficient stream cipher. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, 2002.

10. D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel. A new keystream generator MUGI. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 2002.
11. N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno. Helix fast encryption and authentication in a single cryptographic primitive. In *Fast Software Encryption 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 330–346. Springer-Verlag, 2003.
12. M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansed, and O. Scavenius. Rabbit: A new high-performance stream cipher. In *Fast Software Encryption 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 307–329. Springer-Verlag, 2003.
13. G.G. Rose and P. Hawkes. Turing: A fast stream cipher. In T. Johansson, editor, *Fast Software Encryption 2003*, To be published in Lecture Notes in Computer Science. Springer-Verlag, 2003.
14. NESSIE. New European Schemes for Signatures, Integrity, and Encryption. Available at *http://www.cryptonessie.org*, Accessed August 18, 2003, 1999.
15. SKEW. Symmetric key encryption workshop. Available at *http://www2.mat.dtu.dk/people/Lars.R.Knudsen/stvl/*, Accessed August 6, 2005, 2005.
16. K. Chen, M. Henricksen, W. Millan, J. Fuller, L. Simpson, E. Dawson, H. Lee, and S. Moon. Dragon: A fast word based stream cipher. *ECRYPT Stream Cipher Project Report 2005/006*.
17. D. Coppersmith, S. Halevi, and C.S. Jutla. Cryptanalysis of stream ciphers with linear masking. In M. Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 515–532. Springer-Verlag, 2002.
18. J. Golić. Intrinsic statistical weakness of keystream generators. pages 91–103, 1994.
19. A. Maximov and T. Johansson. Fast computation of large distributions and its cryptographic applications. To appear at Asiacrypt 2005.
20. T. Johansson and A. Maximov. A Linear Distinguishing Attack on Scream. In *Information Symposium in Information Theory—ISIT 2003*, page 164. IEEE, 2003.

[0] The work described in this paper has been supported in part by Grant VR 621-2001-2149, and in part by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.