2012 Seventh International Conference on Availability, Reliability and Security

On the Weak State in GGHN-like Ciphers

Aleksandar Kircanski
Dept. of Computer Science and Software Engineering
Concordia University
Montreal, Qc, Canada
a_kircan@cs.concordia.ca

Amr M. Youssef
Concordia Institute for Information Systems Engineering
Concordia University
Montreal, Qc, Canada
youssef@ciise.concordia.ca

***Abstract*—RC4 is a stream cipher that makes use of an internal state table, S, which represents a permutation over Z_{2^8}. GGHN is a relatively more efficient stream cipher whose design is inspired from RC4 but whose S table, however, does not represent a permutation over Z_{2^m}. In this paper, we point out one challenging aspect of the latter design principle. In particular, we assess GGHN-like algorithms with respect to weak states, in which all internal state words and output elements are even. Once GGHN is absorbed in a weak state, the least significant bit of the plaintext words will be revealed only by looking at the ciphertext. By modelling the algorithm by a Markov chain and calculating chain's absorption time, we show that the average number of steps required by these algorithms to enter this weak state can be lower than expected at first glance and hence caution should be exercised when estimating this number.**

Key words: Cryptography, stream ciphers, RC4, GGHN-like ciphers, Markov chains

I. Introduction

A variety of stream ciphers based on Linear Feedback Shift Registers (LFSRs) were proposed in the last two decades [12]. Although fast when implemented in hardware, these ciphers do not have good performance when implemented in software. One of the first stream ciphers not based on LFSRs was RC4, designed in 1987 by Ron Rivest. RC4 was kept as a trade secret until it was leaked in 1994 by an anonymous Internet post. Today, due to its simplicity and speed, RC4 is probably the most commonly used stream cipher. It is implemented in many protocols and applications such as Secure Socket Layer (SSL), and Wired Equivalent Privacy (WEP).

RC4 produces 8-bit keystream words, by slowly changing an internal state which consists of a permutation of 8-bit values and two counters. The ciphertext is obtained by XOR-ing keystream bytes to the plaintext. From the perspective of 32 and 64 bit processors, which are becoming more common today, this is inefficient. RC4-like cipher that would produce 32-bit or 64-bit keystream words would be around 4-8 times faster, since an 8 bit operation on these processors takes equal time as a 32, or 64 bit operation. To address this problem, Gong *et al.* [14] proposed a generalized version of RC4, namely RC4(n,m), where m denotes the bit-length of the keystream output word. Later, after an attack was discovered by Wu [20], another version of this cipher [5] was introduced. Established names for these ciphers are NGG(n,m) for the first version of RC4(n,m) and GGHN(n,m) for the second version.

The internal state of RC4 includes an S table of 256 8-bit values. The important property of this table is that, at each step, it represents a permutation, i.e., all the possible combinations of the m bit values appear in the table. In NGG(n,m) and GGHN(n,m), the size of the table that would be needed to store the whole permutation is 2^m where m is the keystream word size, which is impractical for $m > 8$. Accordingly, smaller table of size N had to be used in both versions of the ciphers where $N = 2^n << 2^m$ represents the table size and m is the keystream word size in bits. Since a table can now hold only a subset of the possible m bit values, to avoid distinguishing attack which would test whether keystream words belong to only a subset of all possible m-bit values, internal state update function had to not only perform swap operation as in the original RC4 but to introduce new values to the S table. In NGG, this was done by replacing the S table element chosen as keystream word by a sum of two other elements from the table at each step. However, this was proven to be susceptible to an attack [20] and GGHN(n,m) was proposed. In GGHN(n,m) another counter, k, was introduced and attacks similar to [20] are supposedly eliminated.

A. Our contribution

By weak state of GGHN, we denote a state in which k and all the S table entries are even. Since the update and output functions are defined as additions of these elements, once the cipher reaches this weak state, it will remain in it forever and the output keystream words will always be even. Therefore, once GGHN is in a weak state, the least significant bit of the plaintext words will be revealed only by looking at the ciphertext.

In [5], it was noted that the probability that all state entries as well as k become even is very low, $2^{-(N+1)}$, which implies that the average number of steps before GGHN(n,m) enters weak state is $O(2^N)$. In this paper we revisit this claim and provide evidence that this number of steps is much less than 2^N.

978-0-7695-4775-6/12 $26.00 © 2012 IEEE
DOI 10.1109/ARES.2012.32

Based on our analysis, it follows that with GGHN-like ciphers that are based on non bijective S tables, caution should be made when estimating the time needed for approaching weak state.

The rest of the paper is organized as follows. In Section 2, we review the specifications of RC4, NGG and GGHN ciphers. Section 3 gives an overview of some attacks on these ciphers. In Section 4, we provide a brief theoretical background for Markov chains. In Section 5, an idealized model of GGHN-like ciphers is presented and then modelled by a Markov chain. Section 6 presents a way to calculate the absorption times for the defined chain. The conclusion is given in Section 7.

II. RC4 AND GGHN SPECIFICATION

In all three ciphers, the size of the key is l bytes. Usually, $l = 8$ or $l = 16$. The RC4 KSA procedure initializes the S array to an identity permutation and then swaps each byte with a pseudorandom element of the table. In RC4 PRGA, two counters i and j are used. Counter i is publicly known and j is incremented in a pseudorandom way. At each step, element $S[i]$ and $S[j]$ are swapped. Element $S[S[i] + S[j]]$ is chosen as the output keystream byte. A pseudo code for RC4 is shown in Fig. I.

KSA	**PRGA**
Initialization:	*Initialization:*
For $i = 0, ..255$	$i = j = 0$
$S[i] = i$	*Loop:*
$j = 0$	$i = i + 1$
Scrambling:	$j = j + S[i]$
For $i = 0, ..255$	Swap$(S[i], S[j])$
$j = j + S[i] + K[i \bmod l]$	$t = S[i] + S[j]$
Swap$(S[i], S[j])$	Output $z = S[t]$

Figure I
RC4 SPECIFICATION

NGG(n,m) is a family of ciphers defined by two parameters, n and m. The size of the S table is equal to $N = 2^n$, and the word size the cipher operates with is m. For example, for NGG(8,32), the internal state table S consists of 256 32-bit elements and produces keystream words that are 32 bit long. The NGG KSA procedure initializes the table to a predefined set of random values (one particular set of these values is proposed in [14]). The scrambling part of the KSA iterates operations of swap and addition on S table entries. As for the NGG PRGA, it is similar to RC4 PRGA, with an additional update step which changes the values in the table.

KSA	**PRGA**
Initialization:	*Initialization:*
For $i = 0, ..N - 1$	$i = 0$
$S[i] = a_i$	$j = 0$
$j = k = 0$	*Loop:*
Scrambling:	$i = (i + 1) \bmod N$
Repeat r times	$j = (j + S[i]) \bmod N$
For $i = 0, ..N - 1$	$k = (k + S[j]) \bmod M$
$j = (j + S[i] + K[i \bmod l]) \bmod N$	$t = (S[i] + S[j]) \bmod N$
Swap$(S[i], S[j])$	out=$(S[t] + k) \bmod M$
$S[i] = (S[i] + S[j]) \bmod M$	$S[t] = (k + S[i]) \bmod M$
$k = (k + S[i]) \bmod M$	

Figure II
GGHN(n,m) SPECIFICATION

In GGHN(n,m), another pseudorandom counter, k, is added to the algorithm. Again, the S table size is $N = 2^n$ and the internal and keystream word size is m bits. The GGHN KSA is similar to the NGG KSA with the only difference is that it is repeated r times (it is recommended to set $r = 20$ for $n = 8$). As for k, after it is initialized to 0 at the beginning of KSA, the S table entries are added to it iteratively. In comparison to the NGG PRGA, the GGHN PRGA makes use of a pseudorandom counter, k, both in the update and the output steps. Instead of outputting plain pseudorandom entry of the S table as in NGG, value masked by k will constitute the keystream word. Similarly, new value that will be written to the array depends on k. The pseudo code of the algorithm is shown in Fig. II.

III. EXISTING ATTACKS

In this section, we provide a brief summary of some of these results as well as existing attacks on NGG and GGHN.

The security of RC4 has been a subject of extensive research effort. For example, the key schedule of RC4 was examined in [3], [8], [9], [13], [15], [19], some distinguishing attacks were presented in [2], [4], [9], [10], [16] and internal state recovery attacks were presented in [7], [11].

In [20] it was shown that NGG is distinguishable from a random sequence with only about 100 keystream words. The attack relies on the fact that for three random S table entries, relation $S[X] = S[Y] + S[Z]$ holds with biased probability, due to the update step of NGG.

In [16], it was found that the least significant bit (LSB) of NGG keystream word is biased, due to bias inducing state, which occurs with probability 2^{-16} and implies that the LSB of the output word is 0. Distinguisher based on this can be built using $2^{32.89}$ keystream words. This paper discusses NGG in a broader context of ciphers that use arrays and additions and shows that this family of ciphers is prone to attacks due to bias inducing states.

In Klein's work [6], in which RC4 used in WEP mode is shown to be weak, NGG is also shown to be prone to a similar attack which makes it insecure when IVs are

concatenated to keys.

Just as with NGG, the least significant bit of the keystream output is biased, due to a bias inducing state which occurs with probability 2^{-16} and induces that the LSB of keystream words is 0 with certainty [16]. The distinguisher can be built upon $2^{32.89}$ keystream words.

Tsunoo *et al.* [18] presented a distinguisher based on the first two words of keystream associated with approximately 2^{30} keys.

IV. Markov Chains background

In this section we briefly review the Markov chains, absorption states and a way to calculate the average absorption times from the chain transition matrix [1].

A Markov chain can be described as follows. Let $S = \{s_1, s_2, \dots s_n\}$ denote the set of *states*. The chain starts in some state, and in each step, moves from one state to another. The probability that the chain will move to certain state s_j depends only on its previous state s_i, and not on any of the states before that. Therefore, we can write p_{ij} for transition probabilities, denoting the probability that a process will go from s_i to s_j. The matrix $P = (p_{ij})_{r\times r}$ is called the transition matrix. The beginning state of the chain is usually described by a probability distribution over the set S. The following lemma is a rudimentary result on Markov chains.

Lemma 1: Let P be a transition matrix of a Markov chain with set of states $S = \{s_1, s_2, \dots s_r\}$. The p_{ij} entry of matrix P^n then represents the probability that the chain, starting in state s_i, will be in s_j after n steps.

When a pseudorandom number generator that we are modelling by a Markov chain enters the weak state, it cannot go out of it anymore. This behavior corresponds to absorbing state notion, known in Markov chain theory.

Definition 1: A state s_i of a Markov chain is called *absorbing state* if $p_{ii} = 1$, i.e., if once entered, it is impossible for the chain to leave it. A Markov chain is called *absorbing* if it has at least one absorbing state and if, from every state, it is possible to go to absorbing state (not necessarily in one step). Non-absorbing states of an absorbing chain are called *transient* states. When a chain enters absorption state, we say that it got *absorbed*.

Lemma 2: In an absorbing Markov chain, the probability that the process will be absorbed is 1.

To measure the number of steps needed for absorption, it is convenient to order states from set S so that absorbing states come at the end. Suppose that there are t transient states and r absorbing states. Then, the transition matrix will be of the form:

$$\begin{bmatrix} \mathbf{Q} & \mathbf{R} \\ \mathbf{0} & \mathbf{Id} \end{bmatrix}$$

where **Id** is the identity matrix and **0** is zero matrix. The dimensions of **Q**, **R**, **0** and **Id** are $t \times t$, $t \times r$, $r \times t$ and $r \times r$, respectively. The matrix **I-Q** is called the *fundamental matrix* and its inverse reveals information with respect to absorption time. In particular, we use the following result.

Lemma 3: For an absorbing Markov chain, the matrix **I-Q** is invertible. The ij-entry of $(\mathbf{I\text{-}Q})^{-1}$ is the expected number of times the chain is in state s_j, given that it started from s_i. The initial state is counted if $i = j$.

That way, the inverse of fundamental matrix gives information on how many times will a process pass through each state before absorption. We can now calculate the expected number of steps before absorption. Namely, if we sum all the values of i-th row of matrix $(\mathbf{I\text{-}Q})^{-1}$, we get the expected number of steps before absorption, given that a process starts from state s_i.

V. Modeling GGHN(n,m) by a chain

As specified in GGHN PRGA procedure, counter i iterates from 0 to $N-1$ and j and k are incremented in a pseudorandom way. At each step, k is updated as $k = (k + S[j]) \bmod M$, $(S[(S[i] + S[j]) \bmod N] + k) \bmod M$ element is sent to the output and the S table is updated by $S[(S[i] + S[j]) \bmod N] = (k + S[i]) \bmod M$.

To model GGHN-like ciphers, we idealize it as follows. For the update step of k, we take $k = (k + S[X]) \bmod M$ where X is random variable taking values from 0 to $N - 1$, thus approximating pseudorandom number j by a uniformly distributed random value X. As for the output, we model it as $output = (S[Y] + k) \bmod M$, where Y is a uniform random variable independent from X and takes values from 0 to $N-1$ and substituting pseudorandom value $(S[i] + S[j]) \bmod N$. Finally, for the update step, we take $S[Y] = (k+S[Z]) \bmod M$. Here we use already defined Y, and introduce Z, uniform random value independent of X and Y, as substitution for counter i. The PRGA algorithm of this idealized GGHN(n,m) is then modelled as follows:

$$\begin{array}{lcl} k & = & k + S[X] \bmod M \\ output & = & k + S[Y] \bmod M \\ S[Y] & = & k + S[Z] \bmod M \end{array}$$

Note that we do not initialize S by any KSA, but randomly. An eventual bias towards even numbers in beginning table may amplify the results given in the paper.

We say GGHN(n,m) is in weak state if all elements of S as well as value k are even.

A. Defining the chain and calculating state transition probabilities

We map each possible internal state of the cipher to a state from the chain's state set. Let $Z(S)$ denote the number of odd numbers in the S table. We say GGHN(n,m) is in state (z, p) if $Z(S) = z$ and $p = k \bmod 2$ for current S and k. For example, if GGHN(8,32) is in state (130,0), that means

that exactly 130 values in S are odd, and k is even. Since $Z(S) \in \{0..N\}, p \in \{0,1\}$, we have that for GGHN($n$,$m$) the number of states is equal to $2 \times (N+1)$. It follows that GGHN is in weak state if and only if it is in $(0,0)$ state.

Due to the independence between X, Y and Z and other parts of internal state, this chain satisfies the Markovian property. In other words, if current state is (z_1, p_1), probability distribution of next state (z_2, p_2) does not depend on previous state (z_0, p_0) or states before that. In the following theorem, we calculate state transition probabilities.

Theorem 1: Let state transition probabilities not mentioned in the table amount to zero. In rows 1-4 of the table, let $z = 1 \dots N$. In rows 5-8 and 9-12 let $z = 0 \dots N$ and $z = 0 \dots N-1$, respectively. Then, the table specifies state transition probabilities of the defined chain.

State transition	Probability
$P[(z,0) \to (z-1,0)]$	$(\frac{N-z}{N})^2 \times \frac{z}{N}$
$P[(z,0) \to (z-1,1)]$	$(\frac{z}{N})^3$
$P[(z,1) \to (z-1,0)]$	$(\frac{z}{N})^2 \times \frac{N-z}{N}$
$P[(z,1) \to (z-1,1)]$	$\frac{N-z}{N} \times (\frac{z}{N})^2$
$P[(z,0) \to (z,0)]$	$\frac{N-z}{N} \times ((\frac{N-z}{N})^2 + (\frac{z}{N})^2)$
$P[(z,0) \to (z,1)]$	$2 \times (\frac{z}{N})^2 \times \frac{N-z}{N}$
$P[(z,1) \to (z,0)]$	$\frac{z}{N} \times ((\frac{N-z}{N})^2 + (\frac{z}{N})^2)$
$P[(z,1) \to (z,1)]$	$2 \times (\frac{N-z}{N})^2 \times \frac{z}{N}$
$P[(z,0) \to (z+1,0)]$	$(\frac{N-z}{N})^2 \times \frac{z}{N}$
$P[(z,0) \to (z+1,1)]$	$\frac{z}{N} \times (\frac{N-z}{N})^2$
$P[(z,1) \to (z+1,0)]$	$\frac{N-z}{N} \times (\frac{z}{N})^2$
$P[(z,1) \to (z+1,1)]$	$(\frac{N-z}{N})^3$

Proof: Due to the fact that only one S table element is updated at each PRGA iteration, the number of even values in the table can either decrease by one, stay the same or increase by one. Thus, all probabilities not mentioned in the table are equal to 0.

Let $P[even(\cdot)]$ and $P[odd(\cdot)]$ denote the probability that the enclosed argument is even and odd, respectively. We prove rows 1-4 of the table. Let $z = 1..N$. Then,

$$P[(z,0) \to (z-1,0)] = P[even(S[X])] \times P[odd(S[Y])] \times P[even(S[Z])] = \frac{N-z}{N} \times \frac{z}{N} \times \frac{N-z}{N}.$$

$$P[(z,0) \to (z-1,1)] = P[odd(S[X])] \times P[odd(S[Y])] \times P[odd(S[Z])] = \frac{z}{N} \times \frac{z}{N} \times \frac{z}{N}.$$

$$P[(z,1) \to (z-1,0)] = P[odd(S[X])] \times P[odd(S[Y])] \times P[even(S[Z])] = \frac{z}{N} \times \frac{z}{N} \times \frac{N-z}{N}.$$

$$P[(z,1) \to (z-1,1)] = P[even(S[X])] \times P[odd(S[Y])] \times P[odd(S[Z])] = \frac{N-z}{N} \times \frac{z}{N} \times \frac{z}{N}.$$

Now let $z = 0 \dots N$. Then, rows 5-8 can be proven as follows.

$$P[(z,0) \to (z,0)] = P[even(S[X])] \times (P[even(S[Y])] \times P[even(S[Z]] + P[odd(S[Y])] \times P[odd(S[Z])]) = \frac{N-z}{N} \times (\frac{N-z}{N} \times \frac{N-z}{N} + \frac{z}{N} \times \frac{z}{N}).$$

$$P[(z,0) \to (z,1)] = P[odd(S[X])] \times (P[even(S[Y])] \times P[odd(S[Z])] + P[odd(S[Y])] \times P[even(S[Z])]) = \frac{z}{N} \times (\frac{N-z}{N} \times \frac{z}{N} + \frac{z}{N} \times \frac{N-z}{N}).$$

$$P[(z,1) \to (z,0)] = P[odd(S[X])] \times (P[even(S[Y])] \times P[even(S[Z])] + P[odd(S[Y])] \times P[odd(S[Z])]) = \frac{z}{N} \times (\frac{N-z}{N} \times \frac{N-z}{N} + \frac{z}{N} \times \frac{z}{N}).$$

$$P[(z,1) \to (z,1)] = P[even(S[X])] \times (P[even(S[Y])] \times P[odd(S[Z])] + P[odd(S[Y])] \times P[even(S[Z])]) = \frac{N-z}{N} \times (\frac{N-z}{N} \times \frac{z}{N} + \frac{z}{N} \times \frac{N-z}{N}).$$

Proof of rows 9-12 is analogous to the proof of rows 1-4. □

VI. Absorption times

To calculate the absorption time for the above defined chain, we follow guidelines from section IV. First, we encode states (z,p) ($z = 0 \dots N$, $p = 0,1$) by numbers from 0 to $2N+1$ using one-to-one function $f(z,p) = 2N - 2z + 1 - p$. Note that $f((0,0)) = 2N+1$, so in this ordering, weak state of GGHN, in which there is 0 odd numbers in the S table and k is even, comes last, as suggested in section IV. The transition matrix **P** indexed from 0 to $2N+1$ in both dimensions will contain value $P[(z_0,p_0) \to (z_1,p_1)]$ (given by Theorem 1) at index (i,j), where $i = f(z_0,p_0)$ and $j = f(z_1,p_1)$. Then, discarding last row and last column gives matrix **Q** and **B=I-Q** represents the fundamental matrix. Finally we calculate $\mathbf{B}^{-1}$ and then, the sum of i-th row of this matrix represents average number of steps GGHN will take before absorption if the algorithm started from $f^{-1}(i)$ state. We approximate the beginning state distribution by assuming that half of the numbers in S table are odd and value k is even and thus we take the

sum of $f((N/2,0))$ row as an estimate for the expected absorption time of the defined chain. Using the above model, the absorbtion time for $n = 6$ is $2^{39.1} << 2^{64}$.

It should be noted that, unlike this idealized model of GGHN, it is not guaranteed for the original GGHN to enter the weak state. Instead, it can enter a cycle with respect to least significant bit [21]. This kind of cycles occurs if all LSB internal state values at different $i = 0$ moments are equal and this repeats to infinity.

VII. Conclusion

In this paper, we showed that GGHN-like ciphers with table size N may need substantially less than 2^N steps to enter a weak state. This was done by modelling an idealized GGHN(n,m) by a Markov chain, and calculating chain's absorption time.

References

[1] Charles M. Grinstead, J. Laurie Snell, Introduction to Probability, American Mathematical Society, 2nd edition, 1997

[2] S.R. Fluhrer and D.A. McGrew, *Statistical Analysis of the Alleged RC4 Keystream Generator*, Fast Software Encryption 2000, LNCS-1978, Springer-Verlag, pp. 66-71, 2000.

[3] S.R. Fluhrer, I. Mantin and A. Shamir, *Weaknesses in the Key Scheduling Algorithm of RC4*, Selected Areas in Cryptography 2001, LNCS-2259, Springer-Verlag, pp. 1-24, 2001.

[4] J.D. Golić, *Linear Statistical Weakness of Alleged RC4 Keystream Generator*, LNCS-1233, EUROCRYPT '97, pp. 226-238, Springer-Verlag, 1997.

[5] G. Gong, K. C. Gupta, M. Hell and Y. Nawaz, *Towards a General RC4-Like Keystream Generator,* CISC 2005, LNCS-3822, Springer-Verlag, 2005, pp. 162-174.

[6] A. Klein, *Attacks on the RC4 stream cipher,*, Des. Codes Cryptography 48(3), pp. 269-286, 2008.

[7] L.R. Knudsen, W. Meier, B. Prenel, V. Rijmen and S. Verdoolaege, *Analysis Methods for (Alleged) RC4,* LNCS-1514, ASIACRYPT'98, pp. 327-341, Springer-Verlag, 1998.

[8] I. Mantin and A. Shamir, *A practical Attack on Broadcast RC4*, Fast Software Encryption 2001, LNCS-2355, Springer-Verlag, pp. 152-164, 2001.

[9] I. Mantin, *Predicting and Distinguishing Attacks on RC4 Keystream Generator*, EUROCRYPT '2005, LNCS-3494, Springer-Verlag, pp. 491-506, 2005.

[10] A. Maximov, *Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers*, Fast Software Encryption 2005, LNCS-3357, Springer-Verlag, pp. 342-358, 2005.

[11] A. Maximov and D. Khovratovich, *New State Recovery Attack on RC4*, CRYPTO'08, LNCS-5365, Springer-Verlag, pp. 40-52, 2008.

[12] A J. Menezes, P. C. van Oorschot and S A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[13] I. Mironov, *Not (So) Random Shuffle RC4*, CRYPTO' 02,, LNCS-2442, Springer Verlag, pp. 304-319, 2002.

[14] Y. Nawaz, K.C. Gupta and G. Gong, *A 32-bit RC4-like Keystream Generator*, Cryptology ePrint Archive, 2005-175, IACR 2005.

[15] S. Paul and B. Preneel, *A New Weakness in RC4 Keystream Generator and an Approach to Improve the Security of the Cipher*, Fast Software Encryption 2004, LNCS-3017, Springer-Verlag, pp 245-259, 2004.

[16] S. Paul, B. Preenel, *On the (In)security of Stream Ciphers Based on Arrays and Modular Addition*, ASIACRYPT 2006, LNCS-4284, Springer-Verlag, 2006, pp. 69-83.

[17] E. Tews, R.P. Weinmann and A. Pyshkin, *Breaking of 104 Bit WEP in Less than 60 Seconds*, 2007, Cryptology ePrint Archive, 2007-120, IACR 2007.

[18] Y. Tsunoo, T. Saito, H. Kubo, and T. Suzaki, *A Distinguishing Attack on a Fast Software-Implemented RC4-Like Stream Cipher*, IEEE Trans. on Information Theory, Vol. 53, No. 9, Sept. 2007.

[19] S. Vaudenay and M. Vuagnoux, *Passive-Only Key Recovery Attacks on RC4*, Selected Areas in Cryptography 2007, LNCS-4876, Springer-Verlag, pp. 344-359, 2007.

[20] H. Wu, *Cryptanalysis of a 32-bit RC4-like Stream*, Cryptology ePrint Archive, 2005-219, IACR 2005.

[21] A. Kircanski, *Cryptanalysis of Symmetric Key Primitives*, Master Thesis, Concordia University, Montreal, Canada, 2009.