Cryptanalysis of Stream Ciphers with Linear Masking

Don Coppersmith, Shai Halevi, and Charanjit Jutla

IBM T. J. Watson Research Center, NY, USA
{copper,shaih,csjutla}@watson.ibm.com

Abstract. We describe a cryptanalytical technique for distinguishing some stream ciphers from a truly random process. Roughly, the ciphers to which this method applies consist of a "non-linear process" (say, akin to a round function in block ciphers), and a "linear process" such as an LFSR (or even fixed tables). The output of the cipher can be the linear sum of both processes. To attack such ciphers, we look for any property of the "non-linear process" that can be distinguished from random. In addition, we look for a linear combination of the linear process that vanishes. We then consider the same linear combination applied to the cipher's output, and try to find traces of the distinguishing property.
In this report we analyze two specific "distinguishing properties". One is a linear approximation of the non-linear process, which we demonstrate on the stream cipher SNOW. This attack needs roughly 2^{95} words of output, with work-load of about 2^{100}. The other is a "low-diffusion" attack, that we apply to the cipher Scream-0. The latter attack needs only about 2^{43} bytes of output, using roughly 2^{50} space and 2^{80} time.

Keywords: Hypothesis testing, Linear cryptanalysis, Linear masking, Low-Diffusion attacks, Stream ciphers.

1 Introduction

A stream cipher (or pseudorandom generator) is an algorithm that takes a short random string, and expands it into a much longer string, that still "looks random" to adversaries with limited resources. The short input string is called the seed (or key) of the cipher, and the long output string is called the output stream (or key-stream). Although one could get a pseudorandom generator simply by iterating a block cipher (say, in counter mode), it is believed that one could get higher speeds by using a "special purpose" stream cipher.

One approach for designing such fast ciphers, is to use some "non-linear process" that may resemble block cipher design, and to hide this process using linear masking. A plausible rationale behind this design, is that the non-linear process behaves roughly like a block cipher, so we expect its state at two "far away" points in time to be essentially uncorrelated. For close points, on the other hand, it can be argued they are masked by independent parts of the linear process, and so again they should not be correlated.

M. Yung (Ed.): CRYPTO 2002, LNCS 2442, pp. 515–532, 2002.

Some examples of ciphers that use this approach include SEAL [18] and Scream [2], where the non-linear process is very much like a block cipher, and the output from each step is obtained by adding together the current state of the non-linear process and some entries from fixed (or slowly modified) secret tables. Other examples are PANAMA [4] and MUGI [21], where the linear process (called buffer) is an LFSR (Linear Feedback Shift Register), which is used as input to the non-linear process, rather than to hide the output. Yet another example is SNOW [5], where the linear LFSR is used both as input to the non-linear finite state machine, and also to hide its output.

In this work we describe a technique that can be used to distinguish such ciphers from random. The basic idea is very simple. We first concentrate on the non-linear process, looking for a characteristic that can be distinguished from random. For example, a linear approximation that has noticeable bias. We then look at the linear process, and find some linear combination of it that vanishes. If we now take the same linear combination of the output stream, then the linear process would vanish, and we are left with a sum of linear approximations, which is itself a linear approximation. As we show below, this technique is not limited to linear approximations. In some sense, it can be used with "any distinguishing characteristic" of the non-linear process. In this report we analyze in details two types of "distinguishing characteristics", and show some examples of its use for specific ciphers.

Perhaps the most obvious use of this technique, is to devise linear attacks (and indeed, many such attacks are known in the literature). This is also the easiest case to analyze. In Section 4 we characterize the statistical distance between the cipher and random as a function of the bias of the original approximation of the non-linear process, and the *weight distribution* of a linear code related to the linear process of the cipher.

Another type of attacks uses the low diffusion in the non-linear process. Namely, some input/output bits of this process depend only on very few other input/output bits. For this type of attacks, we again analyze the statistical distance, as a function of the number of bits in the low-diffusion characteristic. This analysis is harder than for the linear attacks. Indeed, here we do not have a complete characterization of the possible attacks of this sort, but only an analysis for the most basic such attack.

We demonstrate the usefulness of our technique by analyzing two specific ciphers. One is the cipher SNOW [5], for which we demonstrate a linear attack, and the other is the variant Scream-0 of the stream cipher Scream [2], for which we demonstrate a low-diffusion attack.

1.1 Relation to Prior Work

Linear analyses of various types are the most common tool for cryptanalyzing stream ciphers. Much work was done on LFSR-based ciphers, trying to discover the state of the LFSRs using correlation attacks (starting from Meier and Staffelbach [17], see also, e.g., [14,13]). Golić [9,10] devised linear models (quite similar to our model of linear attacks) that can be applied in principle to any stream

cipher. He then used them to analyze many types of ciphers (including, for example, a linear distinguisher for RC4 [11]). Some examples of linear distinguishers for LFSR based ciphers, very similar to our analysis of SNOW, are [1,6], among others. Few works used also different cryptanalytical tools. Among them are the distinguishers for SEAL [12,7] and for RC4 [8].

The main contribution of the current work is in presenting a simple framework for distinguishing attacks. This framework can be applied to many ciphers, and for those ciphers it incorporates linear analysis as a special case, but can be used to devise many other attacks, such as our "low-diffusion attacks". (Also, the attacks on SEAL due to [12] and [7] can be viewed as special cases of this framework.) For linear attacks, we believe that our explicit characterization of the statistical distance (Theorem 1) is new and useful. In addition to the cryptanalytical technique, the explicit formulation of attacks on stream ciphers, as done in Section 3, is a further contribution of this work.

Organization. In Section 2 we briefly review some background material on statistical distance and hypothesis testing. In Section 3 we formally define the framework in which our techniques apply. In Section 4 we describe how these techniques apply to linear attacks, and in Section 5 we show how they apply to low-diffusion attacks.

2 Elements of Statistical Hypothesis Testing

If $\mathcal{D}$ is a distribution over some finite domain X and x is an element of X, then by $\mathcal{D}(x)$ we denote probability mass of x according to $\mathcal{D}$. For notational convenience, we sometimes denote the same probability mass by $\Pr_{\mathcal{D}}[x]$. Similarly, if $S \subseteq X$ then $\mathcal{D}(S) = \Pr_{\mathcal{D}}[S] = \sum_{x \in S} \mathcal{D}(x)$.

Definition 1 (Statistical distance). *Let $\mathcal{D}_1, \mathcal{D}_2$ be two distributions over some finite domain X. The statistical distance between $\mathcal{D}_1, \mathcal{D}_2$, is defined as*

$$|\mathcal{D}_1 - \mathcal{D}_2| \stackrel{\text{def}}{=} \sum_{x \in X} |\mathcal{D}_1(x) - \mathcal{D}_2(x)| = 2 \cdot \max_{S \subseteq X} \mathcal{D}_1(S) - \mathcal{D}_2(S)$$

(We note that the statistical distance is always between 0 and 2.) Below are two useful facts about this measure:

• Denote by $\mathcal{D}^N$ the distribution which is obtained by picking independently N elements $x_1, ..., x_n \in X$ according to $\mathcal{D}$. If $|\mathcal{D}_1 - \mathcal{D}_2| = \epsilon$, then to get $|\mathcal{D}_1^N - \mathcal{D}_2^N| = 1$, the number N needs to be between $\Omega(1/\epsilon)$ and $O(1/\epsilon^2)$. (A proof can be found, for example, in [20, Lemma 3.1.15].) In this work we sometimes make the heuristic assumption that the distributions that we consider are "smooth enough", so that we really need to set $N \approx 1/\epsilon^2$.

• If $\mathcal{D}_1, ..., \mathcal{D}_N$ are distributions over n-bit strings, we denote by $\sum \mathcal{D}_i$ the distribution over the sum (exclusive-or), $\sum_{i=1}^N x_i$, where each x_i is chosen according to $\mathcal{D}_i$, independently of all the other x_j's. Denote by $\mathcal{U}$ the uniform distribution

over $\{0,1\}^n$. If for all i, $|\mathcal{U} - \mathcal{D}_i| = \epsilon_i$, then $|\mathcal{U} - \sum \mathcal{D}_i| \leq \prod_i \epsilon_i$. (We include a proof of this simple "xor lemma" in the long version of this report [3].) In the analysis in this paper, we sometimes assume that the distributions $\mathcal{D}_i$ are "smooth enough", so that we can use the approximation $|\mathcal{U} - \sum \mathcal{D}_i| \approx \prod_i \epsilon_i$.

Hypothesis testing. We provide a brief overview of (binary) hypothesis testing. This material is covered in many statistics and engineering textbooks (e.g., [16, Ch.5]). In a binary hypothesis testing problem, there are two distributions $\mathcal{D}_1, \mathcal{D}_2$, defined over the same domain X. We are given an element $x \in X$, which was drawn according to either $\mathcal{D}_1$ or $\mathcal{D}_2$, and we need to guess which is the case. A *decision rule* for such hypothesis testing problem is a function $DR : X \to \{1,2\}$, that tells us what should be our guess for each element $x \in X$. Perhaps the simplest notion of success for a decision rule DR, is the statistical advantage that it gives (over a random coin-toss), in the case that the distributions $\mathcal{D}_1, \mathcal{D}_2$ are equally likely a-priori. Namely, $\text{adv(DR)} = \frac{1}{2}\left(\Pr_{\mathcal{D}_1}[DR(x)=1] + \Pr_{\mathcal{D}_2}[DR(x)=2]\right) - \frac{1}{2}$.

Proposition 1. *For any hypothesis-testing problem $\langle \mathcal{D}_1, \mathcal{D}_2 \rangle$, the decision rule with the largest advantage is the* maximum-likelihood *rule, $ML(x) = 1$ if $\mathcal{D}_1(x) > \mathcal{D}_2(x)$, and 2 otherwise. The advantage of the ML decision rule equals a quarter of the statistical distance, $adv(ML) = \frac{1}{4}|\mathcal{D}_1 - \mathcal{D}_2|$.*

3 Formal Framework

We consider ciphers that are built around two repeating functions (processes). One is a non-linear function $NF(x)$ and the other is a linear function $LF(w)$. The non-linear function NF is usually a permutation on n-bit blocks (typically, $n \approx 100$). The linear function LF is either an LFSR, or just fixed tables of size between a few hundred and a few thousand bits. The state of such a cipher consists of the "non-linear state" x and the "linear state" w. In each step, we apply the function NF to x and the function LF to w, and we may also "mix" these states by xor-ing some bits of w into x and vice versa. The output of the current state is also computed as an xor of bits from x and w. To simplify the presentation of this report, we concentrate on a special case, similar to Scream[1]. In each step i we do the following:

1. Set $w_i := LF(w_{i-1})$
2. Set $y_i := L1(w_i)$, $z_i = L2(w_i)$ // $L1, L2$ are some linear functions
3. Set $x_i := NF(x_{i-1} + y_i) + z_i$ // '+' denotes exclusive-or
4. Output x_i

[1] We show how our techniques can handle other variants when we describe the attack on SNOW, but we do not attempt to characterize all the variants where such techniques apply.

3.1 The Linear Process

The only property of the linear process that we care about, is that the string $y_1 z_1 y_2 z_2 \ldots$ can be modeled as a random element in some known linear subspace of $\{0,1\}^\star$. Perhaps the most popular linear process is to view the "linear state" w as the contents of an LFSR. The linear modification function LF clocks the LFSR some fixed number of times (e.g., 32 times), and the functions $L1, L2$ just pick some bits from the LFSR. If we denote the LFSR polynomial by p, then the relevant linear subspace is the subspace orthogonal to $p \cdot Z_2[x]$.

A different approach is taken in Scream. There, the "linear state" resides in some tables, that are "almost fixed". In particular, in Scream, each entry in these tables is used 16 times before it is modified (via the non-linear function NF). For our purposes, we model this scheme by assuming that whenever an entry is modified, it is actually being replaced by a new random value. The masking scheme in Scream can be thought of as a "two-dimensional" scheme, where there are two tables, which are used in lexicographical order[2]. Namely, we have a "row table" $R[\cdot]$ and a "column table" $C[\cdot]$, each with 16 entries of $2n$-bit string. The steps of the cipher are partitioned into batches of 256 steps each. At the beginning of a batch, all the entries in the tables are "chosen at random". Then, in step $i = j + 16k$ in a batch, we set $(y_i|z_i) := R[j] + C[k]$.

3.2 Attacks on Stream Ciphers

We consider an attacker that just watches the output stream and tries to distinguish it from a truly random stream. The relevant parameters in an attack are the amount of text that the attacker must see before it can reliably distinguish the cipher from random, and the time and space complexity of the distinguishing procedure. The attacks that we analyze in this report exploit the fact that for a (small) subset of the bits of x and $NF(x)$, the joint distribution of these bits differs from the uniform distribution by some noticeable amount. Intuitively, such attacks never try to exploit correlations between "far away" points in time. The only correlations that are considered, are the ones between the input and output of a single application of the non-linear function[3].

Formally, we view the non-linear process not as one continuous process, but rather as a sequence of uncorrelated steps. That is, for the purpose of the attack, one can view the non-linear state x at the beginning of each step as a new random value, independent of anything else. Under this view, the attacker sees a collection of pairs $\langle x_j + y_j, NF(x_j) + z_j \rangle$, where the x_j's are chosen uniformly at random and independently of each other, and the y_j, z_j's are taken from the linear process.

One example of attacks that fits in this model are linear attacks. In linear cryptanalysis, the attacker exploits the fact that a one-bit linear combination of

[2] The scheme in Scream is actually slightly different than the one described here, but this difference does not effect the analysis in any significant way.

[3] When only a part of x is used as output, we may be forced to look at a few consecutive applications of NF. This is the case in SNOW, for example.

$\langle x, NF(x)\rangle$ is more likely to be zero than one (or vice versa). In these attack, it is always assumed that the bias in one step is independent of the bias in all the other steps. Somewhat surprisingly, differential cryptanalysis too fits into this framework (under our attack model). Since the attacker in our model is not given chosen-input capabilities, it exploits differential properties of the round function by waiting for the difference $x_i + x_j = \Delta$ to happen "by chance", and then using the fact that $NF(x_i) + NF(x_j) = \Delta'$ is more likely than you would expect from a random process. It is clear that this attack too is just as effective against pairs of uncorrelated steps, as when given the output from the real cipher.

We are now ready to define formally what we mean by "an attack on the cipher". The attacks that we consider, observe some (linear combinations of) input and output bits from each step of the cipher, and try to decide if these indeed come from the cipher, or from a random source. This can be framed as a hypothesis testing problem. According to one hypothesis (Random), the observed bits in each step are random and independent. According to the other (Cipher), they are generated by the cipher.

Definition 2 (Attacks on stream ciphers with linear masking). *An attack is specified by a linear function ℓ, and by a decision rule for the following hypothesis-testing problem: The two distributions that we want to distinguish are*

Cipher. *The Cipher distribution is $\mathcal{D}_c = \langle \ell(x_j + y_j, NF(x_j) + z_j)\rangle_{j=1,2,\ldots}$, where the $y_j z_j$'s are chosen at random from the appropriate linear subspace (defined by the linear process of the cipher), and the x_j's are random and independent.*

Random. *Using the same notations, the "random process" distribution is $\mathcal{D}_r \stackrel{\text{def}}{=} \left\langle \ell(x_j, x'_j)\right\rangle_{j=1,2,\ldots}$, where the x_j's and x'_j's are random and independent.*

We call the function ℓ, the distinguishing characteristic *used by attack.*

The amount of text needed for the attack is the smallest number of steps for which the decision rule has a constant advantage (e.g., advantage of 1/4) in distinguishing the cipher from random. Other relevant parameters of the attack are the time and space complexity of the decision rule. An obvious lower bound on the amount of text is provided by the statistical distance between the Cipher and Random distributions after N steps.

4 Linear Attacks

A linear attack [15] exploits the fact that some linear combination of the input and output bits of the non-linear function is more likely to be zero than one (or vice versa). Namely, we have a (non-trivial) linear function $\ell : \{0,1\}^{2n} \to \{0,1\}$, such that for a randomly selected n bit string x, $\Pr[\ell(x, NF(x)) = 0] = (1+\epsilon)/2$. The function ℓ is called a *linear approximation* (or characteristic) of the non-linear function, and the quantity ϵ is called the *bias* of the approximation.

When trying to exploit one such linear approximation, the attacker observes for each step j of the cipher, a bit $\sigma_j = \ell(x_j + y_j, NF(x_j) + z_j)$. Note that

σ_j by itself is likely to be unbiased, but the σ's are correlated. In particular, since the y, z's come from a linear subspace, it is possible to find some linear combination of steps for which they vanish. Let J be a set of steps such that $\sum_{j\in J} y_j = \sum_{j\in J} z_j = 0$. Then we have

$$\sum_{j\in J}\sigma_j \;=\; \sum_{j\in J}\ell(x_j, NF(x_j)) + \sum_{j\in J}\ell(y_j, z_j) \;=\; \sum_{j\in J}\ell(x_j, NF(x_j))$$

(where the equalities follow since ℓ is linear). Therefore, the bit $\xi_J = \sum_{j\in J}\sigma_j$ has bias of $\epsilon^{|J|}$. If the attacker can observe "sufficiently many" such sets J, it can reliably distinguish the cipher from random.

This section is organized as follows: We first bound the effectiveness of linear attacks in terms of the bias ϵ and the *weight distribution* of some linear subspace. As we explain below, this bound suggests that looking at sets of steps as above is essentially "the only way to exploit linear correlations". Then we show how to devise a linear attack on SNOW, and analyze its effectiveness.

4.1 The Statistical Distance

Recall that we model an attack in which the attacker observes a single bit per step, namely $\sigma_j = \ell(x_j + y_j, NF(x_j) + z_j)$. Below we denote $\tau_j = \ell(x_j, NF(x_j))$ and $\rho_j = \ell(y_j, z_j)$. We can re-write the Cipher and Random distributions as

Cipher. $\mathcal{D}_c \stackrel{\text{def}}{=} \langle \tau_j + \rho_j \rangle_{j=1,2,\ldots}$, where the τ_j's are independent but biased, $\Pr[\tau_j = 0] = (1+\epsilon)/2$, and the string $\rho_1\rho_2\ldots$ is chosen at random from the appropriate linear subspace (i.e., the image under ℓ of the linear subspace of the $y_j z_j$'s).

Random. $\mathcal{D}_r \stackrel{\text{def}}{=} \langle \sigma_j \rangle_{j=1,2,\ldots}$, where the σ_j's are independent and unbiased.

Below we analyze the statistical distance between the Cipher and Random distributions, after observing N bits $\sigma_1 \ldots \sigma_N$. Denote the linear subspace of the ρ's by $L \subseteq \{0,1\}^N$, and let $L^\perp \subseteq \{0,1\}^N$ be the orthogonal subspace. The *weight distribution* of the space $L^\perp$ plays an important role in our analysis. For $r \in \{0, 1, \ldots, N\}$, let $\mathcal{A}_N(r)$ be the set of strings $\boldsymbol{\chi} \in L^\perp$ of Hamming weight r, and let $A_N(r)$ denote the cardinality of $\mathcal{A}_N(r)$. We prove the following theorem:

Theorem 1. *The statistical distance between the Cipher and Random distributions from above, is bounded by* $\sqrt{\sum_{r=1}^{N} A_N(r)\epsilon^{2r}}$.

Proof. Included in the long version [3].

Remark. Heuristically, this bound is nearly tight. In the proof we analyzed the random variable Δ and used the bound $E[|\Delta - E[\Delta]|] \le \sqrt{\mathrm{VAR}[\Delta]}$. One can argue heuristically that as long as the statistical distance is sufficiently small, "Δ should behave much like a Gaussian random variable". If it were a Gaussian, we would have $E[|\Delta|] = \sqrt{\mathrm{VAR}[\Delta]} \cdot \sqrt{2/\pi}$. Thus, we expect the bound from Theorem 1 to be tight up to a constant factor $\sqrt{2/\pi} \approx 0.8$.

4.2 Interpretations of Theorem 1

There are a few ways to view Theorem 1. The obvious way is to use it in order to argue that a certain cipher is resilient to linear attacks. For example, in [2] we use Theorem 1 to deduce a lower-bound on the amount of text needed for any linear attack on Scream-0.

Also, one could notice that the form of Theorem 1 exactly matches the common practice (and intuition) of devising linear attacks. Namely, we always look at sets where the linear process vanishes, and view each such set J as providing "statistical evidence of weight $\epsilon^{2|J|}$" for distinguishing the cipher from random. Linear attacks work by collecting enough of these sets, until the weights sum up of to one. One can therefore view Theorem 1 as asserting that this is indeed the best you can do.

Finally, we could think of devising linear attacks, using the heuristic argument about this bound being tight. However, the way Theorem 1 is stated above, it usually does not imply efficient attacks. For example, when the linear space L has relatively small dimension (as is usually the case with LFSR-based ciphers, where the dimension of L is at most a few hundreds), the statistical distance is likely to approach one for relatively small N. But it is likely that most of the "mass" in the bound of Theorem 1 comes from terms with a large power of ϵ (and therefore very small "weight"). Therefore, if we want to use a small N, we would need to collect very many samples, and this attack is likely to be more expensive than an exhaustive search for the key.

Alternatively, one can try and use an efficient sub-optimal decision rule. For a given bound on the work-load W and the amount of text N, we only consider the first few terms in the power series. That is, we observe the N bits $\boldsymbol{\sigma} = \sigma_1 \ldots \sigma_N$, but only consider the W smallest sets J for which $\boldsymbol{\chi}(J) \in L^\perp$. For each such set J, the sum of steps $\sum_{j \in J} \sigma_j$ has bias $\epsilon^{|J|}$, and these can be used to distinguish the cipher from random. If we take all the sets of size at most R, we expect the advantage of such a decision rule to be roughly $\frac{1}{4}\sqrt{\sum_{r=1}^{R} A_N(r)\epsilon^{2r}}$. The simplest form of this attack (which is almost always the most useful), is to consider only the minimum-weight terms. If the minimum-weight of $L^\perp$ is r_0, then we need to make N big enough so that $\frac{1}{4}\sqrt{A_N(r_0)} = \epsilon^{-r_0}$.

4.3 The Attack on SNOW

The stream cipher SNOW was submitted to NESSIE in 2000, by Ekdahl and Johansson. A detailed description of SNOW is available from [5]. Here we outline a linear attack on SNOW along the lines above, that can reliably distinguish it from random after observing roughly 2^{95} steps of the cipher, with work-load of roughly 2^{100}.

SNOW consists of a non-linear process (called there a Finite-State Machine, or FSM), and a linear process which is implemented by an LFSR. The LFSR of SNOW consists of sixteen 32-bit words, and the LFSR polynomial, defined over $GF(2^{32})$, is $p(z) = z^{16} + z^{13} + z^7 + \alpha$, where α is a primitive element of $GF(2^{32})$. (The orthogonal subspace $L^\perp$ is therefore the space of (bitwise

reversal of) polynomials over Z_2 of degree $\leq N$, which are divisible by the LFSR polynomial p.) At a given step j, we denote the content of the LFSR by $L_j[0..15]$, so we have $L_{j+1}[i] = L_j[i-1]$ for $i > 0$ and $L_{j+1}[0] = \alpha \cdot (L_j[15] + L_j[12] + L_j[6])$.

The "FSM state" of SNOW in step j consists of only two 32-bit words, denoted $R1_j, R2_j$. The FSM update function modifies these two values, using one word from the LFSR, and also outputs one word. The output word is then added to another word from the LFSR, to form the step output. We denote the "input word" from the LFSR to the FSM update function by f_j, and the "output word" from the FSM by F_j. The FSM uses a "32×32 S-box" $S[\cdot]$ (which is built internally as an SP-network, from four identical 8×8 boxes and some bit permutation). A complete step of SNOW is described in Figure 1. In this figure, we deviate from the notations in the rest of the paper, and denote exclusive-or by $\oplus$ and integer addition mod 2^{32} by $+$. We also denote 32-bit cyclic rotation to the left by $\lll$.

1. $f_j := L_j[0]$
2. $F_j := (f_j + R1_j) \oplus R2_j$
3. output $F_j \oplus L_j[15]$
4. $R1_{j+1} := R1_j \oplus ((R2_j + F_j) \lll 7)$
5. $R2_{j+1} := S[R1_j]$
6. update the LFSR

Fig. 1. One step of SNOW: $\oplus$ is xor and $+$ is addition mod 2^{32}.

To devise an attack we need to find a good linear approximation of the non-linear FSM process, and low-weight combinations of steps where the $L_j[\cdot]$ values vanish (i.e., low-weight polynomials which are divisible by the LFSR polynomial p). The best linear approximation that we found for the FSM process, uses six bits from two consecutive inputs and outputs, $f_j, f_{j+1}, F_j, F_{j+1}$. Specifically, for each step j, the bit

$$\sigma_j \stackrel{\text{def}}{=} (f_j)_{15} + (f_j)_{16} + (f_{j+1})_{22} + (f_{j+1})_{23} + (F_j)_{15} + (F_{j+1})_{23}$$

is biased. (Of these six bits, the bits $(f_j)_{15}, (F_j)_{15}$ and $(F_{j+1})_{22}$ are meant to approximate carry bits.) We measured the bias experimentally, and it appears to be at least $2^{-8.3}$.

At first glance, one may hope to find weight-4 polynomials that are divisible by the LFSR polynomial p. After all, p itself has only four non-zero terms. Unfortunately, one of these terms is the element $\alpha \in GF(2^{32})$, whereas we need a low-weight polynomial with 0-1 coefficients. What we can show, however, is the existence of 0-1 polynomials of weight-six that are divisible by p.

Proposition 2. *The polynomial* $q(z) = z^{16 \times 2^{32}-7} + z^{13 \times 2^{32}-7} + z^{7 \times 2^{32}-7} + z^9 + z^6 + 1$ *is divisible by the LFSR polynomial* $p(z) = z^{16} + z^{13} + z^7 + \alpha$.

Proof. Included in the long version [3].

Corollary 1. *For all m, n, the polynomial $q_{m,n}(z) \stackrel{\text{def}}{=} q(z)^{2^m} \cdot z^n$ is divisible by $p(z)$.*

If we take, say, $m = 0, 1, \ldots 58$ and $n = 0, 1, \ldots 2^{94}$, we get about 2^{100} different 0-1 polynomials, all with weight 6 and degree less than $N = 2^{95}$, and all divisible by $p(z)$. Each such polynomial yields a sequence of six steps, $J_{m,n}$, such that the sum of the $L_j[\cdot]$ values in these steps vanishes. Therefore, if we denote the output word of SNOW at step j by S_j, then for all m, n we have,

$$\tau_{m,n} \stackrel{\text{def}}{=} \sum_{j \in J_{m,n}} (S_j)_{15} + (S_{j+1})_{23} \quad = \quad \sum_{j \in J_{m,n}} \sigma_j$$

and therefore each $\tau_{m,n}$ has bias of $2^{-8.3 \times 6} = 2^{-49.8}$. Since we have roughly 2^{100} of them, we can reliably distinguish them from random.

5 Low-Diffusion Attacks

In low-diffusion attacks, the attacker looks for a small set of (linear combinations of) input and output bits of the non-linear function NF, whose values completely determine the values of some other (linear combinations of) input and output bits. The attacker tries to guess the first set of bits, computes the values of the other bits, and uses the computed value to verify the guess against the cipher's output. The complexity of such attacks is exponential in the number of bits that the attacker needs to guess.

We introduce some notations in order to put such attacks in the context of our framework. To simplify the notations, we assume that the guessed bits are always input bits, and the determined bits are always output bits. (Eliminating this assumption is usually quite straightforward.) As usual, let $NF : \{0,1\}^n \to \{0,1\}^n$ be the non-linear function. The attack exploits the fact that some input bits $\ell_{\text{in}}(x)$ are related to some output bits $\ell_{\text{out}}(NF(x))$ via a known deterministic function f. That is, we have $\ell_{\text{out}}(NF(x)) = f(\ell_{\text{in}}(x))$. Here, $\ell_{\text{in}}, \ell_{\text{out}}$ are linear functions, and f is an arbitrary function, all known to the attacker. We denote the output size of $\ell_{\text{in}}, \ell_{\text{out}}$ by m, m', respectively. That is, $\ell_{\text{in}} : \{0,1\}^n \to \{0,1\}^m$, $\ell_{\text{out}} : \{0,1\}^n \to \{0,1\}^{m'}$, and $f : \{0,1\}^m \to \{0,1\}^{m'}$.

In each step j, the attacker observes the bits $\ell_{\text{in}}(x_j + y_j)$ and $\ell_{\text{out}}(NF(x_j) + z_j)$ (where y_j, z_j are from the linear process, as in Section 3.1). Below we denote $u_j = \ell_{\text{in}}(x_j)$, $u'_j = \ell_{\text{out}}(NF(x_j))$, $v_j = \ell_{\text{in}}(y_j)$, $v'_j = \ell_{\text{out}}(z_j)$, and $w_j = u_j + v_j$, $w'_j = u'_j + v'_j$. We can re-write the Cipher and Random distributions as

Cipher. $\mathcal{D}_c \stackrel{\text{def}}{=} \left\langle (w_j = u_j + v_j,\ w'_j = u'_j + v'_j) \right\rangle_{j=1,2,\ldots}$, where the u_j's are uniform and independent, $u'_j = f(u_j)$, and the string $v_1 v'_1 v_2 v'_2 \ldots$ is chosen at random from the appropriate linear subspace (i.e., the image under $\ell_{\text{in}}, \ell_{\text{out}}$ of the linear subspace of the y, z's).

Random. $\mathcal{D}_r \stackrel{\text{def}}{=} \left\langle (w_j, w'_j) \right\rangle_{j=1,2,\ldots}$, all uniform and independent.

It is not hard to see that there may be enough information there to distinguish these two distributions after only a moderate number of steps of the cipher. Suppose that the dimension of the linear subspace of the v_j's and v'_j's is a, and the attacker observes N steps such that $m'N > a$. Then, the attacker can (in principle) go over all the 2^a possibilities for the v_j's and v'_j's. For each guess, the attacker can compute the u_j's and u'_j's, and verify the guess by checking that $u'_j = f(u_j)$ for all j. This way, the attacker guesses a bits and gets $m'N$ bits of consistency checks. Since $m'N > a$ we expect only the "right guess" to pass the consistency checks.

This attack, however, is clearly not efficient. To devise an efficient attack, we can again concentrate on sets of steps where the linear process vanishes: Suppose that we have a set of steps J, such that $\sum_{j\in J}[v_j, v'_j] = [0, 0]$. Then we get

$$\sum_{j\in J}(w_j, w'_j) \;=\; \sum_{j\in J}(u_j, u'_j) \;=\; \sum_{j\in J}(u_j, f(u_j))$$

and the distribution over such pairs may differ from the uniform distribution by a noticeable amount. The distance between this distribution and the uniform one, depends on the specific function f, and on the cardinality of the set J. Below we analyze in details perhaps the simplest cases, where f is a random function. Later we explain how this analysis can be extended for other settings, and in particular for the case of the functions in Scream.

5.1 Analysis for Random Functions

For a given function, $f : \{0,1\}^m \to \{0,1\}^{m'}$, and an integer n, we denote $\mathcal{D}_f^n \stackrel{\text{def}}{=} \left\langle d = \sum_{j=1}^n u_j,\ d' = \sum_{j=1}^n f(u_j) \right\rangle$, where the u_j's are uniform in $\{0,1\}^m$ and independent. We assume that the attacker knows f, and it sees many instances of $\langle d, d'\rangle$. The attacker needs to decide if these instances come from $\mathcal{D}_f^n$ or from the uniform distribution on $\{0,1\}^{m+m'}$. Below we denote the uniform distribution by $\mathcal{R}$. If the function f "does not have any clear structure", it makes sense to analyze it as if it was a random function. Here we prove the following:

Theorem 2. *Let n, m, m' be integers with $n^2 \ll 2^m$ [4]. For a uniformly selected function $f : \{0,1\}^m \to \{0,1\}^{m'}$, $E_f[|\mathcal{D}_f^n - \mathcal{R}|] \le c(n) \cdot 2^{\frac{m'-(n-1)m}{2}}$, where*

$$c(n) = \begin{cases} \sqrt{(2n)!\ /\ (n!\ 2^n)} & \text{if } n \text{ is odd} \\ (1+o(1))\sqrt{\frac{(2n)!}{n!\ 2^n} - \left(\frac{n!}{(n/2)!\ 2^{n/2}}\right)^2} & \text{if } n \text{ is even} \end{cases}$$

Proof. Included in the long version [3]. We note that the term $2^{\frac{m'-(n-1)m}{2}}$ is due to the fact that the attacker guesses $(n-1)m$ bits and gets m' bits of consistency check, and the term $c(n)$ is due to the symmetries in the guessed bits. (For example, the vector $\boldsymbol{u} = u_1 \ldots u_n$ is equivalent to any permutation of $\boldsymbol{u}$.)

[4] It can be shown that the same bounds hold also for larger n's, but assuming $n^2 \ll 2^m$ makes some proofs a bit easier.

How tight is this bound? Here too we can argue heuristically that the random variables in the proof "should behave like Gaussian random variables", and again we expect the ratio between $E[|X - E[X]|]$ and $\sqrt{\mathrm{VAR}[X]}$ to be roughly $\sqrt{2/\pi}$. Therefore, we expect the constant $c(n)$ to be replaced by $\sqrt{2/\pi} \cdot c(n) \approx 0.8c(n)$. Indeed we ran some experiments to measure the statistical distance $|\mathcal{D}_f^n - \mathcal{R}|$, for random functions with $n = 4$ and a few values of m, m'. (Note that $c(4) = (1 + o(1))\sqrt{96} \approx 9.8$ and $\sqrt{2/\pi} \cdot c(4) \approx 7.8$). These experiments are described in the long version of this report [3]. The results confirm that the distance between these distributions is just under $7.8 \cdot 2^{(m'-3m)/2}$.

5.2 Variations and Extensions

Here we briefly discuss a few possible extensions to the analysis from above.

Using different f's for different steps. Instead of using the same f everywhere, we may have different f's for different steps. I.e., in step j we have $\ell_{\mathrm{out}}(NF(x_j)) = f_j(\ell_{\mathrm{in}}(x_j))$, and we assume that the f_j's are random and independent. The distribution that we want to analyze is therefore $\langle d = \sum u_j, d' = \sum f_j(u_j) \rangle$. The analysis from above still works for the most part (as long as $\ell_{\mathrm{in}}, \ell_{\mathrm{out}}$ are the same in all the steps). The main difference is that the factor $c(n)$ is replaced by a smaller one (call it $c'(n)$).

For example, if we use n independent functions, we get $c'(n) = 1$, since all the symmetries in the proof of Theorem 2 disappear. Another example (which is used in the attack on Scream-0) is when we have just two independent functions, $f_1 = f_3 = \cdots$ and $f_2 = f_4 = \cdots$. In this case (and when n is divisible by four), we get $c'(n) = (1 + o(1))\sqrt{\left(\frac{n!}{(n/2)!\ 2^{n/2}}\right)^2 - \left(\frac{(n/2)!}{(n/4)!\ 2^{n/4}}\right)^4}$.

When f is a sum of a few functions. An important special case, is when f is a sum of a few functions. For example, in the functions that are used in the attack on Scream-0, the m-bit input to f can be broken into three *disjoint* parts, each with $m/3$ bits, so that $f(x) = f^1(x^1) + f^2(x^2) + f^3(x^3)$. (Here we have $|x^1| = |x^2| = |x^3| = m/3$ and $x = x^1x^2x^3$.) If f^1, f^2, f^3 themselves do not have any clear structure, then we can apply the analysis from above to each of them. That analysis tells us that each of the distributions $\mathcal{D}^i \stackrel{\mathrm{def}}{=} (\sum_j u_j^i, \sum_j f^i(u_j^i))$ is likely to be roughly $c(n) \cdot 2^{(m'-(n-1)m/3)/2}$ away from the uniform distribution.

It is not hard to see that the distribution $\mathcal{D}_f^n$ that we want to analyze can be cast as $\mathcal{D}^1 + \mathcal{D}^2 + \mathcal{D}^3$, so we expect to get $|\mathcal{D}_f^n - \mathcal{R}| \approx \prod |\mathcal{D}^i - \mathcal{R}| \approx \left(c(n) \cdot 2^{(m'-(n-1)m/3)/2}\right)^3 = c(n)^3 2^{(3m'-(n-1)m)/2}$. More generally, suppose we can write f as a sum of r functions over disjoint arguments of the same length. Namely, $f(x) = \sum_{i=1}^r f^i(x^i)$, where $|x^1| = ... = |x^r| = m/r$ and $x = x^1...x^r$. Repeating the argument from above, we get that the expected distance $|\mathcal{D}_f^n - \mathcal{R}|$ is about $c(n)^r 2^{(rm'-(n-1)m)/2}$ (assuming that this is still smaller than one). As before, one could use the "Gaussian heuristics" to argue that for the "actual distance" we should replace $c(n)^r$ by $(c(n) \cdot \sqrt{2/\pi})^r$. (And if we have different functions for different steps, as above, then we would get $(c'(n) \cdot \sqrt{2/\pi})^r$.)

Linear masking over different groups. Another variation is when we do linear masking over different groups. For example, instead of xor-ing the masks, we add them modulo some prime q, or modulo a power of two. Again, the analysis stays more or less the same, but the constants change. If we work modulo a prime $q > n$, we get a constant of $c'(n) = \sqrt{n!}$, since the only symmetry that is left is between all the orderings of $\{u_1, \ldots, u_n\}$. When we work modulo a power of two, the constant will be somewhere between $c'(n)$ and $c(n)$, probably closer to the former.

5.3 Efficiency Considerations

The analysis from above says nothing about the computational cost of distinguishing between $\mathcal{D}_f^n$ and $\mathcal{R}$. It should be noted that in a "real life" attack, the attacker may have access to many different relations (with different values of m, m'), all for the same non-linear function NF. To minimize the amount of needed text, the attacker may choose to work with the relation for which the quantity $(n-1)m - m'$ is minimized. However, the choice of relations is limited by the attacker's computational resources. Indeed, for large values of m, m', computing the maximum-likelihood decision rule may be prohibitively expensive in terms of space and time. Below we review some strategies for computing the maximum-likelihood decision rule.

Using one big table. Perhaps the simplest strategy, is for the attacker to prepare off-line a table of all possible pairs $\langle d, d' \rangle$ with $d \in \{0,1\}^m$, $d' \in \{0,1\}^{m'}$. For each pair $\langle d, d' \rangle$ the table contains the probability of this pair under the distribution $\mathcal{D}_f^n$ (or perhaps just one bit that says whether this probability is more than $2^{-m-m'}$).

Given such a table, the on-line part of the attack is trivial: for each set of steps J, compute $(d, d') = \sum_{j \in J}(w_j, w'_j)$, and look into the table to see if this pair is more likely to come from $\mathcal{D}_f^n$ or from $\mathcal{R}$. After observing roughly $2^{(n-1)m-m'}/c(n)^2$ such sets J, a simple majority vote can be used to determine if this is the cipher or a random process. Thus, the on-line phase is linear in the amount of text that has to be observed, and the space requirement is $2^{m+m'}$.

As for the off-line part (in which the table is computed), the naive way is to go over all possible values of $u_1 \ldots u_n \in \{0,1\}^m$, for each value computing $d = \sum u_i$ and $d' = \sum f(u_i)$ and increasing the corresponding entry $\langle d, d' \rangle$ by one. This takes 2^{mn} time. However, in the (typical) case where $m' \ll (n-1)m$, one can use a much better strategy, whose running time is only $O(\log n(m+m')2^{m+m'})$.

First, we represent the function f by a $2^m \times 2^{m'}$ table, with $F[x,y] = 1$ if $f(x) = y$, and $F[x,y] = 0$ otherwise. Then, we compute the convolution of F with itself, $E \stackrel{\text{def}}{=} F \star F$ [5],

[5] Recall that the convolution operator is defined on one-dimensional vectors, not on matrices. Indeed, in this expression we view the table F as a one-dimensional vector, whose indexes are $m + m'$-bits long.

$$E[s,t] \;=\; \sum_{x+x'=s} \;\sum_{y+y'=t} F[x,y]\cdot F[x',y'] \;=\; |\{x \;:\; f(x)+f(x+s)=t\}|$$

(Note that E represents the distribution $\mathcal{D}_f^2$.) One can use the Walsh-Hadamard transform to perform this step in time $O((m+m')2^{m+m'})$ (see, e.g., [19]). Then, we again use the Walsh-Hadamard transform to compute the convolution of E with itself,

$$\begin{aligned} D[d,d'] \stackrel{\text{def}}{=} (E\star E)[d,d'] \;&=\; \sum_{s+s'=d} \;\sum_{t+t'=d'} E(s,t)\cdot E(s',t') \\ &= |\{\langle x,s,z\rangle :\; f(x)+f(x+s)+f(z)+f(z+s+d)=d'\}| \\ &= |\{\langle x,y,z\rangle :\; f(x)+f(y)+f(z)+f(x+y+z+d)=d'\}| \end{aligned}$$

thus getting the distribution $\mathcal{D}_f^4$, etc. After $\log n$ such steps, we get the distribution of $\mathcal{D}_f^n$.

When f is a sum of functions. We can get additional flexibility when f is a sum of functions on disjoint arguments, $f(x) = f^1(x^1) + \cdots + f^r(x^r)$ (with $x = x^1 \ldots x^r$). In this case, one can use the procedure from above to compute the tables $D^i[d,d']$ for the individual f^i's. If all the x^i's are of the same size, then each of the D^i's takes up $2^{m'+(m/r)}$ space, and can be computed in time $O(\log n(m'+(m/r))2^{m'+(m/r)})$. Then, the "global" D table can again be computed using convolutions. Specifically, for any fixed $d = d^1 ... d^r$, the $2^{m'}$-vector of entries $D[d,\cdot]$ can be computed as the convolutions of the $2^{m'}$-vectors $D^1[d^1,\cdot]$, $D^2[d^2,\cdot]$, ..., $D^r[d^r,\cdot]$,

$$D[d,\cdot] \;=\; D^1[d^1,\cdot]\star D^2[d^2,\cdot]\star\cdots\star D^r[d^r,\cdot]$$

At first glance, this does not seem to help much: Computing each convolution takes time $O(r\cdot m'2^{m'})$, and we need to repeat this for each $d\in\{0,1\}^m$, so the total time is $O(rm'2^{m+m'})$. However, we can do much better than that.

Instead of storing the vectors $D^i[d^i,\cdot]$ themselves, we store their image under the Walsh-Hadamard transform, $\Delta^i[d^i,\cdot] \stackrel{\text{def}}{=} \mathcal{H}(D^i[d^i,\cdot])$. Then, to compute the vector $D[\langle d^1 ... d^r\rangle,\cdot]$, all we need is to multiply (point-wise) the corresponding $\Delta^i[d^i,\cdot]$'s, and then apply the inverse Walsh-Hadamard transform to the result. Thus, once we have the tables $D^i[\cdot,\cdot]$, we need to compute $r\cdot 2^{m/r}$ "forward transforms" (one for each vector $D^i[d^i,\cdot]$), and 2^m inverse transforms (one for each $\langle d^1 ... d^r\rangle$. Computing each transform (or inverse) takes $O(m'2^{m'})$ time. Hence, the total time (including the initial computation of the D^i's) is $O\left(\log n(rm'+m)2^{m'+(m/r)} + m'2^{m+m'}\right)$, and the total space that is needed is $O(2^{m+m'})$.

If the amount of text that is needed is less than 2^m, then we can optimize even further. In this case the attacker need not store the entire table D in memory. Instead, it is possible to store only the D^i tables (or rather, the $\Delta^i[\cdot,\cdot]$ vectors), and compute the entries of D during the on-line part, as they are needed. Using this method, the off-line phase takes $O(\log n(rm'+m)2^{m'+(m/r)})$ time and $O(r2^{m'+m/r})$ space to compute and store the vectors $\Delta_i[\cdot,\cdot]$, and the

on-line phase takes $O(m'2^{m'})$ time per sample. Thus the total time complexity here is $O(\log n(rm'+m)2^{m'+(m/r)} + Sm'2^{m'})$, where S is the number of samples needed to distinguish $\mathcal{D}$ from $\mathcal{R}$.

5.4 An Attack on Scream-0

The stream cipher Scream (with its variants Scream-0 and Scream-F) was proposed very recently by Coppersmith, Halevi and Jutla. A detailed description of Scream is available in [2]. Below we only give a partial description of Scream-0, which suffices for the purpose of our attack.

Scream-0 maintains a 128-bit "non-linear state" x, two 128-bit "column masks" $c1, c2$ (which are modified every sixteen steps), and a table of sixteen "row masks" $R[0..15]$. It uses a non-linear function NF, somewhat similar to a round of Rijndael. Roughly speaking, the steps of Scream-0 are partitioned to chunks of sixteen steps. A description of one such chunk is found in Figure 2.

1. for $i = 0$ to 15 do
2. $\quad x := NF(x + c1) + c2$
3. $\quad$ output $x + R[i]$
4. $\quad$ if i is even, rotate $c1$ by 64 bits
5. $\quad$ if i is odd, rotate $c1$ by some other amount
6. end-for
7. modify $c1, c2$, and one entry of R, using the function $NF(\cdot)$

Fig. 2. sixteen steps of Scream-0.

Here we outline a low-diffusion attack on the variant Scream-0, along the lines above, that can reliably distinguish it from random after observing merely 2^{43} bytes of output, with memory requirement of about 2^{50} and work-load of about 2^{80}. This attack is described in more details in the long version of [2].

As usual, we need to find a "distinguishing characteristic" of the non-linear function (in this case, a low-diffusion characteristic), and a combination of steps in which the linear process vanishes. The linear process consists of the c_i's and the $R[i]$'s. Since each entry $R[i]$ is used sixteen times before it is modified, we can cancel it out by adding two steps were the same entry is used. Similarly, we can cancel c_2 by adding two steps within the same "chunk" of sixteen steps. However, since $c1$ is rotated after each use, we need to look for two different characteristics of the NF function, such that the pattern of input bits in one characteristic is a rotated version of the pattern in the other.

The best such pair of "distinguishing characteristics" that we found for Scream-0, uses a low-diffusion characteristic for NF in which the input bits pattern is 2-periodic (and the fact that $c1$ is rotated every other step by 64 bits). Specifically, the four input bytes x_0, x_5, x_8, x_{13}, together with two bytes of linear combinations of the output $NF(x)$, yield the two input bytes x_2, x_{10}, and two other bytes of linear combinations of the output $NF(x)$. In terms of the

parameters that we used above, we have $m = 48$ input and output bits, which completely determine $m' = 32$ other input and output bits.

To use this relation, we can observe these ten bytes from each of four steps, (i.e., $j, j+1, j+16k, j+1+16k$ for even j and $k < 16$). We can then add them up (with the proper rotation of the input bytes in steps $j+1, j+17$), to cancel both the "row masks" $R[i]$ and the "column masks" $c1, c2$. This gives us the following distribution $\mathcal{D} = \langle u_1 + u_2 + u_3 + u_4,\ f_1(u_1) + f_2(u_2) + f_1(u_3) + f_2(u_4)\rangle$, where the u_i's are modeled as independent, uniformly selected, 48-bit strings, and f_1, f_2 are two known functions $f_j : \{0,1\}^{48} \to \{0,1\}^{32}$. (The reason that we have two different functions is that the order of the input bytes is different between the even and odd steps.) Moreover, each of the two f_j's can be written as a sum of three functions over disjoint parts, $f_j(x) = f_j^1(x^1) + f_j^2(x^2) + f_j^3(x^3)$ where $|x^1| = |x^2| = |x^3| = 16$.

This is one of the "extensions" that were discussed in Section 5.2. Here we have $n = 4$, $m = 48$, $m' = 32$, $r = 3$, and two different functions. Therefore, we expect to get statistical distance of $c'(n)^3 \cdot 2^{(3m'-(n-1)m)/2}$, with

$$c'(n) \approx \sqrt{2/\pi} \cdot \sqrt{\left(\frac{n!}{(n/2)!\ 2^{n/2}}\right)^2 - \left(\frac{(n/2)!}{(n/4)!\ 2^{n/4}}\right)^4}$$

Plugging in the parameters, we have $c'(4) \approx \sqrt{2/\pi} \cdot \sqrt{8}$, and the expected statistical distance is roughly $(16/\pi)^{3/2} \cdot 2^{-24} \approx 2^{-20.5}$. We therefore expect to be able to reliably distinguish $\mathcal{D}$ from random after about 2^{41} samples. Roughly speaking, we can get $8 \cdot \binom{14}{2} \approx 2^{10}$ samples from 256 steps of Scream-0. (We have 8 choices for an even step in a chunk of 16 steps, and we can choose two such chunks from a collection of 14 in which the three row masks in use remain unchanged.) So we need about $2^{31} \cdot 256 = 2^{39}$ steps, or 2^{43} bytes of output.

Also, in Section 5.3 we show how one could efficiently implement the maximum-likelihood decision rule to distinguish $\mathcal{D}$ from $\mathcal{R}$, using Walsh-Hadamard transforms. Plugging the parameters of the attack on Scream-0 into the general techniques that are described there, we have space complexity of $O(r2^{m'+m/r})$, which is about 2^{50}. The time complexity is $O(\log n(rm'+m)2^{m'+(m/r)} + Sm'2^{m'})$, where in our case $S = 2^{41}$, so we need roughly 2^{80} time.

6 Conclusions

In this work we described a general cryptanalytical technique that can be used to attack ciphers that employ a combination of a "non-linear" process and a "linear process". We analyze in details the effectiveness of this technique for two special cases. One is when we exploit linear approximations of the non-linear process, and the other is when we exploit the low diffusion of (one step of) the non-linear process. We also show how these two special cases are useful in attacking the ciphers SNOW [5] and Scream-0 [2].

It remains an interesting open problem to extend the analysis that we have here to more general "distinguishing characteristics" of the non-linear process.

For example, extending the analysis of the low-diffusion attack from Section 5.1 to the case where the functions f is key-dependent (and thus not known to the adversary) may yield an effective attack on Scream [2].

In addition to the cryptanalytical technique, we believe that another contribution of this work is our formulation of attacks on stream ciphers. We believe that explicitly formalizing an attack as considering sequence of uncorrelated steps (as opposed to one continuous process) can be used to shed light on the strength of many ciphers.

References

1. A. Canteaut and E. Filiol. Ciphertext only reconstruction of stream ciphers based on combination generators. In *Fast Software Encryption*, volume 1978 of *Lecture Notes in Computer Science*, pages 165–180. Springer-Verlag, 2000.
2. D. Copersmith, S. Halevi, and C. Jutla. Scream: a software-efficient stream cipher. In *Fast Software Encryption*, Lecture Notes in Computer Science. Springer-Verlag, 2002. to appear. A longer version is available on-line from `http://eprint.iacr.org/2002/019/`.
3. D. Coppersmith, S. Halevi, and C. Jutla. Cryptanalysis of stream ciphers with linear masking. Available from the ePrint archive, at `http://eprint.iacr.org/2002/020/`, 2002.
4. J. Daemen and C. S. K. Clapp. Fast hashing and stream encryption with Panama. In S. Vaudenay, editor, *Fast Software Encryption: 5th International Workshop*, volume 1372 of *Lecture Notes in Computer Science*, pages 23–25. Springer-Verlag, 1998.
5. P. Ekdahl and T. Johansson. SNOW – a new stream cipher. Submitted to NESSIE. Available on-line from `http://www.it.lth.se/cryptology/snow/`.
6. P. Ekdahl and T. Johansson. Distinguishing attacks on SOBER-t16 and t32. In *Fast Software Encryption*, Lecture Notes in Computer Science. Springer-Verlag, 2002. to appear.
7. S. Fluhrer. Cryptanalysis of the SEAL 3.0 pseudorandom function family. In *Proceedings of the Fast Software Encryption Workshop (FSE'01)*, 2001.
8. S. R. Fluhrer and D. A. McGraw. Statistical analysis of the alleged RC4 keystream generator. In *Proceedings of the 7th Annual Workshop on Fast Software Encryption, (FSE'2000)*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer-Verlag, 2000.
9. J. D. Golić. Correlation properties of a general binary combiner with memory. *Journal of Cryptology*, 9(2):111–126, 1996.
10. J. D. Golić. Linear models for keystream generators. *IEEE Trans. on Computers*, 45(1):41–49, Jan 1996.
11. J. D. Golić. Linear statistical weakness of alleged RC4 keystream generator. In W. Fumy, editor, *Advances in Cryptology – Eurocrypt'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 226–238. Springer-Verlag, 1997.
12. H. Handschuh and H. Gilbert. χ^2 cryptanalysis of the SEAL encryption algorithm. In *Proceedings of the 4th Workshop on Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 1997.
13. T. Johansson and F. Jönsson. Fast correlation attacks based on turbo code techniques. In *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 1999.

14. T. Johansson and F. Jönsson. Improved fast correlation attacks on stream ciphers via convolution codes. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, 1999.
15. M. Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology, EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1993.
16. R. N. McDonough and A. D. Whalen. *Detection of Signals in Noise*. Academic Press, Inc., 2nd edition, 1995.
17. W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.
18. P. Rogaway and D. Coppersmith. A software optimized encryption algorithm. *Journal of Cryptology*, 11(4):273–287, 1998.
19. D. Sundararajan. *The Discrete Fourier Transform: Theory, Algorithms and Applications*. World Scientific Pub Co., 2001.
20. S. P. Vadhan. *A Study of Statistical Zero-Knowledge Proofs*. PhD thesis, MIT Department of Mathematics, August 1999.
21. D. Watanabe, S. Furuya, H. Yoshida, and B. Preneel. A new keystream generator MUGI. In *Fast Software Encryption*, Lecture Notes in Computer Science. Springer-Verlag, 2002. Description available on-line from `http://www.sdl.hitachi.co.jp/crypto/mugi/index-e.html`.