Cryptanalysis of stream ciphers with linear masking

Don Coppersmith Shai Halevi Charanjit Jutla

IBM T. J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA

{copper,shaih,csjutla@watson.ibm.com}

February 16, 2002

Abstract

We describe a cryptanalytical technique for distinguishing some stream ciphers from a truly random process. The ciphers to which this method applies consist of two processes: one is a "non-linear process" (say, akin to a typical round function in a block ciphers), and the other is a "linear process" such as an LFSR (or even fixed tables). These processes can feed each other, and the output of the cipher can be the linear sum of both processes. This combination seems to be attractive for designing fast stream ciphers. Examples of such ciphers include SEAL and PANAMA, and the newer SNOW, MUGI and Scream.

The idea behind our technique is very simple. For the "non linear process", we look for any property 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 the effectiveness of this approach, when applied to specific "distinguishing properties". One property is a linear approximation of the non-linear process, and the other is a "low-diffusion" attack. We demonstrate these attacks by analyzing a few ciphers. Specifically, we show a linear attack that can distinguish SNOW from a random process after seeing roughly 2^{95} words of output, with work-load of about 2^{100}. We also show a low-diffusion attack on Scream-0, that distinguishes it from a random process after seeing only 2^{43} bytes of output, using 2^{50} space and 2^{80} time. We believe that similar attacks can be devised against PANAMA and MUGI, but we did not try to look at them yet.

Key words: 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). Stream ciphers can be used for shared-key encryption, by using the output stream as a one-time-pad. 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, it 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.

Some examples of ciphers that use this approach include SEAL [6] and Scream [1], 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 [2] and MUGI [8], 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 [3], 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.

Linear attacks. Perhaps the most obvious use of our technique, is to devise linear attacks. This is also the easiest case to analyze. We obtain an exact formula for the amount of text that must be observed in order to distinguish the cipher from random using a linear approximation, as a function of the quality of the original approximation of the non-linear process, and the *weight distribution* of some linear code that is related to the linear process of the cipher.

Low-diffusion attacks. Another type of attacks uses the low diffusion in the non-linear process. Namely, some input and output bits of this process depend only on very few other input and output bits. For this type of attacks, we again analyze the amount of text that an attacker needs to see, as a function of the number of bits in the low-diffusion characteristic. This analysis is substantially 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.

Specific ciphers. We demonstrate the usefulness of our technique by analyzing two specific ciphers. One is the cipher SNOW [3], for which we demonstrate a linear attack, and the other is Scream-0 [1] for which we demonstrate a low-diffusion attack.

In addition to the cryptanalytical technique, we believe that our 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 Preliminaries

We recall some basic definitions and facts about statistical distance between distributions, and about 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 the element x according to the distribution $\mathcal{D}$. For notational convenience, we sometimes denote the same probability mass by $\Pr_{\mathcal{D}}[x]$, or even just $\Pr[x]$, if the distribution $\mathcal{D}$ is clear from the context. Similarly, if $S \subseteq X$ then $\mathcal{D}(S) = \Pr_{\mathcal{D}}[S] = \sum_{x \in S} \mathcal{D}(x)$. By $x \leftarrow \mathcal{D}$, we denote drawing an element $x \in X$ according to $\mathcal{D}$. If $\mathcal{D}$ is a distribution over X and $f : X \to Y$ is some function, then we denote by $f(\mathcal{D})$ the distribution over Y, which is induced by picking $x \leftarrow \mathcal{D}$ and setting $y := f(x)$.

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

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

In our analysis, it would often be convenient to adopt the following interpretation of the statistical distance: Note that we can write

$$|\mathcal{D}_1 - \mathcal{D}_2| = |X| \cdot \sum_x \tfrac{1}{|X|} \cdot |\mathcal{D}_1(x) - \mathcal{D}_2(x)| = |X| \cdot E_x[\ |\mathcal{D}_1(x) - \mathcal{D}_2(x)|\ ]$$

In words, the statistical distance $|\mathcal{D}_1 - \mathcal{D}_2|$ is (a scaling of) the expected value of $|\mathcal{D}_1(x) - \mathcal{D}_2(x)|$, where *$x$ is chosen according to the uniform distribution.* Some other useful facts about this measure are as follows:

1. 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 $|\mathcal{D}_1 - \mathcal{D}_2| = 2\epsilon$, then $2(1 - 2e^{-N\epsilon^2/2}) \leq |\mathcal{D}_1^N - \mathcal{D}_2^N| \leq 2N\epsilon$. It follows that to get $|\mathcal{D}_1^N - \mathcal{D}_2^N| = 1$, the number N needs to be between $\Omega(1/\epsilon)$ and $O(1/\epsilon^2)$.

   In the analysis below, we sometimes make the heuristic assumption that the distributions that we consider are "smooth enough", so that to get the statistical distance up to 1, we need to set $N \approx 1/\epsilon^2$.

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 $|\mathcal{U} - \sum \mathcal{D}_i| \leq \prod_i \epsilon_i$. (For the sake of self-containment, we include a proof of this simple "xor lemma" in Section 2.1 below.) 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 $|\mathcal{U} - \sum \mathcal{D}_i| \prod_i \epsilon_i$.

Hypothesis testing. The following is a very brief overview of (binary) hypothesis testing. This material is covered in many statistics and engineering textbooks (e.g., [5, 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,

$$\text{adv(DR)} = \frac{1}{2}\left(\Pr_{x \leftarrow \mathcal{D}_1}[DR(x) = 1] + \Pr_{x \leftarrow \mathcal{D}_2}[DR(x) = 2]\right) - \frac{1}{2}$$

The following is easy to verify:

Proposition 2 *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,*

$$ML(x) = \begin{cases} 1 & \text{if } \mathcal{D}_1(x) > \mathcal{D}_2(x) \\ 2 & \text{otherwise} \end{cases}$$

The advantage of the ML decision rule equals half the statistical distance, adv(ML) $= \frac{1}{4}|\mathcal{D}_1 - \mathcal{D}_2|$.

2.1 Proof of the xor-lemma for statistical distance

Lemma 3 *Let $\mathcal{D}_1, \mathcal{D}_2$ be two distributions over $\{0,1\}^k$, let $\mathcal{D}_3 = \mathcal{D}_1 + \mathcal{D}_2$, and denote by $\mathcal{U}$ the uniform distribution over $\{0,1\}^k$, and $\epsilon_i = |\mathcal{U} - \mathcal{D}_i|$. Then $\epsilon_3 \le \epsilon_1 \epsilon_2$.*

Proof: For each $r, s \in \{0,1\}^k$, denote $e_r = \mathcal{D}_1(r) - 2^{-k}$ and $f_s = \mathcal{D}_2(s) - 2^{-k}$. By definition of statistical distance, we have $\epsilon_1 = |\mathcal{U} - \mathcal{D}_1| = \frac{1}{2}\sum_r |e_r|$ and similarly $\epsilon_2 = \frac{1}{2}\sum_s |f_s|$. For each $t \in \{0,1\}^k$, we then have

$$\begin{aligned} \mathcal{D}_3(t) &= \sum_{r+s=t} (2^{-k} + e_r)(2^{-k} + f_s) \\ &= 2^k \cdot 2^{-2k} + \sum_{r+s=t} 2^{-k}(e_r + f_s) + \sum_{r+s=t} e_r f_s \;=\; 2^{-k} + \sum_{r+s=t} e_r f_s \end{aligned}$$

(where the last equality holds since $\sum_r e_r = \sum_s f_s = 0$). Therefore we have

$$\begin{aligned} |\mathcal{U} - \mathcal{D}_3| &= \sum_t \left|\mathcal{D}_3(t) - 2^{-k}\right| = \sum_t \left|\sum_{r+s=t} e_r f_s\right| \\ &\le \sum_t \sum_{r+s=t} |e_r f_s| = \sum_{r,s} |e_r f_s| = \left(\sum_r |e_r|\right)\left(\sum_s |f_s|\right) = \epsilon_1 \epsilon_2 \end{aligned}$$

□

Corollary 4 *If $\mathcal{D}_i$, $i = 1...N$ are distributions with $|\mathcal{U} - \mathcal{D}_i| = \epsilon_i$, then $|\mathcal{U} - \sum_i \mathcal{D}_i| \le \prod_i \epsilon_i$.*

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 $LF(y)$. 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 (that can be viewed as cyclic registers with no feedback additions) of size between a few hundred and a few thousand bits.

The operation of the cipher may be as follows: The state of the cipher consists of the "non-linear state" x and the "linear state" y. In each step, some bits of y are added to some bits of x, and some other bits of x are added to some other bits of y. Then, the non-linear function NF is applied to the resulting x and the linear function LF is applied to the resulting y. Finally, some bits of x and y are added together to form the output from this step.

To simplify the presentation of this report, we concentrate on a special case, similar to Scream. (When we describe the attack on SNOW, we show how our techniques can handle other variants.) We view the linear state as $y = (y1, y2, y3)$, where $y1, y2$ are used in the current step, and $y3$ is the rest of the linear state. In each step we do the following:

```
1. Set $x := NF(x + y1) + y2$    // '+' denotes exclusive-or
2. Set $y := LF(y)$
3. Output the new $x$
```

3.1 The linear process

For the linear process, we assume that the initial "linear state" y_0 is chosen uniformly at random, and then it keeps evolving according to the linear function LF. Hence, the string $y_0 y_1 \ldots$ is a random element in some linear subspace of $\{0,1\}^\star$.

Perhaps the most popular linear process is to view the "linear state" y as the contents of an LFSR. The linear modification function LF just clocks the LFSR some fixed number of times (e.g., 32 times). This scheme is used for example in PANAMA, SNOW, and MUGI. Often, the LFSR is defined over some extension field (e.g., $GF(2^{32})$), but it always has an equivalent representation as an LFSR over Z_2. If we denote the LFSR polynomial by p, then the linear subspace of $y_0 y_1 \ldots$ is the subspace orthogonal to $p \cdot Z_2[x]$.

A different approach is taken in Scream. There, the "linear state" y resides in some tables, that are "almost fixed". [1] In particular, in Scream, each entry in these tables is modified (via the non-linear function NF) after 16 times that it is used. To simplify the analysis in this paper, we consider a variation of Scream, in which the tables are modified "in sync". That is, we analyze a process in which all the entries in all the tables are modified at once, and then the tables are used without further modifications until each entry is used exactly 16 times. Then all the entries are modified again, etc.

For our purposes, we model this scheme by assuming that whenever an entry is modified, it is actually being replaced by a new, random and independent value. This assumptions means that we do not try to exploit correlations between the values of the entries before and after the modification. A justification of this assumption, is that "correlations between an entry and itself" are far stronger than correlation between an entry and its modification. Therefore, we concentrate solely on exploiting the fact that each entry is used several times, and we expect that ignoring the

[1] One can view a fixed table as a special case of LFSR, with no feedback additions.

correlation between a mask and its modification will have only a small influence on the effectiveness of the attacks.

The masking scheme in Scream can be thought of as a "two-dimensional" scheme, where there are two tables, each with 16 entries, and these entries are used in lexicographical order. Namely, we have a "row table" $R[\cdot]$ and a "column table" $C[\cdot]$, each with 16 entries, and 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 $i + 16j$ in a batch, we set $(y1, y2) := R[i] + C[j]$.

3.2 Attacks on stream ciphers

The attack model that we consider here, is an attacker that 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 the attack can reliably distinguish the cipher from random, as well as 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.[2]

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 $\langle x_j + y1_j, NF(x_j) + y2_j \rangle$, where the x_j's are chosen uniformly at random and independently of each other, and the y_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 $\langle x, NF(x) \rangle$ 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 5 (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 distribution that we want to distinguish are*

[2]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.

Cipher. *The Cipher distribution is* $\mathcal{D}_c = \langle \ell\,(x_j + y1_j, NF(x_j) + y2_j)\rangle_{j=1,2,\ldots}$*, where the* x_j*'s are random and independent and the* y_j*'s are chosen at random from the appropriate linear subspace (as defined by the linear process of the cipher).*

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

We often refer to the function ℓ *as 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. An obvious lower bound on the amount of text is provided by the statistical distance between the Cipher and Random distributions after N steps. Other relevant parameters of the attack are the time and space complexity of the decision rule.

4 Linear attacks

A linear attack [4] 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 the masked bit $\sigma_j = \ell(x_j + y1_j, NF(x_j) + y2_j)$. Note that σ_j by itself is likely to be unbiased, but the σ's are correlated. In particular, since the y'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 $\sum_{j\in J} y1_j = \sum_{j\in J} y2_j = 0$. Then we have

$$\sum_{j\in J} \sigma_j \;=\; \sum_{j\in J} \ell(x_j, NF(x_j)) + \sum_{j\in J} \ell(y1_j, y2_j) \;=\; \sum_{j\in J} \ell(x_j, NF(x_j))$$

(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.

The rest of this section is organized as follows: We first characterize the effectiveness of linear attacks in terms of the bias ϵ and the *weight distribution* of some linear subspace. Then we apply this characterization to an LFSR-based linear process, and show how it can be used for an attack on SNOW. Finally, we briefly describes how it applies to the "two-dimensional masking" of Scream.

4.1 The statistical distance

Recall that we model an attack in which the attacker observes a single bit per step, namely the bit $\sigma_j = \ell(x_j + y1_j, NF(x_j) + y2_j)$. Below we denote $\tau_j = \ell(x_j, NF(x_j))$ and $\rho_j = \ell(y1_j, y2_j)$. We can re-write the Cipher and Random distributions for this case as

Cipher. $\mathcal{D}_c \overset{\text{def}}{=} \langle \tau_j + \rho_j \rangle_{j=1,2,\ldots}$, 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's).

Random. $\mathcal{D}_r \overset{\text{def}}{=} \langle \sigma_j \rangle_{j=1,2,\ldots}$, 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 $\vec{\chi} \in L^\perp$ of Hamming weight r, and let $A_N(r)$ denote the cardinality of $\mathcal{A}_N(r)$. We prove the following theorem:

Theorem 6 *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: Recall that the statistical distance $|\textsf{Cipher} - \textsf{Random}|$ (for N observed bits) can be expressed in terms of the expected value of $|\Pr_{\textsf{Cipher}}[\vec{\sigma}] - \Pr_{\textsf{Random}}[\vec{\sigma}]|$, where $\vec{\sigma}$ is chosen *uniformly at random* from $\{0,1\}^N$. Fix a string $\vec{\sigma} \in \{0,1\}^N$, and we want to analyze the probability $\Pr_{\textsf{Cipher}}[\vec{\sigma}]$. We can express that probability as

$$\Pr_{\textsf{Cipher}}[\vec{\sigma}] = \sum_{\vec{\rho} \in L} \frac{1}{|L|} \cdot \prod_{j=1}^{N} \left(\frac{1}{2} + \frac{\epsilon}{2} \cdot \textsf{sign}(\rho_i + \sigma_i) \right)$$

where the sign indicator is taken to be $(+1)$ if $\rho_i = \sigma_i$, and (-1) otherwise. In other words, $\textsf{sign}(x) \overset{\text{def}}{=} (-1)^x$. We can break the expression above into a power series in ϵ. In this power series, the constant term is 2^{-N}, and the series looks as follows $\Pr_{\textsf{Cipher}}[\vec{\sigma}] = 2^{-N}\left(1 + \sum_{r=1}^{N} \epsilon^r \textsf{coef}_r\right)$, where the coefficients $\textsf{coef}_r$ are defined as

$$\textsf{coef}_r \overset{\text{def}}{=} \sum_{\vec{\rho} \in L} \frac{1}{|L|} \cdot \sum_{\{j_1 \ldots j_r\}} \prod_{t=1}^{r} \textsf{sign}(\sigma_{j_t} + \rho_{j_t}) \quad = \quad \sum_{\{j_1 \cdots j_r\}} \frac{1}{|L|} \cdot \sum_{\vec{\rho} \in L} \textsf{sign}\left(\sum_{t=1}^{r} \sigma_{j_t} + \rho_{j_t} \right) \qquad (1)$$

The summation over $\{j_1 ... j_r\}$ in the expression above ranges over all ordered sets of cardinality r in $[1, N]$. (In other words, we have $1 \le j_1 < j_2 \cdots < j_r \le N$.) Consider one such r-set $J = \{j_1 ... j_r\}$, and we analyze its contribution to the total sum. Let $\chi(J)$ be the characteristic vector of this set. That is, $\chi(J)$ is an N-bit string, which is 1 in bit positions $\{j_1 ... j_r\}$, and 0 everywhere else.

Proposition 7 *Let $J = \{j_1 ... j_r\}$ be a set of cardinality r. If $\chi(J) \notin L^\perp$, then the total contribution to* $\textsf{coef}_r$ *due to the set J is zero. If $\chi(J) \in L^\perp$ then the total contribution to* $\textsf{coef}_r$ *due to the set J is* $\textsf{sign}\left(\sum_{j \in J} \sigma_j\right)$.

Proof: If $\vec{\chi} = \chi(J)$ is not in $L^\perp$, then for exactly half of the strings $\vec{\rho} \in L$ it holds that $\sum_{j \in J} \rho_j = \langle \vec{\chi}, \vec{\rho} \rangle = 0 \pmod 2$. Thus, for exactly half of the strings $\vec{\rho} \in L$ we have $\textsf{sign}\left(\sum_{t=1}^{r} \sigma_{j_t} + \rho_{j_t}\right) = +1$, and for the other half we have $\textsf{sign}\left(\sum_{t=1}^{r} \sigma_{j_t} + \rho_{j_t}\right) = -1$, so $\sum_{\vec{\rho} \in L} \textsf{sign}\left(\sum_{t=1}^{r} \sigma_{j_t} + \rho_{j_t}\right) = 0$.

If $\chi(J) \in L^\perp$, then for all $\vec{\rho} \in L$ we have $\sum_{t=1}^{r} \rho_{j_t} = 0 \pmod 2$, and therefore $\textsf{sign}\left(\sum_{t=1}^{r} \sigma_{j_t} + \rho_{j_t}\right) = \textsf{sign}\left(\sum_{t=1}^{r} \sigma_{j_t}\right)$. Thus, we get $\frac{1}{|L|} \cdot \sum_{\vec{\rho} \in L} \textsf{sign}\left(\sum_{t=1}^{r} \sigma_{j_t} + \rho_{j_t}\right) = \textsf{sign}\left(\sum_{t=1}^{r} \sigma_{j_t}\right)$. □

We now view the terms in the power series above as random variables. Formally, for any set J with $\chi(J) \in L^\perp$, denote $\xi_J(\vec{\sigma}) \stackrel{\text{def}}{=} \mathsf{sign}\left(\sum_{j\in J}\sigma_j\right)$, and we view the ξ_J'es as random variables, which are defined over the choice of $\vec{\sigma}$ *uniformly at random* in $\{0,1\}^N$. Then, we define the normalized probability difference

$$\Delta(\vec{\sigma}) \stackrel{\text{def}}{=} 2^N \cdot \left(\Pr_{\mathsf{Cipher}}[\vec{\sigma}] - \Pr_{\mathsf{Random}}[\vec{\sigma}]\right) = \sum_{r=1}^{N} \epsilon^r \sum_{\chi(J)\in\mathcal{A}_N(r)} \xi_J(\vec{\sigma})$$

Again, we stress that we view $\Delta(\vec{\sigma})$ as a random variable over the *uniform* choice of $\vec{\sigma} \in \{0,1\}^N$. It is easy to see that for any non-empty J, we have $E[\xi_J] = 0$ and $\text{VAR}[\xi_J] = 1$. Also, if $J_1 \neq J_2$, then ξ_{J_1}, ξ_{J_2} are independent. Therefore, the variable Δ has zero mean, and its variance equals the weighted sum of the ξ_J variances. Namely, $\text{VAR}[\Delta] = \sum_{r=1}^{N} A_N(r)\epsilon^{2r}$

We can now write the statistical distance between the Cipher and Random distributions as

$$|\mathsf{Cipher} - \mathsf{Random}| = \sum_{\vec{\sigma}} \left| \Pr_{\mathsf{Cipher}}[\vec{\sigma}] - \Pr_{\mathsf{Random}}[\vec{\sigma}] \right| = \sum_{\vec{\sigma}} 2^{-N}|\Delta(\vec{\sigma})| = E_{\vec{\sigma}}[|\Delta|]$$

By the convexity of the squaring function, we have $E[|\Delta|] \le \sqrt{\text{VAR}[\Delta]}$, and therefore

$$|\mathsf{Cipher} - \mathsf{Random}| = E_{\vec{\sigma}}[|\Delta|] \le \sqrt{\text{VAR}[\Delta]} = \sqrt{\sum_{r=1}^{N} A_N(r)\epsilon^{2r}} \tag{2}$$

□

Remark. It turns out that this bound is nearly tight. Indeed, since Δ is a sum of many "mostly independent" random variables, it usually "behaves very much like a Gaussian". If it were a Gaussian, we would have $E[|\Delta|] = \sqrt{2\text{VAR}[\Delta]/\pi}$. This estimate suggests that to get the statistical distance up to 1, we need to make N large enough so that $\sum_{r=1}^{N} A_N(r)\epsilon^{2r} = \pi/2$.

4.2 LFSR masking

When the linear process is an LFSR, the orthogonal subspace $L^\perp$ is the space of (bitwise reversal of) polynomials over Z_2 of degree $\le N$, which are divisible by the LFSR polynomial p.

Since the linear space L itself typically has dimension of just a few hundreds, the statistical distance is likely to approach 1 for relatively small N. However, this still does not suggest an efficient procedure for distinguishing the cipher from random. Indeed, it is likely that the maximum-likelihood decision rule for the hypothesis testing problem from above is at least as expensive as an exhaustive search for the key. The reason is that most of the "mass" in the power series in Eq. (2) is likely to come from terms with relatively high weight. (The largest contributions to the sum are likely to come from terms $\epsilon^{2r}A_N(r)$ with $r \approx N/(\log(1/\epsilon)\log\log(1/\epsilon))$, where we expect the growth of $A_N(r)$ to more than compensate for the increase in the exponent ϵ^{2r}.)

However, the analysis does suggest 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 $\vec{\sigma} = \sigma_1 \ldots \sigma_N$, but only consider the W combinations $\vec{\chi} \in L^\perp$ with the smallest (non-zero) weight. For each such combination $\vec{\chi}$, the inner product (i.e., sum of steps) $\langle \vec{\chi}, \vec{\sigma} \rangle$ has bias of $\epsilon^{\text{weight}(\vec{\chi})}$, and these can be used to distinguish the cipher from random. If we

```
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
```

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

take all the combinations with weight upto 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 probably 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[3] so that $\sqrt{A_N(r_0)} \cdot \epsilon^{r_0} = 1$.

4.3 An 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 [3]. 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 $GF(2^{32})$. 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 $\lll$.

According to the paradigm above, 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

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

is biased. (Of these six bits, the bits $(f_j)_{15}, (F_j)_{15}$ and $(F_{j+1})_{22}$ are meant to approximate carry

[3]The minimum-distance r_0 may decrease as N increases, but for LFSR codes, which are truncated cyclic codes, we expect r_0 to be essentially fixed (after a short initial period).

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 8 *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: Since $\alpha \in GF(2^{32})$, then the polynomial $t+\alpha$ divides $t^{2^{32}}+t$. That is, there is a polynomial $r(\cdot)$ (with coefficients in $GF(2^{32})$) such that $r(t)\cdot(t+\alpha) = t^{2^{32}}+t$, as formal polynomials over $GF(2^{32})$. It follows that for any polynomial $t(z)$ over $GF(2^{32})$, we have $r(t(z))\cdot(t(z)+\alpha) = t(z)^{2^{32}}+t(z)$, again, as formal polynomials over $GF(2^{32})$. Specifically, if we take $t(z) = z^{16}+z^{13}+z^7$, we get

$$r(t(z))\cdot(z^{16}+z^{13}+z^7+\alpha) \;=\; z^{16\times 2^{32}} + z^{13\times 2^{32}} + z^{7\times 2^{32}} + z^{16} + z^{13} + z^7$$

so the polynomial on the right hand side is divisible by $p(z)$. Since $p(z)$ is co-prime with the polynomial z, we can divide the right-hand-side polynomial by z^7 and still get a polynomial divisible by $p(z)$. $\square$

Corollary 9 *For all m, n, the polynomial*

$$\begin{aligned} q_{m,n}(z) \overset{\text{def}}{=} q(z)^{2^m}\cdot z^n \;=\; & z^{16\times 2^{32+m}-7\times 2^m+n} \;+\; z^{13\times 2^{32+m}-7\times 2^m+n} \;+\; z^{7\times 2^{32+m}-7\times 2^m+n} \\ & +\; z^{9\times 2^m+n} \;+\; z^{6\times 2^m+n} \;+\; z^n \end{aligned}$$

is divisible by $p(z)$.

If we take, say, $m = 0, 1, \dots 58$ and $n = 0, 1, \dots 2^{94}$, 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, such that the sum of the $L_j[\cdot]$ values in these steps vanish. Specifically, the polynomial $q_{m,n}(z)$ corresponds to the sequence of steps

$$\begin{aligned} J_{m,n} \;=\; \{ \;\; & N-n-16\cdot 2^{32+m}+7\cdot 2^m, \quad N-n-9\cdot 2^m, \\ & N-n-13\cdot 2^{32+m}+7\cdot 2^m, \quad N-n-6\cdot 2^m, \\ & N-n-\;\,7\cdot 2^{32+m}+7\cdot 2^m, \quad N-n \qquad\qquad \} \end{aligned}$$

with the property that for all m, n, $\sum_{j\in J_{m,n}} L_j[0..15] \;=\; \sum_{j\in J_{m,n}} L_{j+1}[0..15] \;=\; [0, 0, \dots, 0]$. Therefore, if we denote the output word of SNOW at step j by S_j, then for all m, n we have,

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

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.

4.4 Two-dimensional masking

For two-dimensional masking (with k rows and k columns), since we model each "batch" of k^2 steps as independent, it is enough to analyze one such batch. For one such batch, the linear subspace L is defined by picking $2k$ random values $R[0..k-1]$ and $C[0..k-1]$, and then for $0 \le i, j < k$, setting $y_{i+n\times j} = R[i] + C[j]$. If we have a set of steps $J = \{i_1 + kj_1, \ldots i_r + kj_r\}$, then $\chi(J)$ belongs to the orthogonal space $L^\perp$ iff each index i (resp. j) appears even number of times as the first (resp. second) coordinate in J. Formally, for every $0 \le m < k$, the sets $\{t \le r : i_t = m\}$ and $\{t \le r : j_t = m\}$ have even cardinality. A set of steps J is called an *even set* if it satisfies the above condition.

It is easy to see that even sets must have even cardinality of at least 4, there are exactly $\binom{k}{2}^2$ even sets of cardinality 4, and there are less than $(2k+r)^{2r}$ even sets of cardinality $2r$ for $3 \le r \le k^2/2$. In terms of Eq. (2), we have $N = k^2$, $A_N(4) = \binom{k}{2}^2$, $A_N(2r) < (2k+r)^{2r}$ for $3 \le r \le k^2/2$, and $A_N(r) = 0$ for any other r. It follows that when $\epsilon \ll 1/k$, the dominant term in Eq. (2) is $\epsilon^8 A_N(4) = \epsilon^8 \binom{k}{2}^2$. Thus, for each batch of k^2 steps we have statistical distance of roughly $\epsilon^4 k^2/2$, so we need to observe about $4\epsilon^{-8}/k^4$ such batches (or a total of $4\epsilon^{-8}/k^2$ steps) to reliably distinguish the cipher from random.

Indeed, this lower bound matches the following obvious attack: for each batch of k^2 steps, we consider sets of steps of the form $J = \{i + kj,\ i' + kj,\ i + kj',\ i' + kj'\}$ (aka even sets of cardinality 4). For each set we add the four σ's, to get a bit with bias ϵ^4. This way, each batch gives us $\binom{k}{2}^2$ bits with bias ϵ^4, so after seeing $\epsilon^{-8}/\binom{k}{2}^2$ batches (i.e., total of $4\epsilon^{-8}/k^2$ steps) we have enough bits to distinguish them from random.

5 Low-diffusion attacks

In low-diffusion attacks, the attacker tries to find 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. Then, the attacker uses some guessing strategy 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, we have $\ell_{\text{in}} : \{0,1\}^n \to \{0,1\}^m$, $\ell_{\text{out}} : \{0,1\}^n \to \{0,1\}^{m'}$, and $f : \{0,1\}^m \to \{0,1\}^{m'}$.

In each step j, the attacker observes the bits $\ell_{\text{in}}(x_j + y1_j)$ and $\ell_{\text{out}}(NF(x_j) + y2_j)$. Below we denote $u_j = \ell_{\text{in}}(x_j)$, $u'_j = \ell_{\text{out}}(NF(x_j))$, $v_j = \ell_{\text{in}}(y1_j)$, $u'_j = \ell_{\text{out}}(y2_j)$, and $w_j = u_j + v_j$, $w'_j - u'_j + v'_j$. We can re-write the Cipher and Random distributions for this case as

```
1. for $i = 0$ to 15 do
2.     $x := NF(x + c1) + c2$
3.     output $x + R[i]$
4.     if $i$ is even, rotate $c1$ by 64 bits
5.     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 $NF(\cdot)$
```

Figure 2: sixteen steps of Scream-0.

Cipher. $\mathcal{D}_c \overset{\text{def}}{=} \left\langle (w_j = u_j + v_j,\ w'_j = u'_j + v'_j) \right\rangle_{j=1,2,\ldots}$, 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's).

Random. $\mathcal{D}_r \overset{\text{def}}{=} \left\langle (w_j, w'_j) \right\rangle_{j=1,2,\ldots}$, where the u_j's and u'_j's are 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 d, and the attacker observes N steps such that $m'N > d$. Then, the attacker can (in principle) go over all the 2^d 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. Then, the attacker can verify the guess by checking that indeed $u'_j = f(u_j)$ for all j. This way, the attacker guesses d bits and gets $m'N$ bits of consistency checks. Since $m'N > d$ 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.

5.1 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 [1]. 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.

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 will be described in much more details in the full version of [1].

As usual, we need to find a "distinguishing characteristic" of the non-linear function (in this case, a low-diffusion characteristic as above), 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, e.g., $j, j+1, j+16, j+17$ for some even j (in general, we can observe steps $j, j+1, j+16k, j+1+16k$ for some $k < 16$). 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 $f_i : \{0,1\}^{48} \to \{0,1\}^{32}$. (The reason that we have two different functions is that the order of the input bytes is different between the even and odd steps.) It remains to estimate the statistical distance between $\mathcal{D}$ and the uniform distribution $\mathcal{R}$, and to show how to implement the decision rule to distinguish between them.

In Section 6 we analyze several "types" of distributions such as $\mathcal{D}$, and also explain how to efficiently implement the maximum-likelihood decision rule for distinguish $\mathcal{D}$ from $\mathcal{R}$. Plugging the values $m = 48, m' = 32$ into the general bounds from Section 6 (using the special form of the functions f_i, as discussed in Section 6.1), we estimate the statistical distance $|\mathcal{D} - \mathcal{R}|$ to be as high as $2^{-20.5}$. We therefore need about 2^{41} samples to reliably distinguish $\mathcal{D}$ from random. Roughly speaking, we can get $8 \cdot \binom{15}{2} \approx 2^{10}$ 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 15 in which both the 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 6.3 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 into the general techniques that are described there, we get space complexity about 2^{50}, and time complexity about 2^{80}.

6 Analysis of low-diffusion attacks

Below we analyze in details the simplest case of "low-diffusion" attack, where we use the same functions f in all the steps, and we need to add up four steps in order to cancel the masks. Later we explain how this analysis can be extended for other settings, an in particular for the case of the functions in Scream. For a given function, $f : \{0,1\}^m \to \{0,1\}^{m'}$, we denote

$$\mathcal{D}^f \stackrel{\text{def}}{=} \left\langle d = \sum_{j=1}^{4} u_j,\ d' = \sum_{j=1}^{4} f(u_j) \right\rangle$$

where the u_j's are uniform in $\{0,1\}^m$ and independent. In the simple case that we analyze here, the attacker knows f, and it sees many instances of $\langle d, d \rangle$. The attacker needs to decide these instances come from $\mathcal{D}^f$ 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:

Theorem 10 *For a uniformly selected function $f : \{0,1\}^m \to \{0,1\}^{m'}$, we have*

$$E_f[|\mathcal{D}^f - \mathcal{R}| \le \sqrt{96} \cdot (1 + o(1)) \cdot 2^{(m'-3m)/2}$$

Proof: For any value $d \in \{0,1\}^m$, denote by $\mathcal{D}^f_d$ the distribution which is induced by picking at random the u_j's, subject to $\sum u_j = d$, and computing $d' = \sum f(u_j)$. In other words, the distribution $\mathcal{D}^f_d$ is induced by picking uniformly at random three strings $x, y, z \in \{0,1\}^m$, and computing $d' = f(x) + f(y) + f(z) + f(d + x + y + z)$. Also, denote by $\mathcal{U}$ the uniform distribution over $\{0,1\}^{m'}$. Since the "d" part of both the $\mathcal{D}^f$ and $\mathcal{R}$ distributions is uniform over $\{0,1\}^m$, we can write the distance between them as the expected value, over a random choice of $d \in \{0,1\}^m$, of $|\mathcal{D}^f_d - \mathcal{U}|$. Namely, for any function f we have

$$\begin{aligned} |\mathcal{D}^f - \mathcal{R}| &= \sum_d 2^{-m} |\mathcal{D}^f_d - \mathcal{U}| \\ &= \sum_d 2^{-m} \sum_{d'} \left| \Pr_{x,y,z}[f(x) + f(y) + f(z) + f(x + y + z + d) = d'] - 2^{-m'} \right| \end{aligned}$$

In the analysis below, we view the quantity $\Pr_{x,y,z}[\cdots]$ as a random variable, defined over the choice of $d \in \{0,1\}^m$ and the choice of $f : \{0,1\}^m \to \{0,1\}^{m'}$. Formally, this is done as follows: For fixed $d' \in \{0,1\}^{m'}$ and $x, y, z \in \{0,1\}^m$, we define the random variables $\chi^{d'}_{x,y,z}$ and $S^{d'}$, over the choices of f and of d:

$$\begin{aligned} \chi^{d'}_{x,y,z}(f, d) &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } f(x) + f(y) + f(z) + f(x + y + z + d) = d' \\ 0 & \text{otherwise} \end{cases} \\ S^{d'}(f, d) &\stackrel{\text{def}}{=} 2^{-3m} \sum_{x,y,z} \chi^{d'}_{x,y,z}(f, d) \end{aligned}$$

Then, for any f, d, d', we have $S^{d'}(f, d) = \Pr_{x,y,z}[f(x) + f(y) + f(z) + f(x + y + z + d) = d']$. Using these notations, we can write the distance $|\mathcal{D}^f - \mathcal{R}|$ (for a fixed function f) as

$$E_d\left[|\mathcal{U} - \mathcal{D}^f_d| \right] = E_d\left[\sum_{d'} \left| S^{d'}(f, d) - 2^{-m'} \right| \right] = \sum_{d'} E_d\left[\left| S^{d'}(f, d) - 2^{-m'} \right| \right] \tag{3}$$

For a random function f, we can bound the expected value of Eq. (3) as follows:

$$E_f[|\mathcal{D}^f - \mathcal{R}|] = \sum_{d'} E_{f,d}\left[\left| S^{d'} - 2^{-m'} \right| \right]$$

$$\begin{aligned}
&\overset{(a)}{\leq} \sum_{d'} E_{f,d}\left[\left|S^{d'} - E_{f,d}[S^{d'}]\right|\right] \;+\; \sum_{d'} \left|E_{f,d}[S^{d'}] - 2^{-m'}\right| \\
&\overset{(b)}{\leq} \sum_{d'} \sqrt{\mathrm{VAR}_{f,d}\,[S^{d'}]} \;+\; \sum_{d'} \left|E_{f,d}[S^{d'}] - 2^{-m'}\right| \qquad (4)
\end{aligned}$$

where (a) is just the triangle inequality, and (b) follows from the convexity of the squaring function. To bound the last expression, we use the following proposition (whose proof is given later in Section 6.2):

Proposition 11 *For a random function $f : \{0,1\}^m \to \{0,1\}^{m'}$, and a random $d \in \{0,1\}^m$,*

$$\left|E_{f,d}\left[S^{d'}\right] - 2^{-m'}\right| \;<\; \begin{cases} 3 \cdot 2^{-2m-m'} & \text{for } d' \neq 0 \\ 3 \cdot 2^{-2m} & \text{for } d' = 0 \end{cases}$$

$$VAR_{f,d}[S^{d'}] \;<\; \begin{cases} 96 \cdot 2^{-3m-m'} & \text{for } d' \neq 0 \\ 9 \cdot 2^{-3m}(1 + 12\tfrac{2}{3} \cdot 2^{-m'} + 3 \cdot 2^{-2m}) & \text{for } d' = 0 \end{cases}$$

Let us now denote $3 + \epsilon = \sqrt{9\left(1 + 12\frac{2}{3} \cdot 2^{-m'} + 3 \cdot 2^{-2m}\right)}$. We plug the bounds from Proposion 11 into Eq. (4), and get:

$$\begin{aligned}
E_f[|\mathcal{D}^f - \mathcal{R}|] &< \left((3+\epsilon) \cdot 2^{-3m/2} + 2^{m'} \cdot \sqrt{96} \cdot 2^{(-3m-m')/2}\right) + \left(3 \cdot 2^{-2m} + 2^{m'} \cdot 3 \cdot 2^{-2m-m'}\right) \\
&= (3+\epsilon) \cdot 2^{-3m/2} + \sqrt{96} \cdot 2^{(m'-3m)/2} + 6 \cdot 2^{-2m} \\
&= \sqrt{96} \cdot 2^{(m'-3m)/2}\left(1 + \tfrac{3+\epsilon}{\sqrt{96}} \cdot 2^{-m'/2} + \tfrac{6}{\sqrt{96}} \cdot 2^{-(m'+m)/2}\right) \\
&= \sqrt{96} \cdot (1 + o(1)) \cdot 2^{(m'-3m)/2}
\end{aligned}$$

That completes the proof of Theorem 10. □

How tight is this bound? It turns out that the only inequality in the proof that is not asymptotically tight, is Inequality (b) from Eq. (4). Making the heuristic assumption that the $S^{d'}$'s "behave like Gaussian random variables", we expect the ratio between $E[|S^{d'} - E[S^{d'}]|]$ and $\sqrt{\mathrm{VAR}[S^{d'}]}$ to be roughly $\sqrt{2/\pi}$. Therefore, we expect that the constant $\sqrt{96} \approx 9.8$ be replaced by $\sqrt{192/\pi} \approx 7.8$. Indeed we ran some experiments to measure the statistical distance $|\mathcal{D}^f - \mathcal{R}|$, for random function with a few values of m, m'. These experiments are described in Appendix A. The results confirm that the distance between these distributions is just under $7.8 \cdot 2^{(m'-3m)/2}$.

6.1 Variations and extensions

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

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, 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 for a random $d \in \{0,1\}^m$ (which we view as a triple, $d = d_1 d_2 d_3$, with $|d_1| = |d_2| = |d_3| = m/3$) we expect each of the three distributions $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$ below to be $c \cdot 2^{(m'-m)/2}$ away from the uniform distribution (with $c \approx \sqrt{192/\pi}$). The distributions $\mathcal{D}_b$ ($b = 1,2,3$) are defined

$$\mathcal{D}_b = \Big\{ f_b(x) + f_b(y) + f_b(z) + f_b(x+y+z+d_b) \ : \ x,y,z \text{ are random in } \{0,1\}^{m/3} \Big\}$$

The distribution $\mathcal{D}_f^d$ that we want to analyze, can be expressed as $\mathcal{D}_f^d = \mathcal{D}_1 + \mathcal{D}_2 + \mathcal{D}_3$, so we expect to get $|\mathcal{D}^f - \mathcal{R}| = E_d[|\mathcal{D}_f^d - \mathcal{U}|] \approx \prod_b |\mathcal{U} - \mathcal{D}_b| \approx \left(c \cdot 2^{(m'-m)/2}\right)^3 = c^3 2^{(3m'-3m)/2}$.

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 calculations from above, we get that the expected distance $|\mathcal{D}^f - \mathcal{R}|$ is about $c^r 2^{(rm'-3m)/2}$ (assuming that this is still smaller than one).

Using different f's for different steps. Instead of using the same f everywhere, we can have different f's for different steps. I.e., in step j we have $\ell_{\text{out}}(NF(x_j)) = f_j(\ell_{\text{in}}(x_j))$, and we make the assumptions that the f_j's are random and independent. The analysis from above still works for the most part, as long as $\ell_{\text{in}}, \ell_{\text{out}}$ do not change. The main difference is that the factor $c = \sqrt{96}$ is replaced by a smaller one. If we use four independent functions, then we get $c = 1$, since all the symmetries in the proof of Proposion 11 disappear.

In the case of the attack on Scream, where we use two functions, $f_1 = f_3$ and $f_2 = f_4$, we get a constant of $c = \sqrt{8}$. Again, when we factor in the heuristic argument that the S^d's "behaving like Gaussians", we can reduce this factor from $c = \sqrt{8} \approx 2.83$ to $c = \sqrt{16/\pi} \approx 2.26$. Indeed, we ran some experiments for this case (similar to the ones reported in Appendix A), and they confirm this factor of 2.26.

When each of the f_i's is the sum of r functions, we can repeat the argument from above, and conclude that the expected statistical distance is $c^r 2^{(rm'-3m)/2}$, this time with the smaller constant c. For the case of Scream, we have $c = \sqrt{16/\pi}$, $r = 3$ and $m = 48, m' = 32$, so we expect the statistical distance $|\mathcal{D} - \mathcal{R}|$ to be roughly $(16/\pi)^{3/2} \cdot 2^{-24} \approx 2^{-20.5}$.

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 $q > 4$, we get a constant of $c = \sqrt{6}$ (instead of $\sqrt{96}$), since the only symmetry that is left is between $\{x,y,z\}$ and its other five orderings. When we work modulo a power of two, the constant will be somewhere between $\sqrt{6}$ and $\sqrt{96}$, probably closer to the former.

6.2 Proof of Proposion 11

We use the following notation: For a multi-set M, denote by $[M]_2$ the set of values that appear *odd number of times* in M. We observe the following:

- Since f is a random function, then for any x,y,z,d such that $[x,\ y,\ z,\ x+y+z+d]_2 \neq \emptyset$, we have $\Pr_f[\chi^{d'}_{x,y,z}(f,d) = 1] = 2^{-m'}$.

  If $[x,\ y,\ z,\ x+y+z+d]_2 = \emptyset$, then for that d, $\chi^{d'}_{x,y,z}(f,d)$ is either identically one (if $d' = 0$) or identically zero (if $d' \neq 0$). Notice that to get $[x,\ y,\ z,\ x+y+z+d]_2 = \emptyset$, we must have $d = 0$ and x,y,z that are not all distinct.

- Similarly, if for some fixed x, y, z, x', y', z', d we have $[x,\ y,\ z,\ x+y+z+d]_2 \neq [x',\ y',\ z',\ x'+y'+z'+d]_2$, then the variables $\chi^{d'}_{x,y,z}, \chi^{d'}_{x',y',z'}$ are independent (conditioned on d). If $[x,\ y,\ z,\ x+y+z+d]_2 = [x',\ y',\ z',\ x'+y'+z'+d]_2$, then the variables $\chi^{d'}_{x,y,z}, \chi^{d'}_{x',y',z'}$ are identical (again, conditioned on d).

Using the first observation, we can calculate the expected values of the χ variables.

$$E_{f,d}\left[\chi^{d'}_{x,y,z}\right] = \begin{cases} 2^{-m'} & \text{if } x,y,z \text{ are distinct} \\ 2^{-m'}(1-2^{-m}) & \text{if } x,y,z \text{ are not distinct, and } d' \neq 0 \\ 2^{-m'}(1-2^{-m}) + 2^{-m} & \text{if } x,y,z \text{ are not distinct, and } d' = 0 \end{cases}$$

The number of choices where x, y, z are not distinct is $3 \cdot 2^{2m} - 2 \cdot 2^m$. We can therefore calculate the expected value of $S^{d'}$:

$$\begin{aligned}
\text{For } d' \neq 0,\ E_{f,d}\left[S^{d'}\right] &= (1 - 3\cdot 2^{-m} + 2^{-2m}) \cdot 2^{-m'} + \ (3\cdot 2^{-m} - 2^{-2m}) \cdot 2^{-m'}(1-2^{-m}) \\
&= 2^{-m'}(1 - 3\cdot 2^{-2m} + 2\cdot 2^{-3m}) \\
\text{For } d' = 0,\ E_{f,d}\left[S^{d'}\right] &= (1 - 3\cdot 2^{-m} + 2^{-2m}) \cdot 2^{-m'} + \ (3\cdot 2^{-m} - 2^{-2m}) \cdot (2^{-m'}(1-2^{-m}) + 2^{-m}) \\
&= 2^{-m'} + (3\cdot 2^{-2m} - 2\cdot 2^{-3m})(1-2^{-m'})
\end{aligned}$$

In particular, we get that

$$\left| E\left[S^{d'}\right] - 2^{-m'} \right| \ < \ \begin{cases} 3\cdot 2^{-2m-m'} & \text{for } d' \neq 0 \\ 3 \cdot 2^{-2m} & \text{for } d' = 0 \end{cases}$$

Next we analyze the variance of $S^{d'}$. Recall that the variance of a sum is the sum of all the covariances of the summands, namely

$$\mathrm{VAR}_{f,d}[S^{d'}] = 2^{-6m} \sum_{x,y,z} \ \sum_{x',y',z'} \mathrm{COV}_{f,d}[\chi^{d'}_{x,y,z},\ \chi^{d'}_{x',y',z'}]$$

where $\mathrm{COV}[\chi, \chi'] = E[\chi \cdot \chi'] - E[\chi]E[\chi']$. (We allow $\langle x, y, z\rangle = \langle x', y', z'\rangle$ in the summation above, in which case $\mathrm{COV}[\chi^{d'}_{x,y,z},\ \chi^{d'}_{x',y',z'}] = \mathrm{VAR}[\chi^{d'}_{x,y,z}]$.) Fix two specific variables, $\chi = \chi^{d'}_{x,y,z}, \chi' = \chi^{d'}_{x',y',z'}$, and we examine their covariance. There are three main cases to consider:

1. The first case is where x, y, z are distinct, and x', y', z' are also distinct. It is easy to see that if $\{x, y, z\}$ and $\{x', y', z'\}$ are disjoint, or if they have just a single element in common, then the variables $\chi^{d'}_{x,y,z},\ \chi^{d'}_{x',y',z'}$ are independent, and therefore $\mathrm{COV}[\chi^{d'}_{x,y,z},\ \chi^{d'}_{x',y',z'}] = 0$. This leave two sub-cases to consider:

   (a) If the sets $\{x, y, z\}, \{x', y', z'\}$ have two elements in common, there are exactly two values of d for which $[x,\ y,\ z,\ x+y+z+d]_2 = [x',\ y',\ z',\ x'+y'+z'+d]_2$. (Assume, for example, that $x = x', y = y'$, but $z \neq z'$. In this case, these values are $d = x + y$ and $d = x + y + z + z'$.) Therefore, we have $E[\chi\chi'] = 2^{1-m} \cdot 2^{-m'} + (1 - 2^{1-m}) \cdot 2^{-2m'}$, and so $\mathrm{COV}[\chi, \chi'] = 2^{1-m}(2^{-m'} - 2^{-2m'}) < 2 \cdot 2^{-m-m'}$.
   The number of pairs in this category can be bounded by $18 \cdot 2^{4m}$: We have less than 2^{3m} ways of choosing disjoint x, y, z, then $\binom{3}{2} = 3$ ways of choosing the two elements in the intersection with $\{x', y', z'\}$, then less than 2^m ways of choosing the last element in $\{x', y', z'\}$, and finally six ways of ordering $\{x', y', z'\}$.

(b) When $\{x,y,z\} = \{x',y',z'\}$, the variables $\chi^{d'}_{x,y,z}$, $\chi^{d'}_{x',y',z'}$ are identical, and therefore $\mathrm{COV}[\chi^{d'}_{x,y,z},\ \chi^{d'}_{x',y',z'}] = \mathrm{VAR}[\chi^{d'}_{x,y,z}] = 2^{-m'} - 2^{-2m'} < 2^{-m'}$.

There are $6^2 \cdot \binom{2^m}{3} < 6 \cdot 2^{3m}$ pairs $\chi^{d'}_{x,y,z}, \chi^{d'}_{x',y',z'}$ in this category.

2. The second case is where neither x,y,z nor x',y',z' are distinct. Here too there are two sub-cases to consider:

   (a) When $[x,y,z]_2 \neq [x',y',z']_2$, there are exactly two d's for which $[x,\ y,\ z,\ x+y+z+d]_2 = [x',\ y',\ z',\ x'+y'+z'+d]_2$. One value is $d=0$, for which both sets are empty, and another value for which they are equal but non-empty. Omitting some (easy) intermediate calculations, the covariance can be bounded by

$$\mathrm{COV}[\chi^{d'}_{x,y,z},\ \chi^{d'}_{x',y',z'}] < \begin{cases} 2^{-m-m'} & \text{if } d' \neq 0 \\ 2^{-m}(1+2^{-m'}) & \text{if } d' = 0 \end{cases}$$

   The number of pairs in this category is bounded by $(3 \cdot 2^{2m} - 2 \cdot 2^m)^2 < 9 \cdot 2^{4m}$.

   (b) When $[x,y,z]_2 = [x',y',z']_2$, the variables $\chi^{d'}_{x,y,z}$, $\chi^{d'}_{x',y',z'}$ are identical, and therefore

$$\mathrm{COV}[\chi^{d'}_{x,y,z},\ \chi^{d'}_{x',y',z'}] = \mathrm{VAR}[\chi^{d'}_{x,y,z}] < \begin{cases} 2^{-m'} & \text{if } d' \neq 0 \\ 2^{-m'} + 3 \cdot 2^{-2m} & \text{if } d' = 0 \end{cases}$$

   The number of pairs in this category is bounded by $9 \cdot 2^{3m}$: We have less than $3 \cdot 2^{2m}$ choices for x,y,z. Since x,y,z are not all distinct, then $[\{x,y,z\}]_2$ contains a single element, and this element must be also in $\{x',y',z'\}$. We then have 2^m choices for another element that appears twice in $\{x',y',z'\}$, and then at most 3 ways of ordering the resulting multi-set.

3. The third case is where x,y,z are distinct, but x',y',z' are not, or vice versa. (In the description below we assume that x,y,z are distinct and x',y',z' are not.) If $[x',y',z']_2 \not\subseteq \{x,y,z\}$ then the variables $\chi^{d'}_{x,y,z}$, $\chi^{d'}_{x',y',z'}$ are independent, and their covariance is zero.

   When $[x',y',z']_2 \subseteq \{x,y,z\}$ then, just as in case 1a above, there are two values of d for which $[x',y',z',x'+y'+z'+d]_2 = [x,y,z,x+y+z+d]_2$. (Assume, for example, that $x = x'$ and $y' = z'$, then these values are $d = x+y$ and $d = x+z$). Another "special value" is $d = 0$, for which $[x',y',z',x'+y'+z'+d]_2 = \emptyset$. For all other values of d we have $[x',y',z',x'+y'+z'+d]_2 \neq [x,y,z,x+y+z+d]_2$, and both are non-zero.

   Therefore, for $d' \neq 0$ we have $E[\chi\chi'] = 2^{1-m} \cdot 2^{-m'} + (1 - 3 \cdot 2^{-m}) \cdot 2^{-2m'}$, and for $d' = 0$ we have $E[\chi\chi'] = (3 \cdot 2^{-m}) \cdot 2^{-m'} + (1 - 3 \cdot 2^{-m}) \cdot 2^{-2m'}$. Omitting some more (easy) calculations, we get

$$\mathrm{COV}[\chi^{d'}_{x,y,z},\ \chi^{d'}_{x',y',z'}] < \begin{cases} 2 \cdot 2^{-m-m'} & \text{if } d' \neq 0 \\ 3 \cdot 2^{-m-m'} & \text{if } d' = 0 \end{cases}$$

   There are at most $18 \cdot 2^{4m}$ pairs in this category: When $\{x,y,z\}$ are distinct, we have less than 2^{3m} ways to choose $\{x,y,z\}$, then 3 possibilities of choosing the single element in $[x',y',z']_2$ from the elements $\{x,y,z\}$, then 2^m possibilities for the element that appears twice in $\{x',y',z'\}$, and then at most 3 ways to order the resulting multi-set. This gives at most $9 \cdot 2^{4m}$ pairs with $\{x,y,z\}$ distinct, and $9 \cdot 2^{4m}$ pairs with $\{x',y',z'\}$ distinct.

Summing up the contributions from all the cases above, we get

case	# of pairs	contribution $d' = 0$	contribution $d' \neq 0$
1a.	$< 18 \cdot 2^{4m}$	$2 \cdot 2^{-m-m'}$	$2 \cdot 2^{-m-m'}$
1b.	$< 6 \cdot 2^{3m}$	$2^{-m'}$	$2^{-m'}$
2a.	$< 9 \cdot 2^{4m}$	$2^{-m}(1 + 2^{-m'})$	$2^{-m-m'}$
2b.	$< 9 \cdot 2^{3m}$	$2^{-m'} + 3 \cdot 2^{-2m}$	$2^{-m'}$
3.	$< 18 \cdot 2^{4m}$	$3 \cdot 2^{-m-m'}$	$2 \cdot 2^{-m-m'}$
total:	2^{-6m} ·	$9 \cdot 2^{3m}(1 + 12\frac{2}{3} \cdot 2^{-m'} + 3 \cdot 2^{-2m})$	$96 \cdot 2^{3m-m'}$

This complete the proof. □

6.3 Efficiency considerations

The analysis from above says nothing about the computational cost of distinguishing between $\mathcal{D}^f$ 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 $3m - m'$ 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 $\mathcal{D}^f$ (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 $J = \{j_1, j_2, j_3, j_4\}$, 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 $\mathcal{D}^f$ or from $\mathcal{R}$. After observing roughly $2^{3m-m'-6}$ such steps 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 in $2^{m+m'}$.

As for the off-line part (in which the table is computed), the naive way is to go over all possible values of $u_1, u_2, u_3, u_4 \in \{0,1\}^m$, 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^{4m} time. However, in the (typical) case where $m' \ll 3m$, one can use a much better strategy, whose running time is only $O((m + m')2^{m+m'})$.

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,[4]

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

One can use the Walsh-Hadamard transform to perform this step in time $O((m + m')2^{m+m'})$ (see, e.g., [7]). Then, we again use the Walsh-Hadamard transform to compute the convolution of E

[4]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 indices are $m + m'$-bits long.

with itself,

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

When f is a sum of functions. We can get additional flexibility when f is a sum of functions on disjoint arguments, $f(x) = f_1(x_1) + ... + f_r(x_r)$ (with $x = x_1..x_r$). 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 $O((m'+(m/r))2^{m'+(m/r)})$. 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, $\Delta_i[d_i,\cdot] \overset{\text{def}}{=} \mathcal{H}(D_i[d_i,\cdot])$. 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 $O\left(r(m'+(m/r)2^{m'+(m/r)}) + (r2^{m/r}+2^m)m'2^{m'}\right)$ $= O\left((rm'+m)2^{m'+m/r} + m'2^{m+m'}\right)$, 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 $O((rm'+m)2^{m'+m/r})$ 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 $O((rm'+m)2^{m'+m/r} + Sm'2^{m'})$, where S is the number of samples needed to distinguish $\mathcal{D}$ from $\mathcal{R}$.

In the case of the attack on Scream, we have $m = 48$, $m' = 32$, $r = 3$, and $S = 2^{41}$, so we get space complexity of $3 \times 2^{32+48/3} \approx 2^{50}$, and time complexity of roughly $(3 \times 32 + 48) \times 2^{32+48/3} + 2^{41} \times 32 \times 2^{32} \approx 2^{78}$.

7 Conclusions

In this work we described a general cryptanalytical technique, which 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 [3] and Scream-0 [1].

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] 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.

[2] 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.

[3] P. Ekdahl and T. Johansson. SNOW – a new stream cipher. Submitted to NESSIE. Available on-line from `http://www.it.lth.se/cryptology/snow/`.

[4] 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.

[5] R. N. McDonough and A. D. Whalen. *Detection of Signals in Noise.* Academic Press, Inc., 2nd edition, 1995.

[6] P. Rogaway and D. Coppersmith. A software optimized encryption algorithm. *Journal of Cryptology*, 11(4):273–287, 1998.

[7] D. Sundararajan. *The Discrete Fourier Transform: Theory, Algorithms and Applications.* World Scientific Pub Co., 2001.

[8] 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`.

A Experimental results

We tested our analysis from Section 6, by choosing a few random functions $f : \{0,1\}^m \rightarrow \{0,1\}^{m'}$ (for several settings of m, m'), and evaluating the distance $|\mathcal{D}^f - \mathcal{R}|$. For each function f, we used the techniques from Section 6.3 (based on the Welsh-Hadamard transform) to compute the statistical distance. In this computation, we used the SPIRAL implementation of the Welsh-Hadamard transform, due to Markus Pueschel, Bryan Singer, and Adrian Sox (see `http://www.ece.cmu.edu/~spiral`).

For each setting of m, m', we chose sixteen random functions, and computed the average distance for these functions. For each setting we also computed the average variance of S^0, and the average variance of the $S^{d'}$'s for $d' \neq 0$. The results are presented below. One can see that the only deviation from the expected values in our analysis, is in the cases where m is significantly smaller than m'.

In these cases, the distance is less than what we expect from the analysis. We speculate that the reason for this deviation, is that for such settings the $S^{d'}$'s are "not as smooth", and therefore, there is a larger gap between the quantities $E[|S^{d'} - E[S^{d'}]|]$ and $\sqrt{\mathrm{VAR}[S^{d'}]}$.

```
Testing 16 random functions, m=6, m'=6:
  average statistical distance is 1.174e-01 = 7.514 * 2^{(m'-3m)/2}
  average variance (d'=0) is 3.731e-05      = 9.780 * 2^{-3m}
  average variance (d'!=0) is 5.201e-06     =87.259 * 2^{-m'-3m}

Testing 16 random functions, m=8, m'=8:
  average statistical distance is 3.022e-02 = 7.736 * 2^{(m'-3m)/2}
  average variance (d'=0) is 5.505e-07      = 9.235 * 2^{-3m}
  average variance (d'!=0) is 2.207e-08     =94.802 * 2^{-m'-3m}

Testing 16 random functions, m=10, m'=10:
  average statistical distance is 7.569e-03 = 7.750 * 2^{(m'-3m)/2}
  average variance (d'=0) is 8.441e-09      = 9.063 * 2^{-3m}
  average variance (d'!=0) is 8.707e-11     =95.736 * 2^{-m'-3m}


Testing 16 random functions, m=6, m'=12:
  average statistical distance is 5.700e-01 = 4.560 * 2^{(m'-3m)/2}
  average variance (d'=0) is 3.314e-05      = 8.687 * 2^{-3m}
  average variance (d'!=0) is 8.190e-08     =87.941 * 2^{-m'-3m}

Testing 16 random functions, m=8, m'=12:
  average statistical distance is 8.417e-02 = 5.387 * 2^{(m'-3m)/2}
  average variance (d'=0) is 5.329e-07      = 8.941 * 2^{-3m}
  average variance (d'!=0) is 1.371e-09     =94.211 * 2^{-m'-3m}

Testing 16 random functions, m=10, m'=12:
  average statistical distance is 1.310e-02 = 6.706 * 2^{(m'-3m)/2}
  average variance (d'=0) is 8.375e-09      = 8.993 * 2^{-3m}
  average variance (d'!=0) is 2.167e-11     =95.297 * 2^{-m'-3m}


Testing 16 random functions, m=12, m'=6:
  average statistical distance is 2.380e-04 = 7.799 * 2^{(m'-3m)/2}
  average variance (d'=0) is 1.485e-10      = 10.205 * 2^{-3m}
  average variance (d'!=0) is 2.169e-11     =95.405 * 2^{-m'-3m}

Testing 16 random functions, m=12, m'=8:
  average statistical distance is 4.767e-04 = 7.811 * 2^{(m'-3m)/2}
  average variance (d'=0) is 1.354e-10      = 9.306 * 2^{-3m}
  average variance (d'!=0) is 5.449e-12     =95.857 * 2^{-m'-3m}

Testing 16 random functions, m=12, m'=10:
  average statistical distance is 9.520e-04 = 7.799 * 2^{(m'-3m)/2}
  average variance (d'=0) is 1.321e-10      = 9.075 * 2^{-3m}
  average variance (d'!=0) is 1.365e-12     =96.043 * 2^{-m'-3m}
```