Improvements to Correlation Attacks Against Stream Ciphers with Nonlinear Combiners

Brian Stottler
Elizabethtown College

Spring 2018

1 Background

1.1 Stream Ciphers

Throughout the multi-thousand year history of cryptography, a huge number of encryption methods have been developed to protect sensitive information. Many of these techniques do not live up to modern standards of cryptographic security, but there are still a vast number of options among those that do. In reality, the optimal choice of cipher depends heavily on both the type of data being encrypted and the context in which it will be transferred or used. For example: if the data is produced all at once or in large segments, a block-based cipher such as AES may be appropriate. In a field such as real-time communication, however, there is a need to perform high-speed, bit-by-bit encryption of messages with a priori unknown length. These needs are addressed by a class of symmetric-key substitution ciphers collectively known as *stream ciphers*.

The high-level operation of a stream cipher is quite simple: for each bit (binary value 0 or 1) of plaintext data fed into the system, a corresponding bit of encrypted ciphertext is produced. To examine the details of this encryption, however, a formal notion of addition for bits is required. If x and y are individual bits, we define the operation denoted by $\oplus$ (often called "exclusive or" or XOR) according to the following table:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Equivalently, $x \oplus y = 1$ whenever $x \neq y$ and $x \oplus y = 0$ whenever $x = y$. XOR is equivalent to addition modulo 2 and is thus both associative and commutative. Given a secret binary key of the same length as our plaintext, we can now encrypt in a straightforward manner. To do so, we XOR each bit of the plaintext data with the corresponding bit of the key. As an

example, consider the following encryption operation performed on a short binary plaintext. Note that we will represent binary sequences in our examples in the form $b_1 b_2 b_3 \ldots b_n$, and that performing an XOR operation on two-equal length sequences is equivalent to performing an XOR operation between every pair of corresponding bits.

$$\begin{array}{cll} & 0011001100110011 & \text{(plaintext)} \\ \oplus & 1001101110101001 & \text{(key)} \\ \hline & 1010100010011010 & \text{(ciphertext)} \end{array}$$

Although the above sequences are presented in entirety, there is no dependence between the encryption operations performed at each position. That is to say, the encryption could easily have been performed separately on each pair of plaintext and key bits as they were produced.

Conveniently, the same key used for encryption can be used to decrypt the ciphertext (this makes the cipher *symmetric*). Because $x \oplus x = 0$ for any value of x, the plaintext can be recovered by adding the key to the ciphertext. That is,

$$\text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \oplus 0 = \text{plaintext}.$$

To continue our example, we have

$$\begin{array}{cll} & 1010100010011010 & \text{(ciphertext)} \\ \oplus & 1001101110101001 & \text{(key)} \\ \hline & 0011001100110011 & \text{(plaintext).} \end{array}$$

While this method of combining keys with messages is straightforward, constructing secure keys is a challenging problem. In an ideal world, we would always endeavor to perform a *one-time pad*, in which a truly-random key of exactly the same length as the plaintext is used. Indeed, Claude Shannon famously proved this type of cipher to be unbreakable in his 1949 paper [2]. In practice, the difficulty of creating and transmitting truly-random keys is prohibitive – if a secure channel through which to communicate such keys existed, that channel would presumably be used to send the messages themselves. Instead, stream ciphers

attempt to approximate the security of a one-time pad by beginning with a relatively short key and recursively generating a *keystream* of arbitrary length. Although the key is used to initialize the system, the ciphertext is produced by XORing the plaintext and the keystream. The initial key data in this context is known as a *seed*. In the stream ciphers of interest in our research, the mechanism used to generate keystream bits based on the seed is known as a *linear feedback shift register*.

1.2 Linear Feedback Shift Registers

Stream ciphers - at least those of interest in our research - make use of one or more linear feedback shift registers (LFSRs) to recursively generate keys. An LFSR is a fixed-width array of bits on which a *shift* operation can be performed. Each LFSR has an "input side" and an "output side," and performing a shift causes every bit in the LFSR to move one position closer to the output side. As a result, the bit that was closest to the output side is produced as output. In order to keep the LFSR from exhausting its initial supply of bits, bits from predetermined positions in the array are XOR'd together prior to the shift taking place. After each bit has been shifted, the bit resulting from the previous XOR operation is inserted at the input side of the register. The positions involved in this operation are referred to as the *taps*, and the recursion that they specify is referred to as the *rule* for the register.

A visual example helps to clarify this definition. Since we will later distinguish between the bits produced by different registers using subscripts, we will make the unconventional choice of indexing the bits in our examples using superscripts. Hence, b^t would describe the bit b produced as output following the t-th shift of a register. Consider the example register shown in figure 1. In order to perform the first shift, we compute $b^9 = b^8 \oplus b^4 \oplus b^3 \oplus b^1$. Each bit is then moved one position to the left, causing b^1 to be produced as output. Finally, b^9 is inserted at the right-hand side of the register. More generally, the recursive rule for this register is $b^n = b^{n-1} \oplus b^{n-5} \oplus b^{n-6} \oplus b^{n-8}$. The specific bits stored within a register at a given

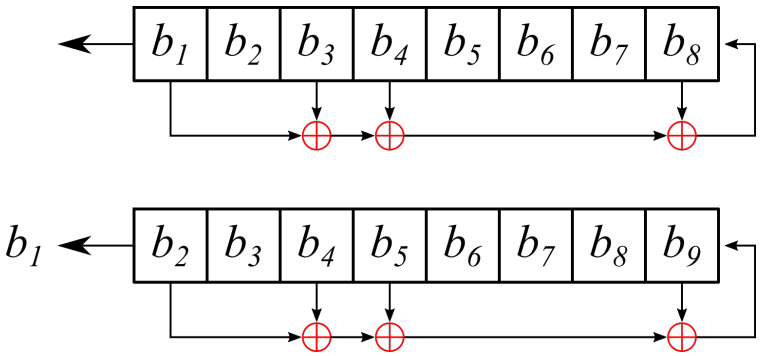

Figure 1: An 8-bit LFSR before and after a shift operation is performed. The newly inserted bit is $b^9 = b^8 \oplus b^4 \oplus b^3 \oplus b^1$.

time are referred to as its *fill*. The diagram in figure 2 shows the output of the same register with initial fill $(1, 1, 0, 1)$ over the course of three shifts.

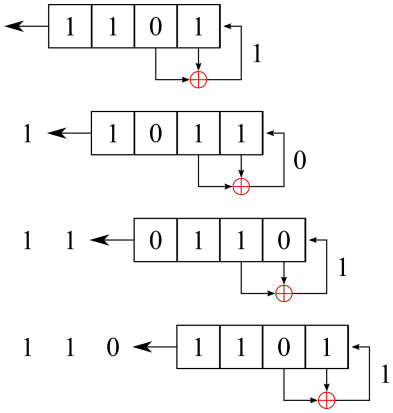

Figure 2: A 4-bit register undergoing three shifts.

Since our goal is to generate arbitrarily long, pseudo-random keys using LFSRs, it is of interest to maximize the period of the output they produce. Because an LFSR can only take on a finite number of states and since each state is determined only by the previous state, it should be apparent that the output of an LFSR must eventually become periodic. By considering the number of possible fills for an LFSR of length k, we can see that the maximum possible period of its output is $2^k - 1$. Note that an all-zero fill is excluded, since any recursive rule given all-zero parameters will indefinitely produce zeros. Although we omit the details, the period of a register's output is largely determined by its recursive rule. For each possible register length k, there is at least one recursive rule that will generate

output with period $2^k - 1$ (assuming a non-zero initial fill is used). We will thus assume that all registers employed in the cryptosystems we discuss have been selected in order to generate maximal-period output.

2 Combiner Functions

While individual LFSRs may efficiently generate pseudorandom output streams, their linearity makes cryptanalysis extremely simple. If an attacker possesses knowledge of a register's length, tap configuration, and a relatively small segment of corresponding output stream, determining the initial fill of the register is reduced to an easily solvable system of linear equations. Even combining the output of multiple, unrelated LFSRs with XOR does not add substantial complexity to this problem. Hence, modern stream ciphers include additional mechanisms to add nonlinear elements to the key generation process. Our research is focused specifically on the use of nonlinear *combiner* functions to fulfill this need.

When making use of such a combiner, multiple LFSRs are configured to shift simultaneously, each outputting a single bit used in the argument of a Boolean function. We will define a *generator* as a set of n LFSRs whose output at each shift is fed directly into a nonlinear Boolean function f of n variables. At each time t, the first register is shifted to produce bit x_1^t, the second register is shifted to produce bit x_2^t, and so on. The output of the entire generator at time t is then computed as $f(x_1^t, x_2^t, \ldots, x_n^t)$. For brevity, we often denote the vector $(x_1^t, x_2^t, \ldots, x_n^t)$ by $\boldsymbol{x}^t$ and denote the output of the generator as $f(\boldsymbol{x}^t)$. More generally, we use $\mathbb{F}_2$ to denote the field of integers modulo 2 (single bits), and $\mathbb{F}_2^n$ to denote n-length vectors of elements from $\mathbb{F}_2$. When we are not concerned about the specific time at which a vector $\boldsymbol{x}^t$ or values $x_1^t, x_2^t, \ldots, x_n^t$ were produced, we will further simplify our notation to $\boldsymbol{x}$ and $x_1, x_2, \ldots, x_n$. The diagram in figure 3 depicts a generator containing four registers with outputs given as arguments to combiner function f. We will always assume that an attacker has complete knowledge of a generator except for the initial fills of its registers.

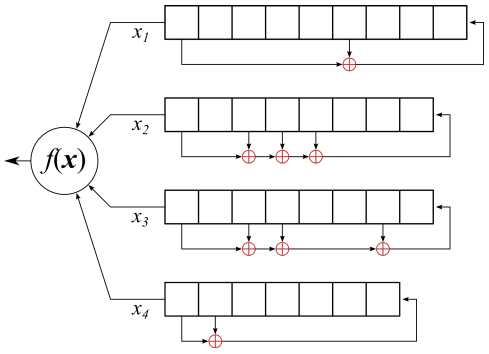

Figure 3: A generator sourcing input from four LFSRs.

This both simplifies our analysis and is usually the case in practice. Hence, the key required to perform encryption and decryption with a given generator is a list of initials fills and the registers they correspond to. While systems of this form are nontrivial to cryptanalyze, later discussion of correlation attacks against these generators will expose an even stricter set of requirements on the function f.

3 Correlation Attacks

We now discuss the method of cryptanalysis introduced by T. Siegenthaler in his 1985 paper [3]. Although Siegenthaler described a ciphertext-only attack, we will reduce the complexity of our development by assuming access to an excerpt of keystream produced by the generator in question. If this were not the case, it would be necessary to test each candidate keystream produced during the attack by XORing it with the ciphertext. Each candidate plaintext produced by this process could then be somehow scored for similarity to the expected language of the true plaintext. Siegenthaler also assumed that the tap configuration of each register was unknown, an assumption we discard for simplicity and practicality. Our goal is thus as follows: Given all details of a generator except for the fills of its registers, and given an excerpt of keystream produced by the generator, we will seek to recover the correct initial fills used to generate that keystream.

Suppose that the generator to be attacked contains n registers of lengths r_1, r_2, ..., r_n

and an n-variable, nonlinear combiner function f. For convenience we will denote a set of registers as an ordered list of integers corresponding to those registers' indices within the generator. To begin, we let $R = (1, 2, \ldots, n)$ describe the set of all registers in the given generator. On top of knowledge of its registers, suppose also that we have N consecutive bits of keystream produced by that generator using unknown initial fills. In order to test hypotheses about those fills, we first create our own copy of the generator, referred to as the *test generator*. Naively, we might then try every possible combination of register fills in our test generator, each time generating a sequence of N output bits and testing them for equality with the known keystream. Unfortunately, this entails $\prod(2^{r_i} - 1)$ possible tests. Instead, we would prefer a means of attacking and recovering the initial fill of each register independently, for a total of $\sum(2^{r_i} - 1)$ possible tests. This is not always possible, and in general the goal of each attack is to recover the fills for some subset of the registers in R.

Throughout the paper, we will let $A \subset R$ be the set of registers with known or previously recovered fills and $B \subset R$ be the set of registers targeted in the current attack. A broad outline of the procedure for performing such an attack is as follows: find a function q that depends only on the registers in $A \cup B$ (that is, on their corresponding variables) and for which a non-trivial correlation (detailed below) exists between f and q. Although q only depends on the registers in $A \cup B$, we find it convenient to describe it as a function of all n registers. Instead of f, we will use q as the combiner function for our test generator. Figure 4 depicts a generator with four registers alongside a test generator depending on only two of those registers. Suppose that $\boldsymbol{x}^t = (x_1^t, x_2^t, \ldots, x_n^t)$ is the unknown "correct" input to f produced by the original generator at time t. Suppose also that $\boldsymbol{y}^t = (y_1^t, y_2^t, \ldots, y_n^t)$ is the input to q produced by our test generator at the same time. The previously mentioned correlation between f and q must be such that each possible pair of the form $(f(\boldsymbol{x}^t), q(\boldsymbol{y}^t))$ occurs with different probability depending on whether or not the correct initial fills have been chosen for our test generator. We will formalize this procedure as a simple hypothesis test, after which we will discuss how to find the best q functions for this purpose. The

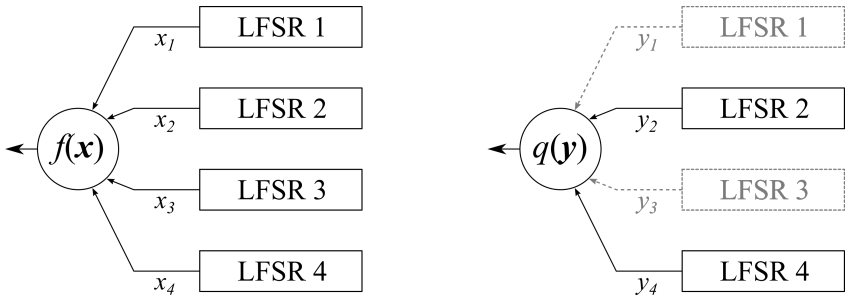

Figure 4: A generator with combiner function f alongside a test generator with combiner function q, which depends on only y_2 and y_4.

question of determining which groups of registers are vulnerable to attack is tied to the choice of q function and will also be addressed at the end of this section.

3.1 Correlation Attack without Prior Knowledge

First, we consider attacks in which no prior knowledge about register fills is being leveraged. That is, situations in which we are attacking a group of registers with unknown fills without making use of previously recovered fills for other registers.

As before, we consider f and q as n-variable functions, with q depending only on the registers in $A \cup B$. Initially, we will assume that we have no prior knowledge and thus that $A = \emptyset$ and that q depends only on the registers in B. Later, we will also allow q to depend on previously-attacked registers of known fill. If $\mathcal{B}_n$ is the set of Boolean functions over $\mathbb{F}_2^n$, then $f, q \in \mathcal{B}_n$. The two hypotheses under consideration are H_0 and H_a. H_0 is the hypothesis that, at any time t, the inputs $\boldsymbol{x}^t, \boldsymbol{y}^t \in \mathbb{F}_2^n$ to f and q, respectively, are effectively independent. H_a is the hypothesis that $q(\boldsymbol{x}^t) = q(\boldsymbol{y}^t)$ at every time t, meaning that the components of $\boldsymbol{x}^t$ and $\boldsymbol{y}^t$ corresponding to the registers in $A \cup B$ are equal. We refrain from asserting that $\boldsymbol{x}^t = \boldsymbol{y}^t$, since we may not be generating output from the registers that q doesn't depend on. If we have chosen the wrong initial fills for the registers in $A \cup B$, we would desire not to reject H_0. If we have chosen the correct fills, we would desire to reject H_0 in favor of H_a. This necessitates the development of a statistic that adequately distinguishes between the two hypotheses.

Frequently, we find it useful to perform counting operations on the truth tables of combiner functions. For this we reserve the notation $C(\ldots)$, where the argument is an indicator of the quantity being counted. In the interest of concise formulas, we will often represent these quantities with notation like $C(g = i)$ instead of $C(\boldsymbol{x} \in \mathbb{F}_2^n : g(\boldsymbol{x}) = i)$. In that vein, for any function $g \in \mathcal{B}_n$ and any bit $i \in \mathbb{F}_2$, we define $C(g = i)$ to be the number of arguments $\boldsymbol{x} \in \mathbb{F}_2^n$ for which $g(\boldsymbol{x}) = i$. Equivalently,

$$C(g = i) = \sum_{\boldsymbol{x} \in \mathbb{F}_2^n} g(\boldsymbol{x}) \oplus i \oplus 1.$$

If h is another function in $\mathcal{B}_n$, we define $C(g = i, h = j)$ to be the number of distinct arguments $\boldsymbol{x} \in \mathbb{F}_2^n$ for which $g(\boldsymbol{x}) = i$ and $h(\boldsymbol{x}) = j$. A crucial point here is that $C(g = i, h = j)$ counts the number of times where g and h take the specified values *given the same argument*. Equivalently,

$$C(g = i, h = j) = \sum_{\boldsymbol{x} \in \mathbb{F}_2^n} (g(\boldsymbol{x}) \oplus i \oplus 1)(h(\boldsymbol{x}) \oplus j \oplus 1).$$

Under H_0, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are assumed to be independent and uniformly chosen from $\mathbb{F}_2^n$, we have

$$P(f(\boldsymbol{x}) = i) = \frac{C(f = i)}{2^n} \text{ and } P(q(\boldsymbol{y}) = j) = \frac{C(q = j)}{2^n}.$$

Thus,

$$P(f(\boldsymbol{x}) = i \cap q(\boldsymbol{y}) = j) = P(f(\boldsymbol{x}) = i)P(q(\boldsymbol{y}) = j) = \frac{C(f = i)C(q = j)}{2^{2n}}.$$

Under H_a, where it is assumed that the components of $\boldsymbol{x}$ and $\boldsymbol{y}$ corresponding to $A \cup B$ are equal, we have

$$P(f(\boldsymbol{x}) = i \cap q(\boldsymbol{y}) = j) = \frac{C(f = i, q = j)}{2^n}.$$

For convenience, we use $p_{ij}(H_0)$ and $p_{ij}(H_a)$ to indicate the probability $P(f(\boldsymbol{x}) = i \cap q(\boldsymbol{y}) = j)$

under the specified hypothesis. That is,

$$p_{ij}(H_0) = \frac{C(f=i)C(q=j)}{2^{2n}} \tag{1}$$

$$p_{ij}(H_a) = \frac{C(f=i, q=j)}{2^n} \tag{2}$$

Next, we define the *probability perturbation* θ by

$$\theta = p_{11}(H_a) - p_{11}(H_0).$$

Using this definition and the basic concepts of probability, we can derive that

$$\begin{aligned} p_{00}(H_a) &= p_{00}(H_0) + \theta, \quad p_{01}(H_a) = p_{01}(H_0) - \theta, \\ p_{10}(H_a) &= p_{10}(H_0) - \theta, \quad p_{11}(H_a) = p_{11}(H_0) + \theta. \end{aligned}$$

We often find it convenient and intuitive to display these probabilities in tables like those depicted in figure 5. It is reasonable to expect that the two cases can be distinguished

H_0:

$f(\boldsymbol{x})$ \ $q(\boldsymbol{y})$	0	1
0	$p_{00}(H_0)$	$p_{01}(H_0)$
1	$p_{10}(H_0)$	$p_{11}(H_0)$

H_a:

$f(\boldsymbol{x})$ \ $q(\boldsymbol{y})$	0	1
0	$p_{00}(H_0) + \theta$	$p_{01}(H_0) - \theta$
1	$p_{10}(H_0) - \theta$	$p_{11}(H_0) + \theta$

Figure 5: The probability distributions for pairs of the form $(f(\boldsymbol{x}), q(\boldsymbol{y}))$ under H_0 and H_a. Note that $\theta = p_{11}(H_a) - p_{11}(H_0)$.

between if and only if $\theta \neq 0$, and that the statistical power of an attack will increase with $|\theta|$.

Let us consider an example in which $f(x_1, x_2, x_3, x_4) = x_1 + x_2 + x_3 x_4$, the registers under attack are specified by $B = (1, 2)$, and $q(x_1, x_2, x_3, x_4) = x_1 + x_2$. The truth tables for f and

q are shown below:

x_1	x_2	x_3	x_4	$f(\boldsymbol{x})$	$q(\boldsymbol{x})$
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	0	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	0

Since $C(f = 1) = C(f = 0) = 2^3$ and $C(q = 1) = C(q = 0) = 2^3$, equation 1 yields the probabilities (all $\frac{1}{4}$) in the H_0 table shown below. Using equation 2, we can compute the corresponding values in the H_a table. Since $p_{11}(H_a) = \frac{3}{8}$, we have that $\theta = \frac{3}{8} - \frac{1}{4} = \frac{1}{8}$, suggesting that we can successfully perform an attack on registers 1 and 2.

H_0:

$f(\boldsymbol{x})$ \ $q(\boldsymbol{y})$	0	1
0	1/4	1/4
1	1/4	1/4

H_a:

$f(\boldsymbol{x})$ \ $q(\boldsymbol{y})$	0	1
0	3/8	1/8
1	1/8	3/8

3.2 Correlation Attack with Prior Knowledge

Suppose that we are seeking to recover the initial fills for the registers in $B \subset R$ and that we have previously recovered the initial fills in another subset $A \subset R$. Although A and B are conceptually sets of registers, we again find it notationally useful to treat them as ordered lists of register indices. Consider any group of k registers $S \subset R$ – we denote S by $S = (i_1, i_2, \ldots i_k)$, where the i_j are integers corresponding to the desired registers (hence, $0 < i_1 < i_2 < \ldots < i_k \leq n$). Given a vector $\boldsymbol{x} \in \mathbb{F}_2^n$, we will often find it convenient to put

constraints on the components of $\boldsymbol{x}$ corresponding to some subset of the registers. We use $\boldsymbol{x}|S$ to denote the k-length vector $(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$ of components from $\boldsymbol{x}$ corresponding to the registers in S. We refer to the vector $\boldsymbol{x}|S$ as the *restriction* of $\boldsymbol{x}$ to S. If $\boldsymbol{s} \in \mathbb{F}_2^k$ is a vector of fixed values, we will write $\boldsymbol{x}|S = \boldsymbol{s}$ to imply $x_{i_1} = s_1, x_{i_2} = s_2, \ldots x_{i_k} = s_k$.

We now consider a function $q \in \mathcal{B}^n$ depending only on the registers in $A \cup B$. Because the input $\boldsymbol{y}^t$ to q at time t is partially determined by the "known" registers in A, our null hypothesis H_0 can no longer assume that the "correct" inputs $\boldsymbol{x}^t$ are completely independent from $\boldsymbol{y}^t$. Instead, we must assume that the components of $\boldsymbol{x}$ and $\boldsymbol{y}$ corresponding to the registers in A are equal (i.e. that $\boldsymbol{x}|A = \boldsymbol{y}|A$), and that all other components are independent. This affects the probability $p_{ij}(H_0)$, which we now adjust accordingly. We define the *tuple count* $C(g = i \mid S = \boldsymbol{s})$ to be a count of the number of arguments $\boldsymbol{x} \in \mathbb{F}_2^n$ for which $\boldsymbol{x}|S = \boldsymbol{s}$ and $g(\boldsymbol{x}) = i$. In order to compute $p_{ij}(H_0)$ while taking the equal components of $\boldsymbol{x}$ and $\boldsymbol{y}$ into account, we sum the conditional probabilities that $f(\boldsymbol{x}) = q(\boldsymbol{y})$ given that $\boldsymbol{x}|A = \boldsymbol{y}|A = \boldsymbol{a}$ for each value of $\boldsymbol{a} \in \mathbb{F}_2^{|A|}$. Hence,

$$\begin{aligned} p_{ij}(H_0) &= \sum_{\boldsymbol{a} \in \mathbb{F}_2^{|A|}} P(f(\boldsymbol{x}) = i \mid A = \boldsymbol{a}) P(q(\boldsymbol{y}) = j \mid A = \boldsymbol{a}) P(A = \boldsymbol{a}) \\ &= \sum_{\boldsymbol{a} \in \mathbb{F}_2^{|A|}} \left(\frac{C(f = i \mid A = \boldsymbol{a})}{2^{n-|A|}} \right) \left(\frac{C(q = j \mid A = \boldsymbol{a})}{2^{n-|A|}} \right) \left(\frac{1}{2^{|A|}} \right) \\ &= \frac{1}{2^{2n-|A|}} \sum_{\boldsymbol{a} \in \mathbb{F}_2^{|A|}} C(f = i \mid A = \boldsymbol{a}) C(q = j \mid A = \boldsymbol{a}) \end{aligned} \tag{3}$$

Note that if $A = \emptyset$, this expression reduces to the probability given in equation 1. The tables shown in figure 5 still apply in this case, with the caveat that equation 3 must be used to compute $p_{ij}(H_0)$.

Continuing our above example with $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3 x_4$, we now assume that the fills of registers 1 and 2 have been successfully recovered, and that our next goal is to attack register 3. In this case, $A = (1, 2)$ and $B = (3)$. In order to leverage the known information about registers 1 and 2, we would like to choose a q function that depends exactly on registers 1,

2, and 3. Although conventional practice is to use the linear combination of outputs from those registers, we will choose $q(\boldsymbol{x}) = 1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3$ to better illustrate the need for equation 3. The truth tables of f and this new q are shown below, alongside the corresponding probability tables:

x_1	x_2	x_3	x_4	$f(\boldsymbol{x})$	$q(\boldsymbol{x})$
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	0	0
1	1	1	1	1	0

H_0:

$f(\boldsymbol{x})$ \ $q(\boldsymbol{y})$	0	1
0	5/16	3/16
1	7/16	1/16

H_a:

$f(\boldsymbol{x})$ \ $q(\boldsymbol{y})$	0	1
0	1/4	1/4
1	1/2	0

Note that $\theta = -1/16$ and that the probabilities in the H_0 table differ from those that would have been produced by equation 1. We are not aware of any prior authors explicitly observing or addressing the differences that occur in this case.

3.3 Test Statistic

Although there are a number of valid statistics for distinguishing H_0 from H_a, we will focus on the classical statistic presented by Siegenthaler [3]. When generalized to support multi-

register attacks, that statistic is defined to be

$$\gamma = N - 2\sum_{t=1}^{N} f(\boldsymbol{x}^t) \oplus q(\boldsymbol{y}^t),$$

where $f(\boldsymbol{x}^t)$ is the given keystream output at time t and $q(\boldsymbol{y}^t)$ is the output from the test generator at time t.

Consider the generic hypothesis $H \in \{H_0, H_a\}$ and let $p_{f\neq q}(H) = p_{01}(H) + p_{10}(H)$. Note that

$$p_{f\neq q}(H_a) = p_{01}(H_a) + p_{10}(H_a) = p_{01}(H_0) + p_{10}(H_0) - 2\theta = p_{f\neq q}(H_0) - 2\theta.$$

If we define $Y = \sum_{t=1}^{N} f(\boldsymbol{x}^t) \oplus q(\boldsymbol{y}^t)$, then Y is a binomial random variable with N trials and probability of success $p_{f\neq q}(H)$. Hence,

$$\begin{aligned} E(Y) &= Np_{f\neq q}(H), \\ V(Y) &= Np_{f\neq q}(H)(1 - p_{f\neq q}(H)). \end{aligned}$$

Denoting $E(\gamma \mid H)$ by $\mu(H)$ and $V(\gamma \mid H)$ by $\sigma^2(H)$, we thus have

$$\begin{aligned} \mu(H_0) &= N - 2Np_{f\neq q}(H_0), \\ \sigma^2(H_0) &= 4Np_{f\neq q}(H_0)(1 - p_{f\neq q}(H_0)). \end{aligned}$$

and

$$\begin{aligned} \mu(H_a) &= N - 2Np_{f\neq q}(H_a) \\ &= N - 2Np_{f\neq q}(H_0) + 4N\theta, \\ \sigma^2(H_a) &= 4Np_{f\neq q}(H_a)(1 - p_{f\neq q}(H_a)) \\ &= 4N(p_{f\neq q}(H_0) - 2\theta)(1 - p_{f\neq q}(H_0) + 2\theta). \end{aligned}$$

Because Y is binomial and N is often large, we approximate γ as a normal random variable. It

should be clear that $\mu(H_0) = \mu(H_a)$ and $\sigma^2(H_0) = \sigma^2(H_a)$ when $\theta = 0$, making it impossible to distinguish between the two distributions.

As is generally the case with hypothesis testing, we would like to either be aware of the type I error (α) and type II error (β) for a fixed number of keystream bits N, or to determine N based on values of α and β. If we consider a rejection region for H_0 of the form $\gamma \geq \gamma^*$ for $\theta > 0$ and $\gamma \leq \gamma^*$ for $\theta < 0$, then

$$\alpha = 1 - \Phi\left(\left|\frac{\gamma^* - \mu(H_0)}{\sqrt{\sigma^2(H_0)}}\right|\right) \tag{4}$$

and

$$\beta = 1 - \Phi\left(\left|\frac{\gamma^* - \mu(H_a)}{\sqrt{\sigma^2(H_a)}}\right|\right). \tag{5}$$

Solving these equations for γ^* yields

$$\gamma^* = \operatorname{sgn}(\theta)\sqrt{\sigma^2(H_0)}\Phi^{-1}(1-\alpha) + \mu(H_0) \tag{6}$$

and

$$\gamma^* = -\operatorname{sgn}(\theta)\sqrt{\sigma^2(H_a)}\Phi^{-1}(1-\beta) + \mu(H_a), \tag{7}$$

respectively. By combining either equations 5 and 6 or equations 4 and 7, we can derive that

$$N = \frac{\left(\Phi^{-1}(1-\beta)\sqrt{p_{f\neq q}(H_a)(1-p_{f\neq q}(H_a))} + \Phi^{-1}(1-\alpha)\sqrt{p_{f\neq q}(H_0)(1-p_{f\neq q}(H_0))}\right)^2}{4\theta^2}. \tag{8}$$

This is the number of keystream bits required to achieve desired α and β errors.

We now return to our previous example where $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3x_4$. During our first attack on registers 1 and 2 in which we chose $q(\boldsymbol{x}) = x_1 \oplus x_2$, for which $p_{f\neq q}(H_0) = 1/2$, $p_{f\neq q}(H_a) = 1/4$, and $\theta = 1/8$. If we fix $\alpha = 2^{-20}$ and $\beta = 2^{-20}$, we can use equation 8 to

derive the following value for N:

$$N = \frac{\left(\Phi^{-1}(1-2^{-20})\sqrt{\frac{1}{4}\left(\frac{3}{4}\right)} + \Phi^{-1}(1-2^{-20})\sqrt{\frac{1}{2}\left(\frac{1}{2}\right)}\right)^2}{4\left(\frac{1}{8}\right)^2}$$

$$\approx 316.$$

Hence, to perform the attack while achieving our desired levels of α and β, we need at least 316 bits of known keystream data.

We can perform the same calculations for our extended example in which the now-recovered fills of registers 1 and 2 are used to attack register 3. With our choice of $q(\boldsymbol{x}) = 1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3$, we can compute $p_{f\neq q}(H_0) = 5/8$ and $p_{f\neq q}(H_a) = 3/4$. With the same α and β as before,

$$N = \frac{\left(\Phi^{-1}(1-2^{-20})\sqrt{\frac{5}{8}\left(\frac{3}{8}\right)} + \Phi^{-1}(1-2^{-20})\sqrt{\frac{3}{4}\left(\frac{1}{4}\right)}\right)^2}{4\left(-\frac{1}{16}\right)^2}$$

$$\approx 1221.$$

In this case, 1221 bits of keystream would be required to perform the desired attack. It should not be surprising that this number is'' larger than in the first case, since θ is significantly smaller.

4 Correlation-Immune Functions

Thus far, we have not provided a method for determining which sets of registers A and B will admit q functions that allow meaningful correlation attacks. An extremely useful concept in making this determination is that of *correlation immunity*, a property first defined by Siegenthaler in 1984 [4]. Consider again a function $f \in \mathcal{B}_n$ and the set of registers $R = (1, 2, \ldots, n)$. f is said to be *correlation immune of order k* if for every subset $S \subset R$ of

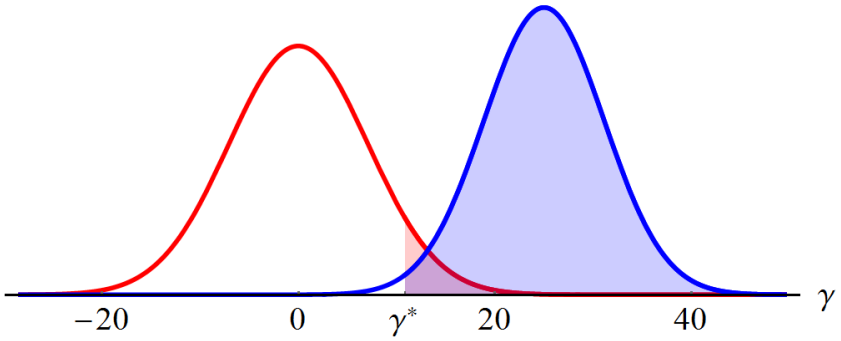

Figure 6: Distributions of γ under H_0 (red) and H_a (blue) for $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3x_4$ and $q(\boldsymbol{x}) = x_1 \oplus x_2$ with $N = 50$ bits. The red shaded area indicates $\alpha = 2^{-4}$ and the blue shaded area is the corresponding power, $1 - \beta \approx 0.99$.

k registers and an unknown value $\boldsymbol{x} \in \mathbb{F}_2^n$, the probability that $\boldsymbol{x}|S = \boldsymbol{s}$ given a value $f(\boldsymbol{x})$ is equal for all choices of $\boldsymbol{s} \in \mathbb{F}_2^k$. That is, the value of $f(\boldsymbol{x})$ is statistically independent of the value of $\boldsymbol{x}|S$. For example, if f is correlation immune of order 3 and $S = (1, 3, 4)$ then there must be equally many values of $\boldsymbol{x} \in \mathbb{F}_2^n$ for which $\boldsymbol{x}|S = (0, 1, 0)$ and $f(\boldsymbol{x}) = 1$ as for which $\boldsymbol{x}|S = (1, 1, 0)$ and $f(\boldsymbol{x}) = 1$.

To see why this property prevents correlation attacks, consider a function $f \in \mathcal{B}_n$, with f being correlation immune of order k. Suppose also that q depends on the k or fewer variables in some $S \subset R$ and that $q(\boldsymbol{x}) = j$ exactly when $\boldsymbol{x}|S$ is one of $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_m \in \mathbb{F}_2^{|S|}$. For every $\boldsymbol{x} \in \mathbb{F}_2^n$, this means that

$$\begin{aligned}
p_{ij}(H_a) &= P(f(\boldsymbol{x}) = i \mid q(\boldsymbol{x}) = j)\, P(q(\boldsymbol{x}) = j) \\
&= \sum_{i=1}^{m} P(f(\boldsymbol{x}) = i \mid \boldsymbol{x}|S = \boldsymbol{s}_i)\, P(\boldsymbol{x}|S = \boldsymbol{s}_i) \\
&= \sum_{i=1}^{m} P(f(\boldsymbol{x}) = i)\, P(\boldsymbol{x}|S = \boldsymbol{s}_i) \\
&= P(f(\boldsymbol{x}) = i)\, P(q(\boldsymbol{x}) = j) \\
&= p_{ij}(H_0).
\end{aligned}$$

Hence, a correlation attack against f and using q will not succeed.

4.1 The Walsh Transform

Luckily, a more practical method exists for determining the correlation immunity of a function f. First, note that we consider the dot product of two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^n$ to be defined as $\boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 \oplus a_2 b_2 \oplus \ldots \oplus a_n b_n$. Next, we define the *Walsh transform* $W(f)$ at $\boldsymbol{\omega} \in \mathbb{F}_2^n$ by

$$W(f)(\boldsymbol{\omega}) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(\boldsymbol{x}) \oplus \boldsymbol{\omega} \cdot \boldsymbol{x}}.$$

In effect, the term $f(\boldsymbol{x}) \oplus \boldsymbol{\omega} \cdot \boldsymbol{x}$ measures how similar f is to the linear combination of variables specified by $\boldsymbol{\omega} \cdot \boldsymbol{x}$. Exponentiating with a base of -1 maps values of 0 and 1 to values of 1 and -1, respectively. Hence, the transform can be equivalently defined as

$$W(f)(\boldsymbol{\omega}) = C(f = \boldsymbol{\omega} \cdot \boldsymbol{x}) - C(f \neq \boldsymbol{\omega} \cdot \boldsymbol{x}),$$

where $C(f = \boldsymbol{\omega} \cdot \boldsymbol{x})$ is a count of how many distinct values $\boldsymbol{x} \in \mathbb{F}_2^n$ make $f(x)$ equal to the linear function $\boldsymbol{\omega} \cdot \boldsymbol{x}$ and $C(f \neq \boldsymbol{\omega} \cdot \boldsymbol{x})$ is its complement

Consider the Walsh transform of $f(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 x_4$ depicted below:

$\boldsymbol{\omega}$	$C(f = \boldsymbol{\omega} \cdot \boldsymbol{x})$	$C(f \neq \boldsymbol{\omega} \cdot \boldsymbol{x})$	$W(f)(\boldsymbol{\omega})$
0000	8	8	0
0001	8	8	0
0010	8	8	0
0100	8	8	0
1000	8	8	0
0011	8	8	0
0101	8	8	0
0110	8	8	0
1010	8	8	0
1100	12	4	8
0111	8	8	0
1011	8	8	0
1101	12	4	8
1110	12	4	8
1111	4	12	-8

Note that we have ordered the values of $\boldsymbol{\omega}$ by their *Hamming weight*, or the number of 1s in each vector. From a theorem presented by Xiao and Massey [5], we have the important result that a function $g \in \mathcal{B}_n$ is correlation immune of order k if and only if its Walsh transform is 0 for all values of $\boldsymbol{\omega} \in \mathbb{F}_2^n$ with Hamming weight of k or less. Hence, the function f whose Walsh transform is shown above is correlation immune of order 1. In practical terms, this means that no correlation attack is feasible against two-or-fewer registers from a generator with combiner f.

4.2 Prior Results for Choosing q

While the Walsh transform gives us useful information about which groups of registers can be successfully attacked, we still lack a means of determining which combiner q for our test generator will admit the largest value of θ. As mentioned previously, the convention is to always choose a linear function of the variables/registers in question. This intuition is not

entirely baseless, as demonstrated by Canteaut and Trabbia in their 2000 paper [1]. The paper first defines a notion of t-resilience: A function $f \in \mathcal{B}_n$ is said to be *t-resilient* if it is correlation immune of order t and if $C(f = 1) = C(f = 0)$. Canteaut proves that, given a generator with a t-resilient combiner f, the optimal choice of combiner q to use during a correlation attack on a group $B = (i_1, i_2, \ldots, i_t, i_{t+1})$ of $t + 1$ registers is the affine function $q(\boldsymbol{x}) = x_{i_1} \oplus x_{i_2} \oplus \ldots \oplus x_{i_{t+1}} \oplus c$. The value of the constant term c is determined as follows: Let 1_B be the bit vector in $\mathbb{F}_2^n$ for which the i-th component is 1 if and only if $i \in B$. Then $c = 0$ if $W(f)(1_B) > 0$ and $c = 1$ otherwise.

Since our example combiner $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3x_4$ is both balanced and correlation immune of order 2, it is also 2-resilient. Hence, we are guaranteed that the optimal choice of q for a correlation attack on the registers in $B = (1, 2)$ is $q(\boldsymbol{x}) = x_1 \oplus x_2$. Whether or not a linear q function is always optimal for a group of more than $t + 1$ registers appears to be an open question. Although it does not answer this question universally, our work provides a systematic method of determining the optimal q function to use in any stage of an attack. We also introduce a novel test method that in certain cases allows for an attack with 100% statistical power.

5 Our Results

5.1 A Method for an Optimal Choice of q

Our procedure for choosing an optimal q function requires carefully expanding the formula for θ. To do so, we first partition the set R of all registers into three disjoint subsets. As before, let A be the set of registers with previously recovered fill and let B be the set of registers targeted in the current attack. We also let C be the set of registers not present in $A \cup B$. We use $f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ to indicate the value $f(\boldsymbol{x})$ where $x|A = \boldsymbol{a}$, $x|B = \boldsymbol{b}$, and $x|C = \boldsymbol{c}$. $q(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ will have similar meaning. Since q only depends on the registers in $A \cup B$, however, it must be the case that for a fixed $\boldsymbol{a}$ and $\boldsymbol{b}$, $q(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = q(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d})$ for all $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{F}_2^{|C|}$. We

will thus use the notation $q(\boldsymbol{a}, \boldsymbol{b})$ to denote any and all of these values. To de-clutter our notation, we also write sums of the form $\sum_{\boldsymbol{s} \in \mathbb{F}_2^{|S|}}$ simply as $\sum_{\boldsymbol{s}}$ wherever the usage of $\boldsymbol{s}$ makes its parent set clear. We now proceed to derive our expansion of θ.

Using the fact that

$$\begin{aligned} p_{11}(H_a) &= \frac{C(f=1, q=1)}{2^n} \\ &= \frac{1}{2^n} \sum_{\boldsymbol{a}} \sum_{\boldsymbol{b}} \sum_{\boldsymbol{c}} f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) q(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \\ &= \frac{1}{2^n} \sum_{\boldsymbol{a}} \sum_{\boldsymbol{b}} q(\boldsymbol{a}, \boldsymbol{b}) \sum_{\boldsymbol{c}} f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \end{aligned}$$

and

$$\begin{aligned} p_{11}(H_0) &= \frac{1}{2^{2n-|A|}} \sum_{\boldsymbol{a}} C(f=1 \mid A=\boldsymbol{a}) C(q=1 \mid A=\boldsymbol{a}) \\ &= \frac{1}{2^{2n-|A|}} \sum_{\boldsymbol{a}} \left(\sum_{\boldsymbol{d}} \sum_{\boldsymbol{c}} f(\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{c}) \right) \left(\sum_{\boldsymbol{b}} \sum_{\boldsymbol{c}} q(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \right) \\ &= \frac{1}{2^{2n-|A|}} \sum_{\boldsymbol{a}} \left(\sum_{\boldsymbol{d}} \sum_{\boldsymbol{c}} f(\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{c}) \right) \left(2^{|C|} \sum_{\boldsymbol{b}} q(\boldsymbol{a}, \boldsymbol{b}) \right) \\ &= \frac{1}{2^{n+|B|}} \sum_{\boldsymbol{a}} \sum_{\boldsymbol{b}} q(\boldsymbol{a}, \boldsymbol{b}) \left(\sum_{\boldsymbol{d}} \sum_{\boldsymbol{c}} f(\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{c}) \right) \end{aligned}$$

we can derive that

$$\begin{aligned} \theta &= p_{11}(H_a) - p_{11}(H_0) \\ &= \frac{1}{2^n} \sum_{\boldsymbol{a}} \sum_{\boldsymbol{b}} q(\boldsymbol{a}, \boldsymbol{b}) \sum_{\boldsymbol{c}} f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) - \frac{1}{2^{n+|B|}} \sum_{\boldsymbol{a}} \sum_{\boldsymbol{b}} q(\boldsymbol{a}, \boldsymbol{b}) \left(\sum_{\boldsymbol{d}} \sum_{\boldsymbol{c}} f(\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{c}) \right) \\ &= \frac{1}{2^{n+|B|}} \sum_{\boldsymbol{a}} \sum_{\boldsymbol{b}} q(\boldsymbol{a}, \boldsymbol{b}) \left[\sum_{\boldsymbol{c}} \left(2^{|B|} f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) - \sum_{\boldsymbol{d}} f(\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{c}) \right) \right] \\ &= \frac{1}{2^{n+|B|}} \sum_{\boldsymbol{a}} \sum_{\boldsymbol{b}} F(\boldsymbol{a}, \boldsymbol{b}) q(\boldsymbol{a}, \boldsymbol{b}), \end{aligned}$$

where

$$F(\boldsymbol{a}, \boldsymbol{b}) = \sum_{\boldsymbol{c}} \left(2^{|B|} f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) - \sum_{\boldsymbol{d}} f(\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{c}) \right).$$

Because we know that the coefficient of each $q(\boldsymbol{a}, \boldsymbol{b})$ term is $F(\boldsymbol{a}, \boldsymbol{b})$, we can choose each value of $q(\boldsymbol{a}, \boldsymbol{b})$ (that is, the values of $q(\boldsymbol{x})$ for all $\boldsymbol{x}$ where $\boldsymbol{x}|A = \boldsymbol{a}$ and $\boldsymbol{x}|B = \boldsymbol{b}$) to maximize the overall expression. Hence, we let $q(\boldsymbol{a}, \boldsymbol{b}) = 1$ if $F(\boldsymbol{a}, \boldsymbol{b}) > 0$ and let $q(\boldsymbol{a}, \boldsymbol{b}) = 0$ if $F(\boldsymbol{a}, \boldsymbol{b}) < 0$. The value of $q(\boldsymbol{a}, \boldsymbol{b})$ can be chosen arbitrarily if $F(\boldsymbol{a}, \boldsymbol{b}) = 0$, although we generally prefer to make q a linear function whenever possible.

Using this procedure, it should be possible to confirm the optimal choices of q for our earlier examples with $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3 x_4$. With respect to the initial attack on registers 1 and 2, we have $A = \emptyset$, $B = (1, 2)$, and $C = (3, 4)$. The following table shows each value of $F(\boldsymbol{a}, \boldsymbol{b})$ for this scenario and our corresponding choices for $q(\boldsymbol{a}, \boldsymbol{b})$:

$\boldsymbol{a}$	$\boldsymbol{b}$	$F(\boldsymbol{a}, \boldsymbol{b})$	$q(\boldsymbol{a}, \boldsymbol{b})$
$\emptyset$	$(0, 0)$	-4	0
$\emptyset$	$(0, 1)$	4	1
$\emptyset$	$(1, 0)$	4	1
$\emptyset$	$(1, 1)$	-4	0

In other words, the truth table for q must be

x_1	x_2	x_3	x_4	$f(\boldsymbol{x})$	$q(x)$
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	0	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	0

which corresponds to the expected (via Canteaut's result) linear function $q(\boldsymbol{x}) = x_1 \oplus x_2$ with $\theta = 1/8$. If we perform the same procedure when $A = (1, 2)$, $B = (3)$, and $C = (4)$, we derive the following values of F:

$\boldsymbol{a}$	$\boldsymbol{b}$	$F(\boldsymbol{a}, \boldsymbol{b})$	$q(\boldsymbol{a}, \boldsymbol{b})$
$(0, 0)$	(0)	-1	0
$(0, 0)$	(1)	1	1
$(0, 1)$	(0)	1	1
$(0, 1)$	(1)	-1	0
$(1, 0)$	(0)	1	1
$(1, 0)$	(1)	-1	0
$(1, 1)$	(0)	-1	0
$(1, 1)$	(1)	1	1

Although we omit the full 4-variable truth table of q, the resulting function is $q(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3$ with $\theta = 1/8$.

5.2 The Impossible Pairs Test

For the same combiner $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3 x_4$, recall our previous example attack in which $A = (1, 2)$, $B = (3)$, and $q(\boldsymbol{x}) = 1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_1 x_3 \oplus x_2 x_3$. The probability tables under each hypothesis are reproduced below.

H_0:

$f(\boldsymbol{x})$ \ $q(\boldsymbol{y})$	0	1
0	5/16	3/16
1	7/16	1/16

H_a:

$f(\boldsymbol{x})$ \ $q(\boldsymbol{y})$	0	1
0	1/4	1/4
1	1/2	0

Although a classical correlation attack is possible in this case, the fact that $(f(\boldsymbol{x}), q(\boldsymbol{y})) = (0, 0)$ occurs with probability 0 under H_a suggests a different type of test. In general, we refer to a pair of bits (b_1, b_2) for which $p_{b_1 b_2}(H_0) \neq 0$ and $p_{b_1 b_2}(H_a) = 0$ as an *impossible pair* and, assuming such a pair can be found, we perform an *impossible pair test*.

We will first focus on the details of exploiting an impossible pair (b_1, b_2) given combiners $f, q \in \mathcal{B}_n$ that produce it. Suppose we are given sets of registers $A, B \subset R$ with their usual meanings and N bits of keystream, denoted as the sequence $\{s^t\} = s^1, s^2, \ldots, s^N$ (recall that we use a superscript to indicate the time/index of each output). For each combination of initial fills assigned to the registers in B, we produce an equal length sequence $\{p^t\}$ from our test generator. If at any time i between 0 and N it is the case that $(s^i, p^i) = (b_1, b_2)$, then we reject the fills under consideration. We can have complete confidence in this decision, since such a pair will never be produced if the correct fills have been chosen. Thus, this resulting test has the rare advantage of 100% statistical power.

Although the type II error for this test is 0, we may still wish to fix one of α or N and to compute the other. Under H_0, the impossible pair (b_1, b_2) occurs with probability $p_{b_1 b_2}(H_0)$. In this case, the number of occurrences of (b_1, b_2) is a binomial random variable in N trials.

Hence,

$$\begin{aligned}\alpha &= P\left((f(\boldsymbol{x}^t), q(\boldsymbol{y}^t)) \neq (b_1, b_2) \text{ for any } t \leq N\right) \\ &= P\left((b_1, b_2) \text{ first occurs at time } N+1 \text{ or later}\right) \\ &= \sum_{t=N+1}^{\infty} (1 - p_{b_1 b_2}(H_0))^{t-1} p_{b_1 b_2}(H_0) \\ &= (1 - p_{b_1 b_2}(H_0))^N\end{aligned} \tag{9}$$

and

$$N = \frac{\ln \alpha}{\ln(1 - p_{b_1 b_2}(H_0))}. \tag{10}$$

Continuing the preceding example, performing an impossible pair test for $(1, 1)$ while achieving an α of 2^{-20} would require at least

$$N = \frac{\ln 2^{-20}}{\ln \frac{15}{16}} \approx 215$$

bits of keystream data. Using the same q function and fixing $N = 215$ and $\alpha = 2^{-20}$, the attack performed with Siegenthaler's statistic has a type II error of $\beta \approx 0.186 \approx 2^{-2.43}$ (computed using equations 5 and 6). Even when using the known-optimal q function $q(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3$ with the same N and α, the standard attack yields $\beta \approx 0.0015 \approx 2^{-9.37}$. This illustrates the fact that impossible pairs attacks are worth considering whenever they are available.

5.3 Finding Impossible Pairs

Once again, let $f \in \mathcal{B}_n$ be a combiner that depends on all registers/variables in R. The following definitions and results provide a means of determining if and when a given set of registers will be susceptible to an impossible pairs attack.

Definition 1. We say the set $S \subset R$ with $|S| = k$ has the *all-cases property* if for every $\boldsymbol{s} \in \mathbb{F}_2^k$, both $C(f = 0 \mid S = \boldsymbol{s}) > 0$ and $C(f = 1 \mid S = \boldsymbol{s}) > 0$.

If $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3 x_4$, for example, and $S = (1, 2)$, we have the following table of tuple counts:

$\boldsymbol{s}$	$C(f = 0 \mid S = \boldsymbol{s})$	$C(f = 1 \mid S = \boldsymbol{s})$
$(0, 0)$	3	1
$(0, 1)$	1	3
$(1, 0)$	3	1
$(1, 1)$	1	3

Hence, S has the all cases property. Note that R itself never has the all-cases property, since for any $\boldsymbol{x} \in \mathbb{F}_2^n$, either $f(\boldsymbol{x}|R) = f(\boldsymbol{x}) = 0$ or $f(\boldsymbol{x}|R) = f(\boldsymbol{x}) = 1$, but not both.

Proposition 1. *If $S \subset R$ has the all-cases property, then every subset of S also has the all-cases property.*

Proof. Let $S = (i_1, i_2, \ldots, i_k)$ have the all-cases property. Since any subset of S can be constructed by removing a finite number of elements from S, we will simply prove that if $|S| = k > 1$, any subset of S with $k - 1$ elements also has the all-cases property. The more general result follows from induction.

Let $S' \subset S$ be such that $|S'| = k - 1$, and let i_j be the single remaining element in $S - S'$. For any $\boldsymbol{b} = (b_1, b_2, \ldots, b_{k-1}) \in \mathbb{F}_2^{k-1}$, we can define $\boldsymbol{a}_0, \boldsymbol{a}_1 \in \mathbb{F}_2^k$ by $\boldsymbol{a}_0 = (b_1, b_2, \ldots, b_{j-1}, 0, b_j, b_{j+1}, \ldots, b_{k-1})$ and $\boldsymbol{a}_1 = (b_1, b_2, \ldots, b_{j-1}, 1, b_j, b_{j+1}, \ldots, b_{k-1})$. Since both $\boldsymbol{a}_0|S' = \boldsymbol{b}$ and $\boldsymbol{a}_1|S' = \boldsymbol{b}$, we see that

$$C(f = 0 \mid S' = \boldsymbol{b}) = C(f = 0 \mid S = \boldsymbol{a}_0) + C(f = 0 \mid S = \boldsymbol{a}_1) > 0,$$

and

$$C(f = 1 \mid S' = \boldsymbol{b}) = C(f = 1 \mid S = \boldsymbol{a}_0) + C(f = 1 \mid S = \boldsymbol{a}_1) > 0.$$

□

Definition 2. We say $S \subset R$ has the all-cases property *of order* k if every subset $S' \subset S$ with $|S'| = k$ possesses the all-cases property. We say f itself possesses the all-cases property of order k if R does.

Note that by the previous proposition, if $S \subset R$ has the all-cases property of order k then it also has the all-cases property of order j wherever $1 \leq j \leq k$.

Consider a function $q \in \mathcal{B}_n$ for use in our test generator. If f and q have impossible pair (b_1, b_2) for some $b_1, b_2 \in \mathbb{F}_2$, then we call q a *pair-prevention function*. We also refer to the set of registers on which q depends (always $A \cup B$ in our attacks) as *depend*(q).

Lemma 1. *Let $S \subset R$. If S possesses the all-cases property, then any pair-prevention function q with depend$(q) \subset S$ must be constant.*

Proof. Let $S \subset R$ be such that S possess the all-cases property. Suppose that q is a pair-prevention function depending only on the variables in S and let (b_1, b_2) be the corresponding impossible pair.

Let $\boldsymbol{a} \in \mathbb{F}_2^n$, and let $\boldsymbol{b} = \boldsymbol{a}|S$. Since q depends only on the variables in S, it must be that for any other $\boldsymbol{a}' \in \mathbb{F}_2^n$ with $\boldsymbol{a}'|S = \boldsymbol{b}$ we have $q(\boldsymbol{a}) = q(\boldsymbol{a}')$. By the all-cases property, we know $C(f = b_1 \mid S = \boldsymbol{b}) > 0$, implying that there is an $\boldsymbol{a}' \in \mathbb{F}_2^n$ with $\boldsymbol{a}'|S = \boldsymbol{b}$ such that $f(\boldsymbol{a}') = b_1$. Since q prevents (b_1, b_2), we must have $q(\boldsymbol{a}') = b_2 \oplus 1$ and thus $q(\boldsymbol{a}) = b_2 \oplus 1$.

Since q takes the value $b_2 \oplus 1$ for every $\boldsymbol{a} \in \mathbb{F}_2^n$, it must be constant. □

Lemma 2. *If $S \subset R$ does not possess the all-cases property, then there is a nonconstant pair-prevention function $q \in \mathcal{B}_n$ with depend$(q) \subset S$.*

Proof. Let $S \subset R$ be a set without the all-cases property. Then there must exist $p \in \mathbb{F}_2$ and $\boldsymbol{b} \in \mathbb{F}_2^{|S|}$ such that $C(f = p \mid S = \boldsymbol{b}) = 0$. Since f is nonconstant and depends on all of its variables, there must also be values $\boldsymbol{b'} \in \mathbb{F}_2^{|S|}$ for which $C(f = p \mid S = \boldsymbol{b'}) > 0$.

Let $\boldsymbol{a} \in \mathbb{F}_2^n$. If $C(f = p \mid S = \boldsymbol{a}|S) = 0$, define $q(\boldsymbol{a}) = 0$. Otherwise, define $q(\boldsymbol{a}) = 1$. By the preceding logic regarding f, q must be nonconstant. Since the value of $q(\boldsymbol{a})$ depends

only on the value of $\boldsymbol{a}|S$, we must have $depend(q) \subset S$. Additionally the pair $(p, 0)$ never occurs, since $f(\boldsymbol{a}) = p$ implies that $C(f = p \mid S = \boldsymbol{a}|S) > 0$ which implies that $q(\boldsymbol{a}) = 1$. $\square$

The construction of q used in this proof is made systematic in the following section. Combining the two lemmas, we have the following useful criterion for the existence of nonconstant pair-prevention functions:

Theorem 1. *Let $S \subset R$. A nonconstant pair-prevention function $q \in \mathcal{B}_n$ with $depend(q) \subset S$ exists if and only if S does not have the all-cases property.*

5.4 The P-Transform

When attempting to find pair-prevention functions for a given Boolean function f, we care only about the "presence" of any given tuple count – not its actual value. Although computing the exact tuple counts for a given subset of R gives us this information, we really only care about the extrema - values for which either the $f = 0$ tuple count or the $f = 1$ tuple count is 0. Since the specific values at which these tuple counts are 0 completely determine the pair-prevention functions admitted, it would be ideal to both locate these values and construct the truth tables of our desired functions in a single operation.

We introduce the following notation to assist with such an operation.

Definition 3. Given $\boldsymbol{a} \in \mathbb{F}_2^k$, define $v(\boldsymbol{a}) = a_1 2^0 + a_2 2^1 + \ldots + a_k 2^{k-1}$ (in $\mathbb{Z}$). We refer to $v(\boldsymbol{a})$ as the *integer value* of $\boldsymbol{a}$.

Definition 4. Given $\boldsymbol{a} \in \mathbb{F}_2^k$, define $\delta(\boldsymbol{a})$ to be the vector of length 2^k for which $\delta(\boldsymbol{a})_i = 1$ when $i = v(\boldsymbol{a}) + 1$ and $\delta(\boldsymbol{a})_i = 0$ otherwise.

Definition 5. Given $S \subset R$, we define $m(S)$ to be a vector of length $|R| = n$ such that $m(S)_i = 1$ when $i \in S$ and $m(S)_i = 0$ otherwise. We refer to $m(S)$ as the *mask* of S.

For example, if $\boldsymbol{a} = (0, 1, 1)$ we have $v(\boldsymbol{a}) = 110_2 = 6$ and $\delta(\boldsymbol{a}) = (0, 0, 0, 0, 0, 0, 1, 0)$. Also, if $|R| = 5$ and $S = (1, 3)$, then $m(S) = (1, 0, 1, 0, 0)$.

We now introduce an operation that, for a given set of variables S and a given value $p \in \mathbb{F}_2$, computes a general form of the truth table for every pair-prevention function q corresponding to impossible pairs of the form (p, b_2) (for both $b_2 = 0$ and $b_2 = 1$) and depending only on the variables in S. Note that $\vee$ corresponds to the Boolean "or" operation defined by the following truth table:

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Definition 6. Let $p \in \mathbb{F}_2$ and $S \subset R$. For $\boldsymbol{x} \in \mathbb{F}_2^n$, we define the *case indicator function* $C_f(S, p, \boldsymbol{x})$ as follows:

$$\begin{aligned} C_f(S, p, \boldsymbol{x}) &= 1 \oplus [1 \oplus p \oplus f(\boldsymbol{x})]\,[1 \oplus f(\boldsymbol{x}) \oplus f(\boldsymbol{x}\, m(S))] \\ &= [p \oplus f(\boldsymbol{x})] \vee [f(\boldsymbol{x}) \oplus f(\boldsymbol{x}\, m(S))]\,, \end{aligned}$$

where $\boldsymbol{x}\, m(S)$ is the component-wise product of the vectors $\boldsymbol{x}$ and $m(S)$.

The two equivalent definitions are provided for clarity – the first is written directly in terms of field operations and the second is more concise and practical for implementation. This function is equal to 1 exactly when any two of $f(\boldsymbol{x})$, $f(\boldsymbol{x}\boldsymbol{s})$, and p differ.

Definition 7. Let $p \in \mathbb{F}_2$ and $S \subset R$. We define the P transform as follows:

$$P_f(S, p) = \prod_{\boldsymbol{x} \in \mathbb{F}_2^n} \mathbf{1} \oplus C_f(S, p, \boldsymbol{x})\, \delta(\boldsymbol{x}|S),$$

where $\mathbf{1}$ is the length $2^{|S|}$ vector with every component equal to 1.

For any $\boldsymbol{b} \in \mathbb{F}_2^{|S|}$, $P_f(S, p)_{v(\boldsymbol{b})} = 0$ exactly when there exists an $\boldsymbol{a} \in \mathbb{F}_2^n$ such that $\boldsymbol{a}|S = \boldsymbol{b}$ and either $f(\boldsymbol{a}) \neq f(\boldsymbol{a}\, m(S))$ or $f(\boldsymbol{a}) \neq p$. Equivalently, $P_f(S, p)_{v(\boldsymbol{b})} = 1$ exactly when the

tuple count $C(f = p \oplus 1 \mid S = \boldsymbol{b}) = 0$. Furthermore, $P_f(S, p)$ can be considered as the truth table for a function q' of the variables in S, defined by letting $q'(\boldsymbol{y}) = P_f(S, p)_{v(\boldsymbol{y})+1}$ for each $\boldsymbol{y} \in \mathbb{F}_2^{|S|}$. It can be then be extended to an equivalent n-variable function q by defining $q(\boldsymbol{x}) = q'(\boldsymbol{x}|S)$ for each $\boldsymbol{x} \in \mathbb{F}_2^n$. This function will prevent pairs of the form $(p \oplus 1, 1)$. The function $q(\boldsymbol{x}) \oplus 1$ will similarly prevent pairs of the form $(p \oplus 1, 0)$.

Remark 1. Although the Boolean operation "or" is a less trivial vector operation than multiplication, we can use it to create an equivalent definition of the P-transform that may be more practical for understanding or implementation:

$$\mathbf{1} \oplus \bigvee_{\boldsymbol{x} \in \mathbb{F}_2^n} C_f(S, p, \boldsymbol{x})\, \delta(\boldsymbol{x}|S)$$

Furthermore, the multiplication by $\delta(\boldsymbol{x}|S)$ could potentially be described in terms of a binary shift operation.

Note that every pair-prevention function $q' \in \mathcal{B}_{|S|}$ depending only on the variables in S will have a truth table vector of the following form: $P_f(S, p)$ with any of its 1 values (but not all) optionally replaced by 0, and with a constant function $\mathbf{1}$ optionally added to it. The P-transform values for $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3 x_4$ are depicted in Tables 1 and 2.

6 Conclusion

In this paper, we have reviewed the existing knowledge surrounding correlation attacks on LFSR-based stream ciphers with nonlinear combiners. Although these details are well established, we are not aware of other authors who explicitly detail the probabilities involved in each possible case. Along with these results, we have introduced a concrete method for deriving the optimal q function to use in any classical correlation attack. In addition to this, we have introduced a novel attack procedure that exploits the existence of "impossible pairs" for f and q functions that admit them. Finally, we have developed the theory necessary to

	S	$P_f(S,0)$	Corresponding q function
	$\emptyset$	0	
	(1)	00	
	(2)	00	
	(3)	00	
	(4)	00	
	(1, 2)	0000	
	(1, 3)	0000	
$\boldsymbol{p = 0}$:	(1, 4)	0000	
	(2, 3)	0000	
	(2, 4)	0000	
	(3, 4)	0000	
	(1, 2, 3)	10000010	$1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3$
	(1, 2, 4)	10000010	$1 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_1x_4 \oplus x_2x_4$
	(1, 3, 4)	00000000	
	(2, 3, 4)	00000000	
	(1, 2, 3, 4)	1110000100011110	$1 \oplus x_1 \oplus x_2 \oplus x_3x_4$

Table 1: The vectors produced by the P-transform with $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3x_4$ and $p = 0$. When the resulting vector is non-constant, the corresponding q function is included.

	S	$P_f(S,1)$	Corresponding q function
	$\emptyset$	0	
	(1)	00	
	(2)	00	
	(3)	00	
	(4)	00	
	(1, 2)	0000	
	(1, 3)	0000	
$\boldsymbol{p = 1}$:	(1, 4)	0000	
	(2, 3)	0000	
	(2, 4)	0000	
	(3, 4)	0000	
	(1, 2, 3)	00101000	$x_1 \oplus x_2 \oplus x_1x_3 \oplus x_2x_3$
	(1, 2, 4)	00101000	$x_1 \oplus x_2 \oplus x_1x_4 \oplus x_2x_4$
	(1, 3, 4)	00000000	
	(2, 3, 4)	00000000	
	(1, 2, 3, 4)	0001111011100001	$x_1 \oplus x_2 \oplus x_3x_4$

Table 2: The vectors produced by the P-transform with $f(\boldsymbol{x}) = x_1 \oplus x_2 \oplus x_3x_4$ and $p = 1$. When the resulting vector is non-constant, the corresponding q function is included.

determine exactly when it will be possible to exploit such pairs. This included the introduction of the P-transform in order to explicitly construct pair-preventing q functions for all susceptible subsets of the registers. Future work may focus on relating the all-cases property to other, better known algebraic properties of Boolean functions, and on the potential use of multiple, independent pair-prevention functions within a single attack.

References

[1] Anne Canteaut and Michaël Trabbia. "Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5". In: *Advances in Cryptology – EUROCRYPT 2000* (2000), pp. 573–588. DOI: https://doi.org/10.1007/3-540-45539-6_40.

[2] C. E. Shannon. "Communication theory of secrecy systems". In: *The Bell System Technical Journal* 28.4 (1949), pp. 656–715. DOI: https://doi.org/10.1002/j.1538-7305.1949.tb00928.x.

[3] T. Siegenthaler. "Decrypting a Class of Stream Ciphers Using Ciphertext Only". In: *IEEE Transactions on Computers* 34.1 (1985), pp. 81–85. DOI: https://doi.org/10.1109/TC.1985.1676518.

[4] Thomas Siegenthaler. "Correlation-immunity of nonlinear combining functions for cryptographic applications (Corresp.)" In: *IEEE Transactions on Information Theory* 30.5 (1984), pp. 776–780. DOI: https://doi.org/10.1109/TIT.1984.1056949.

[5] G-Z. Xiao and James L. Massey. "A spectral characterization of correlation-immune combining functions". In: *IEEE Transactions on information theory* 34.3 (1988), pp. 569–571. DOI: https://doi.org/10.1109/18.6037.