On the Problem of Finding Linear Approximations and Cryptanalysis of Pomaranch Version 2*

Martin Hell and Thomas Johansson

Dept. of Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
{martin,thomas}@it.lth.se

Abstract. We give a simple algorithm that can find biased linear approximations of nonlinear building blocks. The algorithm is useful if the building block is relatively small and exhaustive search is possible. Instead of searching all possible linear relations individually, we show how the most biased relation can be found in just a few steps. As an example we show how we can find a biased relation in the output bits of the stream cipher Pomaranch Version 2, a tweaked variant of Pomaranch, resulting in both distinguishing and key recovery attacks. These attacks will break both the 128-bit variant and the 80-bit variant of the cipher with complexity faster than exhaustive key search.

Keywords: cryptanalysis, linear approximation, Pomaranch, stream ciphers.

1 Introduction

Finding a biased linear approximation of some output bits of a stream cipher can be devastating for the security of the cipher. If the bias of the approximation is sufficiently strong a distinguishing attack can be mounted. In some cases this distinguishing attack can be modified into a key recovery attack which will succeed with computational complexity much lower than exhaustive key search. The process of finding the best approximation usually requires considerable effort from the cryptanalyst. It can be the case that the cipher design does not allow for a comprehensive theoretic analysis and thus a possible biased linear approximation is not found, though it exists. In this paper we suggest a method that finds a linear approximation without considering theoretical aspects of the building blocks of a cipher. The algorithm is suitable if the cipher consists of small building blocks and if the blocks are difficult to analyze theoretically. As

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

E. Biham and A.M. Youssef (Eds.): SAC 2006, LNCS 4356, pp. 220–233, 2007.

an example we show how to find a linear approximation in the output bits of the cipher Pomaranch Version 2.

Pomaranch [4] is one of the candidates in the eSTREAM [3] project. It is designed to be efficient both in hardware and software. It is one of the ciphers that has been moved to the second phase in the hardware category in the eSTREAM project. The cipher is an interesting construction since it introduces a new approach to the design of LFSR based stream ciphers. One common way to introduce nonlinearity into the linearly generated LFSR sequence is to clock the LFSR in an irregular way. Doing this will result in a decimated sequence and the number of bits discarded between consecutive output bits is unknown to an attacker. However, this will introduce some problems. First, an output buffer is needed since the keystream bits will not be produced regularly. Second, for the same reason, the construction is likely to be vulnerable to timing and power attacks. In Pomaranch, the LFSRs are based on a new idea, called jump registers. In each update of the LFSRs, the next state is one of two possible states. Which one it jumps to is key dependent and thus unknown to the attacker. The main advantage is that the update of the ciphers behaves like irregular clocking but the problems involved in irregular clocking are avoided.

Two attacks have been proposed on the first version of Pomaranch. The first attack is a chosen IV attack [1] which succeeds because not all IV bits are diffused into the whole state in the initialization phase. The second attack [8] is a correlation attack on the keystream generation of Pomaranch. This attack uses a biased linear relation in the output bits to mount a key recovery attack. Consequently, because of these two attacks, Pomaranch had to be tweaked. This was done and the tweaked version, referred to as Pomaranch Version 2, was proposed in [5] and [7]. Pomaranch has a key size of 128 bits but since hardware dedicated ciphers are required to support 80 bit keys in the eSTREAM project, a reduced variant was also proposed, supporting 80 bit security level. By using the proposed algorithm we show that the changes made in the tweaked version were not enough. It is still possible to find a linear relation in the output bits that is biased enough to allow distinguishing and key recovery attacks. These attacks are the first on Pomaranch Version 2 and they will break both the 128-bit variant and the 80-bit variant of the cipher with complexity faster than exhaustive key search.

This paper is outlined as follows. Section 2 gives some background theory concerning the problem of binary hypothesis testing. In Section 3 we consider the case of letting a binary linear approximation be written as a binary vector. Section 4 presents an algorithm that finds a linear approximation in certain cases. A practical example is given in Section 5, in which the attack on Pomaranch Version 2 is described. In Section 6 some simulation results are given and in Section 7 we look at a new version of Pomaranch which was designed with our attacks in mind. Finally, the paper is concluded in Section 8.

1.1 Notation

Throughout the paper, random variables $X, Y, \ldots$ are denoted by capital letters and their domains are denoted by $\mathcal{X}, \mathcal{Y}, \ldots$. The realizations of random variables

$x \in \mathcal{X}, y \in \mathcal{Y}, \ldots$ are denoted by small letters. A distribution is denoted by D. The probability function of a random variable X following the distribution D is denoted by $Pr_D[x]$ and sometimes as just $D[x]$.

2 Hypothesis Testing and Distinguishers

In this section we give some background theory and consider the general problem of binary hypothesis testing and distinguishers. For a more thorough treatment of hypothesis testing we refer to [2]. This is a very important concept in cryptology. A distinguisher is a mathematical tool, based on a binary hypothesis tests, which is used very frequently in cryptanalysis. The distinguisher is used not only in distinguishing attacks but it is also often used in key recovery attacks.

In a binary hypothesis test we observe a collection of independent and identically distributed data. Denote the distribution of the observed data by D_{obs}. Further, we have two known distributions, D_0 and D_1. We want to decide whether the observed distribution is actually the distribution D_0 (the null hypothesis H_0) or the distribution D_1 (the alternate hypothesis H_1), knowing that one of them is the true distribution. More formally, we define a *decision rule* which is a function $\delta : \mathcal{X} \to \{0, 1\}$ such that

$$\delta = \begin{cases} 0, & D_{obs} = D_0, \\ 1, & D_{obs} = D_1. \end{cases}$$

The function δ makes a decision for each $x \in \mathcal{X}$. The decision rule divides the domain $\mathcal{X}$ into two regions denoted A and A^c. A is called the *acceptance region* of δ and corresponds to the decision to accept the null hypothesis.

There are two types of errors associated with a binary hypothesis test. We can reject the null hypothesis when it is in fact true, i.e., $\delta : \mathcal{X} \to 1$ when $D_{obs} = D_0$. This is called a type I error, or false alarm and the probability of this error is denoted α. The other alternative is that we accept the null hypothesis when the alternate hypothesis is true, i.e., $\delta : \mathcal{X} \to 0$ when $D_{obs} = D_1$. This is called a type II error, or miss. The probability of this error is denoted by β.

In a binary hypothesis test there are several things that have to be considered. Two important things are how to perform the test in an optimal way and how many samples we need in order to have certain error probabilities, α and β. How to perform the optimal test is given by the Neyman-Pearson lemma.

Lemma 1 (Neyman-Pearson). *Let $X_1, X_2, \ldots, X_m$ be drawn i.i.d. according to mass function D_{obs}. Consider the decision problem corresponding to the hypotheses $D_{obs} = D_0$ vs. $D_{obs} = D_1$. For $T \geq 0$ define a region*

$$A_m(T) = \left\{ \frac{P_0(x_1, x_2, \ldots, x_m)}{P_1(x_1, x_2, \ldots, x_m)} > T \right\}.$$

Let $\alpha_m = D_0^m(\mathcal{A}_m^c(T))$ and $\beta_m = D_1^m(\mathcal{A}_m(T))$ be the error probabilities corresponding to the decision region $\mathcal{A}_m$. Let $\mathcal{B}_m$ be any other decision region with associated error probabilities α^* and β^*. If $\alpha^* \leq \alpha$, then $\beta^* \geq \beta$.

In order to show how the error probabilities are related to the distributions and the number of samples we first define the relative entropy between two distributions.

Definition 1. *The relative entropy between two probability mass functions* $Pr_{D_0}[x]$ *and* $Pr_{D_1}[x]$ *over the same domain* $\mathcal{X}$ *is defined as*

$$D\left(Pr_{D_0}[x] \| Pr_{D_1}[x]\right) = \sum_{x \in \mathcal{X}} Pr_{D_0}[x] \log \frac{Pr_{D_0}[x]}{Pr_{D_1}[x]}.$$

In literature the relative entropy is sometimes also referred to as information divergence or Kullback-Leibler distance. For convenience and ease of reading, in the following we write the relative entropy as $D(D_0 \| D_1)$ or $D(D_0[x] \| D_1[x])$. Note that in general $D(D_0 \| D_1) \neq D(D_1 \| D_0)$.

No general expression for the error probabilities α and β exists. Hence, we know how to perform the optimal test, but we do not know the performance of the test. However, there exist asymptotic expressions for the error probabilities, i.e., expressions that hold when the number of samples is large. The relative entropy is related to the asymptotic error probabilities through Stein's lemma. Stein's lemma states that if we fix the error probability $\alpha < \epsilon$ then

$$\beta = 2^{-nD(D_0 \| D_1)}.$$

The value of α does not affect the exponential rate at which β decreases and according to Stein's lemma, this situation always occurs. Thus, the number of samples needed in a binary hypothesis test is a constant times the inverse of the relative entropy between the two distributions.

In order to have an asymptotic expression that minimizes the overall probability of error $P_e = \pi_1 \alpha + \pi_2 \beta$, where π_i are the *a priori* probabilities of the distributions, the Chernoff information, $C(D_0, D_1)$, can be used. The error probability can then be written as

$$P_e = 2^{-nC(D_0, D_1)},$$

where

$$C(D_0, D_1) = -\min_{0 \leq \lambda \leq 1} \log \left(\sum_x Pr^{\lambda}_{D_0}[x] Pr^{1-\lambda}_{D_1}[x] \right).$$

The Chernoff information between two distributions is often hard to find since we have to minimize over λ. A common way of avoiding this is to pick a value of λ, e.g., $\lambda = 0.5$. This will give a lower bound for the Chernoff information and thus an upper bound for the error probability. In this paper we will only consider the relative entropy between the distributions and the number of samples needed will be computed as

$$\#\text{ Samples needed} = \frac{1}{D(D_0 \| D_1)}.$$

A larger relative entropy means fewer samples for a successful distinguisher.

3 Vectorial Representation of a Linear Approximation

In this section we consider the advantage of representing a biased linear relation as a vector instead of as a binary relation. We give two propositions which will lead us to a simple algorithm that helps us to find biased linear relations. Both propositions follow from basic information theory.

Proposition 1. *Let the vector* $(z_{t_1}, z_{t_2}, \ldots, z_{t_m})$ *follow the size* 2^m *distribution* D_0*. Let* $z = z_{t_1} \oplus z_{t_2} \oplus \ldots \oplus z_{t_m}$ *follow the marginal distribution* D_0'*. Then*

$$D\left(D_0[x] \| D_1[x]\right) \geq D\left(D_0'[x] \| D_1'[x]\right)$$

for some size 2^m *distribution* D_1 *with corresponding marginal distribution* D_1'*.*

Proof. Denote by $a_i^{(e)}$ $(0 \leq i < 2^{m-1})$ the probabilities of the vectors in D_0 with even Hamming weight and correspondingly by $a_i^{(o)}$ $(0 \leq i < 2^{m-1})$ the probabilities of the vectors with odd Hamming weight. Similarly, the probabilities of the vectors in distribution D_1 will be denoted $b_i^{(e)}$ and $b_i^{(o)}$ $(0 \leq i < 2^{m-1})$ respectively. Hence, we want to show that

$$\sum_i a_i^{(e)} \log \frac{a_i^{(e)}}{b_i^{(e)}} + \sum_i a_i^{(o)} \log \frac{a_i^{(o)}}{b_i^{(o)}} \geq \left(\sum_i a_i^{(e)}\right) \log \frac{\sum_i a_i^{(e)}}{\sum_i b_i^{(e)}} + \left(\sum_i a_i^{(o)}\right) \log \frac{\sum_i a_i^{(o)}}{\sum_i b_i^{(o)}}. \quad (1)$$

Let $E\{\cdot\}$ denote the expected value. Jensen's inequality states that for a strictly convex function f and random variable X it holds that

$$E\{f(X)\} \geq f(E\{X\}),$$

with equality if and only if X is a constant. We use the fact that $t \log t$ is a strictly convex function and introduce $\alpha_i = Pr(t \log t = t_i)$. If we consider only the first term on the left hand side and right hand side of (1) and putting $t_i = a_i^{(e)}/b_i^{(e)}$ we can write

$$\sum_i \alpha_i \frac{a_i^{(e)}}{b_i^{(e)}} \log \frac{a_i^{(e)}}{b_i^{(e)}} \geq \left(\sum_i \alpha_i \frac{a_i^{(e)}}{b_i^{(e)}}\right) \log \sum_i \alpha_i \frac{a_i^{(e)}}{b_i^{(e)}}.$$

This will hold for any choice of α_i as long as $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. Hence we can put $\alpha_i = b_i^{(e)} / \sum_i b_i^{(e)}$ and it follows that

$$\sum_i a_i^{(e)} \log \frac{a_i^{(e)}}{b_i^{(e)}} \geq \left(\sum_i a_i^{(e)}\right) \log \frac{\sum_i a_i^{(e)}}{\sum_i b_i^{(e)}}.$$

Doing the same thing for the second terms in (1) will end the proof. □

Proposition 1 implies that we can never lose anything by considering a linear approximation as a binary vector. Moreover, if the distribution of the vectors is such that for all vectors with even Hamming weight the probability is the same and for all vectors with odd Hamming weight the probability is the same, then there is no additional gain in using vectorial representation. We continue the analysis of vectorial representation with the following proposition.

Proposition 2. *Assume that we have a binary vectorial distribution of size* 2^m *denoted* D_0 *and the size* 2^m *uniform distribution* D_1*. Adding a variable to the length* m *vector that is statistically independent with all other variables will not affect the relative entropy between two distributions. Furthermore, adding a variable that is correlated with other variables will increase the relative entropy.*

Proof. Let $\mathbf{x} = (x_0, x_1, \ldots, x_{m-1})$. Using the chain rule for relative entropy we write

$$D\left(D_0[\mathbf{x}, x_m] \parallel D_1[\mathbf{x}, x_m]\right) = \\ D\left(D_0[\mathbf{x}] \parallel D_1[\mathbf{x}]\right) + D\left(D_0[x_m|\mathbf{x}] \parallel D_1[x_m|\mathbf{x}]\right)$$

where the last term is zero if and only if $Pr_{D_0}[x_m|\mathbf{x}] = 1/2,\ x_m \in \{0, 1\}$. □

In the next section we show how these results can be used to find biased linear approximations of nonlinear blocks.

4 Finding a Biased Linear Approximation

In this section the theory developed in Section 3 is used to find biased linear approximations. If we have a linear approximation $x_{t_1} \oplus x_{t_2} \oplus \ldots \oplus x_{t_\mu}$ and add a variable, $x_{t_{\mu+1}}$, that is uniformly distributed and statistically independent with the other variables, the distribution of the resulting approximation is uniform and the approximation is useless. It can not be used in a distinguisher. This is the idea behind a common design principle in stream ciphers, i.e., masking the output with some variable that is assumed to be uniformly distributed. Then the output is guaranteed to be uniform. If we instead write the approximation as a vector, $(x_{t_1}, x_{t_2}, \ldots, x_{t_\mu})$, then, according to Proposition 2, it does not matter how many uniformly distributed and independent variables we add to the vector. As long as all variables from the approximation are present, the relative entropy will never decrease and the vector can be used in a distinguisher.

Consider a cipher containing a relatively small building block B. If the distribution of the output bits, or the distribution of a linear equation of some output bits, can be found, then it is easy to search through all linear relations in order to determine which is most biased. However, as the amount of output bits to be considered increases, the number of possible equations increases exponentially. Considering m consecutive output bits there are 2^m possible equations. If no biased equation is found among these equations, one more output bit has to be considered and an additional 2^m equations has to be checked. If checking one equation requires 2^k computation steps, checking all equations involving m bits will require a computational complexity of 2^{k+m}. Instead, by considering the output bits as a vector, the computational complexity will be limited to 2^k. On the other hand, the memory complexity will be 2^m since the distribution of a length m vector needs to be kept in memory.

Assume that the building block B has a biased linear relation involving some of the output bits $x_0, x_1, \ldots, x_{m-1}$ with a significantly larger bias than

any linear relation involving less than m consecutive output bits. Let $\mathbf{x_m} = (x_0, x_1, \ldots, x_{m-1})$. Then

$$D\left(D_0[\mathbf{x_m}] \parallel D_1[\mathbf{x_m}]\right) \gg D\left(D_0[\mathbf{x_{m-1}}] \parallel D_1[\mathbf{x_{m-1}}]\right). \tag{2}$$

Hence, we can start with the vector $\mathbf{x_1} = (x_0)$ and add one extra variable at a time. If, at step m, considering the vector $\mathbf{x_m} = (x_0, x_1, \ldots, x_{m-1})$, the relative entropy between the distribution of $\mathbf{x_m}$ and the uniform distribution increases significantly, we know that there is a biased linear relation involving at least the bits x_0 and x_{m-1}. In order to find the other bits in the biased linear relation we consider the vector $\mathbf{x_m^{(i)}}$ which we define as the vector $\mathbf{x_m}$ with the variable X_i removed. If

$$D\left(D_0[\mathbf{x_m^{(i)}}] \parallel D_1[\mathbf{x_m^{(i)}}]\right) \approx D\left(D_0[\mathbf{x_m}] \parallel D_1[\mathbf{x_m}]\right) \tag{3}$$

then the variable X_i is *not* present in the linear approximation. Note that we do not use equality in (3) since the variable i can be present in some biased linear relation but still not in the most biased relation. We are only interested in the most biased relation and if the variable X_i is present in this relation the decrease in relative entropy will be significant, not just approximate. Of course we can continue increasing the length of the vector, hoping to find even better linear approximations.

5 Application to the Stream Cipher Pomaranch Version 2

In this section we show how we can use the algorithm described in Section 4 to find a heavily biased linear relation in the stream cipher Pomaranch Version 2. We also show that the existence of this linear relation makes it possible to mount both distinguishing and key recovery attacks on the cipher.

5.1 Description of Pomaranch

The attack described in this paper is independent of the initialization procedure. Because of this, only the keystream generation of Pomaranch will be described here. For a more detailed description of Pomaranch we refer to [5]. The 128-bit variant of Pomaranch is based on 9 jump registers, denoted R_i $(1 \leq i \leq 9)$, see Fig 1.

All registers are identical and of length 14. Half of the register cells are normal delay shift cells and half are feedback cells. The current operation of each cell (shift or feedback) is determined by a jump control bit, JC. Each clock, every jump register is updated in one of two ways. The state is multiplied by the transition matrix A $(JC = 0)$, or it is multiplied by $A + I$ $(JC = 1)$. The state of the registers R_1 to R_8 are filtered through a key dependent function, f_{K_i}, producing outputs c_1 to c_8. The key is divided into 8 parts of 16 bits each and part i is used in f_{K_i}. The jump control for register i, denoted JC_i, is then calculated as

$$JC_i = c_1 \oplus \ldots \oplus c_{i-1}, \quad (i = 2, \ldots, 9).$$

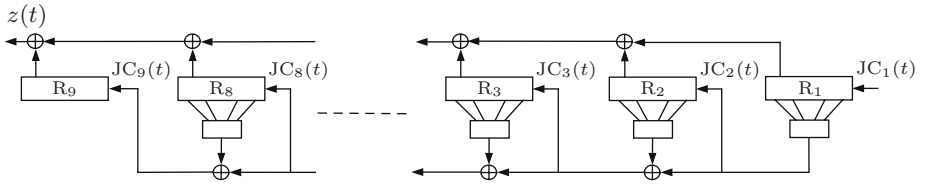

Fig. 1. Overview of Pomaranch (128-bit variant)

The jump control for the first register, JC_1 is always set to 0. The keystream at time t, denoted $z(t)$, is taken as the binary xor of all 9 bits at position 13, denoted $r_i(t)$, in the registers. The 80-bit version is identical to the 128 bit version except that the number of registers is 6 instead of 9.

The design of the second version of Pomaranch is similar to the first version. The differences are the transition matrix A and the choice of taps taken to the function f_{K_i}. Also the IV setup has been changed in order to resist the chosen IV attack in [1].

5.2 Previous Attack on Pomaranch Version 1

The size of the registers is only 14. This suggests that it might be possible to mount a divide-and-conquer kind of attack on the cipher. Since the registers are updated linearly each new output bit will be linearly dependent on the initial state bits. Hence, for any given JC-sequence of length 14 there will be a linear relation in 15 output bits that will always hold, i.e., there is an array $\ell = (\ell_0, \ell_1, \ldots, \ell_{14})$, $\ell_i \in \{0, 1\}$ such that $\sum_{i=0}^{14} \ell_i r(t+i) = 0$. It was shown in [8] that these relations are not evenly distributed among all possible 2^{14} JC-sequences of length 14. Hence, assuming that the JC-sequence is purely random, there will be linear relations in the output bits that are biased. The short register length allows for exhaustive search among all relations and the most biased relation could easily be found.

5.3 New Attack on Pomaranch Version 2

In this section we show that it is still possible, using our proposed algorithm, to find linear relations in the output bits that can be used in an attack. In [7] a theoretical analysis was done based on the attack in [8]. Let $C(x)$ be the characteristic polynomial of A. Then for a given jump control sequence we have the equality

$$\sum_{i=0}^{L} \ell_i x^{i-k_i} (x+1)^{k_i} = C(x),$$

```
for all 2^14 − 1 initial states
    for all 2^(i−1) possible JC-sequences
        clock the jump register i − 1 times
        Dist[(z_0, ..., z_(i−1))]++
    end for
end for
```

Fig. 2. Algorithm to find the distribution for i consequtive output bits. The actual implementation can be recursive to make it faster.

where k_i is the binary weight of the vector $(JC(t), \ldots, JC(t+i-1))$. Using this equality a straightforward $O(L)$ approach to find the values of ℓ_i was described, giving the linear relation for this particular jump control sequence. Hence the most common linear relation among the 2^L sequences was found in $O(L2^L)$. The problem with this analysis is that it only considers relations of length $L+1$. The same analysis is not applicable to relations involving bits further apart since the characteristic polynomial of A is of degree L. Consequently, the design parameters for Pomaranch Version 2 are optimized for the cipher to resist correlation attacks based on relations of length $L+1$ and it successfully does so.

Unfortunately it is not enough to only consider these relations. It is possible that there are relations involving bits further apart that are more biased than any relation involving only bits $L+1$ positions apart. The algorithm proposed in Section 4 seems very suitable for Pomaranch. The shift registers are of length 14 which is easy to search exhaustively. Moreover, all registers are identical. Register R_1 will have $JC_1(t) = 0$, $\forall t$ and will thus behave like a regularly clocked shift register with primitive feedback polynomial. Hence, the register R_1 will not be considered using our algorithm. Instead, it will be exhaustively searched. Registers R_i $(2 \leq i \leq 9)$ will have JC_i determined by a key dependent function. Since these registers are identical, finding a biased linear approximation in the output bits of one register means that all other registers (except R_1) will have this biased approximation. In order to find a good approximation, we look at vectors of consecutive output bits. When calculating the bias of these vectors, the following three assumptions will be used:

1. All states of the registers will have the same probability, except the all-zero state which has probability 0.
2. All JC-sequences will have the same probability.
3. All jump sequences, JC_i $(2 \leq i \leq 9)$, are independent.

Since the shift registers are of length 14, the first vector length we check is 15. The algorithm used to find the distribution for i consequtive output bits are given in Figure 2. The output of Pomaranch is given as the XOR of the output bits of the registers. Hence, we need to find the bias of the xor of all 8 distributions. For k

Table 1. Relative entropy between output vectors and the random distribution

Vector Length	$D(D_0 \| D_1)$
15	$2^{-111.914}$
16	$2^{-108.603}$
17	$2^{-107.671}$
18	$2^{-107.108}$
19	$2^{-75.849}$
20	$2^{-74.849}$
21	$2^{-74.264}$
22	$2^{-73.849}$
23	$2^{-73.527}$

Table 2. Relative entropy when bit i is excluded from the vector $\mathbf{x_{19}}$

i	$x_{19}^{(i)}$	i	$x_{19}^{(i)}$
1	$2^{-75.851}$	10	$2^{-75.849}$
2	$2^{-105.383}$	11	$2^{-75.849}$
3	$2^{-75.849}$	12	$2^{-75.849}$
4	$2^{-75.849}$	13	$2^{-75.849}$
5	$2^{-76.077}$	14	$2^{-75.849}$
6	$2^{-105.264}$	15	$2^{-75.849}$
7	$2^{-75.849}$	16	$2^{-75.849}$
8	$2^{-75.849}$	17	$2^{-76.849}$
9	$2^{-75.849}$		

distributions, this can easily be done in $O(k2^{2n})$ time, where n is the size in bits for each random variable. This can be a bottleneck if the vectors are very large. A much more efficient algorithm for finding the distribution of a sum of random variables was given in [9]. They show that the distribution for $Pr(X_1 \oplus X_2 \oplus \ldots \oplus X_k)$ can be found in $O(kn2^n)$ time where all X_i are n-bit random variables. This algorithm was adopted in our implementation. The relative entropies between the vectors of length 15 to 23 and the random distribution have been given in Table 1.

We see that $D(D_0 \| D_1)$ increases significantly when the vector reaches length 19. The fact that $D(D_0 \| D_1)$ is much higher for $\mathbf{x_i}$ $(i \geq 19)$ tells us that there might exist a heavily biased linear approximation involving the bits x_0, x_{18} and zero or more bits x_i $(1 \leq i \leq 17)$. To find the particular linear approximation that allows us to attack the cipher we look at $D\left(D_0[\mathbf{x_{19}^{(i)}}] \parallel D_1[\mathbf{x_{19}^{(i)}}]\right)$, as suggested by our proposed algorithm. The result is given in Table 2.

We see that when X_2 or X_6 are removed, the relative entropy is almost as when $\mathbf{x_{18}}$ was considered. This implies that the heavily biased linear relation is

$$z(t) \oplus z(t+2) \oplus z(t+6) \oplus z(t+18) = 0. \tag{4}$$

It is possible that the figures in Table 2 stems from the fact that the vector $(z(t), z(t+2), z(t+6), z(t+18))$ is heavily biased but the xor of the variables has a much smaller bias or even no bias at all. However, checking the relative entropy between (4) and random, which is $2^{-77.080}$, confirms that the figures in Table 2 stems from the biased linear relation. In any cipher, constructed using jump registers together with some other function, this relation would have been an important part of linear cryptanalysis. In the specific case of Pomaranch, which uses the output bits from the jump registers immediately in the output function it is even better to consider the total vector of as many bits as possible, since this will give us more information. Though, most of this information stems immediately from this relation. The discovery of the biased linear relation given by our algorithm may help the designers to get additional theoretical knowledge about the design principle of the Pomaranch family of stream ciphers.

5.4 Distinguishing and Key Recovery Attacks

In this section we give the details and complexities of the attacks on Pomaranch Version 2. Using output vectors of length 23, which is the largest vectors we were able to find the distribution for, the relative entropy $D(D_0[\mathbf{x_{23}}] \| D_1[\mathbf{x_{23}}]) = 2^{-73.527}$. The amount of keystream needed in order to distinguish these distributions is about $1/D(D_0[\mathbf{x_{23}}] \| D_1[\mathbf{x_{23}}]) = 2^{73.527}$. Note that the output of register R_1, which always has its jump control sequence $JC_1(t) = 0 \ \forall t$, is not considered in these distributions. Instead the starting state of this register is guessed in the attack. This will add a factor of 2^{14} to the computational complexity. Hence, a distinguishing attack on Pomaranch Version 2 can be mounted using $2^{73.527}$ keystream bits and a computational complexity of $2^{73.527}2^{14} = 2^{87.527}$.

This proposed distinguisher can be turned into a key recovery attack by guessing 16 bits of the key at a time. By mounting the distinguishing attack we will obtain the state of register R_1. In the next step we guess the 16 key bits used in the function f_{K_1} and the 14 bit state of register R_2. The stream generated by the other 7 registers now have relative entropy $2^{-63.897}$ compared to the random distribution. The computational complexity for finding the 16 bits of the key will be $2^{63.897}2^{30} = 2^{93.897}$. Finding the remaining bits will have lower complexity since the bias will increase as we know more register states while we still have to guess 30 bits each time. Hence the key recovery attack will succeed with $2^{73.527}$ keystream bits and computational complexity $2^{93.897}$. The 80-bit hardware oriented variant of Pomaranch Version 2 has the same structure as the 128-bit variant with the exception that only 6 registers are used. The corresponding distinguishing attack on the 80-bit version will require $2^{44.587}$ bits of keystream and a computational complexity of $2^{58.587}$. The key recovery attack will require

Table 3. Complexities of the attacks proposed in this paper

Attack Complexities			
Type of Attack	Variant	Amount of keystream needed	Computational Complexity
Distinguishing Attack	80 bits	$2^{44.587}$	$2^{58.587}$
	128 bits	$2^{73.527}$	$2^{87.527}$
Key recovery Attack	80 bits	$2^{44.587}$	$2^{64.882}$
	128 bits	$2^{73.527}$	$2^{93.897}$

$2^{44.587}$ keystream bits and has a computational complexity of $2^{64.882}$. Table 3 summarizes all proposed attacks on Pomaranch Version 2 and the corresponding complexities.

6 Simulation Results

When the distribution for the output vectors was theoretically calculated, we assumed that the initial states had the same probability, that all jump control sequences had the same probability and that all jump control sequences JC_i $(2 \leq i \leq 9)$ were independent. To verify these assumptions, the real distribution was simulated for scaled down variants of the cipher. An interesting question here is the amount of samples that is needed in the simulation. Let the simulated distribution of the output from the jump registers be denoted D_0^*. As before, the *theoretical* distribution using the assumptions above is denoted D_0 and the uniform distribution is denoted D_1. The size of the distributions is denoted $|\mathcal{X}|$. We use the following theorem taken from [2].

Theorem 1. *Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed according to D_0. Then*

$$Pr\{D(D_0^* \| D_0) > \epsilon\} \leq 2^{-n\left(\epsilon - |\mathcal{X}| \frac{\log(n+1)}{n}\right)}. \tag{5}$$

If $D(D_0 \| D_1) = \mu$, then $\epsilon \leq \mu/2$. To reach the amount of samples where the error probability in (5) is less than or equal to 1, n should satisfy

$$n\left(\mu/2 - |\mathcal{X}| \frac{\log(n+1)}{n}\right) \geq 0 \Rightarrow \frac{n}{\log(n+1)} \geq \frac{2|\mathcal{X}|}{\mu}. \tag{6}$$

It is clear that the amount of samples needed increases exponentially with the vector length and in order to be able to simulate more than one register, we chose to simulate the distribution for the linear relation (4) instead of vectors. (Note that in the hypothesis test used in the cryptanalysis, this is not the case. The amount of samples needed then is still in the order of $1/D(D_0 \| D_1)$. Stein's lemma is still applicable.) We found the distribution for 1, 2 and 3 registers.

Table 4. Simulated values for the distributions

Jump Registers Used	$D(D_0 \| D_1)$	
	Theoretical	Simulated
1	$2^{-10.05}$	$2^{-9.82}$
2	$2^{-19.62}$	$2^{-19.36}$
3	$2^{-29.20}$	$2^{-29.27}$

Using a random key, register R_1 was fed with the all zero JC-sequence. The output was taken as the xor of the output of the other registers, i.e. R_2, $R_2 \oplus R_3$ and $R_2 \oplus R_3 \oplus R_4$ respectively. In all cases, the amount of samples used was 2^{39} which, according to (6), should be enough. The simulated values for the relative entropy can be found in Table 4. From the simulated values, we conclude that the assumptions made in the calculation of the theoretical distributions are valid at least up to 3 registers. From this it should be safe to assume that the theoretical distributions are valid also up to 8 registers. Thus, the theoretical distributions can be used in the hypothesis test.

7 Pomaranch Version 3

Following the results given in a preliminary version of this paper, the designers of Pomaranch proposed a new, improved version of Pomaranch, denoted Pomaranch Version 3 [6]. Before we conclude, we briefly look at the design of this version and see how our results apply to it.

The overall design is kept almost the same but there are a few important changes. First, the length of the registers is increased to 18 instead of 14 as in Version 2. Second, there are two different register types used. Register type 1 is used in the odd numbered jump registers and type 2 is used in the even numbered registers. The two types differ in both feedback polynomials and selection of S-cells and F-cells. A third change is that, in the 80-bit variant, the xor of the outputs from registers 1 to 5 is replaced with a nonlinear function G. The output of G is then xored with the output of register 6 to form the keystream.

Considering binary linear relations in the output bits of the registers, even if a highly biased relation is found it is very likely that this relation will be different for the two types of registers. Combining these two relations in order to get a biased sum of keystream bits will result in an extremely small bias. A second option is to find the same biased linear relation in both registers. However, what is highly biased in one register might have a small bias in the other. By instead considering binary vectors as have been done in this paper, the fact that there are two different types of registers is not a problem. The same variables will be used in vectors from both types of registers and the information from the most biased linear relation will be in each vector. Though, for the attack to be successful, it is still required that the vectors have a high bias. We have computed the bias of output vectors of different lengths for 8 registers, 4 of each type, and it is

clear that the resulting bias is too low. The approach which proved to be very successful on Pomaranch Version 2 will not be successful on Version 3 and we can conclude that Pomaranch Version 3 is immune to the attack described in this paper.

8 Conclusion

We proposed a new simple algorithm that can be helpful in finding linear approximations of a nonlinear block. The algorithm is suitable for small blocks that can be exhaustively searched. As an example we demonstrate how to find a heavily biased linear approximation for the stream cipher Pomaranch Version 2. The linear approximation gives us information that can be used in a distinguishing or a key recovery attack. In the specific case of Pomaranch, we could get even more information by considering the full vector of output bits instead of just the linear relation provided by our algorithm. In a different setting, the linear relation might be used as a part of the cryptanalysis, possibly involving more building blocks. The linear relation provided by our algorithm might help getting a better theoretical understanding of the design principles of the Pomaranch family of stream ciphers. This further theoretical analysis is left as future work.

References

1. Cid, C., Gilbert, H., Johansson, T.: Cryptanalysis of Pomaranch. IEE Proceedings - Information Security 153(2), 51–53 (2006)
2. Cover, T., Thomas, J.A.: Elements of Information Theory. Wiley series in Telecommunication. Wiley (1991)
3. ECRYPT. eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932. Available at `http://www.ecrypt.eu.org/stream/`
4. Jansen, C.J.A, Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator (CJCSG). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/022
5. Jansen, C.J.A., Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator and Pomaranch stream cipher (version 2). eSTREAM, ECRYPT Stream Cipher Project, Report 2006/006 (2006), `http://www.ecrypt.eu.org/stream`
6. Jansen, C.J.A, Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator and Pomaranch stream cipher (version 3). eSTREAM, ECRYPT Stream Cipher Project (2006), `http://www.ecrypt.eu.org/stream`
7. Jansen, C.J.A., Helleseth, T., Kholosha, A.: Pomaranch - design and analysis of a family of stream ciphers. In: The State of the Art of Stream Ciphers, Workshop Record, SASC 2006, Leuven, Belgium (February 2006)
8. Khazaei, S.: Cryptanalysis of pomaranch (CJCSG). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/065 (2005), `http://www.ecrypt.eu.org/stream`
9. Maximov, A., Johansson, T.: Fast computation of large distributions and its cryptographic applications. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 313–332. Springer, Heidelberg (2005)