Two General Attacks on Pomaranch-like Keystream Generators

Håkan Englund, Martin Hell and Thomas Johansson

Dept. of Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden

Abstract. Two general attacks that can be applied to all versions and variants of the Pomaranch stream cipher are presented. The attacks are demonstrated on all versions and succeed with complexity less than exhaustive keysearch. The first attack is a distinguisher which needs keystream from only one or a few IVs to succeed. The attack is not only successful on Pomaranch Version 3 but has also less computational complexity than all previously known distinguishers for the first two versions of the cipher. The second attack is an attack which requires keystream from an amount of IVs exponential in the state size. It can be used as a distinguisher but it can also be used to predict future keystream bits corresponding to an IV if the first few bits are known. The attack will succeed on all versions of Pomaranch with complexities much lower than previously known attacks.

Keywords: Stream ciphers, distinguishing attack, resynchronization attack, eSTREAM, Pomaranch.

1 Introduction

Pomaranch is one of many cipher constructions in the eSTREAM stream cipher project. The Pomaranch family consists of several versions and variants. The first two versions have been cryptanalyzed in [2, 10, 6]. For each new version, the cipher has been changed such that the attacks on the previous versions would not be successful.

In this paper we present two general attacks that can be applied to all versions and variants of the Pomaranch family of stream cipher. The first attack is a statistical distinguisher, which can be applied to all Pomaranch-like ciphers having one or several types of jump registers and both linear and nonlinear filter function. We improve the computational complexity of all known distinguishers on Version 1 and Version 2. Our attack is also applied to Pomaranch Version 3 and it is shown that the attack will succeed on the 80-bit variant with computational complexity $2^{71.10}$, significantly less than exhaustive key search. For the 128-bit variant, the attack will have computational complexity 2^{126}, almost that of exhaustive key search. The complexity of the attack given in the theorems can be seen as design criteria for subsequent versions of Pomaranch.

The second attack is an IV attack. It first stores many samples from the keystream corresponding to one IV in a table. Given short keystream samples from several other IVs, we can find collisions with the samples in the stored table. When a collision is found, future keystream bits can be predicted. The key will not be recovered but the attack is still more powerful than a distinguisher. Also this attack can be applied to all versions and variants of Pomaranch, e.g., by using a table of size $2^{71.3}$ bytes, 2^{98} different IVs and a computational complexity of 2^{104}, the 128 bit version of Pomaranch Version 3 can be attacked.

The outline of the paper is as follows. Section 2 will describe the Pomaranch stream ciphers and Section 3 will briefly describe the previous attack on Pomaranch. In Section 4 the first attack is given and in Section 5 we give the second attack. Section 6 concludes the paper.

2 Description of Pomaranch

In this section, we will give a brief overview of the design of Pomaranch. There are 3 versions of Pomaranch. First the overall design idea is presented and then we give the specific parameters for the different versions. The attacks described in this paper are independent of the initialization procedure and thus, only the keystream generation will be described here. For more details we refer to the respective design documents, see [7–9].

Pomaranch is a synchronous stream cipher designed primarily for being efficient in hardware. It follows the classical design of a filter generator where the contents of an internal state is filtered through a Boolean function to produce the keystream. The design of Pomaranch is illustrated in Figure 1.

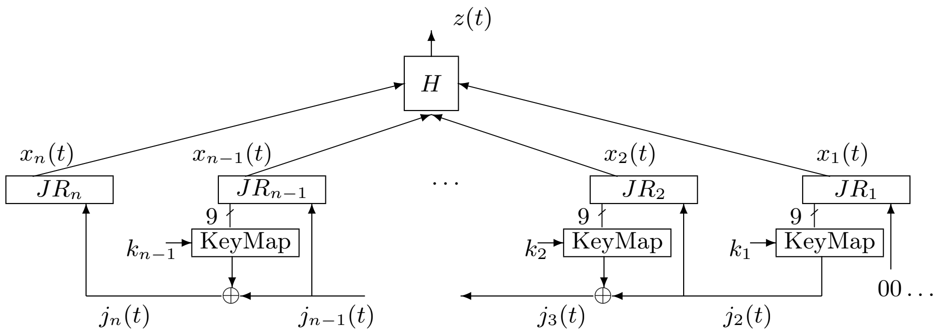

Fig. 1. General overview of Pomaranch.

Pomaranch is based on a cascade of n Jump Registers (JR). A jump register can be seen as an irregularly clocked Linear Finite State Machine (LFSM). The clocking of a register can be done in one of two ways, either it jumps c_0 steps or it jumps c_1 steps. The clocking is decided by a binary jump control sequence, denoted by $j_i(t)$ for register i at time t.

$$j_i(t) = \begin{cases} 0, \; JR_i \text{ is clocked } c_0 \text{ times} \\ 1, \; JR_i \text{ is clocked } c_1 \text{ times} \end{cases}$$

The jump registers are implemented using two different kinds of delay shift cells, S-cells and F-cells. The S-cell is a normal D-element and the F-cell is a D-element where the output is fed back and XORed with the input. Half of the cells are implemented as S-cells and half are implemented as F-cells. When the register is clocked it will jump c_0 steps. When the jump control $j_i(t)$ is one, all cells in JR_i are switched to the opposite mode, i.e., all S-cells become F-cells and vice versa. This switch of cells results in a jump through the state space corresponding to c_1. The jump index of a register is the number of c_0 clockings that is equivalent to one c_1 clocking.

Notations and Assumptions. As seen in Figure 1 we denote Jump Register i by JR_i and its period by T_i. The key is denoted by K and the subkey used for JR_i is denoted k_i. The length of the registers is denoted by L. The filter function is denoted by H (more specifically we will use the notation $H^{(1)}, H^{(2)}_{80}, H^{(2)}_{128}, H^{(3)}_{80}, H^{(3)}_{128}$ for the different filter functions used, where the superscript denotes the version of Pomaranch of interest, and the subscript denotes the key length used in the version). Similarly we will denote the number of registers used by $n^{(1)}, n^{(2)}_{80}, n^{(2)}_{128}, n^{(3)}_{80}, n^{(3)}_{128}$. The bit taken from the register i as input to the filter function at time t is denoted by $x_i(t)$. The keystream bit produced at time t is denoted $z(t)$ and sometimes only z. Both addition modulo 2 and integer addition is denoted by $+$ since there should be no risk of confusion. Only absolute values of biases of approximations will be given.

2.1 Pomaranch Version 1

The first version of Pomaranch was introduced in [7]. In this version, 128 bit keys were used together with an IV in the range of 64 to 112 bits. The cipher is built upon $n^{(1)} = 9$ identical jump registers. Each register uses 14 memory cells together with a characteristic polynomial with the jump index 5945, i.e., $(c_0, c_1) = (1, 5945)$. From register i 9 bits are taken as input to a key dependent function, denoted KeyMap in Figure 1. The output of this function is XORed to the jump sequence from register $i-1$, $j_i(t)$ to produce the jump sequence for the next register, $j_{i+1}(t)$. The keystream bit z is given as the XOR of the bit in cell 13 from all the registers, i.e.,

$$z = H^{(1)}\big(x_1, \ldots, x_9\big) = x_1 + x_2 + \ldots + x_9.$$

2.2 Pomaranch Version 2

The second version of Pomaranch [8], comes in two variants, an 80-bit with $n^{(2)}_{80} = 6$, and a 128-bit key variant using $n^{(2)}_{128} = 9$ registers. The registers are still 14 memory cells long but to prevent the attacks on the first version, different tap positions are used as input to the KeyMap function in Figure 1. The characteristic polynomial is also changed and the new jump index is 13994. A new initialization procedure was introduced to prevent the attacks in [2, 5]. The keystream is still taken as the XOR of the bits in cell 13 of all registers and is given by

$$z = \begin{cases} H_{80}^{(2)}(x_1, \ldots, x_6) = x_1 + x_2 + \ldots + x_6 \\ H_{128}^{(2)}(x_1, \ldots, x_9) = x_1 + x_2 + \ldots + x_9 \end{cases} .$$

2.3 Pomaranch Version 3

As in the second version, there are two variants of Pomaranch Version 3 [9], an 80-bit and a 128-bit key variant. The number of registers used are still the same as in Pomaranch Version 2, i.e., $n_{80}^{(3)} = 6$ respectively $n_{128}^{(3)} = 9$. In the case of Pomaranch Version 3, two different jump registers are used, the first using jump index 84074 which is referred to as type I and the second using jump index 27044, referred to as type II. The type I registers are used for odd numbered sections in Figure 1, and type II for the even numbered sections. Both registers are built on 18 memory cells, and have primitive characteristic polynomials, i.e., when only clocked with zeros or ones they have a period of $2^{18} - 1$. From each register one bit is taken from cell 17 and is fed into H. The filter functions used are

$$z = \begin{cases} H_{80}^{(3)}(x_1, \ldots, x_6) = G(x_1, \ldots, x_5) + x_6 \\ H_{128}^{(3)}(x_1, \ldots, x_9) = x_1 + x_2 + \ldots + x_9 \end{cases} ,$$

where

$$G(x_1, \ldots, x_5) = x_1 + x_2 + x_5 + x_1x_3 + x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_3x_4x_5$$

is a 1-resilient Boolean function. The keystream length per IV/key pair is limited to 2^{64} bits in Version 3.

3 Previous Attacks on the Pomaranch Stream Ciphers

The first attack on Pomaranch, given in [2], was an attack on the initialization procedure. Soon after, an attack on the keystream generation was presented in [10]. This attack considered the best linear approximation of bits distance $L+1$ apart. Register JR_1 was exhaustively searched since this register is always fed with the all zero jump control sequence and the linear approximation is not valid for this register. When the state of JR_1 was known, JR_2 and 16 bits of the key was guessed. This was iterated until the full key was recovered. A distinguisher was not explicitly mentioned in [10] but it is very easy to see that if the attack is stopped after JR_1 is recovered, it would be equivalent to a distinguishing attack. This distinguisher needed $2^{72.8}$ keystream bits and computational complexity $2^{86.8}$.

Pomaranch Version 2 was designed to resist the attacks in [2, 10]. However, it was still possible to find biased linear approximations by looking at keystream bits further apart. This was done in [6] and the new approximation made it possible to mount distinguishing and key recovery attacks on both the 80-bit and the 128-bit variants. The distinguisher for the 80-bit variant needed $2^{44.59}$ keystream bits and a computational complexity of $2^{58.59}$. For the 128-bit variant the complexities were $2^{73.53}$ and $2^{87.53}$ respectively.

4 Distinguishing Attacks on Pomaranch-like Ciphers

In this section we will present general distinguishers for Pomaranch-like ciphers, in particular we will study the constructions that have been proposed in Pomaranch Version 1-3. We will give general results for these cipher families that can be used as design criteria for future Pomaranch ciphers.

4.1 Period of Registers

The first jump register, denoted JR_1 in Figure 1, is during keystream generation mode fed by the jump sequence only containing zeros. Hence JR_1 is a LFSM. The period of the register is denoted by T_1, hence $x_1(t) = x_1(t + T_1)$ with $T_1 = 2^L - 1$.

From JR_1 a jump control sequence is calculated which controls the jumping of JR_2. Assume that after T_1 clocks of JR_1, register JR_2 has jumped C steps. Then after T_1^2 clocks JR_2 has jumped CT_1 steps, a multiple of T_1 and is thus back to its initial state. If primitive characteristic polynomials are used for the registers, it can be shown that the period for register JR_i is

$$T_i = T_1^i,$$

and hence

$$x_i(t) = x_i(t + T_1^i).$$

Consequently, at time t and $t + T_1^p$, the filter function H has p inputs with exactly the same value, namely the contribution from registers $JR_1, \dots, JR_p$. This observation will be used in our attack.

4.2 Filter Function

The filter function used in Pomaranch can be a nonlinear Boolean function or just the linear XOR of the output bits of each jump register. Our attack can be applied to both variants. The keystream bit at time t, denoted by $z(t)$, can be described as

$$z(t) = H\big(x_1(t), \dots, x_n(t)\big).$$

Using the results from Section 4.1 and taking our samples as $z(t) + z(t + T_1^p)$ we can write the expression for the samples as

$$z(t) + z(t + T_1^p) = H\big(x_1(t), \dots, x_n(t)\big) + H\big(x_1(t + T_1^p), \dots, x_n(t + T_1^p)\big). \quad (1)$$

4.2.1 Linear Filter Function

When the filter function H is linear, i.e., $H(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$, and our samples are taken as $z(t) + z(t+T_1^p)$ we know from Section 4.1 that p inputs to the filter function are the same at time t and at time $t + T_1^p$, hence we can rewrite (1) as

$$z(t) + z(t + T_1^p) = \sum_{i=p+1}^{n} x_i(t) + x_i(t + T_1^p).$$

4.2.2 Nonlinear Filter Function

When the filter function H is nonlinear, $x_i(t)$ and $x_i(t+T_1^p)$ will not cancel out in the keystream with probability one, as in Section 4.2.1. But, the input to H at time t and $t+T_1^p$ has p inputs $x_1, \ldots, x_p$ with the exact same value. This might lead to a biased distribution,

$$\begin{aligned}&\Pr\big(H\big(x_1(t), \ldots, x_n(t)\big) + H\big(x_1(t + T_1^p), \ldots, x_n(t + T_1^p)\big) = 0\big) = \\ &\Pr\left(z(t) + z(t + T_i^p) = 0\right) = \frac{1}{2}(1 \pm \epsilon),\end{aligned}$$

where ϵ denotes the bias and $|\epsilon| \leq 1$.

4.3 Linear Approximations of Jump Registers

In our attack, we need to find a linear approximation for the output bits of the jump registers that is biased, i.e., that holds with probability different from one half. We assume that all states of the register are equally probable, except for the all zero state which has probability 0. Further, it is assumed that all jump control sequences have the same probability. Finding the best linear approximation can be done by exhaustive search. Under certain circumstances much faster approaches can be used, see [6]. We search for a set $\mathcal{A}$ of size w such that

$$\Pr(\sum_{i \in \mathcal{A}} x(t + i) = 0) = \frac{1}{2}(1 \pm \varepsilon), \quad |\varepsilon| \leq 1,$$

i.e., the weight of the approximation is w and the terms are given by the set $\mathcal{A}$. For our attack to work it is important that the bias of this approximation is sufficiently high.

It is assumed that jump register JR_1 will always have the all zero jump control sequence. Hence, the linear approximation will never apply for this register.

4.3.1 Different Registers

In the case when not all registers use the same kind of jump registers, we are not interested in the most biased linear approximation of single registers. Instead we have to search for a linear approximation that has a good bias for all types of registers at the same time. This is much harder to find than a single approximation for one register. This is the case in Pomaranch Version 3.

4.4 Attacking Different Versions of Pomaranch

A Pomaranch stream cipher can be designed using one or several types of jump registers. It can also use a linear or a nonlinear Boolean filter function. In this section we take a closer look at the different design possibilities that has been used and give an expression for the number of samples needed in a distinguisher for each possibility.

The general expressions for the amount of keystream needed in an attack can be seen as a new design criteria for Pomaranch-like stream ciphers.

4.4.1 One Type of Registers with Linear Filter Function

In this family of Pomaranch stream ciphers we assume that all jump register sections use the same type of register and that the filter function H is linear, i.e., $H(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$. Pomaranch Version 1 and Pomaranch Version 2 are both included in this family.

Assume that we have found a linear approximation, as described in Section 4.3, of weight w of the register used. We consider samples at t and $t + T_1^p$ such that p positions into H are the same according to Section 4.1. Our samples will be taken as

$$\begin{aligned}\sum_{i\in\mathcal{A}} z(t+i) + \sum_{i\in\mathcal{A}} z(t+i+T_1^p) &= \sum_{j=1}^{n}\sum_{i\in\mathcal{A}}\big(x_j(t+i) + x_j(t+i+T_1^p)\big) \qquad (2)\\ &= \sum_{j=p+1}^{n}\sum_{i\in\mathcal{A}}\big(x_j(t+i) + x_j(t+i+T_1^p)\big).\end{aligned}$$

Since the bias of $\sum_{i\in\mathcal{A}} x_i(t+i)$ is ε and we have $2(n-p)$ such relations the total bias of the samples is given by

$$\varepsilon_{tot} = \varepsilon^{2(n-p)}.$$

Using the approximation that $1/\varepsilon_{tot}^2$ samples are needed to reliably distinguish the cipher from a truly random source, we get an estimate of the keystream length needed. (In practice, this number should be multiplied by a small constant.)

Theorem 1. *The computational complexity and the number N of keystream bits needed to reliably distinguish the Pomaranch family of stream ciphers using a linear filter function and n jump registers of the same type is bounded by*

$$N = T_1^p + \frac{1}{\varepsilon^{4(n-p)}}, \quad p > 0,$$

where ε is the bias of the best linear approximation of the jump register.

4.4.2 Different Registers with Linear Filter Function

In this family of generators different types of jump registers are used and the filter function is assumed to be $H(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$.

This case is very similar to the case when all registers are of the same type. The difference is that, in this case, we are not looking for the best linear approximation of the registers separately. Instead, we have to find a linear approximation that have a bias for all the registers $JR_{p+1}, \ldots, JR_n$. This can be difficult if there are several types of registers. Approximations with a large bias for one type might have a very small bias for other types. Anyway, assume that we have found such a linear approximation. Our samples will still be taken as in (2). If we denote the bias for the approximation of register i by ε_i, then the total bias will be given as

$$\varepsilon_{tot} = \prod_{i=p+1}^{n} \varepsilon_i^2.$$

Theorem 2. *Assume that there is a linear relation that is biased in all registers. The computational complexity and the number N of keystream bits needed to reliably distinguish the Pomaranch family of stream ciphers using a linear filter function and n jump registers of different types is bounded by*

$$N = T_1^p + \frac{1}{\prod_{i=p+1}^{n} \varepsilon_i^4}, \quad p > 0,$$

where ε_i is the bias of jump register JR_i.

The 128-bit variant of Pomaranch Version 3 belongs to a special subclass of this family, namely all registers in odd positions are of type I and registers in even positions are of type II. In this case we only have to search for a linear approximation that is biased for type I and type II registers at the same time. The bias of $\sum_{i \in \mathcal{A}} x_i(t+i)$ is denoted $\varepsilon_{type\ I}$ and $\varepsilon_{type\ II}$, respectively, for the different registers. In total we have $2\lceil \frac{n-p}{2} \rceil$ type I relations and $2\lfloor \frac{n-p}{2} \rfloor$ type II relations when n is odd. Hence, the total bias of the samples is given by

$$\varepsilon_{tot} = \varepsilon_{type\ I}^{2\lceil \frac{n-p}{2} \rceil} \varepsilon_{type\ II}^{2\lfloor \frac{n-p}{2} \rfloor}.$$

If we apply Theorem 2 to the 128-bit variant of Pomaranch Version 3, the number of samples in the distinguisher is given by

$$N = T_1^p + \frac{1}{\varepsilon_{type\ I}^{4\lceil \frac{n-p}{2} \rceil} \varepsilon_{type\ II}^{4\lfloor \frac{n-p}{2} \rfloor}}. \tag{3}$$

4.4.3 Nonlinear Filter Function

Now we consider the case when the Boolean filter function is a nonlinear function. We only consider the case when the filter function H can be written in the form

$$H(x_1, \ldots, x_n) = G(x_1, \ldots, x_{n-1}) + x_n. \tag{4}$$

The attack can easily be extended to filter functions with more (or less) linear terms but to simplify the presentation, and the fact that the 80-bit variant of Pomaranch Version 3 is in this form, we only consider this special case in this paper.

Attacks on this family use a biased linear approximation of JR_n, see Section 4.3, together with the fact that the input to G at time t and $t+T_1^p$ have p inputs in common and hence in some cases a biased distribution, see Section 4.2.2.

Let ϵ denote the bias of $G\big(x_1(t),\ldots,x_{n-1}(t)\big)+G\big(x_1(t+T_1^p),\ldots,x_{n-1}(t+T_i^p)\big)$, and ε the bias of our linear approximation for JR_n, $\sum_{i\in\mathcal{A}} x_i(t+i)$. The samples are taken as

$$\sum_{i\in\mathcal{A}} z(t+i)+\sum_{i\in\mathcal{A}} z(t+i+T_1^p)=\sum_{i\in\mathcal{A}} x_n(t+i)+\sum_{i\in\mathcal{A}} x_n(t+i+T_1^p)$$
$$+\sum_{i\in\mathcal{A}} G\big(x_1(t+i),\ldots,x_{n-1}(t+i)\big)+G(x_1(t+i+T_1^p),\ldots,x_{n-1}(t+i+T_1^p)\big),$$

and the bias of the samples is given by

$$\varepsilon_{tot}=\varepsilon^2\epsilon^w. \tag{5}$$

This relation tells us that we need to keep the weight of the linear approximation of JR_n as low as possible, i.e., there is a trade off between the bias ε of the approximation and the number of terms w in the relation.

Theorem 3. *The computational complexity and the number N of keystream bits needed to reliably distinguish the Pomaranch family of stream ciphers using a filter function of the form* (4) *is bounded by*

$$N=T_1^p+\frac{1}{(\varepsilon^2\epsilon^w)^2}.$$

where ε is the bias of the approximation of weight w of register JR_n and ϵ is the bias of $G\big(x_1(t),\ldots,x_{n-1}(t)\big)+G\big(x_1(t+T_1^p),\ldots,x_{n-1}(t+T_i^p)\big)$.

Note that in this presentation it does not matter if all registers are of the same type or if they are of different types. Since only register JR_n is completely linear in the output function H, we only need to have an approximation of this register.

4.5 Attack Complexities for the Existing Versions of the Pomaranch Family

In this section, we look at the existing versions and variants of Pomaranch that have been proposed so far. These are Pomaranch Version 1, the 80-bit and 128-bit variants of Pomaranch Version 2 and the 80-and 128-bit variants of Pomaranch Version 3. Applying the attack proposed in this paper, we show that we can find distinguishers with better complexity than previously known for *all* 5 ciphers.

4.5.1 Pomaranch Version 1

In Pomaranch Version 1 all registers are the same, so the attack will be according to Section 4.4.1. The best known linear approximation for this register, as given in [10], is

$$\varepsilon = |2\Pr(x(t) + x(t+8) + x(t+14) = 0) - 1| = 2^{-4.286}.$$

Using Theorem 1 for different values of p we get Table 1. We see that the best attack is achieved when $p = 5$. The computational complexity and the amount of keystream needed is then $2^{70.46}$.

Table 1. Number of samples and computational complexity needed to distinguish Pomaranch Version 1 from random.

p	1	2	3	4	5	6	7
$N^{(1)}$	$2^{137.15}$	$2^{120.01}$	$2^{102.86}$	$2^{85.72}$	$2^{70.46}$	$2^{83.99}$	$2^{97.99}$

4.5.2 Pomaranch Version 2

Similarly as in Pomaranch Version 1, in Pomaranch Version 2 all registers are the same and the attack will be performed according to Section 4.4.1. The best bias of a linear approximation for the registers used was found in [6] and is given by

$$\varepsilon = |2\Pr(x(t) + x(t+2) + x(t+6) + x(t+18) = 0) - 1| = 2^{-4.788}.$$

Using Theorem 1 for different values of p gives Table 2. For the 80-bit variant the computational complexity and the number of samples is $2^{56.00}$ and for the 128-bit variant it is $2^{76.62}$.

Table 2. Number of samples needed to distinguish Pomaranch Version 2 according to Theorem 1.

p	1	2	3	4	5	6
$N_{80}^{(2)}$	$2^{95.76}$	$2^{76.61}$	$2^{57.46}$	$2^{56.00}$	$2^{70.00}$	$2^{84.00}$
$N_{128}^{(2)}$	$2^{153.22}$	$2^{134.06}$	$2^{114.91}$	$2^{95.76}$	$2^{76.62}$	$2^{84.00}$

4.5.3 Pomaranch Version 3

There is a significant difference between the 80-bit and the 128-bit variants of Pomaranch Version 3, so this section will be divided into two parts.

80-bit Variant. The 80-bit variant of Pomaranch Version 3 uses a non-linear filter function, the attack will hence follow the procedure described in Section 4.4.3.

We started by estimating the bias of

$$G\big(x_1(t),\dots,x_5(t)\big)+G\big(x_1(t+T_1^p),\dots,x_5(t+T_1^p)\big).$$

The results for different p are summarized in Table 3. The keystream per IV/key pair of Pomaranch Version 3 is limited to 2^{64}. Because of this we limit p to $p\in\{1,2,3\}$, otherwise $T_1^p>2^{64}$. We looked for a linear relation of JR_6 that, together with a value of $p\in\{1,2,3\}$, minimizes the amount of keystream needed as given by Theorem 3. The best approximation found was

$$\Pr\big(x_6(t)+x_6(t+5)+x_6(t+7)+x_6(t+9)+x_6(t+12)+x_6(t+18)=0\big)=\frac{1}{2}(1-2^{-8.774}),$$

using $p=3$. The total bias of our samples using this approximation is

$$\varepsilon_{tot}=(2^{-8.774})^2\cdot(2^{-3})^6=2^{-35.548},$$

according to (5). The samples used in the attack are taken according to

$$\sum_{i\in\mathcal{A}}z(t+i)+\sum_{i\in\mathcal{A}}z(t+i+T_1^3),$$

where $\mathcal{A}=\{0,5,7,9,12,18\}$. According to Theorem 3, the amount of keystream needed is $2^{54}+2^{71.096}=2^{71.096}$. This is also the computational complexity of the attack. In the specification of Pomaranch Version 3 the frame length (keystream per IV/key pair) is limited to 2^{64}. This does not prevent our attack since all samples will have this bias regardless of the key and IV used. We only need to consider 2^{64} keystream bits from $\lceil 2^{7.096}\rceil=137$ different key/IV pairs.

Table 3. The bias of $G\big(x_1(t),\dots,x_5(t)\big)+G\big(x_1(t+T_1^p),\dots,x_5(t+T_1^p)\big)$ in the 80 bit variant of Pomaranch Version 3 for different values of p.

p	1	2	3	4	5
ϵ	0	2^{-4}	2^{-3}	2^{-2}	1

128-bit Variant. In Pomaranch Version 3 two different registers are used, so we start by searching for a linear approximation that is good for both types of registers. The best approximation we found was

$$x(t)+x(t+1)+x(t+2)+x(t+5)+x(t+7)+x(t+11)+x(t+12)+x(t+15)+x(t+21),$$

which has the same bias for both types of registers, namely

$$\varepsilon_{even}=\varepsilon_{odd}=2^{-10.934}.$$

Using (3) for different values of p we get Table 4. Our best distinguishing attack needs $2^{126.00}$ keystream bits. This figure is determined by $T_1^7 = 2^{126.00}$ so it is not possible to look at different key/IV pairs in this case since the distance between the bits in each sample has to be $2^{126.00}$. Since the frame length is limited to 2^{64} it will not be possible to get any biased samples at all with $p = 7$.

Table 4. Number of samples needed to distinguish the 128-bit variant of Pomaranch Version 3 according to (3).

p	1	2	3	4	5	6	7	8
$N_{128}^{(3)}$	$2^{349.89}$	$2^{306.15}$	$2^{262.42}$	$2^{218.68}$	$2^{174.94}$	$2^{131.21}$	$2^{126.00}$	$2^{144.00}$

5 Square Root IV Attack

In this section we will give an attack that works for all families of Pomaranch where the key is longer than half of the total register length. The size of the state in Pomaranch is always larger than twice the key size, e.g., the 128-bit variant of Pomaranch Version 3 has a state size of 290 bits. Thus, the generic time-memory tradeoff attacks will not be applicable in general. Our attack is a variant of the time-memory tradeoff attack and is generic for all stream ciphers. Let us divide the internal state of the cipher into two parts,

$$State = (State_K, State_{K+IV}),$$

where $State_K$ is a part of the state that statically holds the key and $State_{K+IV}$ is a part of the state that is updated, depending on both the key and the IV. If the key size $|K| > |State_{K+IV}|/2$, then the attack will always succeed with complexity below exhaustive key search. In Pomaranch, $State_K$ will consist of the $|K|$ key bits and $State_{K+IV}$ will consist of the register cells.

In the attack scenario, we assume that the key is fixed and that the cipher is initialized with many different IVs. Further, we assume that we have access to one long keystream sequence produced from one of the IVs, denoted IV_0. We intercept the ciphertext corresponding to many other IVs and we know the first l plaintext bits corresponding to every ciphertext. Our goal is to recover the rest of the plaintext for one of the messages.

The key map used to produce the jump control bits is key dependent but independent of the IV. Hence, a fixed key will define a state graph of size $(2^L - 1)^n \approx 2^{nL}$ states, where L is the register length and n the number of registers. We can apply the following attack.

Let a sample of l consecutive keystream bits at time t be denoted $S(t) = \big(z(t), z(t+1), \ldots, z(t+l-1)\big)$. If the sample stems from IV_i we denote it by

$S_{IV_i}(t)$. We first store a large amount of samples from IV_0 in a table. We would like to find another IV, denoted IV_c, that results in a sample such that

$$S_{IV_c}(t_c) = S_{IV_0}(t_0).$$

If a collision is found, then with high probability the following keystream of IV_0 and IV_c will also be identical. That means that if we just know the first l keystream bits generated by IV_c, we can predict future keystream bits from IV_c. The attack is visualized in Figure 2.

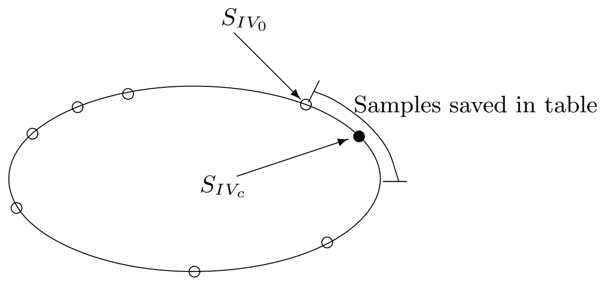

Fig. 2. State graph for a fixed key, a sample is visualized by a small ring.

Assume that $2^{\beta nL}$ $(0 \leq \beta \leq 1)$ samples of length l from a keystream sequence of $2^{\beta nL} + l$ bits, originating from IV_0 and key K, is saved in a table. The table is then sorted with complexity $O(\beta nL \cdot 2^{\beta nL})$. This table covers a fraction of $2^{-(1-\beta)nL}$ of the entire cycle. The number of samples (IVs with l known keystream bits) we need to test to find a collision is geometrically distributed with expected value $2^{(1-\beta)nL}$. For each sample, a logarithmic search with complexity $O(\beta nL)$ in the table is performed to see if there is a collision. To be sure that a collision in the table actually means that we have found a collision in the state cycle, l must be $l \approx nL$. The attack complexities are then given by

$$\begin{array}{ll} \textit{Keystream}: & \begin{array}{l} 2^{\beta nL} + nL, \text{ from one IV and} \\ nL, \text{ from } 2^{(1-\beta)nL} \text{ IVs} \end{array} \\ \textit{Time}: & \beta nL 2^{\beta nL} + \beta nL 2^{(1-\beta)nL} \\ \textit{Memory}: & nL 2^{\beta nL} \end{array}$$

where $0 \leq \beta \leq 1$. By decreasing β it is possible to achieve smaller memory complexity at the expense of more IVs and higher time complexity. We can also see that the best time complexity is achieved when $\beta = 0.5$ for large nL. The proposed attack is summarized in Figure 3. In the figure, $S_{IV_i}(t)$ represents a sample from the keystream from IV_i at time t, and T represents the table where samples are stored.

```
for i = 1, ..., 2^(nL/2)
    T[i] = S_{IV_0}(t + i)
end for
sort T
for i = 1, ..., 2^(nL/2)
    if S_{IV_i}(t) ∈ T
        return cipher
end for
return random
```

Fig. 3. Summary of square root attack.

5.1 Attack Complexities on Pomaranch

In this section we will look at the existing versions of Pomaranch and show that the square root IV attack can be mounted with complexity significantly less than exhaustive key search. We assume $\beta = 0.5$ so the time complexity and the memory complexity in bits are equal. The 128-bit variants of Pomaranch Version 1 and Version 2 can be attacked using a table of size $2^{67.0}$ bytes together with keystream from $2^{63.0}$ different IVs. The 80-bit variant of Pomaranch Version 2 can be attacked using only a table of $2^{45.4}$ bytes and $2^{42.0}$ different IVs. Pomaranch Version 3 uses larger registers, and the complexity of the attack on the 80-bit variant is a table of size $2^{57.8}$ bytes and $2^{54.0}$ IVs. The 128-bit variant needs a table of $2^{85.3}$ bytes and 2^{81} IVs. However, if we respect the maximum frame length of 2^{64} bits, we need to choose $\beta = 0.395$. Then we need a table of $2^{71.3}$ bytes and 2^{98} IVs. The time complexity is in this case 2^{104}.

The success probability of the attack has been simulated on a reduced version of the 128 bit variant of Pomaranch Version 3, using two registers. Choosing $\beta = 0.5$ implies that we know 2^{18} keystream bits from IV_0, we store all samples of length $nL = 36$ in a table, and that we need samples from 2^{18} different IVs in order to find a collision. The simulation results are summarized in Table 5. We also verified the attack using 3 registers. The attack given in this section suggests

Table 5. Simulation results using 2 register Pomaranch version 3 with linear filter funtion, the table summarizes how many times the attack succeeds out of 100 attacks for a specific table size and number of IVs.

Success rate		Number of IVs 2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
	2^{17}	28	45	64	87	98
Table size	2^{18}	38	67	91	98	100
	2^{19}	71	86	98	100	100

a new design criteria for the Pomaranch family of stream ciphers, namely that the total register length must be at least twice the keysize.

6 Conclusions

We have presented two general attacks on Pomaranch-like keystream generators. The attack complexities are summarized in Table 6.

The first attack is a distinguishing attack that can be applied to all known versions and variants of the stream cipher Pomaranch. For Pomaranch Version 1 and Pomaranch Version 2, the distinguisher will succeed with better computational complexity than any other known distinguisher for these versions. For the 80-bit variant of Pomaranch Version 3, the attack will succeed using $2^{71.1}$ bits and computational complexity $2^{71.1}$. Since the frame length is restricted to 2^{64} bits we can instead collect 2^{64} bits from 137 key/IV pairs. For the 128-bit variant our distinguisher needs about 2^{126} samples and will not succeed if the frame length is restricted.

We have also presented a general IV attack that works for all ciphers where the key size is larger than half of the state size, when the part of the state only affected by the key is not considered. The attack was demonstrated on all versions and variants of Pomaranch with complexity far below exhaustive search. This attack has much lower complexity than the first and will work even if the frame length is restricted. The attack will not recover the key but is different from a distinguishing attack. It will recover the plaintext corresponding to one IV if only the ciphertext together with the first few keystream bits are known. On the other hand, this attack will require keystream from a large amount of IVs. This attack will also be applicable to the stream cipher LEX [1] and to any block cipher used in OFB mode of operation as shown in [3]. The attack scenario here is somewhat similar to the attack scenarios in disk encryption, see e.g., [4], where the adversary has write access to the disk encryptor and read access to the storage medium.

Table 6. Summary of attack complexities, for the two proposed attacks, on all versions and variants of Pomaranch.

Attack Complexities		Distinguishing Attack Keystream/Compl.	Square Root IV attack Memory/IVs/Compl.
Pomaranch v1	128 bit	2^{71} / 2^{71}	2^{67} / 2^{63} / 2^{63}
Pomaranch v2	80 bit	2^{56} / 2^{56}	2^{45} / 2^{42} / 2^{42}
	128 bit	2^{77} / 2^{77}	2^{67} / 2^{63} / 2^{63}
Pomaranch v3	80 bit	2^{71} / 2^{71}	2^{58} / 2^{54} / 2^{54}
	128 bit	$*$ 2^{126} / 2^{126}	2^{71} / 2^{98} / 2^{104}

* Without frame length restriction

References

1. A. Biryukov. The design of a stream cipher LEX. Selected Areas in Cryptography—SAC 2006, 2006. Preproceedings.
2. C. Cid, H. Gilbert, and T. Johansson. Cryptanalysis of Pomaranch. *IEE Proceedings - Information Security*, 153(2):51–53, June 2006.
3. H. Englund, M. Hell, and T. Johansson. A note on distinguishing attacks. The State of the Art of Stream Ciphers, Workshop Record, SASC 2007, Bochum, Germany, January 2007.
4. K. Gjøsteen. Security notions for disk encryption. In *Computer Security – ESORICS 2005*, volume 3679 of *Lecture Notes in Computer Science*, pages 455–474. Springer-Verlag, 2005.
5. M. Hasanzadeh, S. Khazaei, and A. Kholosha. On IV setup of Pomaranch. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/082, 2005. *http://www.ecrypt.eu.org/stream.*
6. M. Hell and T. Johansson. On the problem of finding linear approximations and cryptanalysis of Pomaranch version 2. Selected Areas in Cryptography—SAC 2006, 2006. Preproceedings.
7. C.J.A. Jansen, T. Helleseth, and A. Kholosha. Cascade jump controlled sequence generator (CJCSG). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/022, 2005. *http://www.ecrypt.eu.org/stream.*
8. C.J.A. Jansen, T. Helleseth, and A. Kholosha. 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.*
9. C.J.A. Jansen, T. Helleseth, and A. Kholosha. Cascade jump controlled sequence generator and Pomaranch stream cipher (version 3). eSTREAM, ECRYPT Stream Cipher Project. *http://www.ecrypt.eu.org/stream.*
10. S. Khazaei. Cryptanalysis of Pomaranch (CJCSG). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/065, 2005. *http://www.ecrypt.eu.org/stream.*