Cryptanalysis of Pomaranch

Carlos Cid[1], Henri Gilbert[2] and Thomas Johansson[3]

[1] Information Security Group,
Royal Holloway, University of London
Egham, Surrey TW20 0EX, United Kingdom
carlos.cid@rhul.ac.uk
[2] France Télécom, R&D Division
38–40, rue du Général Leclerc
92794 Issy les Moulineaux, Cedex 9, France
henri.gilbert@francetelecom.com
[3] Dept. of Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
thomas@it.lth.se

Abstract Pomaranch [3] is a synchronous stream cipher submitted to eSTREAM, the ECRYPT Stream Cipher Project. The cipher is constructed as a cascade clock control sequence generator, which is based on the notion of jump registers. In this paper we present an attack which exploits the cipher's initialization procedure to recover the 128-bit secret key. The attack requires around 2^{65} computations. An improved version of the attack is also presented, with complexity 2^{52}.

1 Introduction

Pomaranch[1] is one of the 34 stream ciphers submitted to eSTREAM, the ECRYPT Stream Cipher Project [1]. The cipher is implemented as a binary one clock pulse cascade clock control sequence generator, and uses 128-bit keys and IVs of length between 64 and 112 bits [3]. The construction is based on the notion of *jump registers*.

Jump controlled LFSRs were introduced in [2] as alternative to traditional clock-controlled registers. In jump controlled LFSRs, the registers are able to move to a state that is more than one step ahead without having to step through all the intermediate states (thus the name jump registers). The main motivation for the proposal of jump registers is to construct LFSR-based ciphers that can be efficiently protected against side-channel attacks while preserving the advantages of irregular clocking.

2 Outline of Pomaranch

Pomaranch is depicted in Figure 1, where only the key stream generation phase is represented (called Key Stream Generation Mode). The cipher consists of nine cascaded jump registers R_1 to R_9. The jump registers are implemented as autonomous Linear Finite State Machine (LFSM), built on 14 memory cells, which behave either

[1] The cipher is also referred in the specification document [3] as Cascade Jump Controlled Sequence Generator (CJCSG).

as simple delay shift cells or feedback cells, depending on the value of the so-called Jump Control (JC) signal. At any moment, half of the cells in the registers are shift cells, while the other half are feedback cells. The initial configuration of cells is determined by the LFSM transition matrix A, and is used if the JC value is zero. If JC is one, all cells are switched to the opposite mode. This is equivalent to switching the transition matrix to $(A + 1)$ [3].

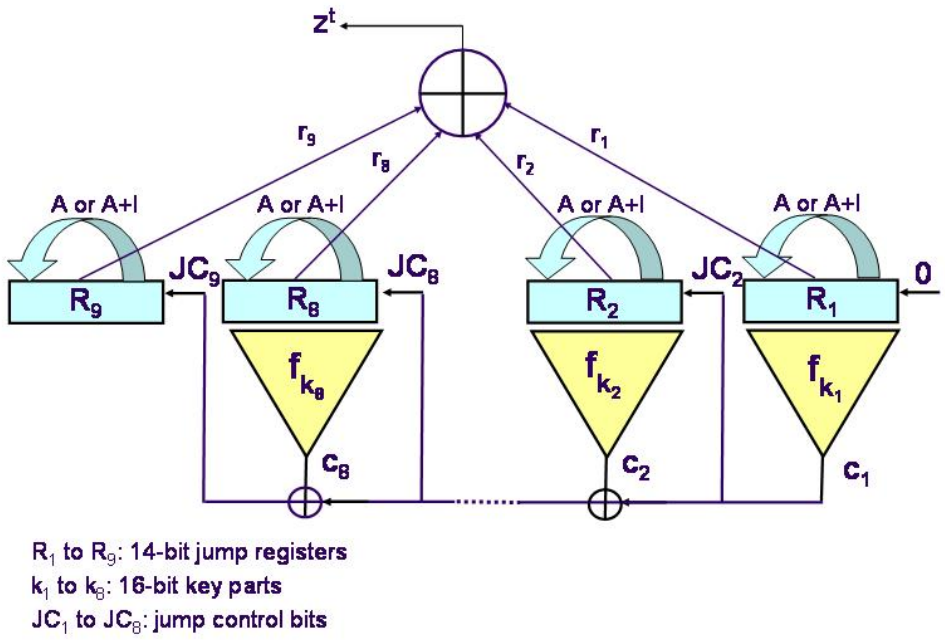

Figure1. The Pomaranch stream cipher

The 128-bit key K is divided into eight 16-bit subkeys k_1 to k_8. At time t, the current state of the registers R_1^t to R_8^t are non linearly filtered, using a function that involves the corresponding subkey k_i. These functions provide as output eight bits c_1^t to c_8^t, which are used to produce the jump control bits JC_2^t to JC_9^t controlling the registers R_2 to R_9 at time t, as following:

$$JC_i^t = c_1^t \oplus \ldots \oplus c_{i-1}^t \quad \text{for } i = 2, \ldots, 9.$$

The jump control bit JC_1 of register R_1 is permanently set to zero. The key stream bit z^t produced at time t is the XOR of nine bits r_1^t to r_9^t selected at fixed positions of the current register states R_1^t to R_9^t.

Key and IV Loading. During the cipher initialization, the content of registers R_1 to R_9 are first set to non-zero constant 14-bit values derived from π, then the

subkeys k_i are loaded and the registers are run for 128 steps in a special mode (called Shift Mode). The main difference between the Key Stream Generation Mode and the Shift Mode is that, in the latter the output of the filtering function of register R_i (denoted by c_i) is added to the feedback of register R_{i+1}, with the tap from cell 1 in the register R_9 being added to the register R_1, making then what can be seen as a "big loop". Note that the configuration of the jump registers do not change in this mode (they all operate as if $JC_i = 0$). This process ensures that the states of the registers R_1 and R_9 after this key loading phase depend upon the entire key K. We denote these states by R_1^K to R_9^K.

Next the IV is loaded into the registers. The IV can have any arbitrary length between 64 and 112 bits. If the IV length is shorter than 112 bits, it is expanded by cyclically repeating it until a length of exactly 112 bits is obtained. This new string is then loaded into the registers as described below. In the remaining of this paper, for the sake of simplicity, we assume that the IV length is exactly 112 bits.

The IV is loaded into the registers in the following manner: the 112-bit IV is split into eight 14-bit parts IV_1 to IV_8, which are XORed with the 14-bit states of registers R_1^K to R_8^K obtained at the end of the key loading. If any of the resulting states consists of 14 null bits, its lowest weight bit is set to one (this ensures that no state will be made up entirely of null bits[2]). The resulting register states R_1 to R_8 form together with R_9^K the nine initial states. We denote these resulting 14-bit state values by R_1^{-128} to R_9^{-128}. The key stream generation mode of Figure 1 is now activated, and the runup consists of 128 steps in which the produced key stream bits are discarded.

3 Description of the Attack

We have identified the following weakness in the Pomaranch IV initialization procedure: if for a given key K and IV value IV, we only modify the IV part IV_8 and keep the remaining parts IV_1 to IV_7 unchanged (thus obtaining a modified IV value IV'), on comparing the key stream generation under the key K with IV and IV', we have that for every $t \geq -128$

$$R_i^t(IV) = R_i^t(IV') \quad \text{for } i = 1, \ldots, 7 \ .$$

In other words, the Key and IV loading procedure does not diffuse all IV bits into the whole state of the generator. Consequently, if IV and IV' are chosen as above, the contributions from registers R_1 to R_7 cancel out on each key stream XOR $z^t(IV) \oplus z^t(IV')$, and we obtain the relation

$$z^t(IV) \oplus z^t(IV') = r_8^t(IV) \oplus r_8^t(IV') \oplus r_9^t(IV) \oplus r_9^t(IV').$$

[2] The Pomaranch specification does not mention this feature, which is described in the source code provided with the submission and has been confirmed by one of the designers [4]. We will show in the next section that, although the cipher can be attacked even if this feature is withdrawn, this represents an additional weakness that leads to improved attacks.

We now show how to exploit this weakness to recover the subkey k_8 of an unknown key K, in a chosen IV attack. Consider 3 distinct chosen IV values IV, IV' and IV'', which only differ by their part IV_8, IV'_8 and IV''_8. We can obtain the corresponding first m-bit key stream $z^t(IV)_{t=0 \texttt{ to } m-1}$, $z^t(IV')_{t=0 \texttt{ to } m-1}$, and $z^t(IV'')_{t=0 \texttt{ to } m-1}$, which in turn provide the pairwise XOR values

$$\begin{aligned}\delta^t &= z^t(IV) \oplus z^t(IV')_{t=0 \texttt{ to } m-1},\\ \delta'^t &= z^t(IV') \oplus z^t(IV'')_{t=0 \texttt{ to } m-1},\end{aligned}$$

In order to recover the value of k_8, we guess the following values:

- Subkey k_8:	16 bits;
- Registers R_8^K and R_9^K:	28 bits;
- $n_8 = \sharp\{t \in \{-128, .., -1\} \mid JC_8^t(IV) = 1\}$:	129 possible values;
- $n_9 = \sharp\{t \in \{-128, .., -1\} \mid JC_9^t(IV) = 1\}$:	129 possible values;
- $n'_9 = \sharp\{t \in \{-128, .., -1\} \mid JC_9^t(IV') = 1\}$:	129 possible values;
- $n''_9 = \sharp\{t \in \{-128, .., -1\} \mid JC_9^t(IV'') = 1\}$:	129 possible values.

The attack exploits the jump registers property that since the transition matrices A and $A + I$ commute, the transition matrix associated with a number s of steps can only take one of the at most $s + 1$ values $A^p(A + I)^q$, with $p + q = s$. Due to this property, the knowledge of the values of (n_8, n_9, n'_9, n''_9) is sufficient to derive the R_8 and R_9 transition matrices of the form $A^{128-n}(A + I)^n$ associated with the 128-step runup for IV values IV, IV' and IV''. Note that although n_8, n_9, n'_9, n''_9 can take any of the 129 values in the $[0 \cdots 128]$ interval, their values are binomially distributed, so that in practice the $2^5 - 1$ middle values in the interval $[49 \cdots 79]$ have an overwhelming occurrence probability.

Now since we have[3]

$$\begin{aligned}&R_8^{-128}(IV) = R_8^K \oplus IV,\\ &R_8^{-128}(IV') = R_8^K \oplus IV',\\ &R_8^{-128}(IV'') = R_8^K \oplus IV'',\\ &R_9^{-128}(IV) = R_9^{-128}(IV') = R_9^{-128}(IV'') = R_9^K,\end{aligned}$$

it follows that knowledge of R_8^K, R_9^K, n_8, n_9, n'_9 and n''_9 allows us to compute $R_8^0(IV)$, $R_8^0(IV')$, $R_8^0(IV'')$, $R_9^0(IV)$, $R_9^0(IV')$, and $R_9^0(IV'')$.

To test a $(k_8, R_8^K, R_9^K, n_8, n_9, n'_9, n''_9)$ assumption we need to compute the resulting values of $R_8^0(IV)$, $R_8^0(IV')$, $R_8^0(IV'')$, $R_9^0(IV)$, $R_9^0(IV')$, and $R_9^0(IV'')$ and iteratively try, for consecutive values of m, to guess the m-bit value $JC_8^t(IV)_{t=0 \texttt{ to } m-1}$ in order to derive the resulting values of $R_8^t(IV)$, $R_8^t(IV')$, $R_8^t(IV'')$, $R_9^t(IV)$, $R_9^t(IV')$, and $R_9^t(IV'')$. Following we verify whether the predicted values $(\delta^t, \delta'^t)_{t=0 \texttt{ to } m-1}$ are in agreement with the observed ones. The average number of m values to be tested until a wrong assumption is discarded (because no $JC_8^t(IV)_{t=0 \texttt{ to } m-1}$ m-tuple fits the observed values) is about 2.

[3] We are ignoring the cipher's non-zero state forcing feature at this stage.

Indeed, for a certain (k_8, R_8^K, R_9^K, n_8, n_9, n_9', n_9'') assumption and a choice of $JC_8^t(IV)$, the pair (δ^t, δ'^t) can take one of four possible values. Assuming the values are randomly generated, there are three events to consider. First the case in which the pairs (δ^t, δ'^t) for both the choices of $JC_8^t(IV) = 0$ and $JC_8^t(IV) = 1$ are in agreement with the observed value. Its probability is 1/16, and it leaves us with two possible configurations that need to be further tested. The second event is when only one pair (δ^t, δ'^t) for either the choices of $JC_8^t(IV) = 0$ or $JC_8^t(IV) = 1$ is in agreement with the observed one. Its probability is 3/8, and it leaves us with one possible configuration that need to be further tested. The third event is when neither the pairs (δ^t, δ'^t) for the choices of $JC_8^t(IV) = 0$ and $JC_8^t(IV) = 1$ is in agreement with the observed one (i.e. the configuration is inconsistent). Its probability is 9/16, and no further tests using this configuration is necessary. Thus if X denotes the number of tests we need to perform, then

$$E(X) = 1 + \frac{1}{16} \cdot 2 \cdot E(X) + \frac{3}{8} \cdot 1 \cdot E(X) + \frac{9}{16} \cdot 0 \cdot E(X),$$

and $E(X) = 2$.

The attack described above allows us to recover the value of k_8. Its complexity is bounded over by $2^{16} \times 2^{28} \times (2^5)^4 \times 2 = 2^{65}$. Note that the attack also recovers the correct values for R_8^K and R_9^K. To recover the other key parts, we can proceed as following: repeat the same attack for another value of (IV, IV', IV''), call it $(\overline{IV}, \overline{IV'}, \overline{IV''})$, such that IV and $\overline{IV}$ only differ by their part IV_7 and $\overline{IV_7}$. Since we know already k_8, R_8^K and R_9^K, this second attack can be mounted much faster. Finally, we can guess the values of R_7^K and n_7 and check whether there exists a sequence $JC_7^t(IV)_{t=0 \texttt{ to } m-1}$ that is consistent with the already known sequences $JC_8^t(IV)_{t=0 \texttt{ to } m-1}$ and $JC_8^t(\overline{IV})_{t=0 \texttt{ to } m-1}$. This can be done for all the remaining key parts, until the entire key K has been recover. The complexity of the entire attack remains about 2^{65}.

Improved Attack. Note that so far we have not exploited the non-zero state forcing feature of Pomaranch, and the above attack works whether this feature is present or not. We now show that this feature results in a low complexity distinguisher, and also allows us to reduce the complexity of the key derivation procedure described above.

The distinguisher works as following: given an unknown key K, we can try the 2^{14} possible IV values obtained by keeping (say) IV_1 to IV_7 unchanged and taking all possible values for (say) IV_8. Now two of these 2^{14} IVs result in exactly the same states R_1^{-128} to R_9^{-128} after key and IV loading, namely the IV value resulting on a 14-bit R_8 state equal to zero (which will have one bit switched to 1 by the cipher non-zero state forcing procedure), and the IV value derived from the former one by swapping the same bit position. The key streams for these two IV values are exactly the same. If the key stream is sufficient long (e.g. more than 27 bits in order for collisions of a pair of IV values to be unlikely), this provides an efficient chosen IV

distinguisher of distinguishing probability close to 1, requiring generation of only 2^{14} key stream sequences of length (say) 64 bits each.

This distinguisher can be used to improve the key derivation attack described above. Indeed, the distinguisher allows us to recover the register value R_8^K up to one single bit, so that a factor of 2^{13} can be saved in the search of (k_8, R_8^K, R_9^K, n_8, n_9,n'_9, n''_9), and the attack complexity is reduced to 2^{52}.

4 Conclusion

We showed in this paper how to mount a chosen IV attack to recover the secret key of Pomaranch with complexity much lower than the one expected with 128-bit keys. The attack exploits a weakness in the cipher initialization procedure, namely the process does not diffuse all the IV bits into the whole state of the key stream generator. By exploiting another feature of the IV loading, we were able to substantially improve the attack.

References

1. eSTREAM, the ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`.
2. C. J. Jansen. Streamcipher Design: Make your LFSRs jump! In *SASC, Workshop Record, ECRYPT Network of Excellence in Cryptology*, pages 94–108, 2004.
3. C.J. Jansen, T. Helleseth, and A. Kholosha. Cascade Jump Controlled Sequence Generator (CJCSG). In *SKEW, Workshop Record, ECRYPT Network of Excellence in Cryptology*, 2005.
4. A. Kholosha. Personal Communication.