

A Guess-and-Determine Attack on the Stream Cipher Polar Bear

John Mattsson^{1,2}

¹ CSC, Royal Institute of Technology, Stockholm, Sweden

² Communications Security Lab, Ericsson Research, Stockholm, Sweden
`john.mattsson@gmail.com`

Abstract. In this paper we present an effective guess-and-determine attack against the stream cipher Polar Bear. The attack requires knowledge of the first 24 bytes of plaintext and recovers the state with a computational complexity of $O(2^{79})$. We also briefly discuss how this weakness can be addressed by the authors in an updated version of Polar Bear.

Keywords: Stream Cipher, Polar Bear, Guess-and-determine, eSTREAM

1 Introduction

There are a variety of efficient and trusted block ciphers. Unfortunately this is not the case for stream ciphers. As a response to this, ECRYPT (a 4-year network of excellence funded by the European Union) manages and co-ordinates a multi-year effort called eSTREAM to identify new stream ciphers suitable for widespread adoption. The new stream cipher Polar Bear [1] is one of 35 candidates submitted to eSTREAM. It was created by Johan Håstad and Mats Näslund and claimed to be suitable for both profile I (software) and profile II (hardware). In this paper we present the first known attack on Polar Bear. Recently a similar attack with improved complexity has been presented by Hasan-zadeh *et al* [2]. We also analyze why this attack is possible and suggest how the cipher can be fixed to avoid this type of attack.

2 Description of Polar Bear

The cipher uses one 7-word (112-bit) LFSR R^0 and one 9-word (144-bit) LFSR R^1 . These are viewed as acting over $\mathbb{F}_{2^{16}}$. Besides these registers, the internal state of the cipher also depends on a word quantity, S , and a dynamic permutation of bytes, D_8 .

The cipher is primarily designed for a key length of 128 bits. The IV can be any number of bytes up to a maximum of 31. The key schedule is (in the case of 128-bit keys) identical to the Rijndael key schedule.

On each message to be processed, the cipher is initialized by taking the key (more precisely, the expanded key), interpreting the IV as a cleartext block, and applying a (slightly modified) five round Rijndael encryption with block length

256. The resulting cipher text block is loaded into R^0 and R^1 . Finally, D_8 is initialized to equal the table T_8 , the Rijndael S-box, and S is set to zero.

Output is produced 4 bytes at a time. To this end, the two LFSRs are first irregularly clocked, determined by S . Eight bytes, selected from R^0 and R^1 , are run through the permutation D_8 to produce the four output bytes. Selected entries in D_8 are swapped. Finally, S and R^0 are modified in preparation for the next output cycle. Entries in R^1 are not modified apart from the LFSR stepping.

2.1 The output cycle

After each update of the cipher's internal state, four bytes are output. Before the first output byte, and between consecutive output pairs of bytes, a state update function is performed as specified below.

Next state function Let $\ell_0 = 7$ and $\ell_1 = 9$ be the lengths of the registers. Register R^i is stepped $2 + (\lfloor S/2^{14+i} \rfloor \bmod 2)$ steps with a sparse feedback where each step consists of

- set $f^i \leftarrow \theta^i R_{j^i}^i + \mu^i R_0^i$ for constants θ^i , j^i , and μ^i , where $j^0 = 1$ and $j^1 = 5$
- set $R_j^i \leftarrow R_{j+1}^i$ for $j = 0, 1, \dots, \ell_i - 2$
- feedback $R_{\ell_i-1}^i \leftarrow f^i$.

After stepping both R^0 and R^1 above, do the following steps, first for $i = 0$, then repeat them for $i = 1$:

- Write $(R_{\ell_i-1}^i, R_{\ell_i-2}^i)$ as four bytes $\alpha_0^i || \alpha_1^i || \alpha_2^i || \alpha_3^i$.
- Let $\beta_j^i = D_8(\alpha_j^i)$ for $j = 0, 1, 2, 3$.
- Swap elements in D_8 by $D_8(\alpha_0^i) \leftrightarrow D_8(\alpha_2^i)$ and $D_8(\alpha_1^i) \leftrightarrow D_8(\alpha_3^i)$.

Next, update S and R^0

- Update S according to $S \leftarrow S +_{16} \beta_0^1 || \beta_1^1$.
- Update R^0 according to $R_5^0 \leftarrow R_5^0 +_{16} \beta_2^1 || \beta_3^1$.

At this point, the internal state is updated, and the output is formed from the above (β_j^0, β_j^1) -pairs as described next.

Output generation Form four output bytes $b_0 || b_1 || b_2 || b_3$ where

$$b_j = \beta_j^0 \oplus \beta_j^1.$$

If more output bytes are required, the output cycle above is repeated. For a more complete description of Polar Bear, see [1].

3 The Attack

In this section we present an effective guess-and-determine attack on Polar Bear requiring only a very small amount of known plaintext. Under the assumption of a certain stepping of the registers, a certain sequence of α -values, and a known plaintext, the state can be recovered in $O(2^{79})$ time. Only knowledge of the first 24 bytes of plaintext is needed.

A first observation of Polar Bear is that it is relatively straightforward that the attack resistance does not meet the key size. For instance, by guessing one (the shorter) LFSR value, it is possible to deduce the value of the other by observing output. Hence, we have an attack with complexity about 2^{112} .

Let the state of LFSR R^i after t steppings of the register be

$$(R_{t+\ell_i-1}^i, R_{t+\ell_i-2}^i, \dots, R_t^i)$$

where ℓ_i is the length of register R^i . The notation $*R_i^0$ will be used for stages in R^0 after their update.

Let the first 24 bytes of plaintext be known and let the corresponding first twelve 16-bit block of keystream be Z_0, Z_1, \dots, Z_{11} .

For the attack to be successful, three assumptions have to be made.

1. During the first six updates of the state, let the steppings for both LFSR R^0 and R^1 be 2-steppings where the register is stepped two steps. This happens if the fourteenth and fifteenth bit of the word quantity S is 0. Because the word quantity S is initialized to zero the first stepping for both registers is always a 2-stepping. The probability that the six first steppings is 2-steppings can therefore be assumed to be $(1/2)^{10} = 2^{-10}$.
2. Let no pair of the first 8 α be equal. The probability for this is

$$\frac{256!}{(256-8)! \cdot 256^8} \approx 0.90$$

3. Let no pair of the following 40 α be equal. The probability for this is

$$\frac{256!}{(256-40)! \cdot 256^{40}} \approx 0.04$$

Because all the steppings for both the registers are 2-steppings all the stages in both LFSRs are used to generate keystream. The probability that all three of the above assumptions holds is greater than 2^{-15} .

Under these assumptions it suffices to guess the four stages $R_9^1, R_{10}^1, R_{11}^1$ and R_{13}^1 (a total of 64 bits) to recover the state. The state can now be recovered with the four equations obtained from the feedback polynomials, the output function and the nonlinear update of R^0 .

$$R_i^0 = \theta^0 \cdot (*R_{i-6}^0 + \mu^0 \cdot (*R_{i-7}^0) \tag{1}$$

$$R_i^1 = \theta^1 \cdot R_{i-4}^1 + \mu^1 \cdot R_{i-9}^1 \tag{2}$$

$$Z_i = \Delta(R_{i+7}^0) + \Delta(R_{i+9}^1) \tag{3}$$

$$*R_i^0 = R_i^0 +_{16} R_{i+2}^1 \tag{4}$$

All operations in (1)–(3) are in the finite field $\mathbb{F}_{2^{16}}$, whereas the $+_{16}$ in (4) is addition modulo 2^{16} . The constants are the ones from the feedback polynomials and the function $\Delta(x)$ is obtained by looking up the two bytes of x in $D8$ and then concatenate them.

From $R_9^1, R_{10}^1, R_{11}^1, R_{13}^1$ and (3) we get $R_7^0, R_8^0, R_9^0, R_{11}^0$. With a knowledge of the four stages R_7^0, R_8^0, R_9^0 and R_{10}^0 we can calculate how $D8$ will be permuted after the first update of the inner state. Let the result of this permutation be $D8'$. As the next 32 α -values are all different we can treat $D8$ as a constant equal to $D8'$ during the next five updates of the state. The rest of the stages in the registers can now be determined in the following order.

(Where $R_i, (3) \rightarrow R_j$ should be read as R_i and (3) gives R_j)

$$\begin{array}{ll}
R_7^0, R_9^0, R_{11}^0, R_9^1, R_{11}^1, R_{13}^1, (4) & \rightarrow *R_7^0, *R_9^0, *R_{11}^0 \\
*R_7^0, R_8^0, *R_9^0, (1) & \rightarrow R_{14}^0, R_{15}^0 \\
R_{14}^0, R_{15}^0, (3) & \rightarrow R_{16}^1, R_{17}^1 \\
R_{15}^0, R_{17}^1, (4) & \rightarrow *R_{15}^0 \\
R_{11}^1, R_{16}^1, (2) & \rightarrow R_{20}^1 \\
R_{20}^1, (3) & \rightarrow R_{18}^0 \\
*R_{11}^0, R_{18}^0, (1) & \rightarrow R_{12}^0 \\
R_{12}^0, (3) & \rightarrow R_{14}^1 \\
R_9^1, R_{14}^1, (2) & \rightarrow R_{18}^1 \\
R_{18}^1, (3) & \rightarrow R_{16}^0 \\
*R_9^0, R_{16}^0, (1) & \rightarrow R_{10}^0 \\
R_{10}^0, (3) & \rightarrow R_{12}^1 \\
R_{10}^0, *R_{11}^0, (3) & \rightarrow R_{17}^0 \\
R_{17}^0, (3) & \rightarrow R_{19}^1 \\
R_{17}^0, R_{19}^1, (4) & \rightarrow *R_{17}^0 \\
R_{10}^1, R_{19}^1, (2) & \rightarrow R_{15}^1 \\
R_{15}^1, (3), (4) & \rightarrow R_{13}^0, *R_{13}^0
\end{array}$$

From R_9^0, \dots, R_{17}^0 and starred and unstarred R_7^0, \dots, R_{13}^0 we can determine $D8$ and S which is the whole state. From this can all future keystream be calculated.

To try all possible values for the 4 register stages takes $O(2^{64})$ time and the probability that such an attack is successful is 2^{-15} . The time complexity for the above attack is therefore $O(2^{79})$.

Hasanzadeh *et al* [2] have lowered the time complexity in a recently presented paper. By using the same attack principle, but with a more careful analysis and selection of 'guessed' values, they reach an overall attack complexity of $O(2^{57.4})$

4 Analysis and update of Polar Bear

There are several unfortunate coincidences that make this attack possible. The most obvious is that the dynamic permutation of bytes $D8$ is not permuted and therefore known initially. Other reasons are the short length of the LFSRs, the use of feedback trinoms, the choice of nonlinear updating of R^0 , and that register stages are too related.

We propose that the security is enhanced by adding a key-dependent pre-mixing of the D8 table in conjunction with the key schedule. We propose that three full rounds of mixing of D8 is used to this end:

1. Expand the key to 768 bytes of expanded key
2. For $i = 0$ to 767
Swap($D8[i \pmod{256}]$, $D8[\text{key}[i]]$)

This will only affect the performance of the key schedule. As far as we have been able to tell, no other change is needed.

Optimization We have been able to optimize the reference code submitted with Polar Bear from 38 cycles/byte on a Pentium M to under 23 cycles/byte. By making a small tweak that change how the permutation of the dynamic permutation $D8$ is done, the code can be optimized further. Instead of reading all β -values and then make the swaps, two β -values is read, the corresponding $D8$ -values are swapped and then the process is repeated for the other two β -values. This makes Polar Bear faster than AES-CTR.

5 Conclusion

The original specification of Polar Bear apparently has weaknesses, but this can easily be fixed with small changes to the algorithm. By making the permutation of $D8$ in the key setup we only lose performance when a new key is exchanged. This is tolerable as the time for key setup is seldom critical as the same key is typically used with a large number of different IVs, and time for key setup is usually small compared to the time used to generate and exchange a new key.

References

1. Johan Håstad and Mats Näslund. *The Stream Cipher Polar Bear*. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/021. 2005. <http://www.ecrypt.eu.org/stream>.
2. Mahdi Hasanzadeh, Elham Shakour and Shahram Khazaei. *Improved Cryptanalysis of Polar Bear*. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/084. 2006. <http://www.ecrypt.eu.org/stream>.