The Stream Cipher Polar Bear

Johan Håstad NADA
Royal Inst. of Technology
SE-10044 Stockholm, Sweden

Mats Näslund*
Communications Security Lab
Ericsson Research
SE-16480 Stockholm, Sweden

April 29, 2005

Abstract

We propose a new stream cipher called "Polar Bear". It combines constructions used successfully in other situations. The key initialization uses Rijndael, the steady state uses a dynamic table as in RC4 and we guarantee a long period through the use of a linear feedback shiftregister. The efficiency for short messages is better than that of RC4 while for long messages Polar Bear is about a factor two behind in efficiency. On the other hand, we believe our construction has a larger safety margin compared to RC4.

1 Introduction

There is a big industry demand for secure and efficient stream ciphers, the mobile/wireless communication sector being one of the foremost "consumers" of such ciphers. The fact that stream ciphers do not expand messages, are tolerant to bit-errors[1], etc, are desirable properties for use with narrow-bandwidth wireless links and voice coders designed to perform well in the presence of a few bit errors. From scientific point of view, there is also an interest to get a better understanding for how to design stream ciphers, since many of the proposed schemes in the past have been more or less severely attacked.

In response to the NESSIE call for primitives a few years ago, a number of stream cipher proposals were submitted, but none "survived" since the there was a feeling that none of the proposals was sufficiently secure and efficient. The IST 6FP ECRYPT Network of Excellence has now therefore issued a new open call, now directed only towards stream ciphers.

In this paper a stream cipher is presented in response to this call. The cipher has been designed with goals:

*Work described here has in part been supported by the Commission of the European Communities through the IST program under contract IST-2002-507932. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

[1]Of course, the error tolerance also open up for attacks on the integrity.

1. The cipher should be secure, meaning that no attacks (key-recovery or distinguishing) better than brute force should be possible.
2. The cipher should be efficient, meaning that it should be substantially faster than AES in counter-mode, when implemented in software.
3. The cipher should support various key sizes, at least up to 128 bits.
4. The cipher should support "re-keying" for packet-based communication, allowing flexible choices for the size of the initialization values (IVs).

We feel we have achieved these goals. We would here also to state formally that we have not inserted any hidden weakness into the construction.

2 Notation

Whenever we say "AES" or "Rijndael", we refer to [2], and not the AES specification, [1], since we need the support for 256-bit blocks.

We use $||$ to denote concatenation, and $\oplus$ denotes bit-wise modulo-2 sum and $+_m$ denotes addition modulo 2^m.

We use byte/word/WORD to denote 8/16/32-bit quantities, respectively. Hexadecimal quantities are prefixed by '0x'. If $a, b, c, \ldots$ are bytes, in $a||b||c||\ldots$, a is the most significant byte etc.

Let T_8 be the Rijndael S-box, i.e. $T_8(0) = $ 0x63, $T_8($0x01$) = $ 0x7c,..., $T_8($0xff$) = $ 0x16. If $w = w'||w''$ is a 2-byte word, we use $T_8(w)$ to denote $T_8(w')||T_8(w'')$, and similarly for WORDs.

We shall make use of arithmetic in $\mathbb{F}_{2^{16}}$, which we throughout represent as

$$\mathbb{F}_2[y]/(y^{16} + y^8 + y^7 + y^5 + 1).$$

Elements of this field shall sometimes represented by integers, where the integer $a = \sum_{i=0}^{15} a_i 2^i$ represents the field element $a(y) = \sum_{i=0}^{15} a_i y^i$ in the standard polynomial basis representation.

3 Cipher Specification

3.1 Informative Description

We first give an informal description of the cipher's operation. 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 but shorter key lengths are accepted and we describe the modifications needed in Section 4.1. The IV can be any number of bytes upto a maximum of 31. The IV size may vary from message to message for a given key, if so desired.

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. Bytes are selected from R^0, R^1 and run through the permutation D_8 to produce the 4 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.

We now turn to the normative cipher specification.

3.2 Initialization and Key Schedule (Normative)

We use Rijndael with the parameters $Nb = 8, Nr = 5, Nk = 4$ with a full final round including the MixColumns operation. The description in the following two subsections is (intended to be) identical to that in [2], and is only reproduced for self-containment.

3.2.1 Key Schedule

As already mentioned, the key schedule is identical to the Rijndael key schedule with the above parameters, and may be done once per key, saving the expanded key.

To be precise, from the 16-byte key $(k_i)_{i=0}^{15}$, create an expanded key, $(w_j)_{j=0}^{47}$ of WORDs by setting $w_j = k_{4j}||k_{4j+1}||k_{4j+3}||k_{4j+3}$ for $0 \leq j \leq 3$ then for j divisible by 4 set

$$w_j = (T_8(\text{rotword}(w_{j-1}) \oplus \text{Rcon}(j/4)) \oplus w_{j-4},$$

where rotword rotates the word one byte and Rcon are the round constants of Rijndael. If j is not divisible by 4 we set

$$w_j = w_{j-1} \oplus w_{j-4}.$$

The w_k-values are in the sequel divided into round keys, the rth round key consisting of $W^r = w_{8r}, \ldots, w_{8r+7}$, $r = 0, \ldots, 5$. Each W^r is interpreted as a 4×8 array of bytes, whose entry in row i, column j is the ith byte of w_{8r+j}, $0 \leq i \leq 3, 0 \leq j \leq 7$.

3.2.2 Initialization

Let the IV contain n bytes, $(IV_i)_{i=0}^{n-1}$ where $n \leq 31$. Set $IV_n = \text{0x10}$ and $IV_i = 0$, $n + 1 \leq i \leq 31$.

Load the IV into an 4×8 array by setting

$$A_{i,j} = IV_{4j+i}, \quad 0 \leq i \leq 3, 0 \leq j \leq 7.$$

Perform a five round encryption of Rijndael as follows.

First the first round key, i.e. do

$$A_{i,j} = W^0_{i,j} \oplus A_{i,j} \quad 0 \leq i \leq 3, 0 \leq j \leq 7$$

Repeat for $r = 1, 2, 3, 4, 5$:

1. Replace entries in A by applying T_8 to each byte of A.
2. ShiftRows. First row is not shifted, second row is shifted one step right, third row 3 is shifted three steps and fourth row is shifted four steps. (All "shifts" are cyclic, allowing wrap around the edges of the matrix A.)
3. Apply MixColumns, i.e. treat each column as a degree 3 polynomial over $\mathbb{F}_{2^8}$ and multiply it by $a(x) = (1 + y)x^3 + x^2 + x + y$ modulo $x^4 + 1$.
4. Add a round key,

$$A_{i,j} = W^r_{i,j} \oplus A_{i,j} \quad 0 \leq i \leq 3, 0 \leq j \leq 7.$$

For $i = 0, 1, \dots 31$, let b_i be the bytes output, $b_{4j+i} = A_{i,j}$, for $0 \leq i \leq 3$ and $0 \leq j \leq 7$. Form words $z_j = b_{2j+1} || b_{2j}$, $0 \leq j \leq 15$. Load z_j into the jth state of R^0 for $j = 0, 1, \dots 6$ and z_{7+j} into the jth state of R^1 for $j = 0, 1, \dots 8$.

Initialize S to 0 and the table D_8 is initialized to agree with T_8, $D_8(j) = T_8(j)$, $0 \leq j \leq 255$. This completes cipher initialization.

3.3 Next State Function (Normative)

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.

3.3.1 Stepping of registers

Let $\ell_0 = 7$ and $\ell_1 = 9$ be the lengths of the registers,

For $i = 0, 1$ let b^i be the $14 + i$th bit of S, i.e. $b^i \triangleq \lfloor S/2^{14+i} \rfloor \bmod 2$. R^i is stepped $2 + b^i$ steps with a sparse feedback where one (1) such step consists of

- set $f^i \leftarrow \theta^i R^i_{j^i} + \mu^i R^i_0$ for constants θ^i, j^i, and μ^i,
- set $R^i_j \leftarrow R^i_{j+1}$, $j = 0, 1, \dots, \ell_i - 2$, and
- feedback $R^i_{\ell_i - 1} \leftarrow f^i$.

Specifically, R^0, R^1 are defined by the primitive polynomials

$$p^0(x) \triangleq \text{0x5ceb} \cdot x^7 + \text{0x8b5a} \cdot x^6 + 1$$

and

$$p^1(x) \triangleq \text{0x2c62} \cdot x^9 + \text{0x689a} \cdot x^4 + 1,$$

so that $j^0 = 1, j^1 = 5$, $\mu^0 =$ 0x5ceb, $\mu^1 =$ 0x2c62, $\theta^0 =$ 0x8b5a, $\theta^1 =$ 0x689a.

After stepping both R^0, R^1 above, repeat the following for $i = 0, 1$ (do steps 1–4 first for $i = 0$, then repeat them for $i = 1$):

1. Let W^i be the two leftmost 16-bit words of R^i (i.e. $R^i_{\ell_i-1}, R^i_{\ell_i-2}$).
2. Write W^i as four bytes $W^i = \alpha^i_0||\alpha^i_1||\alpha^i_2||\alpha^i_3$.
3. Let $\beta^i_j = D_8(\alpha^i_j)$, $j = 0, 1, 2, 3$.
4. Swap elements in D_8 by $D_8(\alpha^i_0) \leftarrow \beta^i_2$, $D_8(\alpha^i_1) \leftarrow \beta^i_0$, $D_8(\alpha^i_2) \leftarrow \beta^i_3$, $D_8(\alpha^i_3) \leftarrow \beta^i_1$.

Next,

- Define two 16-bit words $\gamma_j \triangleq \beta^1_{2j}||\beta^1_{2j+1}$, $j = 0, 1$.
- Update S according to $S \leftarrow S +_{16} \gamma_0$.
- Update R^0 according to $R^0_5 \leftarrow R^0_5 +_{16} \gamma_1$.

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

3.4 Output Generation (Normative)

We form four output bytes $b_0||b_1||b_2||b_3$ where

$$b_j \triangleq \beta^0_j \oplus \beta^1_j.$$

If more output bytes are required, go back and iterate Next-state followed by Output generation as above.

4 Variation

In a situation with frequent reinitialization we replace the dynamic table D_8 used in the running of the algorithm by a smaller table D_4 $\{0,1\}^4 \mapsto \{0,1\}^4$. This variant might also be preferred in certain hardware situations.

We use the standard T_8 as before in the initialization phase and D_4 is only used in the running of the algorithm. The table D_4 is initialized by setting for $x \neq 0$

$$D_4(x) = 1 + y^2 + 1/x$$

where elements are treated as belonging to

$$\mathbb{F}_2[y]/(y^4 + y + 1).$$

and $D_4(0) = 1 + y^2$.

In the running of the algorithm after the stepping of the registers we first set W^i to the leftmost word and run all remaining steps with nibbles replacing bytes, and in particular we produce 4 output nibbles.

This is repeated with W^i being the second leftmost word.

The word quantities γ_0 and γ_1 of the main version of the algorithm are now replaced by two bytes, one in each iteration. The value used to update S and R_5 are the concatenation of the two values produced, the first value giving the eight least significant bits.

4.1 Different length keys

We allow other key sizes in the range 80-120 bits which are supported by the following modifications to the key schedule (no other changes are needed):

- If the key length is 96 bits we use Rijndael expansion with $Nk = 3$.
- If the bytes of the key are $(k_i)_{i=0}^{m-1}$ and the number of bytes, m, is not a multiple of 4 then let k be the smallest multiple of 4 larger than m. Set $k_i = T_8(k_{i-m})$ for $m \leq i \leq k-1$ and then treat this expanded key as a 96-bit or 128-bit key.

5 Motivation

Our philosophy of the design has been to re-use components that have proven reliable in previous cryptographic constructions.

A very classical component is a LFSR which supplies basic properties such as randomness of each byte produced and a guarantee of a long period. This component exists in our construction as the register R^1.

The register R^0 is more speculative. It has partly the character of a traditional LFSR but since we modify its content depending on the contents of R^1 and the dynamic table D_8 we have no guarantees concerning its period. On the other hand it is non-linear and thus it does not have the traditional weaknesses of an LFSR.

A very important stream-cipher is given by RC4 [4] and the heart of that algorithm is given by a dynamically changing table of size 256. The existence of this component seems to rule out linear cryptanalysis. It does also, in a natural way, create a large internal state ruling out many attacks. We find it natural to use this excellent component.

The problems with the security of RC4 have been associated with insufficient initialization and because of this our initialization is very different.

The problem of initializing some variables from a secret key and a publically known initialization vector is in spirit very similar to a block encryption scheme.

In both cases we want mix a secret key with other information in an efficient way. It is hence natural to reuse parts of a widely accepted block encryption scheme such as Rijndael. The requirements for initialization is, however, much less severe than for actual encryption. In view of this we feel that it is sufficient to use only five of the 14 rounds used in 256-bit Rijndael.

It would be possible to use keys up to 256-bits in size, but as the results of the initialization is 256 bits this might open up the possibility for "non-trivial" attacks. Thus, while we do think that an increased length of the key does increase security the current scheme might not be able to use the full potential of a maximally long key.

5.1 The variant

The existence of the dynamic table D_8 makes sure that we have a large internal state and this adds to security. It does carry a cost as it has to be kept in memory and initialized. This cost might be non-trivial in application with frequent initialization and in situations where one processor has to switch between different encryption sessions.

In a situation where we only encrypt small packets the full advantage of this dynamic table is not fulfilled. This is the case as mixing is not immediate and hence D_8 will in fact only take on a very tiny fraction of all possible values as it will remain close to the original table T_8. In such a situation, replacing D_8 by a smaller table eliminates most of the drawbacks while keeping some of the advantages.

6 Security discussion

As already mentioned our construction is based on known primitives and our hope for security is based on the lack of attacks/understanding of potential weaknesses related to RC4 and Rijndael.

When encrypting long messages our construction behaves like a strengthened version of RC4. We note that the dynamically changing table does seem to rule out linear cryptanalysis over long sequences of bits. Indeed no such attack has been proposed on RC4. The LFSRs make sure we do not enter a short loop. It is true that due to the very large state space a small period is exceptionally unlikely to happen in RC4 but turning this probably well founded hope into certainty is certainly an added feature.

An additional feature of the construction is that we do not output values in D_8 but only the exclusive-or of two values. Thus the outputs have an even lower relation to the internal state of the cipher than is the case for RC4. This implies that it seems very difficult to accumulate any information of the internal state by observing the output.

In short we feel that there is no indication that observing long sequences of outputs from the proposed cipher will enable an attacker to mount an interesting attack. In fact the security margins here seem substantial.

Another type of attack is to observe the first few bytes of output for many different IVs. To be on the safe side we should, for the discussion, assume that the IVs are different but chosen by the attacker.

When looking at the encryption of the first couple of bytes the general situation is similar to the analysis of a hash function or a block-cipher and in both cases we have seen progress lately with new attacks. By our choices in the construction in our case the analysis of block ciphers is the most relevant. Had we chosen to use a full Rijndael encryption for the initialization the argument would have been simple. Any method that could have been used to distinguish the first output bytes from random bits could have been used to distinguish the output from Rijndael from random bits. As substantial effort has been devoted to this problem without any noticeable results this would be a strong argument for security had we used the full Rijndael.

To speed up initialization we have limited the number of rounds of Rijndael to 5 and since there are attacks [3] to distinguish the output of Rijndael of upto 7 rounds from random bits the margin of safety does not appear to be overwhelming.

The first output byte of Polar Bear is an exclusive-or of two applications of T_8 each to the linear combination of at least three outputs of Rijndael. Current distinguishing attacks cannot make use of this limited information. In fact it seems very hard to use this type of very special information but as there has been no reason to analyze this situation the lack of such results is not a very strong indication that such information is not useful. We do believe however that it is not possible. It is always difficult to argue the non-existence of attacks but let us at least note the following.

The only general method for extracting information is through linear cryptanalysis. Linear cryptanalysis has been extensively studied in connection with AES on 128-bit blocks and already (see [2]) 4 rounds is sufficient to bring down any characteristic well below 2^{-64} which is the threshold for a nontrivial attack on a 128 bit cipher. The 256-bit variant is not analyzed to the same extent but we have an extra round to compensate for the larger block size.

7 Performance

Preliminary benchmarks are as follows (1400MHz Pentium, MS Visual C compiler) without any deeper optimization of Polar Bear code, except that using tables for $\mathbb{F}_{2^{16}}$ multiplications and optimizations provided by the compiler. The Rijndael coded that was included was of optimized nature. The timings are an average over 1000 messages and includes time for initialization of a new IV but not the key schedule. The key schedule time is the same as for Rijndael on 256 bit blocks. For large packets, the performance levels out at about 190Mb/s, since the initialization overhead then becomes neglgible. Performance in cycles per byte follows using the clock frequency.

Comparing to a simplistic implementation of RC4, the performance is faster than RC4 on short packets (up to about 160Byte), and for large packets, RC4

Msg size (bytes)	Mbits/s
32	135
64	155
128	170
256	180
512	184
1024	190

Table 1: Preliminary benchmarks (M = 2^{20}).

is faster by a factor slightly below 2.

7.1 Implementation Suggestions

While we have not made any optimizations, we point out some obvious techniques that can be used. For initialization, clearly all implementation techniques used to speed up AES/Rijndael applies and there is no need to repeat them here and we refer to [2, 5]. Note that a standard Rijndael implementation needs to tweaked in two ways

- Use 5 rounds.
- Apply MixColumn in all rounds.

7.1.1 Finite field operations

We make use of some arithmetic over $\mathbb{F}_{2^{16}}$. Multiplication by (fixed) constants, θ, in this field, e.g. needed for the LFSR implementation, can be done using linearity

$$\theta \cdot v = \theta \cdot (v_1 y^8 + v_0) = (\theta y^8) \cdot v_1 \oplus \theta \cdot v_0,$$

where v_1, v_0 are the 8 most/least significant bits of v. Hence, two 256-word tables containing the values of $(\theta y^8)v_1$ and θv_0, for all v_1, v_0 can be used together with some bit operations and an XOR.

Alternatively, note that the element y is a generator for the multiplicative group of $\mathbb{F}_{2^{16}}$ for the representation chosen. Hence, pre-computed tables of discrete logarithms and anti-logarithms can be used:

$$\theta \cdot v = \text{ALOG}[(\text{DLOG}[v] + c_\theta) \bmod (2^{16} - 1)],$$

for a constant $c_\theta = \text{DLOG}[\theta]$.

8 Summary

We thank Per Austrin and Karl Norrman for helpful suggestions, including programming assistance.

References

[1] Advanced Encryption Standard (AES), NIST FIPS-PUB 197, Nov 26, 2001.

[2] J. Daemen and V. Rijmen, The design of Rijndael, Springer-Verlag, 2002.

[3] H. Gilbert and M. Minier, A collision attack on 7 rounds of Rijndael. Proceedings of third Advanced Encryption Standard conference, pp 230-241, 2000, NIST.

[4] R. L. Rivest, The RC4 Encryption Algorithm RSA Data Security, Inc, Mar 1992.

[5] The Rijndael page, www.iaik.tu-graz.ac.at/research/krypto/AES/old/~rijmen/rijndael/

A Test-vector

All values in hex.

A.1 Key schedule

input key (len = 16):

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

expanded key (size 192):

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
62 63 63 63 62 63 63 63 62 63 63 63 62 63 63 63
9b 98 98 c9 f9 fb fb aa 9b 98 98 c9 f9 fb fb aa
90 97 34 50 69 6c cf fa f2 f4 57 33 0b 0f ac 99
ee 06 da 7b 87 6a 15 81 75 9e 42 b2 7e 91 ee 2b
7f 2e 2b 88 f8 44 3e 09 8d da 7c bb f3 4b 92 90
ec 61 4b 85 14 25 75 8c 99 ff 09 37 6a b4 9b a7
21 75 17 87 35 50 62 0b ac af 6b 3c c6 1b f0 9b
0e f9 03 33 3b a9 61 38 97 06 0a 04 51 1d fa 9f
b1 d4 d8 e2 8a 7d b9 da 1d 7b b3 de 4c 66 49 41
b4 ef 5b cb 3e 92 e2 11 23 e9 51 cf 6f 8f 18 8e
ab 42 42 63 95 d0 a0 72 b6 39 f1 bd d9 b6 e9 33
```

A.2 Initialization

IV (len = 16):

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

expanded IV:

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

R0 inital value: `e58e 256f f115 9f2a 0d62 559f da11`
R1 initial value: `29e1 740b a80b b5e5 bd53 899e e2ec 541b dbcb`

A.3 Output keystream

key stream prefix:
`0e 3c 44 72 cf 4c 7e aa f6 4b 12 e0 49 78 2a dc`