Evaluation of Serpent, one of the AES finalists, on 8-bit microcontrollers

Vincent Journot

May 11, 2000

Abstract

The algorithm of Serpent [1] is one of finalists of the AES, Advanced Encryption Standard. AES will replace DES, Data Encryption Standard. [2]

More information

- about Serpent is available at `http://www.cl.cam.ac.uk/users/rja14`
- about AES (and DES) is available at `http://csrc.nist.gov/encryption/aes`

1 Description of the algorithm

Serpent is described in the *bitslice* mode.

1.1 The Thirty-two rounds

The Serpent algorithm is a 32-round SP-network, i.e. it is composed of 32 rounds and it is based on substitutions and permutations. The first thirty-one are identical and can be described as below:

Let B_0 be the plain text and B_i its value at round i;

Let K_i be the subkey used in round i;

Let S_i be the substitution box;

Let L be the linear transform:

$$B_{i+1} = L(S_i(B_i \oplus K_i))$$

where $\oplus$ refers to an exclusive-or.

The last round is slightly different as the linear transform is replaced by an exclusive-or with a thirty-third subkey K_{32}:

$$B_{32} = (S_7(B_{31} \oplus K_{31})) \oplus K_{32}$$

As a matter of fact, applying the linear transform during the last round does not increase the security of the algorithm as it can be easily inverted.

1.2 The key schedule

Serpent generates 33 128-bit subkeys from the 256-bit key. Thus, it divides the key into eight 32-bit words w_{-8} ... w_{-1}. Then it generates 132 w_i using the following recurrence:

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \Phi \oplus i) <<< 11$$

Where

Φ stands for the first thirty-two bits of the fractional part of the *golden ratio* $\frac{\sqrt{5}+1}{2}$, i.e. 0x9e3779b9 in hexadecimal.
$<<<$ 11 stands for a rotation by 11 bits to the left.

Adding the golden ratio constant ensures an even distribution of key bits throughout the rounds, and eliminates weak keys. It also prevents complementation, i.e. if we take the complement of both a plaintext and a key, the result is not the complement of the ciphertext.

We apply the same S-box to each group of four w_i. In the first round, we use S-box 3, then in the second round, S-box 2 and so on. Each of the eight S-boxes is used four times, except for S-box 3 which is used five times.

$$\{k_0, k_1, k_2, k_3\} = S_3(w_0, w_1, w_2, w_3)$$

The first subkey is $K_0 = \{k_0, k_1, k_2, k_3\}$. Identically,

$$K_i = \{k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3}\}$$

1.3 The linear transform

The linear transform maximizes the avalanche effect. The S-boxes have the property that a single input bit change will cause two output bits to change. The rotations by 1, 3, 5, 7, 13 and 22 bits have no common factor — modulo 32 — except 1. After three rounds, each plaintext bit affects all the data bits.

X_0, X_1, X_2, X_3 are four 32-bit words.

$<<<$ stands for left rotation

$<<$ stands for left shift: this is a rotation with the loss of the leftmost bit.

$$
\begin{aligned}
X_0, X_1, X_2, X_3 &:= S_i(B_i \oplus K_i) \\
X_0 &:= X_0 <<< 13 \\
X_2 &:= X_2 <<< 3 \\
X_1 &:= X_1 \oplus X_0 \oplus X_2 \\
X_3 &:= X_3 \oplus X_2 \oplus (X_0 << 3) \\
X_1 &:= X_1 <<< 1 \\
X_3 &:= X_3 <<< 7 \\
X_0 &:= X_0 \oplus X_1 \oplus X_3 \\
X_2 &:= X_2 \oplus X_3 \oplus (X_1 << 7) \\
X_0 &:= X_0 <<< 5 \\
X_2 &:= X_2 <<< 22 \\
B_{i+1} &:= X_0, X_1, X_2, X_3
\end{aligned}
$$

2 The implementation of Serpent on an 8051 microcontroller

The 8051 microcontroller family (MCS-51) is used in smartcards, such as bank cards. They are programmed in an assembly language [3].

The S-boxes

This presents Dag Arne Osvik's optimization [4].

The implementation is based on logical operations between five registers:

S-box 0					
R_3	^=	R_0	R_4	=	R_1
R_1	&=	R_3	R_4	^=	R_2
R_1	^=	R_0	R_0	—=	R_3
R_0	^=	R_4	R_4	^=	R_3
R_3	^=	R_2	R_2	—=	R_1
R_2	^=	R_4	R_4	=~	R_4
R_4	—=	R_1	R_1	^=	R_3
R_1	^=	R_4	R_3	—=	R_0
R_1	^=	R_3	R_4	^=	R_3

Table 1: Implementation of S-box 0

R_3 ^= R_0 means $R_3 := R_3 \oplus R_0$

R_4 =~ R_4 means that we replace R_4 by its bit-wise complement

— stands for bit-wise "or"

& stands for bit-wise "and"

We then just have to reorder the registers R_i.

2.1 Serpent version 1.3

Description of the program

The program:

- loads the key and the plain text to encrypt;
- reorders the bits of both the key and the plaintext to have a little endian representation (done by the ORDER2 subroutine);
- calls the ENCRYPT main subroutine. This subroutine can be divided into two parts:
  - the ROUNDS subroutine which consists of four loops. Each loop runs eight rounds.
  - the ROUND32 subroutine whose role is obvious;
- reorders the bits of both the key and the plain text to have a little endian representation (done by the ORDER2 subroutine).

Each round i consists of:

- the generation of four new w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3} thanks to four calls to the SKEY subroutine;
- a loop repeated four times which works on a quarter of the subkey and the text B_i. This loop applies S-box $S_{3-i \mod 8}$ to the subkey, makes the or-exclusive with B_i (XORB subroutine), and then applies S-box $S_{i \mod 8}$;
- the linear transform.

2.2 Performances

Though the version of Serpent which requires 70 000 machine cycles is far slower than DES, and *Triple DES*, one must keep in mind that the algorithm has been designed to take advantage of the available technology, i.e. it computes with 32-bit registers. It is greatly slowed down by an implementation on 8-bit registers. The most striking example is the rotation of a 32-bit word on the left by one bit. It only requires one instruction on 32-bit registers. However, in a 8-bit mode, it needs 14 instructions, as shown below, i.e. 18 machine cycles (we must add 4 cycles for the call of the subroutine)!

```
ROTL1:  MACRO   one
        MOV     one,A
        RLC     A
        MOV     A,one+3
        RLC     A
        MOV     one+3,A
        MOV     A,one+2
        RLC     A
        MOV     one+2,A
        MOV     A,one+1
        RLC     A
        MOV     one+1,A
        MOV     A,one
        RLC     A
        MOV     one,A
ENDM
```

I made some tests in order to compare Serpent with the other candidates and DES. I used the implementations which are provided in the folder *otheraes*. The simulator which was used is provided in the floppy disk — this is an unregistered version. The numbers for DES correspond to the encryption of a 64-bit block in 10ms.

Candidat	RAM used (bytes)	code size (bytes)	Number of machine cycles
MARS seems to be hardly implementable on 8-bit microcontrollers			
RC6	77	706	58056
RIJNDAEL	54	826	3977
SERPENT	56	2021 (7130)	70339 (60300)
TWOFISH	52	2063	19819
DES			6000
TRIPLE DES			18000

Table 2: comparison of the five candidates with DES

Here are the different numbers[1] for the five finalists' implementations. The numbers given for DES and Triple DES are for a 128-bit block encryption, i.e. two encryptions. Serpent is three times as slow as Triple DES but it is much more secure. If we reduced the number of rounds from 32 to 16, which remains secure enough, Serpent is then slightly slower than Triple DES.

Serpent 1.3 requires 56 RAM bytes, which is nearly optimal. As a matter of fact, we need:

- 32 bytes to store eight w_i;

[1]The numbers between parentheses are those of another version of Serpent whose code size is a real drawback. The SKEY subroutine has been replaced by a macro.

- 16 bytes to keep the B_i block;
- 1 byte to store the round number;
- 5 registers to apply a S-box.

The last two bytes are used to store temporary values such as indexes. The program uses very few registers. The use of the stack and the register banks would artificially reduce the number of necessary RAM bytes — since the registers banks are not taken account of in the calculation. This explain the slight difference between Rijndael, Serpent and Twofish.

Remark

If you want to test serpent.asm, you will get the following result:

- plaintext = 00000000000000000000000000000000
- key = 08000000000000000000000000000000000000000000000000000000000000000000000000000000
- ciphertext = EC9D6557EED58E6CF89A746BBD

whereas if you try the C implementation in [1], you will get:

- plaintext = 00000000000000000000000000000000
- key = 00000000000000000000000000000000000000000000000000000000000000000000000000000008
- ciphertext = BD6B749AF86C8ED5EE57659DEC

This is a matter of little endian and big endian.

References

[1] Ross Anderson, Eli Biham, Lars Knudsen.
"Serpent: A Proposal for the Advanced Encryption Standard."
Available on the net at `http://csrc.nist.gov/encryption/aes.`

[2] National Institute of Standards and Technology.
Data Encryption Standard (DES).
FIPS PUB 46-3, reaffirmed 1999 October 25.

[3] Intel Corporation.
MCS 51 Microcontroller Family User's Manual.
February 1994.

[4] Dag Arne Osvik.
"Speeding up Serpent."
To appear in the proceedings of the 3rd AES Candidate Conference.
March 2000.