Serpent Cipher

Final Report

Cryptography

Team Rikki-Tikki-Tavi

Authors:
Nicholas Sereni
Dan Grau
Karl Berger

Prof.:
Alan Kaminsky

February 8, 2013

Contents

1 Serpent Cipher

1.1 Background

The Serpent cipher was designed by Ross Angerson, Eli Biham, and Lars Knudsen. It was created as candidate for the Advanced Encryption Standard. Based on AES requirements, it has a 128 bit block length and a 256 bit key length. It also supports keys sizes of 128 and 192 bits.

1.2 The Algorithm

Serpent splits the 128 bit block into four 32-bit words. There are 32 rounds. Each round uses a subkey generated from the user key. The user key does not have a size requirement, but it becomes fixed at 128, 192, or 256 bits. Padding is achieved by appending a "1" followed by "0" bits. The algorithm can be summarized as:

- An initial permutation
- 32 rounds consisting of:
  - key mixing operation
  - S-boxes
  - linear transformation (replaced by a a key mixing operation in the final round)
- A final permutation

  This process is explained visually in Figure 1.

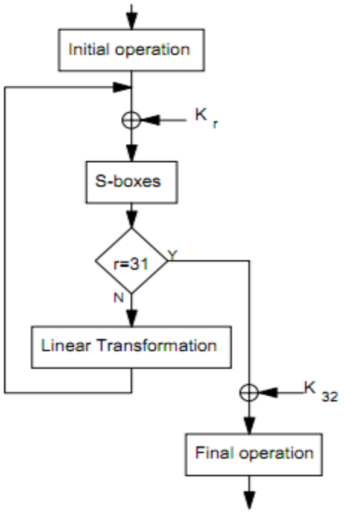

Figure 1: Block Diagram of the Encryption Process

1.2.1 Initial and Final Permutations

The initial and final permutations are simply bit mappings. This is a very simple method and is especially effective in hardware. In permutations, each bit on the input is assigned to a different index on the output. There are no operations performed, only reassignments. Figure 2 shows this general idea. Please note that this diagram does not represent the diagram for Serpent(it's acutally for DES) and is only being used for an example. The actual permutations can be found in [**?**].

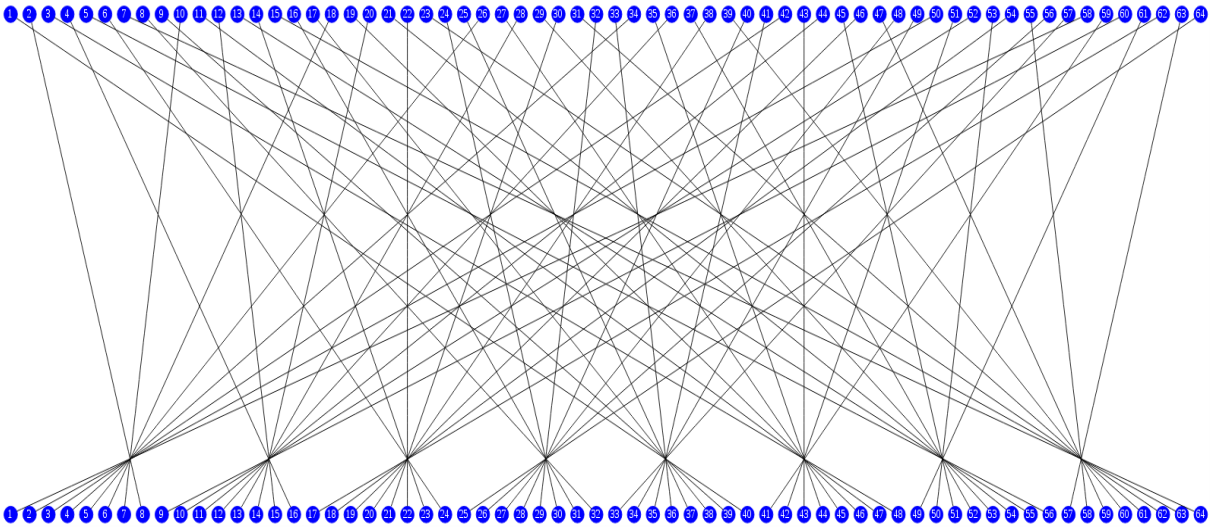

Figure 2: A General Permutation

1.2.2 S-boxes

An S-box is simply a look-up-table. In Serpent, the S-boxes are 4-bit permutations. The advantage of an S-box is that for a 1-bit change of an input value, the output is guaranteed to be altered by more than one bit (at least for the Serpent S-boxes). An exmaple S-box can be seen in Figure 3. Please note that this diagram does not represent the S-box for Serpent. The Serpent S-boxes can be seen in [**?**].

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 3: A General S-box

1.2.3 Linear Transformation

The linear transformation functions acts on the 128-bit block as four 32-bit words. Each word is linearly adjusted and combined with other words according to Figure 4. In this figure, $<<<$ denotes a left rotation, and $<<$ denotes a left shift.

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

Figure 4: Linear Transformation

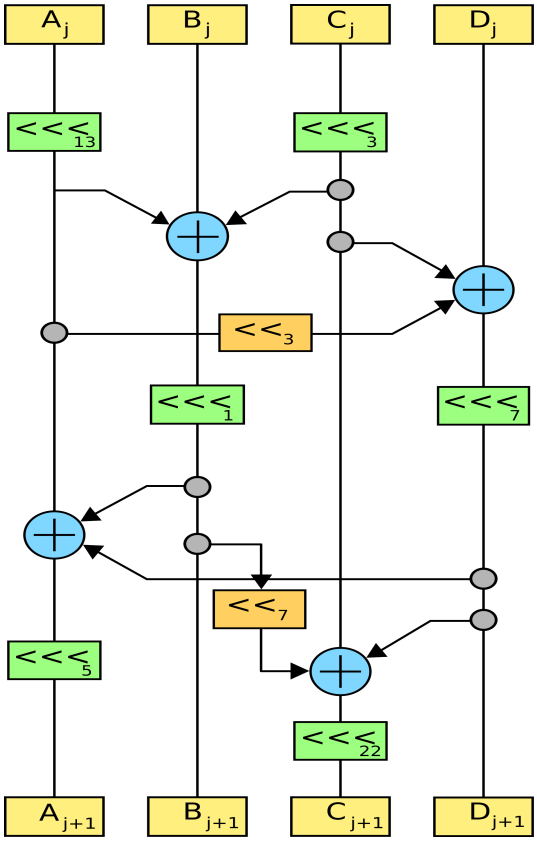

Figure 5: Linear Transformation

1.2.4 Decryption

Decrption is very similar to encryption. However, inverse S-boxes and linear transformations are used as well as a reverse order of subkeys. This is made most clear with the use of Figure 6.

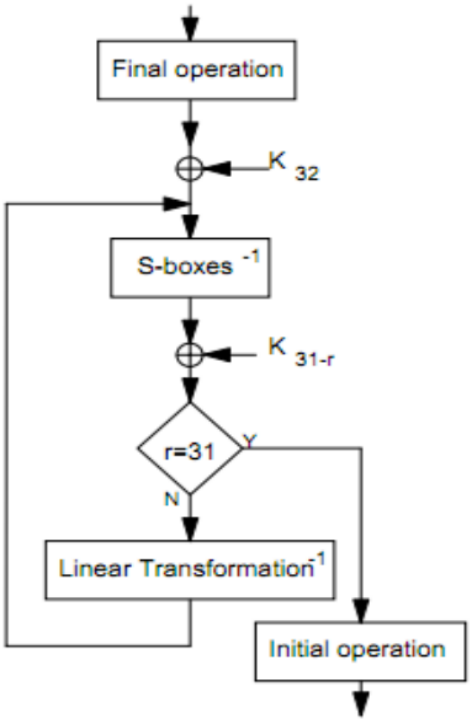

Figure 6: Block Diagram of the Decryption Process

2 Original Implementation

2.1 Timing Results

All timing results were measured on the CS machine, Joplin. Source-code level results were truncated as problem methods are easily identifiable within the first few lines.

2.1.1 Total Running Time with no JIT compiler

$ time java -Xint Serpent 1
49672ba898d98df95019180445491089
real 0m0.135s

user 0m0.040s
sys 0m0.024s

2.1.2 Runtime Profiles of over 100 Seconds

$java -Xint -Xprof Serpent 100000
d3f68d0623563be822d68dde8f4ad282
Flat profile of 156.42 secs (15402 total ticks): main

Interpreted		native	Method
26.7 % 4119	+	0	Serpent.sBox
22.8 % 3511	+	0	Serpent.getRoundKey
21.4 % 3290	+	0	Serpent.initPermutation
21.3 % 3274	+	0	Serpent.linearTransform
3.5 % 540	+	0	Serpent.setKey
3.5 % 537	+	0	Serpent.finalPermutation
0.8 % 128	+	0	Serpent.encrypt
0.0 % 0	+	2	java.io.FileInputStream.readBytes
0.0 % 0	+	1	java.io.UnixFileSystem.getBooleanAttributes0
100.0 % 15399	+	3	Total interpreted

$time java -Xint -agentlib:hprof=cpu=samples,depth=10 Serpent 100000
d3f68d0623563be822d68dde8f4ad282
Dumping CPU usage by sampling running threads ... done.
real 2m38.259s
user 2m38.062s
sys 0m0.488s

rank	self	accum	count	trace	method
1	7.06 %	7.06 %	1092	300041	Serpent.initPermutation
2	6.91 %	13.97 %	1070	300033	Serpent.linearTransform
3	6.57 %	20.54 %	1016	300042	Serpent.getRoundKey
4	6.30 %	26.84 %	975	300027	Serpent.sBox
5	6.19 %	33.03 %	958	300074	Serpent.initPermutation
6	4.25 %	37.27 %	657	300032	Serpent.sBox
7	2.43 %	39.70 %	376	300040	Serpent.initPermutation
8	2.35 %	42.05 %	364	300045	Serpent.initPermutation
9	2.31 %	44.36 %	357	300086	Serpent.finalPermutation
10	2.29 %	46.66 %	355	300034	Serpent.finalPermutation
11	2.27 %	48.93 %	352	300051	Serpent.initPermutation