Serpent: A Flexible Block Cipher With Maximum Assurance

Ross Anderson[1] Eli Biham[2] Lars Knudsen[3]

[1] Cambridge University, England; email `rja14@cl.cam.ac.uk`
[2] Technion, Haifa, Israel; email `biham@cs.technion.ac.il`
[3] University of Bergen, Norway; email `lars.knudsen@ii.uib.no`

1 Introduction

This paper presents a candidate block cipher for the Advanced Encryption Standard (AES). AES is an intriguing challenge to the designer, because of the great length of time the selected algorithm will have to resist attack.

Twenty years after the adoption of its predecessor, DES, equipment containing DES is continuing to be fielded in massive quantities. In the banking sector alone, we find a global population of some 400,000 automatic teller machines growing at perhaps 20% per annum and being replaced on average after seven years, giving annual sales of over 100,000 units. Even if there were an immediate consensus to migrate to a successor algorithm, it would probably take five years to agree the design details and the migration strategy. It is more likely that a further ten or even twenty years will be required to get agreement among the 20,000 or so affected financial institutions. Similar considerations apply to other established application areas such as prepayment utility meters, burglar alarms and road toll tags.

So even if AES only persists for 25 years as a standard, we expect it to be used for a further 25 years in legacy systems. During this period, it will protect not only embedded systems such as ATMs, but also large quantities of government data and personal health information whose confidentiality may have to be maintained for the lifetime of individuals. It follows that AES must be designed to withstand advances in both engineering and cryptanalysis over a period of at least a century.

We therefore decided to adopt a highly conservative design philosophy. We did not feel it appropriate to use relatively novel and untested ideas in a cipher with such extreme assurance requirements. So we did not wish to rely on operations such as the mixing of operations from different algebraic groups, or on data dependent rotations. Although these techniques are promising, they have not been around for very long and the techniques available for their analysis are constantly improving.

This left us with the techniques familiar from DES, namely combining S-boxes with linear mappings in such a way that we can apply the known techniques of differential and linear cryptanalysis [7, 17]. The problem that we now faced was: how could we do this effectively enough to make AES an attractive alternative to (say) triple-DES?

2 The Cipher

The breakthrough came with recent ideas for bitslice implementation of ciphers [2] which were used in the recent software keysearch attack on DES. The basic idea is that just as one can use a 1-bit processor to implement an algorithm such as DES by executing a hardware description of it, using a logical instruction to emulate each gate, so one can also use a 32-bit processor to compute 32 different DES blocks in parallel — using the CPU as a 32-way SIMD machine.

This is much more efficient than the conventional implementation, in which a 32-bit processor is mostly idle as it computes operations on 6 bits, 4 bits, or even single bits. In a bitslice implementation, all the processor's 32 bits can be kept busy. However the problem with using bitslice techniques for DES encryption (as opposed to keysearch) is that one has to process many blocks in parallel, which is not what most applications require.

Serpent was therefore designed so that all operations can be executed using 32-fold parallelism during the encryption or decryption of a single block. We will now describe the algorithm briefly; for a full description and a variety of implementations, see our AES submission package which contains a full paper, and the further materials which are all available at [18].

2.1 The conventional description

Serpent is a 32-round SP-network operating on four 32-bit words, giving a block size of 128 bits. It consists of:

- an initial permutation IP;
- 32 rounds, each consisting of a key mixing operation, a pass through S-boxes, and (in all but the last round) a linear transform. In the last round, this linear transform is replaced by an additional key mixing operation;
- a final permutation IP^{-1}.

Each round uses 32 copies of the same 4-bit to 4-bit S-box. Thus, for example, the first round takes the output of the initial permutation, which we call $\hat{B}_0$, xors it with the first round key $\hat{K}_0$, and passes it through 32 copies of the first S-box S_0. The first copy of S_0 takes bits 0, 1, 2 and 3 of $(\hat{B}_0 \oplus \hat{K}_0)$ as its input and returns the first four bits of an intermediate vector; the next copy of S_0 inputs bits 4–7 of $(\hat{B}_0 \oplus \hat{K}_0)$ and returns the next four bits of the intermediate vector, and so on. The intermediate vector is then transformed using the linear transform, giving $\hat{B}_1$. The initial and final permutations, the linear transform and the S-boxes are specified in tables as with the official definition of DES; these tables are in the full paper at [18].

There are eight different S-boxes each of which is used in four rounds: after using S_7 in round 7, we use S_0 again in round 8, then S_1 in round 9, and so on. The last round R_{31} is slightly different from the others: we apply S_7 on $\hat{B}_{31} \oplus \hat{K}_{31}$, and XOR the result with $\hat{K}_{32}$ rather than applying the linear transform. The result $\hat{B}_{32}$ is then permuted by IP^{-1}, giving the ciphertext.

Thus the cipher may be formally described by the following equations:

$$\begin{aligned}\hat{B}_0 &:= IP(P)\\ \hat{B}_{i+1} &:= R_i(\hat{B}_i)\\ C &:= IP^{-1}(\hat{B}_{32})\end{aligned}$$

where

$$\begin{aligned}R_i(X) &= L(\hat{\mathcal{S}}_i(X \oplus \hat{K}_i)) \qquad i = 0, \ldots, 30\\ R_i(X) &= \hat{\mathcal{S}}_i(X \oplus \hat{K}_i) \oplus \hat{K}_{32} \quad i = 31\end{aligned}$$

here $\hat{\mathcal{S}}_i$ is the application of the S-box $S_{i \bmod 8}$ 32 times in parallel, and L is the linear transform.

Decryption differs from encryption in that the inverse of the S-boxes must be used in the reverse order, as well as the inverse linear transform and reverse order of the subkeys.

An advantage of this design is that it is easy to adapt the extensive cryptanalysis already done on DES. Serpent is actually a better design than DES; we find, for example, that the probability of the best six-round characteristic of Serpent is less than 2^{-58}, while for DES the corresponding figure is about 2^{-20}. Another advantage is that we can provide a very efficient implementation, which we will now describe.

2.2 The bitslice description

Much of the motivation for the above design will become clear as we consider how to implement the algorithm efficiently. We do this in bitslice mode, in which the description of the algorithm is much simpler. No initial and final permutations are required, since the initial and final permutations described above are just those needed to convert the data from and to the bitslice representation. If the bitslice representation of a vector is X, then we denote its standard representation as $\hat{X}$. IP takes the first bit of every nibble, then the second, and so on, packing them into four 32-bit words; we have $IP(X) = \hat{X}$ and $IP^{-1}(\hat{X}) = X$.

In bitslice mode, the cipher consists simply of 32 rounds. The plaintext becomes the first intermediate vector $B_0 = P$, after which the 32 rounds are applied. Each round $i \in \{0, \ldots, 31\}$ consists of three operations:

1. Key Mixing: At each round, a 128-bit subkey K_i is exclusive or'ed with the current intermediate data B_i
2. S-Boxes: The 128-bit combination of input and key is considered as four 32-bit words. The S-box, which is implemented as a sequence of logical operations (as it would be in hardware) is applied to these four words, and the result is four output words. The CPU is thus employed to execute the 32 copies of the S-box simultaneously, resulting in $\mathcal{S}_i(B_i \oplus K_i)$
3. Linear Transform: The 32 bits in each of the output words are linearly mixed by the following sequence of register operations:

$$
\begin{aligned}
X_i &:= \mathcal{S}_i(B_i \oplus K_i) \quad (i = 0, \ldots, 3) \\
X_0 &:= X_0 <<< 13 \\
X_2 &:= X_2 <<< 3 \\
X_1 &:= X_1 \oplus X_0 \oplus X_2 \\
X_3 &:= X_3 \oplus X_2 \oplus (X_0 << 3) \\
X_1 &:= X_1 <<< 1 \\
X_3 &:= X_3 <<< 7 \\
X_0 &:= X_0 \oplus X_1 \oplus X_3 \\
X_2 &:= X_2 \oplus X_3 \oplus (X_1 << 7) \\
X_0 &:= X_0 <<< 5 \\
X_2 &:= X_2 <<< 22 \\
B_{i+1} &:= X_i \quad (i = 0, \ldots, 3)
\end{aligned}
$$

where $<<<$ denotes rotation, and $<<$ denotes shift. In the last round, this linear transform is replaced by an additional key mixing: $B_{32} := S_7(B_{31} \oplus K_{31}) \oplus K_{32}$. At each stage $IP(B_i) = \hat{B}_i$, and $IP(K_i) = \hat{K}_i$.

The first reason for the choice of linear transform is to maximize the avalanche. The S-boxes have the property that a single input bit change will cause two output bits to change, and the choice of linear transform ensures that a single input bit change will cause a maximal number of bit changes after two and more rounds. The effect is that each plaintext bit affects all the data bits after three rounds, as does each round key bit (even if an opponent chooses some subkeys and works backwards, it is still guaranteed that each key bit affects each data bit over six rounds). The second reason is that it is simple, and can be used in a modern processor with a minimum number of pipeline stalls. The third reason is that we analysed it using programs we developed for investigating block ciphers, and we found bounds on the probabilities of the differential and linear characteristics. These bounds show that the choice suits our needs.

2.3 The S-boxes

The S-boxes of Serpent are 4-bit permutations with the following properties:

- each differential characteristic has a probability of at most $1/4$, and a one-bit input difference will never lead to a one-bit output difference;
- each linear characteristic has a probability in the range $1/2 \pm 1/4$, and a linear relation between one single bit in the input and one single bit in the output has a probability in the range $1/2 \pm 1/8$;
- the nonlinear order of the output bits as a function of the input bits is the maximum, namely 3.

The S-boxes were generated by a deterministic pseudorandom process in which we initialised a matrix with the 32 rows of the DES S-boxes and shuffled their values using a key until we found eight S-boxes that fit the above criteria. The details are included in the full length paper.

3 The Key Schedule

As with the description of the cipher, we can describe the key schedule in either standard or bitslice mode. Here we will give the description for the latter case.

Our cipher requires 132 32-bit words of key material. We first pad the user supplied key to 256 bits, if necessary, and write it as eight 32-bit words w_{-8}, $\ldots$, w_{-1}. We then compute a prekey of 132 words $w_0, \ldots, w_{131}$ by the following affine 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 is the fractional part of the golden ratio $(\sqrt{5}+1)/2$ or `0x9e3779b9` in hexadecimal. The underlying polynomial $x^8 + x^7 + x^5 + x^3 + 1$ is primitive, which together with the addition of the round index is chosen to ensure an even distribution of key bits throughout the rounds, while eliminating weak keys and related keys.

The round keys are now calculated from the prekeys using the S-boxes, again in bitslice mode. We use the S-boxes to transform the prekeys w_i into words k_i of round key in the following way:

$$\begin{aligned}
\{k_0, k_1, k_2, k_3\} &:= S_3(w_0, w_1, w_2, w_3) \\
\{k_4, k_5, k_6, k_7\} &:= S_2(w_4, w_5, w_6, w_7) \\
&\ldots \\
\{k_{124}, k_{125}, k_{126}, k_{127}\} &:= S_4(w_{124}, w_{125}, w_{126}, w_{127}) \\
\{k_{128}, k_{129}, k_{130}, k_{131}\} &:= S_3(w_{128}, w_{129}, w_{130}, w_{131})
\end{aligned}$$

We then concatenate the 32-bit values k_j four at a time into the 128-bit subkeys K_i (for i $\in \{0, \ldots, 32\}$).

3.1 Design history

Serpent's design history is given in the full paper; here we give a summary. The first published version of Serpent used the DES S-boxes, as their differential and linear properties are well understood [5]. We then learned how to construct S-boxes with much greater security, as Serpent does not impose the tight constraints on S-box design that DES does.

The hard choice was whether to make Serpent as fast as possible provided it could resist all presently known attacks — in which case it would have sixteen rounds and be about twice as fast as DES — or make it about as fast as DES but with 32 rounds giving a substantial margin of safety against future advances in cryptanalysis. For the reasons discussed in the introduction, we opted for the latter course. We now present the security analysis in more detail.

4 Security

The analysis of our initial published design (with the DES S-boxes) indicated that the number of plaintexts required for the best shortcut attack would be well over 2^{100}. The improved S-boxes enable us to improve this figure to 2^{256}. By a strength of 2^{256}, we mean that a differential or linear attack against any key would take that many texts, assuming that they were available (though they aren't). This figure comes from computing the relevant probabilities over all the keys. There are of course higher probability differentials for fixed keys; for any fixed key, differentials with probability 2^{-120} can be expected in all the AES candidates, as only the required block size of 128 bits affects their probability. However, we expect that these could be found for Serpent only by exhaustive search and would thus not give rise to any practical attack.

In our analysis, we use conservative bounds to enable our claims to resist substantial improvements in existing attacks. For example, our differential and linear analyses use 24-round and 28-round characteristics, shorter by 8 and 4 rounds than the cipher, while the best attacks on DES use characteristics that are shorter by only three rounds. Our estimates of the probabilities of the best characteristics are also very conservative, so our complexity claims are almost certainly much lower than the real values.

The conclusion of our analysis is that there is no indication of any useful shortcut attack. We believe that such an attack would require a major theoretical breakthrough, and that Serpent's safety margin is adequate to cope with foreseeable improvements in current techniques. In any case, it should be noted that regardless of the design of a 128 bit block cipher, it is prudent to change keys well before 2^{64} blocks have been encrypted, in order to avoid various collision attacks (e.g., [3, 13]). This would easily prevent all known kinds of key recovery attack other than keysearch.

We now list the main weaknesses and attacks which we had in mind when designing Serpent.

4.1 Differential Cryptanalysis

We searched for the best characteristics of this cipher. We made a worst case assumption that all the entries in the difference distribution tables have probability 1/4, except that entries with a one-bit input difference and one-bit output difference are impossible. The following results hold independently of the order of the S-boxes used in the cipher, and independently of the choice of the S-boxes, so long as they satisfy these minimal conditions. We searched for the best characteristics with up to seven rounds, and those with the highest probabilities are given in Table 1.

As the probability of a 6-round characteristic is bounded by 2^{-58}, the probability of a 24-round characteristic is bounded by $2^{-4 \cdot 58} = 2^{-232}$. This means that even if an attacker can implement an 8R-attack (which seems unlikely) this will require many more plaintexts than are available. If the linear transform had

Rounds	Differential Probability	Linear Probability $(1/2 \pm p)$	p^{-2}
1	2^{-2}	$1/2 \pm 4/16 = 1/2 \pm 2^{-2}$	2^4
2	2^{-6}	$1/2 \pm 2^2(4/16)^3 = 1/2 \pm 2^{-4}$	2^8
3	2^{-14}	$1/2 \pm 2^7(4/16)^8 = 1/2 \pm 2^{-9}$	2^{18}
4	2^{-26}	$1/2 \pm 2^{13}(4/16)^{14} = 1/2 \pm 2^{-15}$	2^{30}
5	2^{-42}	$1/2 \pm 2^{19}(4/16)^{20} = 1/2 \pm 2^{-21}$	2^{42}
6	2^{-58}	$1/2 \pm 2^{26}(4/16)^{27} = 1/2 \pm 2^{-28}$	2^{56}
7	$< 2^{-70}$	$1/2 \pm 2^{32}(4/16)^{33} = 1/2 \pm 2^{-34}$	$> 2^{68}$

Table 1. Bounds on the Probabilities of Differential and Linear Characteristics

used only rotates, then every characteristic could have 32 equiprobable rotated variants. This is why we also used shift instructions.

We have bounded the probabilities of characteristics. However, it is both much more important and much more difficult to bound the probabilities of differentials. To do this we firstly reduced the probabilities of the characteristics; secondly ensured that there are few characteristics with the highest possible probability, and that they cannot be rotated and still remain valid; and thirdly arranged for characteristics to affect many different bits, so that they cannot easily be unified into differentials.

We conjecture that the probability of the best 28-round differential is not higher than 2^{-120}, and (as noted above) that such a differential would be very hard to find.

4.2 Linear Cryptanalysis

In linear cryptanalysis, it is possible to find one-bit to one-bit relations of the S-boxes. The probability of these relations is bounded by $1/2 \pm 1/8$. Thus, a 28-round linear characteristic with only one active S-box in each round would have probability $1/2 \pm 2^{27}(1/8)^{28} = 1/2 \pm 2^{-57}$, even if the linear transform is eliminated, and that an attack based on such relations would require about 2^{114} known plaintexts. However, the linear transform assures that in the round following a round with only one active S-box, at least two are active.

More general attacks can use linear characteristics with more than one active S-box in some of the rounds. In this case the probabilities of the S-boxes are in the range $1/2 \pm 1/4$. As with differential cryptanalysis, we can bound the probability of characteristics. We searched for the best linear characteristic of Serpent under the assumptions that a probability of any entry is not further from 1/2 than 1/4 and that the probability of a characteristic which relates one bit to one bit is not further from 1/2 than 1/8. The best linear characteristics with up to seven rounds are given in Table 1. We can see that the probability of a 6-round characteristic is in the range $1/2 \pm 2^{-28}$, so the probability of a 24-round characteristic is in the range $1/2 \pm 2^{-109}$. The number of plaintexts needed for such an attack is at least 2^{218}, which is much higher than the number of available texts.

Based on these figures we believe that the probability of the best 28-round linear differential (or linear hull) is in the range $1/2 \pm 2^{-120}$, so an attack would need at least 2^{240} texts. This is a very conservative estimate; we believe the real figure is well over 2^{256}. In any case, linear attacks are infeasible.

4.3 Other Attacks

Related Keys: As the key schedule uses rotations and S-boxes, with different S-boxes in different rounds, and as we XOR the round number into the prekey, it is highly unlikely that keys can be found that allow related key attacks [4, 11, 12]. Serpent has none of the simpler vulnerabilities that can result from exploitable symmetries in the key schedule: there are no weak keys, semi-weak keys, equivalent keys, or complementation properties.

Higher Order Differential Cryptanalysis: Attacks based on dth order differentials [14, 16] are not applicable to Serpent, as the output bits attain the maximum possible nonlinear order after five rounds.

Truncated Differential Cryptanalysis: Because of the strong diffusion over many rounds, we believe that truncated differential attacks [14] are not applicable to Serpent.

Other shortcut attacks: Davies' attack [9] and the improved version of [6] are not applicable, since the S-boxes are invertible and data bits are not duplicated. As far as we know, neither statistical cryptanalysis [20] nor partitioning cryptanalysis [10] provides a less complex attack than differential or linear cryptanalysis. Non-linear cryptanalysis has so far only managed to improve the linear attack by small factors [19], and Serpent has a large margin of safety against such attacks.

Timing Attacks: The number of instructions used to encrypt or decrypt does not depend on either the data or the key, and even cache misses cannot help the attacker. It follows that timing attacks [15] are not applicable.

5 Performance in Various Environments

On a 133MHz Pentium/MMX processor, our bitslice implementation of Serpent runs about as fast as DES: it encrypts 9,791,000 bits per second, or about 1738 clock cycles per block, while the best optimized DES implementation (Eric Young's Libdes) encrypts 9,824,864 bits per second. We estimate that on the NIST platform of a 200 MHz Pentium, it will run at about 14.7 Mbit/s.

Our bitslice Java implementation performs 10,000 encryptions in 3.3 seconds on a 133 MHz Pentium MMX which translates to 388 kbit/s, and we expect 583 kbit/s on a 200 MHz machine.

In many applications, we can use parallelism to exceed these raw figures. If we wish to simultaneously encrypt a string and compute a MAC on it, using two different keys, we can do this easily on an Intel/MMX processor as this can perform SIMD processing of two 32-bit computations at once. With a 64-bit processor such as DEC Alpha, it is only slightly more complicated. It is also possible to hash multiple streams of data simultaneously.

For very high speed applications, one could use dedicated hardware. A fully pipelined chip might use 67,000 gates: about 33,000 gates for each of encryption and decryption, plus control logic and buffers. However, as Serpent consists of four repetitions of the same structure of 8 S-boxes, it would often be adequate to pipeline only eight rounds at a time, leading to a gate count of approximately 18,000. A compact hardware implementation would process one round at a time at a cost of about 4,500 gates; a similar investment of silicon could give a CPU an extra multiplexing instruction that would make Serpent code much faster and smaller, and have other uses too. (For details, see the full paper.)

On 8-bit processors, the instruction count is larger but here one usually optimises for code size rather than speed. Typical 8-bit crypto applications, such as smartcards, toll tags and prepayment utility meters, encrypt or decrypt a few blocks during a transaction lasting a second or more; but the cost of the processors, and thus memory size, is critical [1]. The most compact implementation of Serpent appears to be one that uses a bitslicing main loop (to avoid the initial and final permutations) but table lookups for each S-box (to avoid the code size of the Boolean expression of the S-box). An Ada implementation that uses this strategy indicates a code size of just under 1K and a computational cost of 34,000 clock cycles. Thus on a 3.5 MHz 6805, we expect a throughput of about 100 encryptions per second or 12.8 kbit/s. A full bitslice implementation would occupy more memory (2K) but should deliver 40.7 kbit/s. These figures are more than adequate for the applications.

6 Conclusion

We have presented a cipher which satisfies the AES requirements. It provides users with the highest practical level of assurance that no shortcut attack will be found. To achieve this, we limited ourselves to well understood mechanisms, so that we could rely on the wide experience of block cipher cryptanalysis. We also used twice as many rounds as are necessary to block all currently known shortcut attacks. We believe that this is prudent practice.

Despite these exacting design constraints, Serpent is as fast as DES. Software versions can be optimised for speed or code size, and hardware implementations can start at 4,500 gates. In short, Serpent also offers an unprecedented level of implementation flexibility.

Acknowledgements: We are extremely grateful to Frank Stajano for coding reference implementations in C and Python and for running many of the tests independently; to Intel Corporation for financial support; to Markus Kuhn for the Ada implementation; to the Cryptix group for the initial Java implementation; to Craig Clapp for a discussion on the key schedule; and to Gideon Yuval for suggesting the name of the cipher (see Amos 5.19).

References

1. RJ Anderson, SJ Bezuidenhoudt, "On the Reliability of Electronic Payment Systems", in *IEEE Transactions on Software Engineering* v 22 no 5 (May 1996) pp 294–301
2. E Biham, "A Fast New DES Implementation in Software", in *Fast Software Encryption — 4th International Workshop, FSE '97*, Springer LNCS v 1267 pp 260–271
3. E Biham, *How to Forge DES-Encrypted Messages in* 2^{28} *Steps*, Technical Report CS884, Technion, August 1996
4. E Biham, "New Types of Cryptanalytic Attacks Using Related Keys", in *Journal of Cryptology* v 7 (1994) no 4 pp 229–246
5. E Biham, RJ Anderson, LR Knudsen, "Serpent: A New Block Cipher Proposal", in *Fast Software Encryption — FSE 98*, Springer LNCS vol 1372 pp 222–238
6. E Biham, A Biryukov, "An Improvement of Davies' Attack on DES", in *Journal of Cryptology* v 10 no 3 (Summer 97) pp 195–205
7. E Biham, A Shamir, *'Differential Cryptanalysis of the Data Encryption Standard'* (Springer 1993)
8. CSK Clapp, "Joint Hardware / Software Design of a Fast Stream Cipher", in *Fast Software Encryption — FSE 98*, Springer LNCS vol 1372 pp 75–92
9. D Davies, S Murphy, "Pairs and Triplets of DES S Boxes", in *Journal of Cryptology* v 8 no 1 (1995) pp 1–25
10. C Harpes, JL Massey, "Partitioning Cryptanalysis", in *Fast Software Encryption — 4th International Workshop, FSE '97*, Springer LNCS v 1267 pp 13–27
11. J Kelsey, B Schneier, D Wagner, "Key-Schedule Cryptanalysis of IDEA, GDES, GOST, SAFER and Triple-DES", in *Advances in Cryptology — Crypto 96*, Springer LNCS v 1109 pp 237–251
12. LR Knudsen, "Cryptanalysis of LOKI91", in *Advances in Cryptology — Auscrypt '92* Springer LNCS v 718 pp 196–208
13. L.R. Knudsen, *Block Ciphers – Analysis, Design and Applications*, Ph.D. Thesis, Århus University, Denmark, 1994.
14. LR Knudsen, "Truncated and Higher-Order Differentials", in *Fast Software Encryption — 2nd International Workshop, FSE '94*, Springer LNCS v 1008 pp 196–211
15. PC Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", in *Advances in Cryptology — Crypto 96*, Springer LNCS v 1109 pp 104–113
16. XJ Lai, *'Higher Order Derivative and Differential Cryptanalysis'*, private communication, September 30, 1993.
17. M Matsui, "Linear Cryptanalysis Method for DES Cipher", in *Advances in Cryptology — Eurocrypt 93*, Springer LNCS v 765 pp 386–397
18. `http://www.cl.cam.ac.uk/users/rja14/#Cryptanalysis`
19. T Shimoyama, "Quadratic relation of S-box and Application to the Linear Cryptanalysis of DES", presented at the rump session of Fast Software Encryption 98.
20. S Vaudenay, "An Experiment on DES Statistical Cryptanalysis", in *3rd ACM Conference on Computer and Communications Security, March 14-16, 96, New Delhi, India; proceedings published by ACM* pp 139–147