Serpent: A New Block Cipher Proposal

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

Abstract. We propose a new block cipher as a candidate for the Advanced Encryption Standard. Its design is highly conservative, yet still allows a very efficient implementation. It uses the well-understood DES S-boxes in a new structure that simultaneously allows a more rapid avalanche, a more efficient bitslice implementation, and an easy analysis that enables us to demonstrate its security against all known types of attack. With a 128-bit block size and a 256-bit key, it is faster than DES on the market leading Intel Pentium/MMX platforms (and at least as fast on many others); yet we believe it to be at least as secure as three-key triple-DES.

1 Introduction

For many applications, the Data Encryption Standard algorithm is nearing the end of its useful life. Its 56-bit key is too small, as shown by a recent distributed key search exercise [24]. Although triple-DES can solve the key length problem, the DES algorithm was also designed primarily for hardware encryption, yet the great majority of applications that use it today implement it in software, where it is relatively inefficient.

For these reasons, the US National Institute of Standards and Technology has issued a call for a successor algorithm, to be called the *Advanced Encryption Standard* or *AES*. The essential requirement is that AES should be both faster and more secure than triple DES; specifically, it should have a 128 bit block length and a key length of 256 bits (though keys of 128 and 192 bits must also be supported).

In this paper, we present a candidate for AES. Our design philosophy has been highly conservative; we did not feel it appropriate to use novel and untested ideas in a cipher which, if accepted after a short review period, will be used to protect enormous volumes of financial transactions, health records and government information over a period of decades.

We therefore decided to use the S-boxes from DES, which have been subjected to intense study over many years and whose properties are thus well understood, in a new structure which is optimized for efficient implementation on modern processors while simultaneously allowing us to apply the extensive analysis already done on DES. As a result, we can show that our design resists all known attacks, including those based on both differential [9] and linear [23] techniques.

The cipher is inspired by recent ideas for bitslice implementation of ciphers [11]. However, unlike (say) the bitslice implementation of DES, which encrypts 64 different blocks in parallel in order to gain extra speed, Serpent is designed to allow a single block to be implemented efficiently by bitslicing. This allows the usual modes of operations to be used, so there is no need to change the environment to gain the extra speed.

We propose several variants of the cipher, which we have tentatively named Serpent. The primary variant is a 32-round cipher which we believe to be as secure as three-key triple-DES, but which is only slightly slower than DES when implemented in C on a Pentium (in some assembly languages it may be faster than DES). It is an SP-network operating on four 32-bit words, thus giving a block size of 128 bits.

The additional variants can deal with different increased block sizes. The block size can be doubled to 256 bits either by increasing the word size from 32 to 64 bits (which will be well suited to the new generation of 64-bit processors), or by using the round function in a Feistel construction. These two variants can be combined to give a cipher with 512-bit blocks. By reducing the block size to 16 bits, or by reducing it to 8 bits and using Feistel construction, we get a 64 bit block cipher. In some applications, we may wish to compute a MAC and perform CBC encryption simultaneously with different keys; we can support this by using a word size of half the machine word size.

At this stage all the variants are still tentative. We are still working on improvements and analysis. As usual in this field, we encourage interested parties to analyze the cipher, inform us of any weakness, and pass on any remarks or suggestions for improvements.

All values used in the cipher are represented as bit-streams. The indices of the bits are counted from 0 to bit 31 in one 32-bit word, bit 127 in 128-bit blocks, bit 255 in 256-bit keys, etc. They are printed as hexadecimal numbers of 8 digits, 32 digits, or 64 digits, respectively. For internal computation, all values are represented in little-endian, where the first word (word 0) is the least significant word of the numbers, and the last word is the most significant, and where bit 0 is the least significant bit of word 0. [4]

2 The Cipher

The main variant of our cipher encrypts a 128-bit plaintext P to a 128-bit ciphertext C in r rounds under the control of $r + 1$ 128-bit subkeys $\hat{K}_0, \ldots, \hat{K}_r$. (We have chosen $r = 32$ as the default, and will henceforth replace r by 32 in order to make the description of the cipher more readable.)

The user key can be of any length between 64 and 256 bits.[5] Short keys with

[4] Thus, bit 0 of an intermediate value in the standard implementation is not necessarily the same as bit 0 of the corresponding value in the bitslice implementation.

[5] We define key lengths to be multiples of 32 bits for ease of implementation, and to eliminate possible confusion — such as whether "1F" is a 5-bit key, a 6-bit key or an 8-bit key

less than 256 bits are mapped to full-length keys of 256 bits by appending one "1" bit and as many "0" bits as required to make up 256 bits. This mapping has the advantage that every short key can be specified by a full-length key, but no two short keys are equivalent. We view short keys as unique short representations of full-length keys. Thus every key has exactly one full-length representation and one short representation (except the full-length zero key which does not have any short representation).

The cipher itself is an SP-network and 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 transformation. In the last round, this linear transformation is replaced by an additional key mixing operation;
- a final permutation FP.

The initial and final permutations do not have any cryptographic significance. They are used to simplify an optimized implementation of the cipher, which is described in the next section, and to improve its computational efficiency. Both these two permutations and the linear transformation are specified in the appendix; their design principles will be made clear in the next section.

We use the following notation. The initial permutation IP is applied to the plaintext P giving $\hat{B}_0$, which is the input to the first round. The rounds are numbered from 0 to 31, where the first round is round 0 and the last is round 31. The output of the first round (round 0) is $\hat{B}_1$, the output of the second round (round 1) is $\hat{B}_2$, the output of round i is $\hat{B}_{i+1}$, and so on, until the output of the last round (in which the linear transformation is replaced by an additional key mixing) is denoted by $\hat{B}_{32}$. The final permutation FP is now applied to give the ciphertext C.

Each round function R_i ($i \in \{0, \ldots, 31\}$ uses only a single replicated S-box. For example, R_0 uses S_0, 32 copies of which are applied in parallel. Thus 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 as output 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 transformation, giving $\hat{B}_1$. Similarly, R_1 uses 32 copies of S_1 in parallel on $\hat{B}_1 \oplus \hat{K}_1$ and transforms their output using the linear transformation, giving $\hat{B}_2$.

In the last round R_{31}, we apply S_{31} on $\hat{B}_{31} \oplus \hat{K}_{31}$, and XOR the result with $\hat{K}_{32}$ rather than applying the linear transformation. The result $\hat{B}_{32}$ is then permuted by FP, giving the ciphertext.

Thus the 32 rounds use 32 different S-boxes each of which maps four input bits to four output bits. Each S-box is used only in one round, in which it is used 32 times in parallel. The 32 S-boxes are chosen as the 32 separate lines of the eight DES S-boxes; thus our S_0 (used in round 0) is the first line of the DES $S1$, our S_1 (used in round 1) is the second line of the DES $S1$, our S_4 (used in round 4) is the first line of the DES $S2$ and so on.

As with DES, the final permutation is the inverse of the initial permutation. Thus the cipher may be formally described by the following equations:

$$\begin{aligned} \hat{B}_0 &= IP(P) \\ \hat{B}_{i+1} &= R_i(\hat{B}_i) \\ C &= FP(\hat{B}_r) \end{aligned}$$

where

$$\begin{aligned} R_i(X) &= L(\hat{\mathcal{S}}_i(X \oplus \hat{K}_i)) \qquad i = 0, \ldots, r-2 \\ R_i(X) &= \hat{\mathcal{S}}_i(X \oplus \hat{K}_i) \oplus \hat{K}_r \quad i = r-1 \end{aligned}$$

where $\hat{\mathcal{S}}_i$ is the application of the S-box S_i 32 times in parallel, and L is the linear transformation.

Although each round of the proposed cipher might seem weaker than a round of DES, we shall see below that their combination overcomes the weakness. The greater speed of each round, and the increased number of rounds, make the cipher both almost as fast as DES and much more secure.

2.1 Decryption

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

3 An Efficient Implementation

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. For a full description of a bitslice implementation of DES, see [11]; 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 — in effect, 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. The bitslice approach was used in the recent successful DES key search, in which spare CPU cycles from thousands of machines were volunteered to solve a challenge cryptogram. However the problem with using bitslice techniques for DES encryption (as opposed to keysearch) is that one has to process many blocks in parallel, and although special modes of operation can be designed for this, they are not the modes in common use.

Our cipher has therefore been designed so that all operations can be executed using 32-fold parallelism during the encryption or decryption of a single block. Indeed the bitslice description of the algorithm is much simpler than its conventional description. No initial and final permutations are required, since the initial and final permutations described in the standard implementation above are just

the conversions of the data from and to the bitslice representation. We will now present an equivalent description of the algorithm for bitslice implementation.

The cipher consists simply of 32 rounds. The plaintext becomes the first intermediate data $B_0 = P$, after which the 32 rounds are applied, where 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 with $\mathcal{S}_i(B_i \oplus K_i)$
3. Linear Transformation: The 32 bits in each of the output words are linearly mixed, by

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

where $<<<$ denotes rotation, and $<<$ denotes shift. In the last round, this linear transformation is replaced by an additional key mixing: $B_r = S_{r-1}(B_{r-1} \oplus K_{r-1}) \oplus K_r$. Note that at each stage $IP(B_i) = \hat{B}_i$, and $IP(K_i) = \hat{K}_i$.

The first reason for the choice of linear transformation is to maximize the avalanche effect. The DES S-boxes have the property that a single input bit change will cause two output bits to change; as the difference sets of {0, 1, 3, 5, 7, 13, 22} modulo 32 have no common member (except one), it follows 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. (Some historical information on the design of the linear transformation is given in the appendix.)

The second reason is that it is simple, and can be used in a pipelined processor with a minimum number of pipeline stalls. The third reason is that it was analyzed by programs we developed for investigating block ciphers, and we found bounds on the probabilities of the differential and linear characteristics. These bounds show that this choice suits our needs, although we would like to improve on it.

So we are still considering other, simpler, choices for the linear transformation. One possibility is to adapt an LFSR-like transform of the form $X_i = X_i \oplus \mathrm{ROL}(X_{i-1}, r_i)$ for $i = 1, \ldots, 6$, where the four data words are $X_0, \ldots, X_3$, the indices of X are taken modulo 4, and the r_i's are fixed. The problems with such a scheme are that it is hard to pipeline, and that every characteristic can be rotated in all its words and still remain with the same probability. We are still working on other possible linear transformations.

4 The Key Schedule

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

Our cipher requires 132 32-bit words of key material. We first expand the user supplied 256-bit key K to 33 128-bit subkeys $K_0, \ldots, K_{32}$, in the following way. We write the key K as eight 32-bit words $w_{-8}, \ldots, w_{-1}$ and expand these to an intermediate key (which we call *prekey*) $w_0, \ldots, w_{131}$ 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, and to eliminate weak keys and related keys.

The round keys are now calculated from the prekeys using the S-boxes, again in bitslice mode. The S-box inputs and outputs are taken at a distance of 33 words apart, in order to minimize the key leakage in the event of a differential attack on the last few rounds of the cipher. We use the S-boxes to transform the prekeys w_i into words k_i of round key by dividing the vector of prekeys into four sections and transforming the i'th words of each of the four sections using $S_{(r+3-i) \bmod r}$. This can be seen simply for the default case $r = 32$ as follows:

$$\begin{aligned}
\{k_0, k_{33}, k_{66}, k_{99}\} &= S_3(w_0, w_{33}, w_{66}, w_{99}) \\
\{k_1, k_{34}, k_{67}, k_{100}\} &= S_2(w_1, w_{34}, w_{67}, w_{100}) \\
&\ldots \\
\{k_{31}, k_{64}, k_{97}, k_{130}\} &= S_4(w_{31}, w_{64}, w_{97}, w_{130}) \\
\{k_{32}, k_{65}, k_{98}, k_{131}\} &= S_3(w_{32}, w_{65}, w_{98}, w_{131})
\end{aligned}$$

We then renumber the 32-bit values k_j as 128-bit subkeys K_i (for i $\in \{0, \ldots,$ r$\}$) as follows:

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

Where we are implementing the algorithm in the form initially described in section 2 above rather than using bitslice operations, we now apply IP to the round key in order to place the key bits in the correct column, i.e., $\hat{K}_i = IP(K_i)$.

5 Security

As mentioned above, the differential and linear properties of the DES S-boxes are well understood. Our preliminary estimates indicate that the number of known/chosen plaintexts required for either type of attack would exceed 2^{128} (they are certainly well over 2^{100} and we are working on more accurate bounds). There is thus no indication of any useful shortcut attack; we believe that such an attack would require a new theoretical breakthrough. In any case, it should be noted that regardless of the design of a 128 bit block cipher, it is normally prudent to change keys well before 2^{64} blocks have been encrypted, in order to avoid the collision attack of section 5.2 below. This would easily prevent all known shortcut attacks.

We designed the cipher with a view to reducing or avoiding vulnerabilities arising from the following possible weaknesses and attacks. In our analysis, we use conservative bounds to enable our claims to resist reasonable improvements in the studied attacks. For example, we analyze the cipher using 24-round and 28-round characteristics, shorter by 8 and 4 rounds than the cipher, while the best attack on DES uses characteristics that are shorter by only three rounds. Our estimates of the probabilities of the best characteristics are also very conservative; in practice they should be considerably lower. Therefore, our complexity claims are probably much lower than the real values, and Serpent is expected to be much more secure than we actually claim.

5.1 Dictionary Attacks

As the block size is 128 bits, a dictionary attack will require 2^{128} different plaintexts to allow the attacker to encrypt or decrypt arbitrary messages under an unknown key. This attack applies to any deterministic block cipher with 128-bit blocks regardless of its design.

5.2 Modes of Operation

After encrypting about 2^{64} plaintext blocks in the CBC or CFB mode, one can expect to find two equal ciphertext blocks. This enables an attacker to compute the exclusive-or of the two corresponding plaintext blocks [21]. With progressively more plaintext blocks, plaintext relationships can be discovered with progressively higher probability. This attack applies to any deterministic block cipher with 128-bit blocks regardless of its design.

5.3 Key-Collision Attacks

For key size k, key collision attacks can be used to forge messages with complexity only $2^{k/2}$ [7]. Thus, the complexity of forging messages under 128-bit keys is only 2^{64}, under 192-bit keys it is 2^{96}, and under 256-bit keys it is 2^{128}. This attack applies to any deterministic block cipher, and depends only on its key size, regardless of its design.

5.4 Differential Cryptanalysis

An important fact about Serpent is that any characteristic must have at least one active S-box in each round. At least two active S-boxes are required on average, due to the property that a difference in only one bit in the input causes a difference of at least two bits in the output of each S-box. Therefore, if only one bit differs in the input of some round, then at least two differ in the output, and these two bits affect two distinct S-boxes in the following round, whose output differences affect at least four S-boxes in the following round.

We searched for the best characteristics of this cipher. For this, we made a worst case assumption that all the entries in the difference distribution tables have probability 1/2, except the few entries which have only one bit input difference and one bit output difference, which are assumed impossible (probability zero). These bounds are satisfied by all the S-boxes, except for one entry of S_{30}, where the maximal value is 10/16: the highest probabilities in the various S-boxes are 6/16 and 8/16, except in S_{21} in which it is 4/16 and in S_{30} in which it is 10/16. We assume later that round 30 is not approximated by the characteristic anyway. Thus 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 the ones with the highest probabilities are given in Table 1.

Rounds	Differential Probability	Linear Probability $(1/2 \pm p)$	p^{-2}
1	2^{-1}	$1/2 \pm 6/16 = 1/2 \pm 2^{-1.4}$	$2^{2.8}$
2	2^{-3}	$1/2 \pm (6/16)^3 = 1/2 \pm 2^{-2.2}$	$2^{4.4}$
3	2^{-7}	$1/2 \pm (6/16)^8 = 1/2 \pm 2^{-4.3}$	$2^{8.6}$
4	2^{-13}	$1/2 \pm (6/16)^{14} = 1/2 \pm 2^{-6.8}$	$2^{13.6}$
5	2^{-21}	$1/2 \pm (6/16)^{20} = 1/2 \pm 2^{-9.3}$	$2^{18.6}$
6	2^{-29}	$1/2 \pm (6/16)^{27} = 1/2 \pm 2^{-12.2}$	$2^{24.4}$
7	$< 2^{-35}$	$1/2 \pm (6/16)^{33} = 1/2 \pm 2^{-13.8}$	$> 2^{27.6}$

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

We can see that the probability of a 6-round characteristic is bounded by 2^{-29}. Thus, the probability of a 24-round characteristic is bounded by $2^{-4 \cdot 29} =$

2^{-116}. In practice, the probability of the best 24-round characteristic is expected to be much lower than this. Thus, even if an attacker can implement an 8R-attack, still the attack requires more than 2^{117} chosen plaintexts (and again, this is a very conservative estimate). If the attacker can implement only a 4R-attack, using a 28-round characteristic, the probability of the characteristic is bounded by $2^{-4 \cdot 35} = 2^{-140}$, and the attack requires more plaintexts than are available. A 3R-attack would require even more plaintexts.

Notice that if the linear transformation had used only rotates, then every characteristic could have 32 equiprobable rotated variants, with all the data words rotated by the same number of bits. This is the reason that we also use shift instructions, which avoid most of these rotated characteristics.

We have bounded the probabilities of characteristics. However, it is both much more important and much more difficult to bound the probabilities of differentials. In order to reduce the probabilities of differentials we have (1) reduced the probabilities of the characteristics, (2) ensured that there are few characteristics with the highest possible probability, and that they cannot be rotated and still remain valid, (3) 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 that such a differential if it exists would be very hard to find. (Note that for any fixed key there expected to be differentials with probability 2^{-120}, but averaging over all possible keys reduces this average probability to about 2^{-128}.)

5.5 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 are $1/2 \pm 2/16$. Thus, a 28-round linear characteristic with only one active S-box in each round would have probability $1/2 \pm 2^{27}(2/16)^{28} = 1/2 \pm 2^{-57}$, and that an attack based on such relations would require about 2^{114} known plaintexts, if it were possible at all (as the linear transformation 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 bounded by $1/2 \pm 6/16$. As with differential cryptanalysis, we can bound the probability of characteristics. We searched for the best linear characteristic of this cipher under the assumptions that a probability of any entry is not further from $1/2$ than $6/16$ and that the probability of a characteristic which relates one bit to one bit is not further from $1/2$ than $2/16$. Note that due to the relation between linear and differential characteristics, the searches are very similar; we actually modified the search program used in the differential case to search for the best linear characteristics with up to seven rounds, and those with the highest probabilities are given in Table 1.

We can see that the probability of a 6-round characteristic is bounded by $1/2 \pm 2^{-12.2}$, from which we can conclude that the probability of a 24-round

characteristic is bounded by $1/2 \pm 2^{-45.8}$. The probability of a 28-round characteristic is bounded by $1/2 \pm 2^{-52.2}$, and an attack based on it would require at least 2^{104} known plaintexts. Again, we wish to emphasize that all these figures are conservative lower bounds, and that the actual complexities of attacks are expected to be substantially higher.

Based on these figures we conjecture that the probability of the best 28-round linear differential is bounded by $1/2 \pm 2^{-50}$, so an attack would need at least 2^{100} blocks. Again, this is a very conservative estimate; we believe the real figure is over 2^{128} and that linear attacks are thus infeasible. We are working on more accurate figures; meantime the normal prudent practice of changing keys well before 2^{64} blocks have been encrypted will prevent linear attacks.

5.6 Higher Order Differential Cryptanalysis

It is well known that a dth order differential of a function of nonlinear order d is constant, and this can be exploited in higher order differential attacks [6, 20, 22]. The DES S-boxes all have nonlinear order 5 [21]. From this one would expect that the nonlinear order of the output bits after r rounds is about 3^r, with the maximum value of 127 reachable after five rounds. Therefore we are convinced that higher order differential attacks are not applicable to Serpent.

5.7 Truncated Differential Cryptanalysis

For some ciphers it is possible and advantageous to predict only the values of parts of the differences after each round. This notion, of truncated differential attacks, was introduced by Knudsen in [20]. However, the method seems best applicable to ciphers where all operations are done on larger blocks of bits. Because of the strong diffusion over many rounds, we believe that truncated differential attacks are not applicable to Serpent.

5.8 Related Keys

As the key schedule uses rotations and S-boxes, it is highly unlikely that keys can be found that allow related key attacks [10, 18, 19]. Moreover, different rounds of Serpent use different S-boxes, so even if related keys were found, related-key attacks would not be applicable.

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.

5.9 Other Attacks

Davies' attack [15, 16] and the Improved Davies attack [8] are not applicable, since the S-boxes are invertible, and no duplications of data bits are applied.

As far as we know, neither statistical cryptanalysis [25] nor partitioning cryptanalysis [17] provides a less complex attack than differential or linear cryptanalysis.

5.10 Fault Analysis

We have not been concerned in this design to build in any particular protection against attacks based on induced faults [5, 12, 13]. If an attacker can progressively remove the machine instructions by which this cipher is implemented, or progressively destroy selected gates, or progressively modify the bits of the key register, then he can clearly extract the key. We tend to the view that an attacker with the ability to inspect or modify the implementation detail will have many attacks based not just on compromising keys but on subverting protocols, extracting plaintext directly and so on [4]. The mechanisms required to protect against such attacks are largely independent of the design of any block cipher used [1], and are thus beyond the scope of this work.

6 Performance

We implemented this cipher on a 133MHz Pentium/MMX processor. A 32-round bitslice (unoptimized) implementation gave speeds as fast as DES: it encrypted 9,791,000 bits per second, while the best optimized DES implementation (Eric Young's Libdes) encrypts 9,824,864 bits per second on the same machine and with the same measuring program. With the most obvious optmization — keeping the subkeys in a fixed array in memory rather than passing them as parameters — the speed increases to 10,281,124 bits per second. This implementation is included with the submission and is available online from the authors' web pages.

The performance of the cipher on other processors in bitslice mode should be similar to the standard implementation of DES, and when coded in assembly language Serpent can be faster than DES. It takes about 1830–1940 instructions to encrypt 128 bits versus typically 685 instructions to encrypt 64 bits in DES. The reason our cipher is not 50% slower is that it has been designed to make good use of pipelining, and it does not need so many memory accesses for table lookups.

The instruction count is based on the observation that a gate circuit of the 4x4 (DES-based) S-boxes requires an average of 15.75 gates on the Pentium, about 14.5 on MMX (using only MMX instructions), and even less on the DEC Alpha (the numbers vary due to the different sets of instructions, which are detailed in the appendix). MMX has the additional advantage that it can operate on 64-bit words, or alternatively on two 32-bit words at once (so two encryptions can be done in parallel using the same or different keys, thus enabling simultaneous CBC encryption and MAC computation). It is also implemented with greater parallelism on some recent chips, such as the Pentium II. On the other hand, it does not have rotate operations, so rotates require four instructions (copy, shift left, shift right, and OR).

It is also worth remarking that if this cipher is adopted as the Advanced Encryption Standard, and chip makers wish to support high speed implementation, then it may not be necessary to add a hardware encryption circuit to the CPU. It

would be sufficient to add what we call the 'BITSLICE instruction': this works as a generalised multiplexer in that it executes an arbitrary boolean function on four registers under the control of a truth table encoded in a (64-bit) fifth register. We estimate that the cost of implementing this on an n-bit processor will be only about $100n$ gates, and it would have many uses other than cryptography (image processing is a particular candidate). If supported, one BITSLICE instruction would replace many of the instructions in each round; Serpent would become much faster, and require much smaller code.

It is also worth noting that hardware implementations of the cipher can iteratively apply one round at a time, although the S-boxes in each round are different. The trick is similar to the BITSLICE instruction: the designers of the hardware can design the round function to get a description of the S-boxes as a parameter in some register, and compute the S-boxes according to this description. This trick crucially reduces the number of gates required for the hardware implementation of the cipher. An estimate of the gate count will be provided in the final AES submission.

As it is expected that the selected AES cipher will have to coexist with DES for several years, the choice of S-boxes of Serpent has another advantage: it can share its S-boxes with DES, and thus save a few hundred bytes of memory. This might be important for some low-cost 8-bit processors, and for special purpose enciphering hardware.

7 Other Variants

As we remarked above, there are two ways in which the block size can be changed:

1. we can increase the word length (in the bitslice implementation) from 32 to 64 bits, or decrease it to 16 or even 8 bits;
2. we can double the block size by using the round-function as the F-function in a Feistel construction.

This makes available a family of block ciphers for a wide range of applications. For example, if we wish to generate large quantities of keystream on a 64-bit processor, we might use a Feistel construction with a 64-bit word length and output feedback mode, generating 512 bits with each invocation of the cipher. At the other extreme, where a smartcard application needs a 64-bit block cipher for reasons of backward compatibility in an established protocol (e.g. the VISA electronic purse system [2] and prepayment electricity meters [3]), we could use an 8-bit word size in a Feistel construction. A requirement for a 64-bit block cipher in a mobile application based on 80186 family microcontrollers might be better met using a 16-bit word size and the standard SP construction rather than an 8-bit word size and the Feistel one.

It is also worth noting that in many applications we wish to simultaneously encrypt a string and compute a MAC on it, using two different keys. With the market leading Intel/MMX architecture this can be accomplished without

difficulty using the standard variant of Serpent; as noted above, the MMX processor can perform SIMD processing of two 32-bit computations. With a 64-bit processor such as DEC Alpha, it is only slightly more complicated; the S-box computations for both 32-bit computations can be done in parallel and we simply have to implement two different instances of the linear transformation.

These variants require slight modifications of the cipher, principally in the rotation constants. We believe that they are secure (or can easily be made so). Specific proposals will be included with the final submission.

8 Conclusion

We have presented a cipher which we have engineered to satisfy the AES requirements. It is faster than DES, and with 192 or 256 bit keys it is believed to be more secure than three-key triple DES. Its security is partially based on the reuse of the thoroughly studied components of DES, and thus can draw on the wide literature of block cipher cryptanalysis. Its performance comes from allowing an efficient bitslice implementation on a range of processors, including the market leading Intel/MMX and compatible chips.

This is still a preliminary design and may change between the time of writing and the final AES submission. Readers are invited to attack the cipher, to test implementations in various environments, and to report any interesting findings to the authors. A patent application has been filed, but it is our intention to grant a worldwide royalty-free license for conforming implementations in the event that this cipher is adopted as the Advanced Encryption Standard.

Finally, up to date information on Serpent, including the latest revision of this paper and a C reference implementation, can be found on the authors' home pages:

`http://www.cs.technion.ac.il/~biham/`
`http://www.cl.cam.ac.uk/~rja14/`
`http://www.ii.uib.no/~larsr/`

Acknowledgments

The second author was supported by Intel Corporation during a visit to Cambridge in September 1997 while much of this work was done; and the name of the cipher was suggested by Gideon Yuval (see Amos 5.19). We are also grateful to Frank Stajano for coding reference implementations in C and Python, and to the Cryptix group [14] for writing a Java implementation.

References

1. DG Abraham, GM Dolan, GP Double, JV Stevens, "Transaction Security System", in *IBM Systems Journal* v 30 no 2 (1991) pp 206–229

2. RJ Anderson, "UEPS - a Second Generation Electronic Wallet" in *Computer Security — ESORICS 92*, Springer LNCS vol 648 pp 411–418
3. 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
4. RJ Anderson, MG Kuhn, "Tamper Resistance — a Cautionary Note", in *The Second USENIX Workshop on Electronic Commerce Proceedings* (Nov 1996) pp 1–11
5. RJ Anderson, MG Kuhn, "Low Cost Attacks on Tamper Resistant Devices", in *Security Protocols — Proceedings of the 5th International Workshop*; Springer LNCS v 1361 pp 125–136
6. E Biham, *'Higher Order Differential Cryptanalysis'*, unpublished paper, 1994
7. E Biham, *How to Forge DES-Encrypted Messages in* 2^{28} *Steps*, Technical Report CS884, Technion, August 1996
8. E Biham, A Biryukov, "An Improvement of Davies' Attack on DES", in *Journal of Cryptology* v 10 no 3 (Summer 97) pp 195–205
9. E Biham, A Shamir, *'Differential Cryptanalysis of the Data Encryption Standard'* (Springer 1993)
10. E Biham, "New Types of Cryptanalytic Attacks Using Related Keys", in *Journal of Cryptology* v 7 (1994) no 4 pp 229–246
11. 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
12. E Biham, A Shamir, "Differential Fault Analysis of Secret Key Cryptosystems", in *Advances in Cryptology — Crypto 97*, Springer LNCS v 1294 pp 513–525
13. D Boneh, RA DeMillo, RJ Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", in *Advances in Cryptology — Eurocrypt 97*, Springer LNCS v 1233 pp 37–51
14. "Cryptix AES Kit", `http://www.t-and-g.fl.net.au/java/cryptix/aes/`
15. DW Davies, *'Investigation of a Potential Weakness in the DES Algorithm'*, private communication (1987)
16. D Davies, S Murphy, "Pairs and Triplets of DES S Boxes", in *Journal of Cryptology* v 8 no 1 (1995) pp 1–25
17. C Harpes, JL Massey, "Partitioning Cryptanalysis", in *Fast Software Encryption — 4th International Workshop, FSE '97*, Springer LNCS v 1267 pp 13–27
18. 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
19. LR Knudsen, "Cryptanalysis of LOKI91", in *Advances in Cryptology — Auscrypt'92* Springer LNCS
20. LR Knudsen, "Truncated and Higher-Order Differentials", in *Fast Software Encryption — 2nd International Workshop, FSE '94*, Springer LNCS v 1008 pp 196–211
21. L.R. Knudsen, *Block Ciphers – Analysis, Design and Applications*, Ph.D. Thesis, Århus University, Denmark, 1994.
22. XJ Lai, *'Higher Order Derivative and Differential Cryptanalysis'*, private communication, September 30, 1993.
23. M Matsui, "Linear Cryptanalysis Method for DES Cipher", in *Advances in Cryptology — Eurocrypt 93*, Springer LNCS v 765 pp 386–397
24. RSA Data Security Inc., `www.rsa.com`

25. 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

A Appendix

A.1 The Initial Permutation IP:

Here, having the value v (say, 32) at position p (say, 1) means that the output bit at position p (1) comes from the input bit at position v (32).

```
 0  32  64  96   1  33  65  97   2  34  66  98   3  35  67  99
 4  36  68 100   5  37  69 101   6  38  70 102   7  39  71 103
 8  40  72 104   9  41  73 105  10  42  74 106  11  43  75 107
12  44  76 108  13  45  77 109  14  46  78 110  15  47  79 111
16  48  80 112  17  49  81 113  18  50  82 114  19  51  83 115
20  52  84 116  21  53  85 117  22  54  86 118  23  55  87 119
24  56  88 120  25  57  89 121  26  58  90 122  27  59  91 123
28  60  92 124  29  61  93 125  30  62  94 126  31  63  95 127
```

A.2 The Final Permutation FP:

```
 0   4   8  12  16  20  24  28  32  36  40  44  48  52  56  60
64  68  72  76  80  84  88  92  96 100 104 108 112 116 120 124
 1   5   9  13  17  21  25  29  33  37  41  45  49  53  57  61
65  69  73  77  81  85  89  93  97 101 105 109 113 117 121 125
 2   6  10  14  18  22  26  30  34  38  42  46  50  54  58  62
66  70  74  78  82  86  90  94  98 102 106 110 114 118 122 126
 3   7  11  15  19  23  27  31  35  39  43  47  51  55  59  63
67  71  75  79  83  87  91  95  99 103 107 111 115 119 123 127
```

A.3 The Linear Transformation:

For each output bit of this transformation, we describe the list of input bits whose parity becomes the output bit. The bits are listed from 0 to 127. (Thus, for example, output bit 1 is the exclusive-or of input bits 72, 114 and 125.) In each row we describe four output bits which together make up the input to a single S-box in the next round. It should be noted that this table is IP(LT(FP(x))) where LT is the linear transformation as specified in section 3.

```
{16 52 56  70  83  94 105} {72 114 125} { 2  9 15  30  76  84 126} {36  90 103}
{20 56 60  74  87  98 109} { 1  76 118} { 2  6 13  19  34  80  88} {40  94 107}
{24 60 64  78  91 102 113} { 5  80 122} { 6 10 17  23  38  84  92} {44  98 111}
{28 64 68  82  95 106 117} { 9  84 126} {10 14 21  27  42  88  96} {48 102 115}
{32 68 72  86  99 110 121} { 2  13  88} {14 18 25  31  46  92 100} {52 106 119}
{36 72 76  90 103 114 125} { 6  17  92} {18 22 29  35  50  96 104} {56 110 123}
{ 1 40 76  80  94 107 118} {10  21  96} {22 26 33  39  54 100 108} {60 114 127}
{ 5 44 80  84  98 111 122} {14  25 100} {26 30 37  43  58 104 112} { 3 118    }
{ 9 48 84  88 102 115 126} {18  29 104} {30 34 41  47  62 108 116} { 7 122    }
{ 2 13 52  88  92 106 119} {22  33 108} {34 38 45  51  66 112 120} {11 126    }
```

```
{ 6 17 56  92  96 110 123} {26  37 112} {38 42 49  55  70 116 124} { 2  15  76}
{10 21 60  96 100 114 127} {30  41 116} { 0 42 46  53  59  74 120} { 6  19  80}
{ 3 14 25 100 104 118    } {34  45 120} { 4 46 50  57  63  78 124} {10  23  84}
{ 7 18 29 104 108 122    } {38  49 124} { 0  8 50  54  61  67  82} {14  27  88}
{11 22 33 108 112 126    } { 0  42  53} { 4 12 54  58  65  71  86} {18  31  92}
{ 2 15 26  37  76 112 116} { 4  46  57} { 8 16 58  62  69  75  90} {22  35  96}
{ 6 19 30  41  80 116 120} { 8  50  61} {12 20 62  66  73  79  94} {26  39 100}
{10 23 34  45  84 120 124} {12  54  65} {16 24 66  70  77  83  98} {30  43 104}
{ 0 14 27  38  49  88 124} {16  58  69} {20 28 70  74  81  87 102} {34  47 108}
{ 0  4 18  31  42  53  92} {20  62  73} {24 32 74  78  85  91 106} {38  51 112}
{ 4  8 22  35  46  57  96} {24  66  77} {28 36 78  82  89  95 110} {42  55 116}
{ 8 12 26  39  50  61 100} {28  70  81} {32 40 82  86  93  99 114} {46  59 120}
{12 16 30  43  54  65 104} {32  74  85} {36 90 103 118           } {50  63 124}
{16 20 34  47  58  69 108} {36  78  89} {40 94 107 122           } { 0  54  67}
{20 24 38  51  62  73 112} {40  82  93} {44 98 111 126           } { 4  58  71}
{24 28 42  55  66  77 116} {44  86  97} { 2 48 102 115           } { 8  62  75}
{28 32 46  59  70  81 120} {48  90 101} { 6 52 106 119           } {12  66  79}
{32 36 50  63  74  85 124} {52  94 105} {10 56 110 123           } {16  70  83}
{ 0 36 40  54  67  78  89} {56  98 109} {14 60 114 127           } {20  74  87}
{ 4 40 44  58  71  82  93} {60 102 113} { 3 18 72 114 118 125    } {24  78  91}
{ 8 44 48  62  75  86  97} {64 106 117} { 1  7 22  76 118 122    } {28  82  95}
{12 48 52  66  79  90 101} {68 110 121} { 5 11 26  80 122 126    } {32  86  99}
```

A.4 The Inverse Linear Transformation:

Here we provide the inverse of the above linear transformation, which is used when implementing a non-bitslice version of decryption.

```
{ 53  55  72} { 1  5  20  90    } {   15 102} { 3 31 90              }
{ 57  59  76} { 5  9  24  94    } {   19 106} { 7 35 94              }
{ 61  63  80} { 9 13  28  98    } {   23 110} {11 39 98              }
{ 65  67  84} {13 17  32 102    } {   27 114} { 1  3 15  20  43 102  }
{ 69  71  88} {17 21  36 106    } { 1 31 118} { 5  7 19  24  47 106  }
{ 73  75  92} {21 25  40 110    } { 5 35 122} { 9 11 23  28  51 110  }
{ 77  79  96} {25 29  44 114    } { 9 39 126} {13 15 27  32  55 114  }
{ 81  83 100} { 1 29  33  48 118} { 2 13  43} { 1 17 19  31  36  59 118}
{ 85  87 104} { 5 33  37  52 122} { 6 17  47} { 5 21 23  35  40  63 122}
{ 89  91 108} { 9 37  41  56 126} {10 21  51} { 9 25 27  39  44  67 126}
{ 93  95 112} { 2 13  41  45  60} {14 25  55} { 2 13 29  31  43  48  71}
{ 97  99 116} { 6 17  45  49  64} {18 29  59} { 6 17 33  35  47  52  75}
{101 103 120} {10 21  49  53  68} {22 33  63} {10 21 37  39  51  56  79}
{105 107 124} {14 25  53  57  72} {26 37  67} {14 25 41  43  55  60  83}
{  0 109 111} {18 29  57  61  76} {30 41  71} {18 29 45  47  59  64  87}
{  4 113 115} {22 33  61  65  80} {34 45  75} {22 33 49  51  63  68  91}
{  8 117 119} {26 37  65  69  84} {38 49  79} {26 37 53  55  67  72  95}
{ 12 121 123} {30 41  69  73  88} {42 53  83} {30 41 57  59  71  76  99}
{ 16 125 127} {34 45  73  77  92} {46 57  87} {34 45 61  63  75  80 103}
{  1   3  20} {38 49  77  81  96} {50 61  91} {38 49 65  67  79  84 107}
{  5   7  24} {42 53  81  85 100} {54 65  95} {42 53 69  71  83  88 111}
{  9  11  28} {46 57  85  89 104} {58 69  99} {46 57 73  75  87  92 115}
```

```
{ 13  15  32} {50 61  89  93 108} {62 73 103} {50 61 77  79  91  96 119}
{ 17  19  36} {54 65  93  97 112} {66 77 107} {54 65 81  83  95 100 123}
{ 21  23  40} {58 69  97 101 116} {70 81 111} {58 69 85  87  99 104 127}
{ 25  27  44} {62 73 101 105 120} {74 85 115} { 3 62 73  89  91 103 108}
{ 29  31  48} {66 77 105 109 124} {78 89 119} { 7 66 77  93  95 107 112}
{ 33  35  52} { 0 70  81 109 113} {82 93 123} {11 70 81  97  99 111 116}
{ 37  39  56} { 4 74  85 113 117} {86 97 127} {15 74 85 101 103 115 120}
{ 41  43  60} { 8 78  89 117 121} {    3  90} {19 78 89 105 107 119 124}
{ 45  47  64} {12 82  93 121 125} {    7  94} { 0 23 82  93 109 111 123}
{ 49  51  68} { 1 16  86  97 125} {   11  98} { 4 27 86  97 113 115 127}
```

A.5 S-Boxes

Here are the S-boxes S_0 through S_{31} (each on a separate line).

```
14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13
15 1 8 14 6 11 3 4 9 7 2 13 12 0 5 10
3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5
0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15
13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9
10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8
13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1
13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7
1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12
7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15
13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9
10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4
3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14
2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9
14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6
4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3
12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13
4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6
1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2
6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12
13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7
1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2
7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8
2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11
```

A.6 Lists of Relevant Instructions on Various Processors

The relevant instructions on the following processors are:

Pentium:	AND, OR, XOR, NOT, rotate
MMX:	AND, OR, XOR, NOT, ANDN, only shifts
Alpha:	AND, OR, XOR, NOT, ANDN, ORN, XORN, only shifts

where the ANDN operation on x and y is $x \wedge (\neg y)$, the ORN operation is $x \vee (\neg y)$, and the XORN operation is $x \oplus (\neg y)$ (or equivalently $\neg(x \oplus y)$).

On MMX a rotate takes four instructions, while on an Alpha it takes three. On Pentium and MMX it might be necessary to copy some of the registers before use, as instructions have only two arguments; but some instructions can refer directly to memory. The Alpha instructions have 3 arguments (src1, src2 and destination), but cannot refer directly to memory.

A.7 Historical Remarks

Here we describe some design history. In our first design, the linear transformations were just bit permutations, which were applied as rotations of the 32-bit words in the bitslice implementation. In order to ensure maximal avalanche, the idea was to choose these rotations in a way that ensured maximal avalanche in the fewest number of rounds. Thus, we chose three rotations at each round: we used (0, 1, 3, 7) for the even rounds and (0, 5, 13, 22) for the odd rounds. The reason for this was that (a) rotating all four words is of course useless (b) a single set of rotations did not suffice for full avalanche (c) these sets of rotations have the property that no difference of pairs in either of them coincides with a difference either in the same set or the other set.

However, we felt that the avalanche was still slow, as each bit affected only one bit in the next round, and thus one active S-box affected only 2–4 out of the 32 S-boxes in the next round. As a result, we had to use 64 rounds, and the cipher was only slightly faster than triple-DES. So we moved to a more complex linear transformation; this improved the avalanche, and analysis showed that we could now reduce the number of rounds to 32. We believe that the final result is a faster and yet more secure cipher.

We also considered "improving" the cipher by replacing the XOR operations by seemingly more complex operations, such as additions. We did not do this due to two major reasons: (1) Our analysis takes advantage of the independence between the bits in the XOR operation, as it allows us to describe the cipher in a standard way, and use the known kinds of security analysis. This analysis would not hold if the XOR operations were replaced; (2) in some other ciphers the replacement of XORs by additions (or other operations) has turned out to weaken the cipher, rather than strengthening it.

A.8 Reference Implementation

An unoptimized reference C implementation is available from the authors' home pages. Note however that the cipher may still be modified in the future in the run-up to the AES final submission deadline, and as it progresses through the AES selection process.

This article was processed using the LaTeX macro package with LLNCS style