Introducing the new LOKI97 Block Cipher

Lawrie Brown[*], Josef Pieprzyk[†]

[*]School of Computer Science, Australian Defence Force Academy, Canberra 2600, Australia
[†]School of IT and Computer Science, CCSR, University of Wollongong, Wollongong, NSW 2522, Australia

June 12, 1998

Abstract

This paper describes LOKI97, a new private key block cipher with 128-bit data and a 256-bit key schedule, which can be initialised by 128, 192, or 256-bit keys. The data computation uses 16 rounds of a balanced Feistel network with a complex function f which incorporates two S-P layers. The 256-bit key schedule uses 48 rounds of an unbalanced Feistel network using the same complex function f to generate the subkeys. The cipher specification is given, followed by some background and design considerations, and a preliminary analysis.

1 Introduction

LOKI97 is a new 16 round Feistel cipher on 128-bit data using a complex non-linear function f, with a 256-bit key schedule which can be initialised using 128, 192, or 256-bit keys. It has evolved from the earlier LOKI89 [BPS90], and LOKI91 [BKPS91] 64-bit block ciphers, with a strengthened key schedule, and a larger keyspace. The overall structure design and analysis of LOKI97 was performed by Dr Lawrie Brown during his 1997 sabbatical [1], with assistance and critique from Professors Josef Pieprzyk and Jennifer Seberry. The design and analysis of the S-box functions was done by Prof. Pieprzyk. LOKI97 has been submitted as a candidate algorithm for the US NIST Advanced Encryption Standard [NIS97] call.

The specification for the new LOKI97 cipher is presented. Discussion of its background, design rationale, expected efficiency, and a preliminary analysis then follows.

2 LOKI97 Specification

LOKI97 is a private key block cipher which encrypts 128-bit data blocks using 128, 192, or 256-bit keys.

2.1 Data Computation

LOKI97 encrypts a 128-bit plaintext block to create a 128-bit ciphertext block. The data computation is initialised by dividing the 128-bit plaintext input value $[L|R]$

[*] Phone: +61 2 62688184, Email: Lawrie.Brown@adfa.oz.au
[†] Phone: +61 2 4221 3872, Email: josef@cs.uow.edu.au

[1] at NTNU, Trondheim, Norway; and UOW, Wollongong, Australia

into two 64-bit words:

$$L_0 = L$$
$$R_0 = R$$

These are then processed through 16 rounds ($i = 1, 16$) of a balanced Feistel network:

$$R_i = L_{i-1} \; xor \; f(R_{i-1} + SK_{3i-2}, SK_{3i-1})$$
$$L_i = R_{i-1} + SK_{3i-2} + SK_{3i}$$

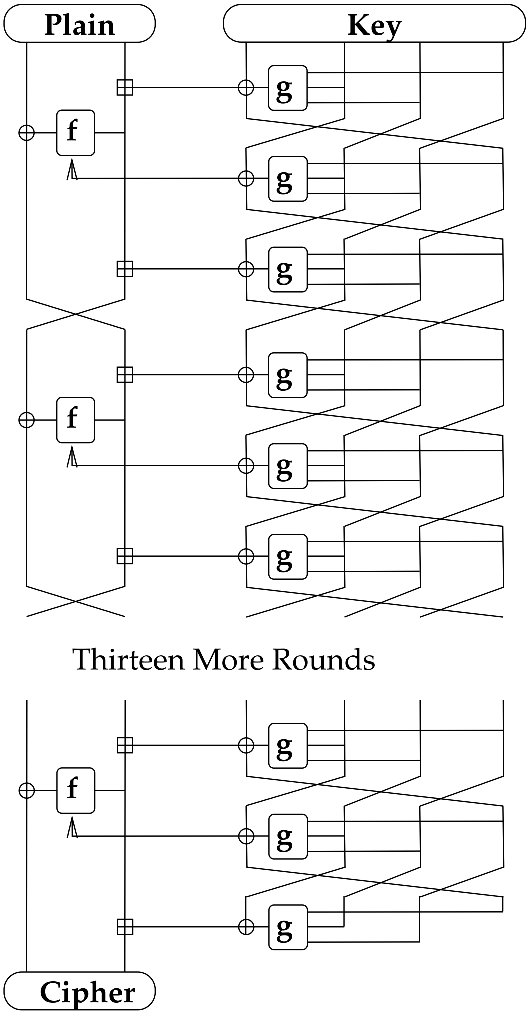
Each round uses xor (addition modulo 2) and integer addition + (modulo 2^{64}) of the 64-bit data values, along with the output of the complex nonlinear function $f(A, B)$ which provides maximal avalanche between all its input bits.

The resulting 128-bit ciphertext output value is composed of:

$$[R_{16}|L_{16}]$$

after the final round, ie with no final swap as usual.

The overall data computation thus looks as follows:



The decryption computation involves splitting the ciphertext into two 64-bit words:

$$[R_{16}|L_{16}]$$

and then running the rounds in reverse. ie: use 16 rounds ($i = 16, 1$)

$$L_{i-1} = R_i \; xor \; f(L_i - SK_{3i}, SK_{3i-1})$$
$$R_{i-1} = L_i - SK_{3i} - SK_{3i-2}$$

The decrypted 128-bit plaintext value is given by

$$[L_0 | R_0]$$

This is equivalent to performing the encryption computation with the subkeys in reverse order, and with additive inverses for SK_{3i}, SK_{3i-2}.

2.2 Key Schedule

LOKI97 uses a key schedule based on an unbalanced Feistel network (as per Schneier and Kelsey [SK96]), operating on four 64-bit words. It uses the same function $f(A, B)$ as the data computation to provide sufficient non-linearity to ensure that computing related keys is infeasible.

The key schedule is initialised, based on the size of the key supplied, into the four 64-bit words $[K4_0|K3_0|K2_0|K1_0]$ as follows:

Given a 256-bit key $[K_a|K_b|K_c|K_d]$**, let** $[K4_0|K3_0|K2_0|K1_0] = [K_a|K_b|K_c|K_d]$

Given a 192-bit key $[K_a|K_b|K_c]$**, let**
$[K4_0|K3_0|K2_0|K1_0] = [K_a|K_b|K_c|f(K_a, K_b)]$

Given a 128-bit key $[K_a|K_b]$**, let**
$[K4_0|K3_0|K2_0|K1_0] = [K_a|K_b|f(K_b, K_a)|f(K_a, K_b)]$

These are then processed through 48 rounds ($i = 1, 48$) to compute the 48 subkeys SK_i as follows:

$$SK_i = K1_i = K4_{i-1} \; xor \; g_i(K1_{i-1}, K3_{i-1}, K2_{i-1})$$
$$K4_i = K3_{i-1}$$
$$K3_i = K2_{i-1}$$
$$K2_i = K1_{i-1}$$

$where$

$$g_i(K1, K3, K2) = f(K1 + K3 + (Delta * i), K2)$$
$$Delta = \lfloor (sqrt(5) - 1) * 2^{63} \rfloor = 9E3779B97F4A7C15_{16}$$

Three rounds of the key schedule are required to produce the three subkeys for each round of the data computation. Thus a total of 48 rounds are required for the key schedule, for a cost of approx three encryption computations.

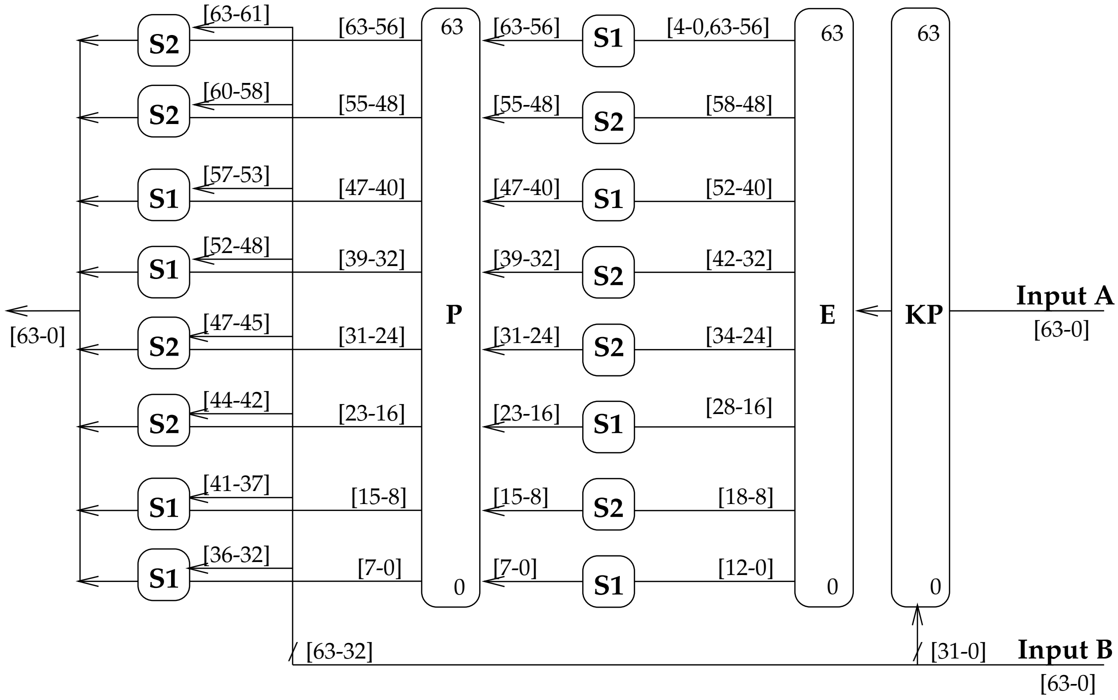
The integer addition (module 2^{64}) of $K1 + K3 + (Delta * i)$ forms an incompatible group with the xor used to compute the previous subkey, as in the data computation. It includes multiples mod 2^{64} of $Delta$, a value derived from the golden ratio by taking the integer part after multiplying it by 2^{63}, and used to reduce any symmetry problems in the key schedule.

Decryption is equivalent to encryption with the subkeys used in reverse order with (additive) inverses of the SK_{3i-2} and SK_{3i} subkeys. These will need to be precomputed in encrypt order first - there is no reverse shortcut as for LOKI89 and LOKI91.

2.3 Function f(A,B)

The highly non-linear function $f(A, B)$ takes two 64-bit input values A and B, processes them using two layers of S-boxes with the highest possible non-linearity, to produce a 64-bit output. The two permutations are used to ensure maximal avalanche between all bits in the function. It is specified as follows:

$$f(A, B) = Sb(P(Sa(E(KP(A, B)))), B)$$



Additional detail of the various component operations used is given below. Note the bit numbering convention used is that within each word, bit 0 is the rightmost, least-significant bit; and the highest numbered bit (31, 63, or 127 in 4, 8, or 16 byte words respectively) is the leftmost, most-significant bit.

KP(A,B) a very simple keyed permutation which splits its 64-bit input A into two 32-bit words, and uses the lower (rightmost) 32-bits of input B to decide whether to exchange corresponding pairs of bits in these words (if key bit is 1), or not (if key bit is 0), similar to that used in ICE [Kwa97]. It may be computed as:

$$KP([Al|Ar], SKr) = [((Al\&\neg SKr)|(Ar\&SKr))|((Ar\&\neg SKr)|(Al\&SKr))]$$

E() an expansion function, similar to LOKI91 but shifted for easier implementation, which selects overlapping groups of 13 bits (S1) or 11 bits (S2), so that at least some bits influence 2 S-boxes simultaneously, and with the preceeding addition, means all bits have some influence on multiple S-boxes. E creates a 96 bit output value from the 64 inputs bits as follows: $[4-0, 63-56|58-48|52-40|42-32|34-24|28-16|18-8|12-0]$.

Sa(), Sb() are two columns of S-boxes, composed by concatenating boxes S1 and S2, with Sa()=[S1,S2,S1,S2,S2,S1,S2,S1], and Sb()=[S2,S2,S1,S1,S2,S2,S1,S1]. In Sa() the inputs are data+key from the output of E, whilst in Sb() the upper bits are pure key bits (from the lower,rightmost 32-bits of B).

P() permutation to diffuse S-Box outputs across full 64-bit width, using a regular, latin-square, pattern, similar to LOKI91, but with a slight change so the same output never goes to the corresponding input. P maps input bits $[63-0]$ to output bits:

$$\begin{array}{l}[56,48,40,32,24,16,08,00,57,49,41,33,25,17,09,01,\\ 58,50,42,34,26,18,10,02,59,51,43,35,27,19,11,03,\\ 60,52,44,36,28,20,12,04,61,53,45,37,29,21,13,05,\\ 62,54,46,38,30,22,14,06,63,55,47,39,31,23,15,07]\end{array}$$

ie. input bit 63 goes to output bit 56, input 62 to output 48 etc.

This function can theoretically be implemented with 24 table lookups (8 each for Sa, P, and Sb), plus some and's, or's, xor's, shifts and adds for each round, making it reasonably fast and efficient. Using compiled code of course wont achieve this ideal.

2.4 S-boxes

The S-boxes chosen for LOKI97 use cubing in a galois field $GF(2^n)$ with n odd, as this has some highly desirable properties (discussed below). In order to use odd sized inputs, S1 uses 13 input bits, and S2 uses 11. These are alternated as specified above to combine to work on an even sized input block. The input value is inverted (so that a 0 or 1 input does not give 0 or 1 output), and the output values are masked down to select the lower 8 output bits only. The S-box functions are:

$$S1[x] = ((x\ xor\ 1FFF)^3\ mod\ 2911)\ \&\ FF,\ \ in\ \ GF(2^{13})$$
$$S2[x] = ((x\ xor\ 7FF)^3\ mod\ AA7)\ \&\ FF,\ \ in\ \ GF(2^{11})$$

(nb. all constants above are written in hex, all computations are done as polynomials in $GF(2^n)$)

2.5 Certification Triples

A sample LOKI97 certification triple is:

```
LOKI97    key: 000102030405060708090A0B0C0D0E0F
               101112131415161718191A1B1C1D1E1F
LOKI97  plain: 000102030405060708090A0B0C0D0E0F
LOKI97 cipher: 75080E359F10FE640144B35C57128DAD
```

A trace of this triple, showing all the subkeys, round values, and function f values, is given in Appendix A.

3 Design Background

LOKI97 is the latest in a family of block ciphers which includes LOKI89 and LOKI91, although in the light of lessons learnt from these earlier ciphers, its structure has evolved somewhat.

LOKI89 is a 64-bit, 16 round, symmetric block cipher with a 64-bit user key. It was originally designed by L. Brown, J. Pieprzyk and J. Seberry in 1990 [BPS90], and later re-designed (and renamed LOKI91), with improved resistance to differential cryptanalysis, by L. Brown, M. Kwan, J. Pieprzyk and J. Seberry [BKPS91].

The original version of LOKI89 was analysed by Biham and Shamir [BS91]. They found that whilst reduced round versions of LOKI89 could be susceptible to differential cryptanalysis, the full 16 round version was not.

Following the redesign to LOKI91 to strengthen its immunity, it was analysed by Knudsen [Knu92]. He found that there was no characteristic with a probablility high enough to do a sucessful differential attack; that the size of the image of the F-function in LOKI91 is $8/13x2^{32}$; and that there is a chosen plaintext attack that reduces an exhaustive key search on LOKI91 by almost a factor of 4 using $2^{32}+2$ chosen plaintexts. In a subsequent paper, Knudsen [Knu94] discusses the concept of weak hash keys in LOKI.

Biham [Bih94] introduced some new types of cryptanalytic attacks which exploit relations between sub-keys. He applied these attacks to both versions of LOKI. For LOKI91 the complexity is of $O(2^{61})$, which is faster than exhaustive search and comparable to Knudsen's results.

Tokita, Sorimachi, and Matsui [TSM94] have examined the susceptibility of LOKI91 to linear cryptanalysis, and have found 12 or more round versions to be immune. Subsequently, Knudsen and Robshaw [KR96] have made incremental improvements to the search efficiency using non-linear approximations, but 12 or more rounds in LOKI91 are still immune. Recently Sakurai and Furuya [SF97] have described further incremental advances.

LOKI91 is currently regarded as a reasonably secure cipher, immune to both differential and linear cryptanalysis, whose main deficiencies are the linear key schedule which leaves it susceptible to some related key attacks, and the keysize which, given those attacks, is effectively about 2^{60}.

LOKI91 is described in Schneier [Sch96], which has led to some continued interest in its use by organisations wanted an unencumbered encryption algorithm. This is likely to continue with the release of a small, fast Java version in the publically available Cryptix library [GN97].

4 Design Considerations

Given experience with the earlier LOKI designs, it was felt that some form of Feistel cipher was still the most appropriate approach. The core design properties for such ciphers are for them to demonstrate appropriate avalanche and completeness effects [WT86]. As with the earlier LOKI designs, the general approach to ensuring these properties was to partition the design into two phases:

- First an overall algorithm structure was designed which provides these properties on the assumption that the S-boxes possess them.
- Second, appropriate S boxes were selected with the desired characteristics.

Consideration was given to the necessary conditions for a secure Feistel cipher noted by Knudsen [Knu93]:

- there are no simple relations
- all keys are equally good
- resistance to differential attacks
- resistance to linear attacks

A number of other modern private-key block ciphers were reviewed in the early design phase, including Blowfish, CAST, ICE, IDEA, SAFER, SPEED and TEA (descriptions of most of which may be found in Schneier [Sch96]). A number of variant approaches were seen in these, including in the use of:

- random vs designed S-boxes

- incompatible mixing operations
- complexity of round function vs number of rounds

Following this review, it was decided that LOKI97 would use designed S-boxes, utilise two incompatible mixing operations (addition and XOR), and have a smaller number of iterations of a complex round function.

Also, to provide a stronger key schedule than previous LOKI variants, it was decided to use the same complex nonlinear round function in the key schedule, as in the data computation. This results in the computation cost of the key schedule being of the same order of magnitude as the data computation, that is not too costly for applications with periodic key changes, but costly enough to modestly penalise exhaustive key search operations which require a key change on each block.

4.1 The LOKI97 Round Function $f(A, B)$

The round function $f : \Sigma^{64} \times \Sigma^{64} \to \Sigma^{64}$ takes a 64-bit input and transforms it to 64-bit output using a 64-bit subkey. The key features in the round function are that:

- it applies two S-boxes: S1 and S2 composed into two layers of an S-P network. In each layer S1 and S2 are used four times each;
- the S-P network satisfies the completeness property, that is each output bit is a complex function of all input bits;
- the 64-bit subkey hides the structure of the round function when the cryptanalyst does not know the subkey.

The major components of the round function are the two S-boxes: S1 and S2; and the permutations KP, E and P.

4.2 Rationale for the S-box Selection

S-boxes are required to satisfy the following properties:

1. S-boxes must be balanced, i.e. each output Boolean function must contain the equal number of zeros and ones in its truth table.
2. Any output (input) Boolean function of an S-box must be highly nonlinear, that is there is no "good" approximation of a single output (input) function by a linear function.
3. S-boxes should satisfy the SAC criterion, i.e. a single bit change on the input causes changes on approximately half of the output bits.
4. S-boxes should have a "good" XOR profile. In other words, the profile must be relatively uniform with the smallest and biggest entries not wildly different from the average. Typically, it is also desirable that the entries related to zero output difference should be as small as possible (preferably all equal to or smaller than the average).

The S-boxes applied in LOKI97 are designed using cubing in GF(2^n) or more precisely: $S(x) : GF(2^n) \to GF(2^n)$ where $S(x) = x^3 \bmod p(x)$ and $p(x)$ is an irreducible polynomial used for $GF(2^n)$ arithmetic computations. Note that the cyclic group in $GF(2^n)$ has the order $2^n - 1$ and consists of all nonzero elements of the field. It is well known that exponentiation $x^\alpha \bmod p(x)$ is a permutation if α does not divide $2^n - 1$. The lowest exponent which generates a nonlinear permutation is 3 provided that 3 does not divide $2^n - 1$. This condition holds whenever n is odd (see [Pie92]). In LOKI97, the S1 and S2 S-boxes use cubing in $GF(2^{13})$ and $GF(2^{11})$, respectively. As these S-boxes use cubing for odd n, all output Boolean functions are balanced. This remains true even after the outputs are truncated to 8 bits. Cubing permutations in $GF(2^n)$ are highly nonlinear as any output function shares the nonlinearity with the function(see [Pie92])

$$\eta(x) = x_1x_2 + \ldots + x_{n-2}x_{n-1} + x_n$$

whose nonlinearity is equal to

$$\mathcal{N}(\eta) = \frac{1}{2} - \frac{1}{2^{(n+1)/2}}.$$

Observe that $\eta(x)$ for $x_n = 0$ and restricted to $(n-1)$ variables is a bent function (attains the maximum nonlinearity in the space of $(n-1)$ Boolean variables). The high nonlinearity of all output functions implies that the SAC property holds.

A number of generators were trialed before the final selection of the ones used in LOKI97. All generators gave functions with maximal non-linearity, but there were small differences in other characteristics, such as variations in the single bit avalanche characteristics. Those chosen were a compromise which resulted in a reasonable balance of the various measures tested (see the later preliminary analysis section for some of the figures).

4.2.1 XOR Profile

Consider the XOR profile of cubing permutations. We follow the standard notation, that is, the XOR profile is represented by $2^n \times 2^n$ table where $\Delta \in GF(2^n)$ is the input difference and $\delta \in GF(2^n)$ is the output difference. A value of Δ labels a row and δ indicates a column in the XOR profile table. Thus $\Delta = x + y$ where x, y are two input values from $GF(2^n)$ and $\delta = x^3 + y^3$ is the corresponding output difference. After some simple transformations, we obtain

$$\delta = \Delta(x^2 + \Delta x + \Delta^2) \tag{1}$$

The entry (Δ, δ) in the XOR profile indicates the number of x values for which Equation (1) holds. Take the first row and column of the XOR profile, i.e. $\Delta = \delta = 0$. Equation (1) holds for all $x \in GF(2^n)$ so the entry $(\Delta, \delta) = (0, 0)$ is equal to 2^n. It is obvious that if Equation (1) holds for x than it must be satisfied for $y = x + \Delta$. To see that this is true, we start from Equation (1) which holds for x. So

$$\begin{aligned} \delta &= \Delta(x^2 + \Delta x + \Delta^2) \\ &= \Delta((\Delta + x)^2 + \Delta x) \\ &= \Delta(y^2 + \Delta(y + \Delta)) \\ &= \Delta(y^2 + \Delta y + \Delta^2) \end{aligned}$$

and Equation (1) holds also for y. We have proved the following lemma.

Lemma 4.2.1 *Given a cubing permutation and its XOR profile. If $x \in GF(2^n)$ satisfies Equation (1), then there is $y = x + \Delta$ for which the same equation holds. In other words, all entries of the XOR profile are even.*

The next lemma shows that the XOR profile has only zero and twos are its entries.

Lemma 4.2.2 *Given a cubing permutation and its XOR profile. If $x \in GF(2^n)$ satisfies Equation (1) for some fixed (Δ, δ), then any transformation $z = x + \alpha$ ($\alpha \neq \Delta$) causes Equation (1) to hold for z with $\delta_1 \neq \delta$.*

Proof : We start from Equation (1). After substituting $x = z + \alpha$, we get

$$\begin{aligned} \delta &= \Delta((z+\alpha)^2 + \Delta(z+\alpha) + \Delta^2) \\ &= \Delta(\Delta^2 + \Delta\alpha + \alpha^2 + \Delta z + z^2) \end{aligned}$$

which gives the following

$$\delta + \Delta^2\alpha + \alpha^2\Delta = \Delta(z^2 + \Delta z + \Delta^2).$$

Write $\delta_1 = \delta + \Delta^2\alpha + \alpha^2\Delta$. What remains to be proved is that $\Delta^2\alpha \neq \alpha^2\Delta$ for $\Delta \neq \alpha$. This part is proved by contradiction. Assume that $\Delta^2\alpha = \alpha^2\Delta$. As we work with $GF(2^n)$, there is a primitive element g which generates all nonzero elements of the field. Let $\Delta = g^b$ and $\alpha = g^a$. Therefore

$$\Delta^2\alpha = g^{2b+a} \text{ and } \alpha^2\Delta = g^{2a+b}$$

The assumption that $\Delta^2\alpha = \alpha^2\Delta$. implies that

$$2b + a \equiv 2a + b \pmod{2^n - 1}$$

which is equivalent to the congruence that $a \equiv b \bmod 2^n - 1$. This is the required contradiction which proves that $\Delta^2\alpha \neq \alpha^2\Delta$ as long as $\Delta \neq \alpha$ and shows that $\delta_1 \neq \delta$. □

Theorem 4.2.1 *Given a cubing permutation in $GF(2^n)$, every row in the XOR profile consists of entries either 0 or 2.*

Proof : (Sketch) Note that the entries in each row must sum up to 2^n. A given entry (Δ, δ) contains the number of x values for which Equation (1) holds. Lemma (4.2.1) says that all entries must be even. According to Lemma (4.2.2), every $z = x + \alpha$ where $\alpha \neq \Delta$ falls into some other entry (Δ, δ_1). □

For further background, see [Nyb94].

The above considerations are valid for the cubing permutation on the full $n \times n$ S-box. What happens if we cut off some output bits ? Clearly the resulting $(n \times m)$ S-box is no longer a permutation $(m < n)$. Some observations for shortened $(n \times m)$ S-boxes based on cubing are:

- the number of rows in the XOR profile is still equal to 2^n but the number of columns is 2^m,
- sum of all entries in every row equals 2^n,
- an entry (Δ, δ) $(\Delta \in GF(2^n), \delta \in GF(2^m))$ is the sum of all entries (Δ, δ') of the full $n \times n$ XOR profile for which $\delta' \in GF(2^n)$ equals $\delta \in GF(2^m)$.

Consider our two selected S-boxes. S1 is 13 × 8 S-box. As the XOR profile of the cubing permutation contains zeros and twos only, we expect that the XOR profile of S1 may have the biggest entry $2 \cdot 2^5 = 64$ when all 2^5 merged entries are equal to 2. Some smaller entries occur when the folding down procedure (caused by dropping the output bits) mixes zero entries with nonzero entries of the XOR profile of the cubing permutation.

S2 is 11 × 8 S-box. The biggest entry is $2 \cdot 2^3 = 16$ if all merged entries for the cubing permutations are equal to 2. It is expected that some other entries are multiple of 2 not exceeding 16.

4.3 Rationale for Permutations KP, E and P

The permutations are required to provide the necessary diffusion of the outputs from the S-boxes over as many of the S-box inputs in the next layer as possible.

The keyed permutation KP acts to obscure which S-box any given input bit will feed to in the round. It does this by selectively exchanging, under the control of some subkey bits, pairs of input bits. Given the arrangement of the following layer of S-boxes, it causes the bits to swap between an S1 input and an S2 input. Thus any attack has to consider probabilities in both boxes. The idea of this keyed permutation is taken from that used in the ICE cipher [Kwa97].

Permutation E selects overlapping groups of either 13 bits (for S1) or 11 bits (for S2). It ensures that some of the input (combined data and key) bits influence 2 S-box inputs simultaneously, restricting which bits can be feasibly used for attacks.

Permutation P provides the main diffusion function. Following the same principles used in the earlier LOKI designs, this was selected to use a regular latin-square pattern, as this has been found to provide the fastest possible diffusion in the least number of rounds [BS90]. The particular pattern used here was selected to ensure that no output bit is permuted to the same bit (as occured in the earlier LOKI designs)

4.4 Rationale for the Key Schedule

As discussed previously, following the analyses of the existing LOKI ciphers, we decided to use a non-linear key schedule with the same round function as used for the data computation. Since the key schedule was to be initialised with up to 256-bit keys, an unbalanced Feistel network was needed [SK96]) being:

$$\begin{aligned}
&SK_i = K1_i = K4_{i-1}\ xor\ g_i(K1_{i-1}, K3_{i-1}, K2_{i-1})\\
&K4_i = K3_{i-1}\\
&K3_i = K2_{i-1}\\
&K2_i = K1_{i-1}
\end{aligned}$$

In order to accomodate the necessary 3 inputs, the function $g(K1, K3, K2)$ was defined from the existing round function as:

$$\begin{aligned}
&g_i(K1, K3, K2) = f(K1 + K3 + (Delta * i), K2)\\
&Delta = \lfloor (sqrt(5) - 1) * 2^{63} \rfloor = 9E3779B97F4A7C15_{16}
\end{aligned}$$

The addition of multiples of $Delta$, a value derived from the golden ratio by taking the integer part after multiplying it by 2^{63} is used to minimise any symmetry affects in the key schedule, as was done in the TEA cipher [WN94].

In order to support the smaller 128 and 192-bit keys, an initialisation phase is used that again uses the same non-linear function $f(A, B)$ to expand the supplied bits up to 256-bits for the key schedule, in a complex non-linear manner.

5 Computational Efficiency

The following computational estimates have been obtained by using the program `aestime` with the C reference implementation of LOKI97. This program, for each key/block size combination, times the algorithm initialisation (which precomputes the 10496 entries for S1, S2 and P), encryption of 1 megabyte of data (ie 65536 blocks of 128 bits), decryption of 1 MB, and setting of 1000 encrypt and decrypt key pairs. From this information, the estimates given below were derived. This program is included in the C reference implementation.

A few comments. Within the statistical timing variations, there is basically no significant difference between the times for the various key/block sizes. This is as expected since, apart from a small change in the initialisation of the key schedule, there is no difference in the work done. Also, once the key schedule has been computed (ie `makeKey` has been called), then since the key schedule information is passed as one parameter to encryptBlock or decryptBlock, changing key incurs no additional cost. Finally, in the raw timing figures, a time appears only for the first initialisation, since the code knows it has been done before, it wont recompute the tables.

5.1 Computational Speed on a Pentium 2 (Reference system)

The following figures were obtained from a number of runs of the `aestime` program on a Pentium 2/233MHz system with 64MB RAM. Runs were done with the code compiled using Microsoft Visual C++ under Win95, as well as using gcc under linux. There was no significant difference in the average values, as one would expect for a computationally bound task (though the variance under Win95 was rather higher). The following figures are typical of those obtained (nb times are in millisecs, with a raw resolution around 5ms):

Key/Blk	Init	Encrypt 1Mb	Decrypt 1Mb	Key Init (1000 pairs)
128/128	30	1370	1340	40
192/128	0	1370	1320	40
256/128	0	1380	1330	35

From the above information, and noting that 1ms = 233000 clocks, we have the following number of clock cycles to:

Key/Blk	Init Cipher	Encrypt 1 Blk	Decrypt 1 Blk	Init 1 Key	Change Key
128/128	6990000	4870	4764	4660	0
192/128	6990000	4870	4692	4660	0
256/128	6990000	4906	4728	4078	0

The code used in the LOKI97 library had 9706 bytes of instructions, and 12290 bytes of data (almost all of that being space for the precomputed S1, S2 and P tables).

Given the functional specification for LOKI97's S-boxes, a time-space tradeoff is possible, since the S-box values can be computed either during the initialisation stage, and saved in 2 tables totaling 10kB, or can be computed as needed during the key scheduling and data computations.

5.2 Computational Speed on a PDP-11 (approx 8-bit system)

The following figures were obtained by running a version of the `aestime` program on a PDP-11 simulator (this being the closest to an 8-bit system we could obtain). The code was changed to traditional K&R C, with signed integer operations only, and compiled using a PDP-11 Version 7 Unix K&R C compiler. It was then run on a simulator which returned the current instruction number to the timing call. Thus the following figures give the number of instructions executed for each stage:

Key/Blk	Init	Encrypt 1Mb	Decrypt 1Mb	Key Init (1000 pairs)
128/128	437918	73016214	66608249	4156598
192/128	18	73016214	66630095	4178731
256/128	18	73016214	65418923	4203200

From the above information, we have the following number of instructions for each stage. To convert these to timings, note that a PDP-11/04 for example took typically around $3 \mu sec$ for integer and memory access instructions.

Key/Blk	Init Cipher	Encrypt 1 Blk	Decrypt 1 Blk	Init 1 Key	Change Key
128/128	437918	1114	1016	2078	0
192/128	437918	1114	1016	2089	0
256/128	437918	1114	998	2101	0

The code used in the LOKI97 library here had 11772 bytes of instructions, and 13754 bytes of data (most being space for the precomputed S1, S2 and P tables).

As before, a time-space tradeoff is possible between precomputing the S-boxes, and computing the S-box functions as needed. Also the current code uses purely integer operations on 8, 16 and 32 bit signed integers. A pure 8-bit implementation would probably better be written to explicitly use byte and short values only, saving probably sub-optimal compiler approximations generated to handle the longer word operations.

6 Expected Strength Against Attack

At the current time, although some possible attack approaches are suggested in the analysis section below, there is no attack currently known on LOKI97 which is faster than exhaustive key search. Also, since there are no known relations between keys, all possible key values need to be searched. Consequently, the expected strength of LOKI97 against attack at present is the cost of exhaustive key search, which is:

Key/Blk	No Trials in Key Search
128/128	2^{128}
192/128	2^{192}
256/128	2^{256}

As noted below, each trials involves a cost of about 4 encryptions.

7 Preliminary Analysis

7.1 Key Search

LOKI97 uses 128, 192, or 256-bit keys in a 256-bit key schedule. Even with the shortest key size of 128-bits, exhaustive searches are currently infeasible, and likely to remain so for some considerable time (barring revolutionary advances like the development of practical quantum computers). The cost of such a search would, for 128-bit keys be, on average, about $O(2^{127})$ trials (ie 50% of all keys), each of which involves a cost of about 4 encryptions (approx 3 for the key schedule and 1 for the actual test encryption).

7.2 Characteristics of the Key Schedule

Given the use of the non-linear function $g(K1, K3, K2)$ in the key schedule, it is clear that there are no simple relations between any of the subkey bits, rather they depend in a complex manner on all the input key bits. Thus there are no obvious weak keys (those which result in constant, or repeated values, for all the subkeys).

The best we can see doing is to derive those key values which result in up to the first 4 subkeys (affecting only the first 1 $^1/_3$ data rounds) being zero. These can obviously be derived by using relationships between the input key values, which take the form: $[K4_0|K3_0|K2_0|K1_0]$, and linear relations between the subkeys.

One round of the LOKI97 key schedule looks as follows:

$$\begin{aligned}
&SK_i = K1_i = K4_{i-1} \ xor \ g_i(K1_{i-1}, K3_{i-1}, K2_{i-1})\\
&K4_i = K3_{i-1}\\
&K3_i = K2_{i-1}\\
&K2_i = K1_{i-1}
\end{aligned}$$

In more specific cases, we could attempt to coerce the first few subkey values to some known (preferably 0) value. Now the first few rounds of the key schedule, given a 256-bit input key of $[K4|K3|K2|K1]$, can be specified as follows:

```
SK1 = K4 xor f(K3+K1+D,K2) nb. D = Delta
SK2 = K3 xor f(K2+SK1+2D,K1)
SK3 = K2 xor f(K1+SK2+3D,SK1)
SK4 = K1 xor f(SK1+SK3+4D,SK2)
SK5 = SK1 xor f(SK2+SK4+5D,SK3)
SK6 = SK2 xor f(SK3+SK5+6D,SK4)
.....
```

so, attempting to solve for successive zero subkeys, we get:

SK1 = 0 implies

```
K4 = f(K3+K1+D,K2)
eg.
[f(D,0)|0|0|0] =
[A6ACC1AD4F7D648E000000000000000000000000000000000000000000000000]
```

SK2 = SK1 = 0 implies

```
K3 = f(K2+2D,K1), and
K4 = f(f(K2+2D,K1)+K1+D,K2)
eg.
[f(f(2D,0)+D,0)|f(2D,0)|0|0] =
[D56AEDDD378763C8B3F8B84B61E6FD2D00000000000000000000000000000000]
```

SK3 = SK2 = SK1 = 0 implies

```
K2 = f(K1+3D,0), and
K3 = f(f(K1+3D,0)+2D,K1), and
K4 = f(f(f(K1+3D,0)+2D,K1)+K1+D,f(K1+3D,0))
eg.
[f(f(f(3D,0)+2D,0)+D,f(3D,0))|f(f(3D,0)+2D,0)|f(3D,0)|0] =
[CC3533C1DAE8E39076AC4AAAF1F443A7802D899C87BB07FD0000000000000000]
```

SK4 = SK3 = SK2 = SK1 = 0 implies

```
K1 = f(4D,0), and
K2 = f(f(4D,0)+3D,0), and
K3 = f(f(f(4D,0)+3D,0)+2D,f(4D,0)), and
K4 = f(f(f(f(4D,0)+3D,0)+2D,f(4D,0))+f(4D,0)+D,f(f(4D,0)+3D,0))
eg.
[DF74B90CE9D04479AF42CF6ACD63B8526E989FA699AB078E6BB007E91228F095]
```

ie. there is only 1, completely specified value, which gives this.

SKn and lower = 0 implies that $SKn = 0 = f(nD, 0)$, which is not true for any n=1,48 (ie all values of use in the LOKI97). Thus it is not possible to have the first n subkeys zero for any $n > 4$.

So as a general statement, we can coerce the first 4 subkeys to zero (or indeed any desired value), with exponentially decreasing possibilities as we fix successive subkeys. After that, there are no obvious related values.

If we consider the shorter keys, then in fact the initialisation which specifies $K2 = f(K_3, K_4)$, and $K1 = f(K_4, K_3)$, make it extremely difficult to solve the above equations, even for just $SK1 = 0$, at all.

Given the use of the highly non-linear function f, we do not believe that any trapdoors exist in the key schedule, or for LOKI97 in general, although no formal analysis has as yet been done to verify this.

Consequently, the analysis of the key schedule can be summarised as follows:

- Weak Keys - no weak keys (keys which result in all subkeys being identical) are known to exist.
- Key Complementation Properties - none are known
- Restrictions on Key Selection - given that no weak keys exist, there are no keys known which must be avoided. It is probably desirable to avoid the 256 bit key specified above which results in the first 4 subkeys being 0, though no attack is known which can exploit it.

7.3 Analysis of the S-boxes

The LOKI97 S-boxes are:

$$S1[x] = ((x\ xor\ 1FFF)^3\ mod\ 2911)\ \&\ FF,\ \ in\ \ GF(2^{13})$$
$$S2[x] = ((x\ xor\ 7FF)^3\ mod\ AA7)\ \&\ FF,\ \ in\ \ GF(2^{11})$$

These are designed to have maximum non-linearity (hence the choice of cubing in an polynomial galois field with n odd), have good immunity to differential cryptanalysis, and have good avalanche properties.

The input values are inverted in order to remove the mapping of $0 \rightarrow 0$, $1 \rightarrow 1$, that otherwise would occur.

A number of generator polynomials are available in each field (630 in $GF(2^{13})$ and 180 in $GF(2^{11})$). All of these seem to generate basically equivalent XOR profiles, so this was not a determiner in their selection. There were however minor differences in their avalanche characteristics, which led to the final selection.

7.3.1 XOR Profiles

The XOR profiles of the S-boxes are remarkably regular and relatively flat.

Analysis shows that for S1, the maximum value is 64 (occuring 32640 times) out of a maximum of 8192; and further, in the zero output column (ie where different inputs give the same output value), the maximum value is just 32 (occuring 7936 times). These are as were predicted theoretically earlier. Should a characteristic be constructed using one of these entries, it would have a probability of $Pr(32/8192) = Pr(1/256)$.

For S2, the maximum value is 16 (occuring 32640 times) out of a maximum of 2048; and further, in the zero output column (ie where different inputs give the same output value), the maximum value is just 8 (occuring 1792 times). The individual row profiles are either all 8's, or combinations of 0 and 16 in various arrangements. A characteristic constructed using one of these entries, would also have a probability of $Pr(8/2048) = Pr(1/256)$.

Whilst these results imply that there are many possibilities with these values, the actual probability of success is much lower than that seen in other ciphers (eg

LOKI91). Further, the construction of the round function is designed to obscure which S-bit any input bit will go to, and thus to greatly reduce the chance of success.

7.3.2 Avalanche Properties

The avalanche characteristics of the chosen functions were evaluated by exhaustively testing for each input bit position, for all other bit values, the effect of flipping that bit on the output value. Counts were kept of which output bits changed (the result being very even for both), as well for when no bits changed, or just one bit changed (ie no avalanche occured). As shown in the table below, these cases occured a small percentage of the time.

```
LOKI97 Candidate S:(x+1fff)^3 mod 2911 GF(2^13)
S_test - Avalanche Test & 1-bit Changes (rule 3)
Col Tot   0   0   0   0   0   0   0   0   0   0   0   0   0 (   0)
No aval  16  16  16  16  16   0   0  16  16  16  16  16  16 ( 176)
1bit av 128 128 128 128 128  96 128 128 128 128 128 128 128 (1632)

LOKI97 Candidate S:(x+7ff)^3 mod aa7 GF(2^11)
S_test - Avalanche Test & 1-bit Changes (rule 3)
Col Tot   0   0   0   0   0   0   0   0   0   0   0 (   0)
No aval   4   0   0   4   4   0   4   4   4   4   4 (  32)
1bit av  32  24  32  32  32  16  32  32  32  32  32 ( 328)
```

In summary:

```
S1
Output bits flip exactly 1/2 the time,
For any input bit, 0 change occurs in at most 16/4096 (ie 1/256)
For any input bit, 1 bit change occurs in at most 128/4096 (ie 1/32)

S2
Output bits flip exactly 1/2 the time,
For any input bit, 0 change occurs in at most 4/1024 (ie 1/256)
For any input bit, 1 bit change occurs in at most 32/1024 (ie 1/32)
```

These results are comparable with those seen in the LOKI89 and LOKI91 S-box function, which have been shown to be well designed with good security.

7.4 Differential Cryptanalysis

One round of the LOKI97 data encryption computation looks as follows:

$$\begin{aligned} R_i &= L_{i-1} \; xor \; f(R_{i-1} + SK_{3i-2}, SK_{3i-1}) \\ L_i &= R_{i-1} + SK_{3i-2} + SK_{3i} \end{aligned}$$

where the non-linear function $f(A, B)$ is specified as:

$$f(A, B) = Sb(P(Sa(E(KP(A, B)))), B)$$

The first thing to note is that the use of integer addition in the R data value will in general result in the loss of any desired fixed XOR difference (though there may be a small fraction of cases for which this is not true). Thus conventional differential cryptanalysis would seem improbable. A modified variant using additive differences on the R data path was considered, however the initial keyed permutation KP in f will ensure that fixed additive differences will be destroyed as the bits are exchanged. So this approach also seems unlikely.

Consider conventional differential cryptanalysis again. Even assuming we have some fraction of cases where the difference survives the addition, the presence of KP ensures that we end up attacking through either S1 or S2 in any round, but we don't know which. Given the attack probability is the same, this suggests that a conventional 2-round characteristic, using $S(x') \to 0$ with $Pr(1/256)$ could iterate at best (modulo the massive difficulties imposed by the additions of subkey values and the keyed permutation KP) over 16 rounds with $Pr(2^{-64})$. It is not expected that anything approaching this probability can be achieved in practise.

Use of the $S(x') \to 0$ characteristic enables the second column of S-boxes in the round to be ignored (since same input leads to same output). Any other characteristic type (eg $S(x') \to x'$) will not be able to do this, and will thus have vastly lower probabilities since both layers of S-boxes will be involved in each round.

7.5 Linear Cryptanalysis

Some bounds on the susceptability of LOKI97 to linear cryptanalysis can be set knowing the non-linearity of its S-boxes.

Given a $n \times n$ S-box with its nonlinearity $\mathcal{N}(S) = \frac{1}{2} - \frac{1}{2^{(n+1)/2}}$, any linear approximation $\ell()$ of its inputs and outputs will correctly reflect the real dependencies with the probability P_ℓ such that

$$\frac{1}{2} - \frac{1}{2^{(n+1)/2}} \leq P_\ell \leq \frac{1}{2} + \frac{1}{2^{(n+1)/2}}.$$

S_1 may be approximated by a linear equation $\ell()$ with the probability

$$\frac{1}{2} - 2^{-7} \leq P_\ell \leq \frac{1}{2} + 2^{-7}.$$

Similarly S_2 may be approximated by a linear equation $\ell()$ with the probability

$$\frac{1}{2} - 2^{-6} \leq P_\ell \leq \frac{1}{2} + 2^{-6}.$$

The round function is a network of S_1 and S_2. To find a linear approximation of the input/output of the round function, we may attempt to find a linear path created by linear approximation of two S_2. Note that any other path will include at least one S_1 whose linear approximation is less accurate. Let the first S_2 be approximated by a linear function

$$\ell_1(s_1, \ldots, s_{11}, u_1, \ldots, u_8) = 0$$

where $s_1, \ldots, s_{11}$ are inputs and $u_1, \ldots, u_8$ outputs and $\ell_1()$ is a linear function. The second S_2 can be approximated by

$$\ell_2(u'_1, \ldots, u'_8, k_1, k_2, k_3, s^*_1, \ldots, s^*_8) = 0$$

where $((u'_1, \ldots, u'_8, k_1, k_2, k_3)$ are inputs and $(s^*_1, \ldots, s^*_8)$ outputs. The two approximations ℓ_1 and ℓ_2 represent two independent random variables. If we combined them then the linear approximation of the single round can be done no better than (see [Mat93])

$$\frac{1}{2} + 2(2^{-6})(2^{-6}) = \frac{1}{2} + 2^{-11}$$

In other words the best linear approximation of a single round is given by

$$\frac{1}{2} - 2^{-11} \leq P_{\text{best}} \leq \frac{1}{2} + 2^{-11}.$$

For full 16 rounds. we can find bounds for the probablity

$$\frac{1}{2} + 2^{15}(2^{-11})^{16} = \frac{1}{2} + 2^{-161}$$

The best linear approximation for full 16 round is

$$\frac{1}{2} - 2^{-161} \leq P_{\text{best}} \leq \frac{1}{2} + 2^{-161}.$$

The best linear charactersitics for 14 rounds is

$$\frac{1}{2} - 2^{-141} \leq P_{\text{best}} \leq \frac{1}{2} + 2^{-141}.$$

The last case might be useful when the approximations in the first and the last rounds can be skipped (somehow).

The above computations are based on the following theorem.

Theorem 7.5.1 *(Matsui [Mat93]) Given n independent random variables $X_1, \ldots, X_n$ such that $P(X_i = 0) = p_i$ and $P(X_i = 1) = 1 - p_i$ for $i = 1, \ldots, n$. Then the probability that $X_1 \oplus \ldots \oplus X_n = 0$ is*

$$\frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} (p_i - 0.5). \qquad (2)$$

7.6 Overall Cipher Characteristics

Some preliminary trials have been run to characterise the overall cipher. In particular, some random trials have been run to determine its avalanche characteristics. Avalanche is one of the key desired properties of block ciphers, where changing 1 bit of input should result in approx half the output bits changing, and for each specific output bit to change about half the time. It is impossible to totally characterise avalanche for a cipher like LOKI97 as there are too may possibilities. However we have run some random trials (with random key and data, and then flipped each bit of data in turn). The results are as follows:

```
LOKI97 avalanche: 640384 tests, 63.99617 mean no bits changed per test,
out bit avalanches range over 0.49830258 - 0.5021237
```

8 Advantages and Limitations

LOKI97 is a general block cipher, optimised for bulk data encryption applications. However it certainly can be used in any mode which has been defined for a block cipher, including stream modes such as CFB (CFB1 is implemented in the reference code) or OFB; MAC modes such as CBC or any newer modes; in OFB mode as a pseudo-random number generator; or as a hashing algorithm using CBC, CFB or other hash modes. Note though, that since changing the key incurs an overhead of approximately 3 data encryptions, any mode of use which treats the key as a data input (such as some of the hashing modes) will be penalised.

Given that LOKI97 is a general block cipher, which requires only simple integer arithmetic and logical operations, along with table lookups for its implementation, there is no reason why its should not be implementable in all the environments specified, including 8-bit processors, ATM, HDTV, B-ISDN, voice applications, satellite applications etc. Data rates of order 1Mb/s have been demonstrated on a pentium grade processor in software with the current code, thus any application requiring this or a lower rate can be supported in software. Faster rates, such as would be needed for ATM etc, would require a hardware implementation.

LOKI97 has been designed explicitly only for keysizes of 128, 192 and 256 bits, and a datasize of 128 bits. No other combinations are supported.

LOKI97 uses mathematically designed S and P-boxes to maximise avalanche and completeness properties for the round function. The functional specification of the S-boxes permits a time-space efficiency tradeoff, as well as providing maximal non-linearity.

9 Conclusions

This paper describes the design of the new LOKI97 block cipher, which has evolved from the earlier LOKI91 design so as to use 128-bit data and 128/192/256 bit keys, with a greatly strengthened key schedule, and using a significantly more complex round function composed of 2 S-P layers. Some discussion of the design rationale is given, followed by the preliminary cipher analysis. It is also an AES candidate algorithm.

References

[Bih94] Eli Biham. New Types of Cryptanalytic Attacks Using Related Keys. *Journal of Cryptology*, 7(4):229–246, 1994.

[BKPS91] Lawrence Brown, Matthew Kwan, Josef Pieprzyk, and Jennifer Seberry. *Improving Resistance to Differential Cryptanalysis and the Redesign of LOKI*, volume 739 of *Lecture Notes in Computer Science*, pages 36–50. Springer-Verlag, 1991. Also issued as ADFA TR CS38/91.

[BPS90] Lawrence Brown, Josef Pieprzyk, and Jennifer Seberry. *LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications*, volume 453 of *Lecture Notes in Computer Science*, pages 229–236. Springer-Verlag, 1990. Also issued as ADFA TR CS1/90.

[BS90] Lawrence Brown and Jennifer Seberry. *On the Design of Permutation P in DES Type Cryptosystems*, volume 434 of *Lecture Notes in Computer Science*, pages 696–705. Springer-Verlag, 1990. Also issued as ADFA TR CS6/89.

[BS91] Eli Biham and Adi Shamir. *Differential Cryptanalysis Snefru, Kharfe, REDOC-II, LOKI and Lucifer*, volume 576 of *Lecture Notes in Computer Science*, pages 156–171. Springer-Verlag, 1991.

[GN97] Ian Grigg and Raif Naffah. Cryptix Java Cryptographic Library, 1997. http://www.systemics.com/docs/cryptix/.

[Knu92] Lars Knudsen. *Cryptanalysis of LOKI91*, volume 718 of *Lecture Notes in Computer Science*, pages 196–208. Springer-Verlag, 1992.

[Knu93] Lars Knudsen. *Practically Secure Feistel Ciphers*, volume 809 of *Lecture Notes in Computer Science*, pages 211–221. Springer-Verlag, 1993.

[Knu94] Lars Knudsen. *New potentially weak keys for DES and LOKI*, volume 950 of *Lecture Notes in Computer Science*, pages 419–424. Springer-Verlag, 1994.

[KR96] Lars Knudsen and M.J.B. Robshaw. *Non-linear Approximations in Linear Cryptanalysis*, volume 1070 of *Lecture Notes in Computer Science*, pages 224–236. Springer-Verlag, 1996.

[Kwa97] Matthew Kwan. *The Design of the ICE Encryption Algorithm*, volume ??? of *Lecture Notes in Computer Science*. Springer-Verlag, 1997. http://www.cs.mu.oz.au/~mkwan/ice/paper.html.

[Mat93] Mitsuru Matsui. *Linear Cryptanalysis Method for DES Cipher*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1993.

[NIS97] NIST. Advanced Encryption Standard Call, 1997. http://csrc.ncsl.nist.gov/encryption/aes/aes_home.htm.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In T. Helleseth, editor, *Advances in Cryptology - Eurocrypt'93*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer-Verlag, 1994.

[Pie92] J. Pieprzyk. Bent Permutations. In G. Mullen and P. Shiue, editors, *Proceedings of 1st International Conference on Finite Fields, Coding Theory, and Advances in Communications and Computing*, volume 141 of *Lecture Notes in Pure andApplied Mathematics*. Springer-Verlag, 1992.

[Sch96] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms and Source Code in C*. John Wiley and Sons, New York, 2nd edition, 1996.

[SF97] Kouichi Sakurai and Souichi Furuya. *Improving Linear Cryptanalysis of LOKI91 by Probabalistic Counting Method*, volume ??? of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[SK96] Bruce Schneier and John Kelsey. *Unbalanced Feistel Networks and Block Cipher Design*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer-Verlag, 1996.

[TSM94] Toshio Tokita, Tohru Sorimachi, and Mitsuru Matsui. *Linear cryptanalysis of LOKI and S2DES*, volume 917 of *Lecture Notes in Computer Science*, pages 293–306. Springer-Verlag, 1994.

[WN94] David J. Wheeler and Roger M. Needham. *TEA, a Tiny Encryption Algorithm*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer-Verlag, 1994. http://www.cl.cam.ac.uk/ftp/papers/djw-rmn/djw-rmn-tea.html.

[WT86] A. F. Webster and S. E. Tavares. *On the Design of S-Boxes*, volume 218 of *Lecture Notes in Computer Science*, pages 523–534. Springer-Verlag, 1986.

Further Information and Source Distributions

The latest version of this paper, as well as the reference sources in C and Java, and sample test data, may be be obtained from:

```
http://www.adfa.oz.au/~lpb/research/loki97
```

Appendix A - Log of Test Triple

The following log details the subkeys and round values computed for the sample test triple, along with the nonlinear function *f* values used to compute them.

nb. the subkey values are indexed from 0 (tracking the array indices used in the reference C and Java code) rather than 1 as was used in the description earlier.

```
  key: 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
plain: 000102030405060708090A0B0C0D0E0F
cipher: 75080E359F10FE640144B35C57128DAD

makeKey(0001020304050607,08090A0B0C0D0E0F,1011121314151617,18191A1B1C1D1E1F)
  SK[0]=ECB82110452BF90A; f=ECB92313412EFF0D
  SK[1]=592CD965E4168E33; f=5125D36EE81B803C
  SK[2]=1A8E5818A655138C; f=0A9F4A0BB240059B
  SK[3]=86E3263EBBBF339C; f=9EFA3C25A7A22D83
  SK[4]=AD6DFD04A887509B; f=41D5DC14EDACA991
  SK[5]=B3A56496E96B4F0D; f=EA89BDF30D7DC13E
  SK[6]=52A0073EA724A264; f=482E5F260171B1E8
  SK[7]=2FF1BF749D54136B; f=A912994A26EB20F7
  SK[8]=BD0BC040D3A54F28; f=10663D447B221FB3
  SK[9]=929220E7076443A8; f=21374471EE0F0CA5
  SK[10]=7FB7D2666220233A; f=2D17D558C504815E
  SK[11]=56FE853A068B8D2E; f=790F3A4E9BDF9E45
  SK[12]=AC1C43D92F7490B6; f=11178399FCD1DF9E
  SK[13]=FBE3491706663C44; f=697169F001027FEC
  SK[14]=E4AC5DE7F5220AB5; f=9B1B8F819702298F
  SK[15]=F10D78B02017847C; f=A7F3FD8A269C0952
  SK[16]=F4FFDEBD5E175312; f=58E39D647163C3A4
  SK[17]=4396B72AE367FF41; f=B875FE3DE501C305
  SK[18]=26ED088C13C3993F; f=C241556BE6E1938A
  SK[19]=C3468074E387B643; f=324BF8C4C390323F
  SK[20]=2752EDD11A129E73; f=D3AD336C4405CD61
  SK[21]=A46AACF6DD57D61F; f=E7FC1BDC3E30295E
  SK[22]=5F06EB99CFD8084F; f=79EBE315DC1B9170
  SK[23]=8252416503BE6B13; f=4114C111E039DD50
  SK[24]=EF7D17F4791630C3; f=C82FFA256304AEB0
  SK[25]=B5536290E13AAD94; f=1139CE663C6D7B8B
  SK[26]=D65073A787DCAF9A; f=8956983E4804A7D5
  SK[27]=3DB329BD5B9BF213; f=BFE168D858259900
  SK[28]=804E42039A6496DA; f=6F3355F7E372A619
  SK[29]=DB97B9E35223D540; f=6EC4DB73B31978D4
  SK[30]=B152C3DD7A6EE03F; f=6702B07AFDB24FA5
  SK[31]=176EECE0F5AA3E62; f=2ADDC55DAE31CC71
  SK[32]=F0B4C6DA31B841FC; f=70FA84D9ABDCD726
  SK[33]=3BDDEA965A9F612D; f=E04A537508BCB46D
  SK[34]=E03718A6FDC7901A; f=5165DB7B87A97025
  SK[35]=710587AB3E6A614F; f=666B6B4BCBC05F2D
  SK[36]=B0C6F115D3ECE6C2; f=407237CFE254A73E
  SK[37]=AF82DA2EF75F6924; f=945F30B8ADC00809
  SK[38]=AA5DE8BDB42A8BDB; f=4A6AF01B49ED1BC1
  SK[39]=50BB552A21F75E7D; f=21BED2811F9D3F32
  SK[40]=B8EB467438FF42E4; f=082DB761EB13A426
  SK[41]=936362030FA48C95; f=3CE1B82DF8FBE5B1
  SK[42]=E55434C694CE74CE; f=4F09DC7B20E4FF15
  SK[43]=BDA3575166DF26BC; f=ED18027B472878C1
  SK[44]=B779C086BDB9551E; f=0F9286F2854617FA
  SK[45]=1322E154E6746255; f=80418357E9D0EEC0
  SK[46]=3441894738B21D3D; f=D115BD81AC7C69F3
  SK[47]=F9539B20F3944405; f=44F0CC71954B62B9

blockEncrypt(000102030405060708090A0B0C0D0E0F)
  L[1]=0F4F8333F78E1AA5; R[1]=64CB99C8CB17F030; f(SK(1))=64CA9BCBCF12F637
  L[2]=9F54249E704272D9; R[2]=627C5CB2B47B02A8; f(SK(4))=6D33DF8143F5180D
  L[3]=722824322F44F434; R[3]=620CAAC8D2EB65DF; f(SK(7))=FD588E56A2A91706
  L[4]=4B9D50E9E0DB36B5; R[4]=70721A7E1E9ED1FB; f(SK(10))=025A3E4C31DA25CF
  L[5]=013ABC3F43356D66; R[5]=1D8AE37EA41C6DA7; f(SK(13))=5617B39744C75B12
  L[6]=522F1359A79BF164; R[6]=5413A42C303D8D5D; f(SK(16))=552918137308E03B
  L[7]=A2539A895E13C50F; R[7]=A6C0DE25F6AB614E; f(SK(19))=F4EFCD7C5130902A
  L[8]=CD7DCC81D7C1A280; R[8]=BBD666242D3B8733; f(SK(22))=1985FCAD7328423C
  L[9]=81A3F1C02E2E6790; R[9]=BCC9279136E34DEA; f(SK(25))=71B4EB10E122EF6A
  L[10]=D6140B31E4A3153D; R[10]=F3194A7408F52869; f(SK(28))=72BABBB426DB4FF9
  L[11]=9520D52BB51C4AA4; R[11]=C108DC9F95015A77; f(SK(31))=171CD7AE71A24F4A
  L[12]=6DEC4EE12E0B1CF3; R[12]=CF23C07BC2D506B7; f(SK(34))=5A03155077C94C13
  L[13]=2A489A4F4AEC7954; R[13]=2CE1F004CFC7122E; f(SK(37))=410DBEE5E1CC0EDD
  L[14]=1100A7320162FD40; R[14]=8FE94D86A97297D4; f(SK(40))=A5A1D7C9E39EEE80
  L[15]=2CB742D3FBFA61C0; R[15]=F4CE36E67D09E753; f(SK(43))=E5CE91D47C6B1A13
  L[16]=0144B35C57128DAD; R[16]=75080E359F10FE64; f(SK(46))=59BF4CE664EA9FA4
  = 75080E359F10FE640144B35C57128DAD

blockDecrypt(75080E359F10FE640144B35C57128DAD)
  L[1]=F4CE36E67D09E753; R[1]=2CB742D3FBFA61C0; f(SK(46))=59BF4CE664EA9FA4
  L[2]=8FE94D86A97297D4; R[2]=1100A7320162FD40; f(SK(43))=E5CE91D47C6B1A13
  L[3]=2CE1F004CFC7122E; R[3]=2A489A4F4AEC7954; f(SK(40))=A5A1D7C9E39EEE80
  L[4]=CF23C07BC2D506B7; R[4]=6DEC4EE12E0B1CF3; f(SK(37))=410DBEE5E1CC0EDD
  L[5]=C108DC9F95015A77; R[5]=9520D52BB51C4AA4; f(SK(34))=5A03155077C94C13
  L[6]=F3194A7408F52869; R[6]=D6140B31E4A3153D; f(SK(31))=171CD7AE71A24F4A
  L[7]=BCC9279136E34DEA; R[7]=81A3F1C02E2E6790; f(SK(28))=72BABBB426DB4FF9
  L[8]=BBD666242D3B8733; R[8]=CD7DCC81D7C1A280; f(SK(25))=71B4EB10E122EF6A
  L[9]=A6C0DE25F6AB614E; R[9]=A2539A895E13C50F; f(SK(22))=1985FCAD7328423C
  L[10]=5413A42C303D8D5D; R[10]=522F1359A79BF164; f(SK(19))=F4EFCD7C5130902A
  L[11]=1D8AE37EA41C6DA7; R[11]=013ABC3F43356D66; f(SK(16))=552918137308E03B
  L[12]=70721A7E1E9ED1FB; R[12]=4B9D50E9E0DB36B5; f(SK(13))=5617B39744C75B12
  L[13]=620CAAC8D2EB65DF; R[13]=722824322F44F434; f(SK(10))=025A3E4C31DA25CF
  L[14]=627C5CB2B47B02A8; R[14]=9F54249E704272D9; f(SK(7))=FD588E56A2A91706
  L[15]=64CB99C8CB17F030; R[15]=0F4F8333F78E1AA5; f(SK(4))=6D33DF8143F5180D
  L[16]=08090A0B0C0D0E0F; R[16]=0001020304050607; f(SK(1))=64CA9BCBCF12F637
  = 000102030405060708090A0B0C0D0E0F
```

.