Int. J. Contemp. Math. Sciences, Vol. 2, 2007, no. 29, 1401 - 1424

A Structure for Fast Data Encryption

Alireza Shafieinejad and Faramarz Hendessi

Dept. of Electrical and Computer Engineering
Isfahan University of Technology, Iran

T. Aaron Gulliver

Dept. of Electrical and Computer Enginering
University of Victoria, PO Box 3055, STN CSC
Victoria, BC Canada

Abstract

Fast Data Encryption (FDE) is a new symmetric block cipher which has a DES-like structure. FDE has been designed with an increased key length, increased key scheduling complexity and an improved round function which can be executed in parallel. FDE uses eight Substitution Boxes (S-boxes) in the round function to provide confusion. In this paper, we present the FDE structure and an algorithm to construct a set of strong S-boxes. Eight suitable S-boxes from this set are suggested for use with FDE.

Mathematics Subject Classification: 94A60

Keywords: Symmetric cryptographic algorithms, DES, FDE, S-box

1 Introduction

Fast Data Encryption (FDE) is a new cryptographic structure based on the principles of the Data Encryption Standard (DES). FDE is suitable for both hardware and/or software implementation [13, 14]. In this paper, we show that FDE is a strong and powerful encryption algorithm that can easily be implemented and is much stronger than DES.

In the next section, the FDE structure is introduced. First, the criteria for designing FDE are discussed. These criteria were chosen to retain the advantages of DES while removing weaknesses. Second, the functionality of FDE is considered. The details of each round of encryption are discussed. Third, the key schedule for generating subkeys from the main key is presented.

In Section 3, an algorithm is presented to construct strong S-boxes. The algorithm employs a bit-to-bit method to design S-boxes with high nonlinearity using the Strict Avalanche Criteria (SAC) [1, 4, 11, 15]. This algorithm has three steps. In the first step, strong 4-bit functions are created, and in the second step 4×4 S-boxes are created from the functions obtained in the first step. The third step generates 6×4 S-boxes from the S-boxes found in Step 2. The first and second steps of this algorithm are exhaustive, while the last step employs a random search because of the size of the search space. With this algorithm, several thousand strong S-boxes were created, and eight were chosen for use in the FDE structure.

The last Section summarizes the results presented in this paper.

2 Fast Data Encryption Structure

Fast Data Encryption (FDE) is a symmetric block cipher with a DES-like structure in which the two fundamental cryptographic operations, confusion and diffusion, are performed by substitution boxes and bit permutations, respectively, as in DES. FDE was designed to eliminate DES shortcomings (such as a short key length and a simple key generation algorithm) [2, 5]. In particular, the key length in FDE is 128 bits. A modified DES round structure is employed and permutations are performed on both halves of the data block. Since these operations are separate, the calculations can be done in parallel. In the FDE round function, eight 6×4 S-boxes are used (as in DES). As is well known, the DES S-boxes are not suitable for use in most encryption structures, as they are far from ideal [2]. In addition, the different round structure in FDE requires different S-box properties (with differing priorities). Thus we present new S-boxes and their design in this paper.

The FDE design criteria are given below.

2.1 FDE Design Criteria

The design of FDE takes into account the following criteria to create a block cipher to replace DES:

1. Improved round function
2. Increased key scheduling algorithm complexity
3. Increased key length (to 128 bits)
4. Improved S-boxes
5. Parallel computations in the round function

6. A symmetric round structure for efficient pipelined hardware implementation

2.2 Functional Structure of FDE

The odd and even FDE round functions are shown in Figs. 1 and 2, respectively. L_i and R_i are the first and last 32 bits, respectively, of the 64 bit data block in the i-th round. According to Figs. 1 and 2, the round operations are done in two stages. First the G or G' data block is XORed with the 32 bit partial keys X_{3i+1}, X_{3i+2} and X_{3i+3}. Then the substitution and permutation operations are done in F with the 16 bit keys Z_{2i+1} and Z_{2i+2}. Note that five partial keys from the 128 bit key are employed in each round.

The substitution and permutation operations are executed concurrently on each of the two halves of the data block. On the right, **S** and **P** are employed, and on the left, the inverse of these are used ($\mathbf{S}^{-1}$ and $\mathbf{P}^{-1}$). As with DES, **S** is a substitution network that consists of eight 6×4 balanced S-boxes that map a 32 bit input to a 32 bit output according to 16 bit keys **Z**. This is shown in Fig. 3. **P** is the same as in DES.

The key for **S** is obtained by XORing the 32 bit keys with the data block. This is used to select the S-boxes according to 16 bit keys (two bits of **Z** determine a 4×4 S-box from a 6×4 S-box).

The FDE structure shown in Fig. 4 consists of 16 rounds, and initial and final permutations. Note that there is no F function in the last round. The G, G', F, T and hl functions are defined as

$$
\begin{aligned}
&G_{X_{i+2},X_{i+1}X_i}(R,L) = (R \oplus X_i \oplus X_{i+2},\ L \oplus R \oplus X_i \oplus X_{i+1}) \\
&G'_{x_{i+2}X_{i+1}X_i}(R,L) = (R \oplus L \oplus X_i \oplus X_{i+1},\ L \oplus X_i \oplus X_{i+2}) \\
&F_{Z_{i+1},Z_i}(R,L) = (P(S_{Z_i}(R)),\ S^{-1}_{Z_{i+1}}(P^{-1}(L))) \\
&T(R,L) = (L,R) \quad T \circ T(R,L) = (R,L) \ \Rightarrow\ T^{-1} = T \\
&hl(r_{63}r_{62}r_{61}\ldots r_2r_1r_0) = (r_{63}r_{62}\ldots r_{33}r_{32}),\ (r_{31}r_{30}\ldots r_1r_0)
\end{aligned}
\tag{1}
$$

where $\oplus$ denotes modulo-2 addition (XOR).

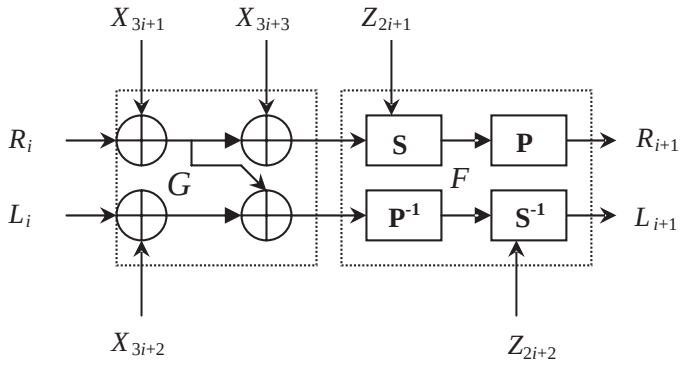

Figure 1: An even round in FDE.

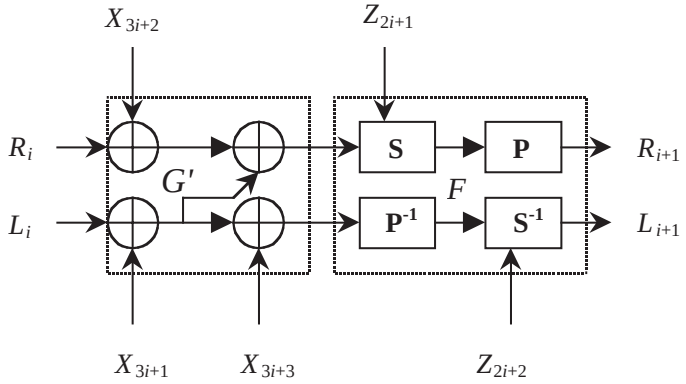

Figure 2: An odd round in FDE.

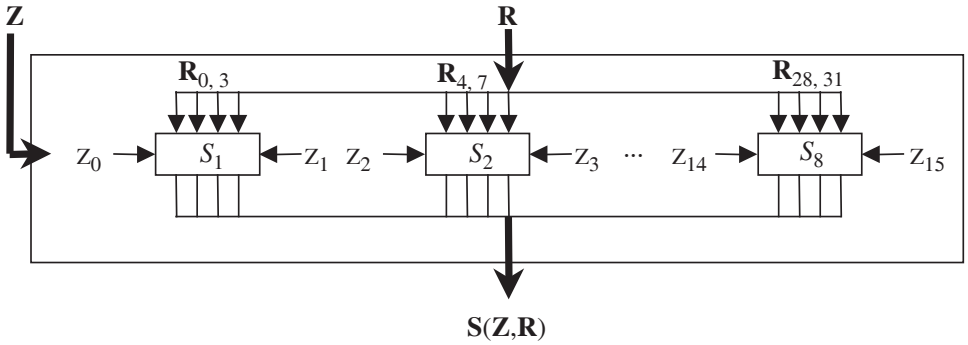

Figure 3: The FDE substitution network structure.

In FDE, T is a function that exchanges the two halves of a 64 bit block and is used after the initial permutation. Obviously the inverse of T is equal to itself. *hl* is a function that converts one 64 bit block into two 32 bit blocks. The inverse of *hl* converts two 32 bit blocks to one 64 bit block. FDE encryption with these definitions is then

$$\begin{aligned}&\text{FDE\_Encrypt} = IP^{-1} \circ hl^{-1} \circ G'_{X_{48},X_{47},X_{46}} \circ F_{Z_{30},Z_{29}} \circ G_{X_{45},X_{44},X_{43}} \circ \ldots \\ &F_{Z_4,Z_3} \circ G'_{X_6,X_5,X_4} \circ F_{Z_2,Z_1} \circ G_{X_3,X_2,X_1} \circ T \circ hl \circ IP(\text{Plaintext})\end{aligned} \tag{2}$$

where IP denotes initial permutation. For decryption, we reverse both sides of (2) giving

$$\begin{aligned}&\text{FDE\_Decrypt} = IP^{-1} \circ hl^{-1} \circ {G'}^{-1}_{X_3,X_2,X_1} \circ F^{-1}_{Z_2,Z_1} \circ G^{-1}_{X_6,X_5,X_{44}} \circ F^{-1}_{Z_4,Z_3} \circ \\ &\ldots \circ G^{-1}_{X_{45},X_{44},X_{43}} \circ F^{-1}_{Z_{30},Z_{29}} \circ G^{-1}_{X_{48},X_{47},X_{46}} \circ hl \circ IP(\text{Ciphertext})\end{aligned} \tag{3}$$

The inverses of F, G and G' are

$$\begin{aligned}F^{-1}_{Z_{i+1},Z_i}(R,L) &= (S^{-1}_{Z_i}(P^{-1}(R)), P(S_{Z_{i+1}}(L)) \\ G^{-1}_{X_{i+2},X_{i+1},X_i}(R,L) &= G'_{X_i,X_{i+1},X_{i+2}}(R,L) \\ {G'}^{-1}_{X_{i+2},X_{i+1},X_i}(R,L) &= G_{X_i,X_{i+1},X_{i+2}}(R,L)\end{aligned} \tag{4}$$

respectively, because

$$\begin{aligned}G_{X_{i+2},X_{i+1},X} \circ G'_{X_i,X_{i+1},X_{i+2}}(R,L) &= (R,L) \\ G'_{X_{i+2},X_{i+1},X_i} \circ G_{X_i,X_{i+1},X_{i+2}}(R,L) &= (R,L)\end{aligned} \tag{5}$$

Thus by substituting (4) in (3) we obtain

$$\begin{aligned}&\text{FDE\_Decrypt} = IP^{-1} \circ hl^{-1} \circ T \circ G_{X_1,X_2,X_3} \circ F^{-1}_{Z_2,Z_1} \circ G'_{X_4,X_5,X_6} \circ F^{-1}_{Z_4,Z_3} \\ &\circ \ldots \circ G'_{X_{43},X_{44},X_{45}} \circ F^{-1}_{Z_{30},Z_{29}} \circ G_{X_{46},X_{47},X_{48}} \circ hl \circ IP(\text{Ciphertext})\end{aligned} \tag{6}$$

Considering the T function, we have

$$\begin{aligned}T \circ G'_{X_i,X_{i+1},X_{i+2}}(R,L) &= G_{X_i,X_{i+1},X_{i+2}} \circ T(R,L) \\ T \circ G_{X_i,X_{i+1},X_{i+2}}(R,L) &= G'_{X_i,X_{i+1},X_{i+2}} \circ T(R,L) \\ T \circ F^{-1}_{Z_{i+1},Z_I}(R,L) &= f_{z_i,z_{i+1}} \circ T(R,L)\end{aligned} \tag{7}$$

This means that T operating on the left of G', G and F^{-1} is equivalent to G, G' and F, respectively, with T moving to the right side. Thus decryption

in (2) can be transformed to

$$\begin{aligned}
&\text{FDE_Decrypt}\\
&= IP^{-1} \circ hl^{-1} \circ T \circ G_{X_1,X_2,X_3} \circ F^{-1}_{Z_2,Z_1} \circ G'_{X_4,X_5,X_6} \circ F^{-1}_{Z_4,Z_3} \circ \ldots \circ\\
&\quad G'_{X_{43},X_{44},X_{45}} \circ F^{-1}_{Z_{30},Z_{29}} \circ G_{X_{48},X_{47},X_{46}} \circ hl \circ IP(\text{Ciphertext})\\
&= IP^{-1} \circ hl^{-1} \circ G'_{X_1,X_2,X_3} \circ T \circ F^{-1}_{Z_2,Z_1} \circ G'_{X_4,X_5,X_6} \circ F^{-1}_{Z_4,Z_3} \circ \ldots \circ\\
&\quad G'_{X_{45},X_{44},X_{43}} \circ F^{-1}_{Z_{30},Z_{29}} \circ G_{X_{48},X_{47},X_{46}} \circ hl \circ IP(\text{Ciphertext})\\
&= IP^{-1} \circ hl^{-1} \circ G'_{X_1,X_2,X_3} \circ F_{Z_1,Z_2} \circ T \circ G'_{X_4,X_5,X_6} \circ F^{-1}_{Z_4,Z_3} \circ \ldots \circ\\
&\quad G'_{X_{45},X_{44},X_{43}} \circ F^{-1}_{Z_{30},Z_{29}} \circ G_{X_{48},X_{47},X_{46}} \circ hl \circ IP(\text{Ciphertext})\\
&= IP^{-1} \circ hl^{-1} \circ G'_{X_1,X_2,X_3} \circ F_{Z_1,Z_2} \circ G_{X_4,X_5,X_6} \circ T \circ F^{-1}_{Z_4,Z_3} \circ \ldots \circ\\
&\quad G'_{X_{45},X_{44},X_{43}} \circ F^{-1}_{Z_{30},Z_{29}} \circ G_{X_{48},X_{47},X_{46}} \circ hl \circ IP(\text{Ciphertext})\\
&= \cdots\\
&= IP^{-1} \circ hl^{-1} \circ G'_{X_1,X_2,X_3} \circ F_{Z_1,Z_2} \circ G_{X_4,X_5,X_6} \circ F_{Z_3,Z_4} \circ \ldots \circ\\
&\quad G_{X_{43},X_{44},X_{45}} \circ F_{Z_{29},Z_{30}} \circ G'_{X_{46},X_{47},X_{48}} \circ hl \circ IP(\text{Ciphertext}) \qquad (8)
\end{aligned}$$

This shows that decryption is the same as encryption, except that the order of the partial keys is reversed. This was a consideration in choosing 16 rounds for FDE implementation. Since G and G' are duals, and encryption starts with G, encryption should end with G'. Thus G' should occur after an odd number of F functions, which implies that the number of FDE rounds be even (as the last round has no F).

$$\begin{aligned}
\text{FDE_Decrypt} = {} & IP^{-1} \circ G'_{X_1,X_2,X_3} \circ F_{Z_1,Z_2} \circ G_{X_4,X_5,X_6} \circ F_{Z_3,Z_4} \circ \ldots \circ\\
& G'_{X_{43},X_{44},X_{45}} \circ F_{Z_{29},Z_{30}} \circ G'_{X_{46},X_{47},X_{48}} \circ IP(\text{Ciphertext}) \qquad (9)
\end{aligned}$$

2.3 The FDE Key Scheduling Structure

Categorizing the keys in each round, we have the following vectors

$$\begin{aligned}
&U_{2i+1} = (X_{3i+1},\ X_{3i+2}) && i = 0,\ 1,\ \ldots,\ 7\\
&U_{2i+2} = (X_{3i+1},\ Z_{2i+1},\ Z_{2i+2}) && i = 0,\ 1,\ \ldots,\ 6\\
&U_{30-2i} = (X_{48-3i},\ X_{47-3i}) && i = 0,\ 1,\ \ldots,\ 7\\
&U_{29-2i} = (X_{3i+1},\ Z_{30-2i},\ Z_{29-2i} && i = 0,\ 1,\ \ldots,\ 6\\
&V = (X_{24},\ Z_{15},\ Z_{16},\ X_{25})
\end{aligned} \qquad (10)$$

Each U_i is a 64 bit vector and V is a 96 bit vector. With this definition, FDE is simply

$$\text{Ciphertext} = \text{FDE\_Encrypt}_{U_1,U_2,U_3,\ldots U_{15},V,U_{16},\ldots U_{30}}(\text{Plaintext}) \qquad (11)$$

The key generation algorithm is shown below. U is the 128-bit main key, and X and Y are sequentially the first and last 32 bits of U, respectively.

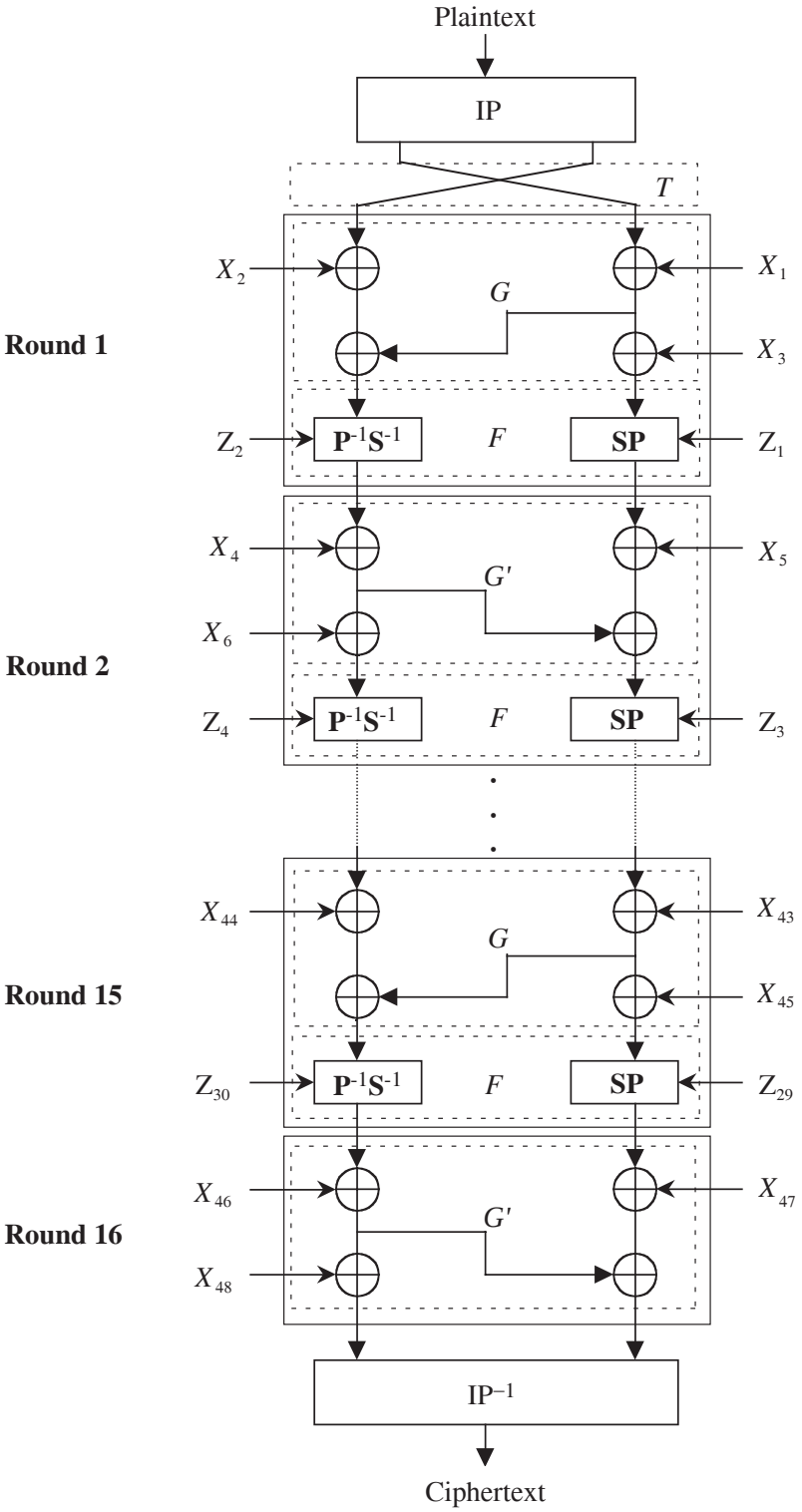

Figure 4: The FDE structure.

$$
\begin{aligned}
V &= \text{Choose\_96bit}(U) \\
U_1 &= \text{FDE\_Encrypt}_{Y,Y,\dots,Y,V,Y,\dots,Y}(X) \\
U_2 &= \text{FDE\_Encrypt}_{U_1,Y,\dots,Y,V,Y,\dots,Y}(X) \\
U_3 &= \text{FDE\_Encrypt}_{U_1,U_2,Y,\dots,Y,V,Y,\dots,Y}(X) \\
&\vdots \\
U_{30} &= \text{FDE\_Encrypt}_{U_1,U_2,\dots,U_{15}V,U_{16},\dots,U_{29},Y}(X)
\end{aligned} \tag{12}
$$

Choose_96bit(U) is a selection function which chooses 96 bits from 128 bits (U) for the partial key V. As encryption is a complicated logical operation, the relationship between partial keys will also be complicated. This complexity can be considered an advantage since in practice the subkeys can be considered independent in different attacks. Another advantage of this key generation algorithm is an increase in the Cost of Exhaustive Search (CES) attack complexity by a factor of 31 over that for the simple DES key generator [13]. One minor drawback of this key generation algorithm is that it requires 128×16 bits (256 bytes) of memory to store the partial keys. Another is that 30 FDE encryptions are required (to create the partial keys related to the new main key), to start encryption with a new key.

3 Designing Strong S-boxes

3.1 Definitions and Preliminaries

An $n \times m$ S-box S performs a mapping of the form $S : \{0,\ 1\}^n \rightarrow \{0,\ 1\}^m$. S can be represented as 2^n m-bit numbers $r_0, r_1, \dots, r_{2^n-1}$ or a $2^n \times m$ binary matrix. In this form these numbers are called S-box rows and each m-bit number gives the S-box output for each of the 2^n possible inputs ($S(x) = r_x,\ 0 \leq x < 2^n - 1$). This is denoted as the binary matrix representation.

S can also be considered as m functions: $S(x) = [f_{m-1}(x),\ f_{m-2}(x),\ \dots,\ f_0(x)]$, where each f_i is a Boolean function. In this form these functions are called S-box columns [10]. Both of these definitions will be used in this section. The linear combination of two functions is defined as

$$(f \oplus g)(x) = f(x) \oplus g(x)$$

L_n is the set of $\{0,\ 1\}^n \rightarrow \{0,\ 1\}$ linear functions. f is linear if it can be written as the multiplication of a constant vector and an input vector

$$f \in L_n \Leftrightarrow f : \{0,\ 1\}^n \rightarrow \{0,\ 1\}\ \exists\ a \in \{0,\ 1\}^n : f(x) = a \cdot x$$

where multiplication is defined as

$$a \cdot x = a_{n-1}x_{n-1} \oplus \ldots \oplus a_0 x_0$$

A_n denotes the set of affine functions mapping $f : \{0,\ 1\}^n \rightarrow \{0,\ 1\}$. f is an affine mapping if

$$f \in A_n \Leftrightarrow f : \{0,\ 1\}^n \rightarrow \{0,\ 1\}\ \exists\ a \in \{0,\ 1\}^n \wedge b \in \{0,\ 1\} : f(x) = a \cdot x \oplus b$$

These definitions will be used below.

Walsh Transform:

The Walsh transform of a function $f : \{0,\ 1\}^n \rightarrow \{0,\ 1\}$ is defined as

$$W(f)(w) = \sum_{x=0}^{2^n-1} (-1)^{f(x)+w\cdot x}$$

The set of bent functions $f : \{0,\ 1\}^n \rightarrow \{0,\ 1\}$ with n even, is defined as

$$W(f)(w) = \pm 2^{n/2} \forall w \in \{0,\ 1\}^n$$

Nonlinearity:

The nonlinearity of a function $f : \{0,\ 1\}^n \rightarrow \{0,\ 1\}$ is defined as

$$nl(f) = \min_{l \in A_n}(\mathrm{wt}(f \oplus l))$$

where wt() denotes the minimum Hamming weight of the function, defined as the least Hamming distance between the function and an affine function. The Hamming weight of an n-bit sequence is equal to the number of ones in the sequence.

It can be shown that the above definition is equivalent to the following expression employing the Walsh transform (which is more suitable for calculation) [12]

$$nl(f) = 2^{n-2} - \frac{1}{2} \max_{w \in \{0,1\}^n} |W(f)(w)|$$

The nonlinearity of an S-box S is defined as

$$nl(S) = \min_{f \in C}(nl(f))$$

where C is the set of all nontrivial linear combinations of the columns of S.

Balance:

The function $f : \{0,\ 1\}^n \rightarrow \{0,\ 1\}$ is called balanced (bidirectional) if the number of zero and ones in its output is equal when the input accepts

all possible values. This means that $\mathrm{wt}(f) = 2^{n-1}$. In general, the function $f : \{0,\ 1\}^n \to \{0,\ 1\}^m$, $n \geq m$ is bidirectional if each output bit of $y = f(x) \in \{0,\ 1\}^m$ is repeated an equal number of times (2^{n-m}), for all values of x in the domain. Also, an S-box is balanced if its mapping is defined by a bidirectional function.

Inverse:

The inverse of a balanced S-box $S : \{0,\ 1\}^n \to \{0,\ 1\}^m$, denoted S^{-1}, is defined by

$$\forall x \in \{0,\ 1\}^m : S^{-1} \circ S(x) = x$$

***XOR* table:**

The XOR table entry for an S-box with $\alpha,\ \beta$ for which $\alpha \in \{0,\ 1\}^n \backslash \{\mathbf{0}\},\ \beta \in \{0,\ 1\}^m$ is defined as [10]

$$XOR(\alpha,\ \beta) = \#\{x \in \{0,\ 1\}^n : S(x) \oplus S(s \oplus \alpha) = \beta\}$$

The table entry for $\alpha,\ \beta$ denotes the number of input pairs with difference α that generate a difference of β in the output. For an S-box, the XOR value is defined as the maximum value in the XOR table

$$XOR(S) = \max_{\alpha,\beta}(XOR(\alpha,\ \beta))$$

Let Nz denote the number of non-zero values in the XOR table. The mean and variance of the XOR table can then be defined as

$$Nz_{\mathrm{avg}}(S) = \frac{Nz}{2^n \times 2^m} \qquad \sigma = \sqrt{\sum_{a=0}^{2^n-1} \sum_{b=0}^{2^m-1} (XOR(a,b) - 2^{n-m})^2 / 2^{n+m}}$$

Dynamic distance:

The dynamic distance of order j of a function $f : \{0,\ 1\}^n \to \{0,\ 1\}$ is defined as [10]

$$DD_j(f) = \max_{\substack{d \in \{0,1\}^n \\ 1 \leq \mathrm{wt}(d) \leq j}} \frac{1}{2} \left| 2^{n-1} - \sum_{x=0}^{2^n-1} f(x) \oplus f(x \oplus d) \right|$$

This distance can be used to quantify dynamic properties such as the strict avalanche criterion and bit independence criterion.

Strict avalanche criterion:

A function $f : \{0,\ 1\}^n \to \{0,\ 1\}$ satisfies the Strict Avalanche Criterion (SAC) if for each input value change, each output bit changes with probability 1/2 [10]. This means that $DD_1(f) = 0$. This is related to the completion of

an encryptor which means that each output bit is related to all input bits (key and plaintext). The SAC for a block cipher means that modifying an input bit causes on average half of the output bits change. The distance to SAC is defined as [10]

$$DSAC(f) = DD_1(f)$$

Bit independence criterion:

For an S-box $S : \{0,\ 1\}^n \rightarrow \{0,\ 1\}$, output bits are independent if for a change in an input bit the resulting correlation between two output bits is zero. The distance Bit Independence Criterion (BIC) is defined as [10]

$$DBIC(S) = \max_{\substack{c\in\{0,1\}^n \\ \mathrm{wt}(c)=2}} (DD_1(M \cdot c))$$

where M is a binary matrix corresponding to S (representation of S as a $2^n \times m$ binary matrix), and matrix multiplication is done with addition modulo-2. A lower BIC means the outputs due to changes in the inputs are more independent.

3.2 S-box Design Criteria

In this paper, the goal is to find S-boxes for which S and S^{-1} have the following properties:

- High nonlinearity values.
- The XOR table has a proper distribution and $XOR(S)$ is small.
- Satisfy the SAC.
- Output bits change independently (satisfy the BIC).

The first property provides resistance against linear attacks. If the S-box columns are close to a linear function, the S-box mapping will be near linear and as a result the encryption process can easily be defined by some linear equations. The second property will aid in protection against differential cryptanalysis [3] (by proper we mean that the table has close to a constant distribution so that σ is small [10]). This property is necessary (but not sufficient), to ensure the r-round characteristic [7, 8] with high probability [3]. If an S-box does not satisfy the SAC, some output bits will be dependent only on some input bits. It is possible to use this dependency in a selected text attack if enough cipher text and plaintext can be obtained. This is useful in key space searches. The second to fourth properties ensure that the S-box has good dynamic properties [4].

The inverse of a balanced 6×4 S-box is obtained by inverting each of the 4×4 S-boxes. In the FDE round functions (encryption and decryption), S^{-1} is employed, hence the above properties must hold for each S-box and its inverse. This is true for most block ciphers which use bidirectional S-boxes.

3.3 The bit-to-bit method of balanced S-box construction

A balanced $m \times m$ S-box is a permutation of 0 to $2^m - 1$, hence the total number of $m \times m$ balanced S-boxes is equal to the number of mappings of $(0,\ 1,\ \ldots\ 2^m - 1)$, which is $(2^m)!$ For 4×4 S-boxes ($m = 4$), this number is $N_{4\times4} = 16! = 2 \times 10^{13}$. For 6×4 S-boxes composed of four 4×4 balanced S-boxes, the number of possible S-boxes is

$$N_{6\times4} = \binom{N_{4\times4}}{4} \approx N_{4\times4}^4/24 \approx 6 \times 10^{39}$$

This shows that it is infeasible to search the complete space to find the optimal S-boxes. A random search of this space will result in good S-boxes with low probability since the solution space is small (given the S-box properties being considered). Hence, it is necessary to limit the search in a controlled way.

The first method we consider to find an $m \times m$ balanced S-box is to generate M functions, each of which is an S-box column. In other words, choose m distinct m bit functions to create a permutation of $0,\ 1,\ \ldots,\ 2^m - 1$. This is called a bit-to-bit design and is based on the idea that a strong $S : \{0,\ 1\}^m \rightarrow \{0,\ 1\}^m$ map is composed of m strong m-bit functions. In the bit-to-bit method we first create a set of N m-bit functions $\Delta = \{f_1,\ f_2,\ \ldots,\ f_N\}$. Then m functions are selected from this set such that $F = \{f'_1,\ f'_2\ \ldots f'_m\}$ is a permutation.

This method was first proposed by Webster [15], and then by Tavares and Adams [1]. Dawson also used this method to create 12,000 S-boxes with good information theoretic properties [4]. O'Connor analyzed this algorithm both theoretically and by simulation [11]. He showed that it is infeasible for $m > 6$. He obtained a lower bound on the size of Δ such that the probability of having an m bit permutation is greater than $1/2$

$$N_m^* > e^{\frac{2^m}{m} - m}$$

Table 1 shows the exponential growth of N_m^* as m varies from 4 to 10. O'Connor also proved that finding an m-bit mapping in Δ is an NP-hard problem, so the algorithm complexity grows exponentially with the size of Δ. Hence, although this S-box design method is better than an exhaustive search, it is still too complex for implementation for practical values of m.

Table 1: Values of N_m^* when Δ has a Map with $p(N_m^*,\ m) > 1/2$

m	N_m^*
4	2^4
5	2^7
6	2^{13}
7	2^{24}
8	2^{45}
9	2^{80}
10	2^{133}

3.4 A New S-box Construction Algorithm

Our new S-box construction algorithm is performed in three steps.

1. Create a set Δ from the functions $f : \{0,\ 1\}^4 \rightarrow \{0,\ 1\}$ which satisfy the SAC.

2. Create 4×4 S-boxes from Δ such that the S-box and its inverse have good properties as given above.

3. Create 6×4 S-boxes which satisfy the criteria from the 4×4 S-boxes in step 2.

These steps are described in detail below.

3.4.1 Creating a set of SAC functions with maximum nonlinearity

It can be shown that the following inequality is true for each n bit function $f : \{0,\ 1\}^n \rightarrow \{0,\ 1\}$

$$nl(f) \leq 2^{n-1} - 2^{\frac{1}{2}n-1}$$

Equality holds only when the function is bent [12]. Bent functions have maximum nonlinearity and also satisfy the SAC [9]. However, bent functions are not balanced (their Hamming distance is $2^{n-1} \pm 2^{\frac{1}{2}n-1}$). Therefore, bent functions are not suitable for inclusion in the set Δ. Nonlinearity for balanced functions is less than that for bent functions. Seberry *et. al* [12] obtained upper and lower bounds on the maximum nonlinearity for a balanced function, and these values are given in Table 2.

The maximum nonlinearity upper bound for V_{2k} is $2^{2k-1} - 2^{k-1} - 2$. The maximum nonlinearity lower bound means that balanced functions can be created which have this nonlinearity, which is $2^{2k-1} - 2^k$ for V_{2k}. Several methods were suggested in [12] to generate functions with this nonlinearity.

Table 2: Upper and Lower Bounds on Nonlinearity

Vector space	V_4	V_6	V_8	V_{10}	V_{12}	V_{14}
Upper bound	4	26	118	494	2014	8126
Lower bound	4	24	112	480	1984	8064

From Table 2, the maximum nonlinearity for V_4 is 4. The lower bound for V_6 is 24, while the upper bound is 26 (the true value can be obtained by checking all 2^{64} 6-bit functions).

In our design, the set Δ was created by a computer search from V_4 balanced functions which satisfy the SAC and have the maximum nonlinearity for a balanced function ($nl(f) = 4$). The number of functions satisfying these conditions is 1368.

3.4.2 Creating 4×4 S-boxes from Δ

Using the functions generated in the first step, and a bit-to-bit method with a branch and bound strategy [11], we created a set of strong 4×4 S-boxes. In this step all groups of 4 members of Δ that produced a permutation were checked. The depth of the tree search was four. In each level of the tree search, one S-box column was selected.

The number of S-boxes that can be generated from Δ is very large, as shown by the following approximation

$$\text{Number of S-boxes} \approx \frac{\binom{N}{4}}{2^4} \approx \frac{N^4}{4! \times 2^4}$$

which is $O(N^4)$, and with $N = 1368$ is a very large number. Because of memory restrictions, it is not feasible to store all of these S-boxes. Hence a Cost function was used to reject some of them. In the design process, there were cases of two S-boxes in which one S-box was better in terms of one property while the other S-box was better in terms of another. A Cost function determines which S-box is better and should be kept. The definition of this function is important as it shows which properties are more important in the design. In this paper, the Cost function is a linear combination of nonlinearity, output bit independency and the SAC for each column

$$\begin{aligned}\text{Cost}(S) = {} & nl(S) + nl(S^{-1}) - DBIC(S) - DBIC(S^{-1}) \\ & - \sum_{i=1}^{4} (DSAC(S \cdot f_i) + DSAC(S^{-1} \cdot f_i)).\end{aligned}$$

The maximum value of this function corresponds to the maximum S-box nonlinearity, and the shortest distances to SAC and BIC. The details of this function are given in the following pseudo-code

Find_S-box 4×4 ***(*** Δ ***set of Strong functions)*** {

$\Delta = \{f_1,\ f_2,\ \ldots,\ f_N\}$
$Cost\_min = 8$
for $i = 1$ *to* $n - 3$ *do*
  for $j = i + 1$ *to* $n - 2$ *do*
    if (f_i, f_j) *is a partial permutation then*
      for $k = j + 1$ *to* $n - 1$ *do*
        if (f_i, f_j, f_k) *is a partial permutation then*
          for $m = k + 1$ *to* n *do*
            if (f_i, f_j, f_k, f_m) *is a permutation then*
              $s = (f_i, f_j, f_k, f_m)$;
              if $(\mathrm{Cost}(s) \geq Cost\_min)$ *then*
                Output and Save s;
                $Cost\_min = \mathrm{Cost}(s)$
}

The search of all possible combinations is done in four loops. At the beginning of the second, third and fourth *for* loops, a restriction is placed on the search based on a partial permutation of the functions selected up to that level. A k-tuple $(f_1, f_2, \ldots, f_k)$ is a partial permutation if the k-bit function defined by the concatenation of f_1 to f_k is a balanced mapping. This ensures that the given combination of functions creates a partial permutation which can be converted to a complete permutation by the proper selection of additional functions.

The *if* instruction within the last *for* loop checks the S-box Cost function. If this value is greater than or equal to *Cost_min*, the S-box is saved and *Cost_min* is updated.

If (f_i, f_j, f_k, f_m) is a balanced mapping, each permutation of the elements will also be balanced, giving $4! = 24$ equivalent solutions. Hence in our algorithm each *for* loop index is greater than the previous loop index to consider only one of these possibilities. This significantly reduces the execution time and memory requirements.

Execution of this algorithm on a Pentium-III (MMX 550 MHZ) machine took approximately 50 hours. The *Cost_min* value was initially 8 but quickly increased to 12 once some S-boxes were obtained. The number of S-boxes with

Cost value 12 is 1535. The properties of these S-boxes are

$$\begin{aligned} &nl(S) = nl(S^{-1}) = 4 \\ &DBIC(S) = DBIC(S^{-1} = 2 \\ &DSAC(S \cdot f_i) = DSAC(S^{-1}) \cdot f_i) = 0 \qquad \text{for } i = 1,\ 2,\ 3,\ 4 \end{aligned}$$

Thus for the best 4×4 S-boxes, the nonlinearity is maximum, the SAC is satisfied for all S-box columns, but the BIC is not the best as the distance is 2. The XOR table distribution values for these S-boxes have

$$\begin{aligned} &nz_{\text{avg}}(S) = nz_{\text{avg}}(S^{-1}) = 41\% \\ &\sigma(S) = \sigma(S^{-1}) = 1.53 \end{aligned}$$

All S-boxes with Cost value less than 12 were discarded at this point.

3.4.3 Creating 6×4 S-boxes from 4×4 S-boxes

Since four 4×4 S-boxes are required to create a 6×4 S-box, we select four 4×4 S-boxes from those obtained in the previous step and construct the corresponding 6×4 S-box. We have

$$N_{6\times4} = \frac{24N_{4\times4} \times 24(N_{4\times4} - 1) \times 24(N_{4\times4} - 2) \times 24(N_{4\times4} - 3)}{4!} \approx 24^3 \times N_{4\times4}^4$$
$$N_{4\times4} = 1535 \Rightarrow N_{6\times4} \approx 7.6 \times 10^{16}$$

If checking one condition takes 10 nanoseconds, it would take 24 years to check all conditions, thus we are forced to do a random selection. The algorithm pseudo-code for this is given below.

Find_S-box_ 6×4 ***(A : set of strong*** 4×4 ***S-boxes)*** **{**

$A = \{f_0, f_1, \ldots, f_n\}$// set of 4×4 S-boxes generated in 2nd stage

while (number of 6×4 *S-boxes <* MAX*)*{

randomly select 4 element of $A : f_a, f_b, f_c, f_d$

where $0 \le a < b < c < d \le n$

$f_\alpha = \text{select\_random\_ permutation}(f_a)$;

$f_\beta = \text{select\_random\_ permutation}(f_b)$;

$f_\gamma = \text{select\_random\_ permutation}(f_c)$;

$f_\lambda = \text{select\_random\_ permutation}(f_d)$;

$S_{6\times4} = (f_\alpha, f_\beta, f_\gamma, f_\lambda)$

if $\sum_{i=1}^{4}(DSAC(S \cdot f_i) + DSAC(S^{-1} \cdot f_i)) > T\_Dsac$ *continue*;

if $nl(S) < T\_nl$ or $nl(S^{-1}) < T\_nl$ *continue*;

if $DBIC(S) + DBIC(S^{-1}) > T\_Dbic$ *continue*;

if $\text{Cost}(S) \ge Cost\_min$ *Output* S;

```
    }
}
```

First four 4×4 S-boxes are selected at random, and a random permutation is applied to their columns (select_random_permutation()). Then the properties of the corresponding 6×4 S-box are checked. The aim is to find S-boxes with Cost functions that have *Cost_min* and SAC, nonlinearity and BIC values not less than T_Dsac, T_nl and T_Dbic. As the calculation of the Cost function is time consuming, these bounds are checked separately to prevent unnecessary computations.

3.5 Results and Final S-box Selection

The initial values were selected as:

$$T\_nl = 20, T\_Dsac = 16, T\_Dbic = 10, Cost\_min = 24, MAX = 2200$$

It took 40 hours to produce 2200 S-boxes. Values of the Cost function ranged from 24 to 32. Checking these S-boxes shows there is a tradeoff between the properties. When one value becomes optimum, others are typically not optimum. For example, five S-boxes with a Cost function of 32 were obtained. These S-boxes (S1-FDE is one of them) have the following properties:

$$\sum_{i=1}^{4}(DSAC(S \cdot f_i) + DSAC(S^{-1} \cdot f_i)) = 0$$

$$DBIC(S) = DBIC(S^{-1}) = 4$$

$$nl(S) = nl(S^{-1}) = 20 \qquad 20 \le nl(S \cdot f_i),\ nl(S^{-1} \cdot f_i) \le 24$$

$$nz_{\text{avg}}(S) = 83.84 \qquad XOR(S) = 16,\ 18,\ 20$$

This shows that this group satisfies the SAC, but output bit independence is not achieved completely (as was the case with the 4×4 S-boxes). The DBIC(S) value was the least value among the S-boxes, so it seems likely that this is the minimum possible value. The nonlinearity of this group is 20 which is not the maximum achieved. Another group has a nonlinearity of 22 (FDE-S2 is in this group). The properties of this group are:

$$\sum_{i=1}^{4}(DSAC(S \cdot f_i) + DSAC(S^{-1} \cdot f_i)) = 8$$

$$DBIC(S) = DBIC(S^{-1}) = 4$$

$$nl(S) = nl(S^{-1}) = 22 \qquad 22 \le nl(S \cdot f_i),\ nl(S^{-1} \cdot f)_i \le 24$$

$$nz_{\text{avg}}(S) = 87 \qquad XOR(S) = 14$$

Note that the XOR table distribution of this group is also better, but the SAC value is worse. The S-box columns in this case have distance 8 from the SAC.

In fact among all 2200 S-boxes, the best SAC value and a nonlinearity of 22 were never achieved simultaneously. Also a nonlinearity of 24 was never achieved, thus it is likely not possible. To have a nonlinearity of 24, the combination of all S-box columns must have a nonlinearity of 24. For example, the following S-box

$$\begin{aligned} S = \{&8,1,3,5,6,9,2,10,14,13,12,4,15,11,0,7 \\ &6,0,9,8,14,15,12,2,13,4,11,1,10,5,3,7 \\ &11,1,12,9,4,0,6,5,10,14,2,13,8,3,7,15 \\ &8,11,12,4,6,10,13,2,7,15,14,9,5,3,1,0\} \end{aligned}$$

has these values.

All columns of S and S^{-1} have a nonlinearity of 24, but the S-box nonlinearity is 22 because some linear combination of the columns of S have a nonlinearity of 22.

The maximum value achieved for non-zero values in the XOR table is 88, and the minimum value of $XOR(S)$ is 12 (FDE-S6 is one of these). In this group, the better XOR table distribution goes with poorer SAC and BIC properties, as shown below

$$\sum_{i=1}^{4}(DSAC(S \cdot f_i) + DSAC(S^{-1} \cdot f_i)) = 10$$

$$DBIC(S) = DBIC(S^{-1}) = 6$$

$$nl(S) = nl(S^{-1}) = 22 \qquad 22 \leq nl(S \cdot f_i),\ nl(S^{-1} \cdot f_i) \leq 24$$

$$nz_{\text{avg}}(S) = 88 \qquad XOR(S) = 12$$

In general, it is difficult to say which S-box is better among these groups, as this depends on the encryption algorithm structure and the possible attacks against it. From the 2200 S-boxes, eight were selected for use in the FDE structure. These S-boxes are given in Appendix A, and are denoted as FDE-S1 to FDE-S8. In the selection of these S-boxes, the following properties were considered a priority:

1. Maximum Cost (nonlinearity and SAC properties for each S-box column and BIC)
2. A good XOR table
3. Correlation coefficients for column pairs having the minimum value

3.6 Comparison with DES S-boxes

The following properties were used in the design of the DES S-boxes [6]:

1. None of the S-boxes is a linear or affine function of the inputs.

2. Modification of one input bit causes at least two changes in the output.
3. S-boxes are selected so that for a constant input bit, the difference in the number of 0 and 1 outputs is minimized.
4. $S(x)$ and $S(x \oplus 001100)$ differ in at least in two bits.
5. $\forall e,\ f : S(x) \neq S(x \oplus (11ef00))$
6. $\forall a,\ b,\ c,\ d,\ abcd \neq 0000 \Rightarrow: S(x) \neq S(x \oplus (0abcd0))$

The first property ensures the nonlinearity of the S-boxes, which is also considered in the design in this paper. The second property concerns the SAC and the third considers the independence of the SAC variables. Both of these are encompassed in the design given here. The sixth property ensures that each DES 6×4 S-box is constructed from four balanced 4×4 S-boxes. In fact the first and sixth bit of these 4×4 S-boxes selects the function. Each 4×4 S-box denotes a permutation of $(0,\ 1,\ \ldots,\ 15)$. In DES, the first and sixth input bits of the S-box are key bits. In the DES algorithm the balanced property is not used and in fact DES encryption will work regardless of the S-box mapping. The fourth and fifth properties result in an interesting characteristic of DES S-boxes as given in the following theorem [6].

Theorem 3.1 *If two inputs to the F-function result in equal outputs, the two inputs must differ in the inputs to at least three neighboring S-boxes.*

This property provides DES with some resistance against differential attacks, in particular, DES has a low probability of 2-round iterative characteristics. This implies that the DES designers were aware of differential attacks. Tables 3 to 6 compare the FDE and DES S-boxes in terms of nonlinearity, SAC, output bit independence and XOR table distribution, respectively, which were our design factors.

By comparing the values in these tables, we find that the S-boxes given here are better than the DES S-boxes in terms of all four properties. The FDE S-boxes were not designed considering the fourth and fifth properties, but the differences in structure (no expanding function and the independence of key and data bits in each S-box), make this less important. Knudsen showed that without these properties, the resulting encryptor may be weak against differential attacks [7, 8]. This weakness is because of the potentially high probability of 2-round iterative characteristics. In addition, Kim states that resistance to a differential attack cannot be decided based on the XOR table [6], and that it requires consideration of all n-round iterative characteristics. This along with the analytical and stochastic evaluation of FDE will be the subject of a future paper.

Table 3: Nonlinearity Comparison

S-box	**DES Nonlinearity**				**FDE Nonlinearity**			
	S	S^{-1}	$s \cdot f_i$	$S^{-1} \cdot f_i$	S	S^{-1}	$s \cdot f_i$	$S^{-1} \cdot f_i$
S1	16	16	18,20,22,18	22,20,20,14	20	20	24,24,22,20	24,20,20,22
S2	16	16	22,20,18,18	16,22,22,20	22	22	22,24,24,22	22,22,22,22
S3	16	16	18,22,20,18	16,22,22,20	22	22	24,22,22,22	22,22,22,22
S4	16	16	22,22,22,22	16,22,16,22	20	20	20,24,24,24	24,24,24,24
S5	12	12	22,20,18,20	16,18,20,12	22	22	22,22,22,24	22,22,22,24
S6	18	18	20,20,20,20	18,22,20,18	22	22	24,22,24,24	22,22,22,22
S7	14	14	18,22,14,20	16,22,18,18	22	22	22,22,22,22	22,22,24,22
S8	16	16	22,20,20,22	16,18,20,16	20	20	24,24,22,24	20,24,24,22

Table 4: DBIC Comparison

S-box	**DES DBIC**		**FDE DBIC**	
	S	S^{-1}	S	S^{-1}
S1	6	6	4	4
S2	6	10	4	4
S3	6	8	4	4
S4	16	12	6	4
S5	6	10	6	4
S6	6	8	6	6
S7	6	12	4	6
S8	8	6	4	4

Table 5: SAC Comparison

S-box	**DES**		**FDE**	
	$DSAC(s \cdot f_i)$	$DSAC(S^{-1} \cdot fi)$	$DSAC(s \cdot f_i)$	$DSAC(S^{-1} \cdot f_i)$
S1	8,8,6,8	8,6,4,8	0,0,0,0	0,0,0,0
S2	8,4,14,8	8,6,6,8	0,0,0,2	2,0,2,2
S3	12,8,12,10	8,6,6,8	0,2,2,0	2,0,2,2
S4	6,6,6,6	4,6,4,8	0,0,2,0	2,0,2,0
S5	6,8,10,6	10,6,6,12	2,0,0,4	2,0,2,0
S6	8,10,8,8	12,8,4,10	0,2,0,0	2,4,0,0
S7	12,8,12,8	10,6,8,8	0,2,0,0	2,4,0,2
S8	10,10,10,8	10,6,8,16	0,0,2,2	2,0,0,2

Table 6: XOR Table Comparison

S-box	DES			FDE		
	$XOR(S)$	$nz_{\text{avg}}(S)(\%)$	$\sigma(S)$	$XOR(S)$	$nz_{\text{avg}}(S)(\%)$	$\sigma(S)$
S1	16	79.0	3.23	18	83.0	2.89
S2	16	78.0	3.30	14	87.0	2.58
S3	16	79.0	3.25	14	87.0	2.50
S4	16	68.0	3.70	14	86.0	2.70
S5	16	76.0	3.34	14	87.0	2.52
S6	16	80.0	3.14	12	88.0	2.50
S7	16	77.0	3.45	16	87.0	2.53
S8	16	77.0	3.30	16	84.0	2.83

4 Conclusions

In this paper, we introduced Fast Data Encryption (FDE), which is a DES-like encryption algorithm with a symmetric round function, a 128 bit key, and complex key scheduling. We presented an algorithm to construct a set of strong S-boxes. Eight S-boxes suitable for use in FDE were constructed using this algorithm.

References

[1] C. Adams and S. Tavares, The structured design of cryptographically good S-boxes, *J. Cryptology*, **3** (1990), 27–41.

[2] M. Aref, F. Hendessi and M. Omoumi, A new algorithm for fast data encryption, (in Persian), *Esteghlal J.*, **11** (1993).

[3] E. Biham and A. Shamir, Differential cryptanalysis of DES-like cryptosystems, *Proc. Crypto'90, Springer-Verlag Lecture Notes in Computer Science*, **537** (1991), 2–21.

[4] M.H. Dawson and S.E. Tavares, An expanded set of S-box design criteria based on information theory, *Proc. Eurocrypt'91, Springer-Verlag Lecture Notes in Computer Science*, *547* (1991), 352–367.

[5] F. Hendessi, *Analyzing of DES Cryptosystem*, M.Sc. Thesis, Isfahan University of Technology, Iran, Dec. 1988.

[6] K. Kim, S. Park and S. Lee, Reconstruction of s^2-DES S-boxes and their immunity to differential cryptanalysis, *Proc. Korea-Japan Joint Workshop on Inform. Security and Crypt.* (1993), 32.01–32.10.

[7] K. Kim, Construction of DES-like S-boxes based on boolean functions satisfying the SAC, *Proc. AsiaCrypt'91, Springer-Verlag Lecture Notes in Computer Science*, **739** (1991), 59–72.

[8] L.R. Knudsen, Iterative characteristics of DES and s^2-DES, *Proc. Crypto'92, Springer-Verlag Lecture Notes in Computer Science*, **740** (1993), 497–511.

[9] W. Meier and O. Staffelbach, Nonlinearity criteria for cryptographic functions, *Proc. EuroCrypt'89, Springer-Verlag Lecture Notes in Computer Science*, **434** (1989), 549–562.

[10] S. Mister and C. Adams, Practical S-box design, *Proc. Workshop Select. Areas Crypt.* (1996), 61–76.

[11] L. O'Connor, An analysis of a class of algorithms for S-box construction, *J. Cryptology*, **7** (1994), 133–151.

[12] J. Seberry, X.-M. Zhang and Y. Zheng, Nonlinearity and propagation characteristics of balanced boolean functions, *Inform. and Computation*, *119* (1995), 1–13.

[13] A. Shafieinezhad, *Analyzing of FDE Cryptosystem and Fast Implementation in Software*, M.Sc. Thesis, Isfahan University of Technology, Iran, 2001.

[14] A. Shafieinezhad, F. Hendessi and M. Esmaeili, Fast FDE implementation in software (bitslicing method), (in Persian), *Proc. Iranian Conf. on Computers*, (2002), 510–518.

[15] A.F. Webster, *Plaintext/Ciphertext Bit Dependencies in Cryptographic Systems*, Master's Thesis, Queen's University, Kingston, Ontario, Canada, 1985.

Received: October 31, 2006

Appendix A: The FDE S-boxes

FDE-S1

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

FDE-S2

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

FDE-S3

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

FDE-S4

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

FDE-S5

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

FDE-S6

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

FDE-S7

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

FDE-S8

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