OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

(12) (19) (CA) **Brevet-Patent**

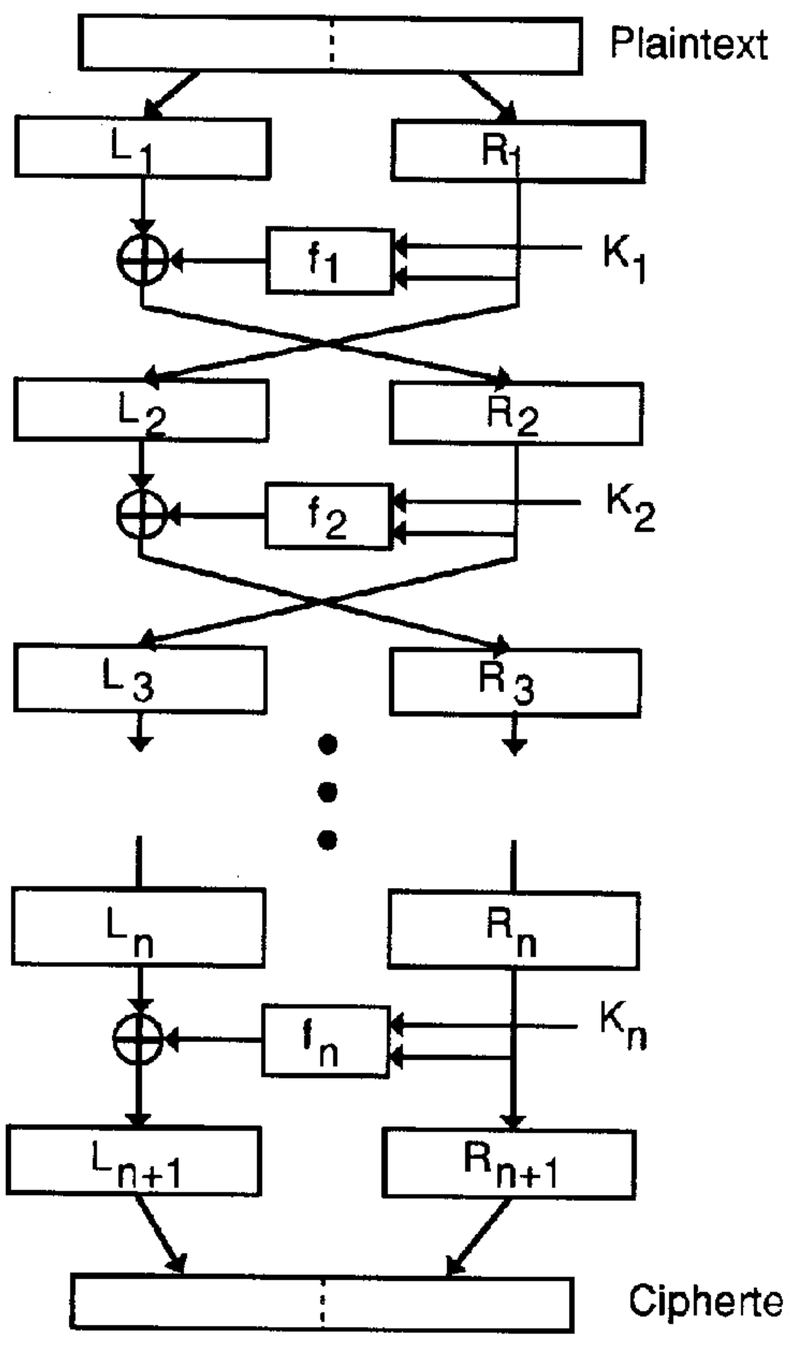
(11) (21) (C) **2,164,768**
(22) 1995/12/08
(43) 1997/06/09
(45) 2001/01/23

(72) Adams, Carlisle Michael, CA
(72) Wiener, Michael James, CA
(72) Lockhart, Roland Thomas, CA
(73) ENTRUST TECHNOLOGIES LTD., CA

(51) Int.Cl.[6] G09C 1/00

(54) **CONSTRUCTION DE CHIFFRES SYMETRIQUES SELON LA METHODE CAST**

(54) **CONSTRUCTING SYMMETRIC CIPHERS USING THE CAST DESIGN PROCEDURE**



(57) A new design procedure for constructing a family of DES-like Substitution-Permutation Network (SPN) cryptosystems with desirable cryptographic properties including provable resistance to differential cryptanalysis, linear cryptanalysis, and related-key cryptanalysis is described. New cryptosystems, called CAST ciphers, constructed according to the procedure are also described. Details of the design choices in the procedure are given, including those regarding the component substitution boxes (s-boxes), the overall framework, the key schedule, and the round function. A fully specified example CAST cipher, an output of this design procedure, is presented as an aid to understanding the concepts and to encourage detailed analysis by the cryptologic community.

Abstract of the Disclosure

A new design procedure for constructing a family of DES-like Substitution-Permutation Network (SPN) cryptosystems with desirable cryptographic properties including provable resistance to differential cryptanalysis, linear cryptanalysis, and related-key cryptanalysis is described. New cryptosystems, called CAST ciphers, constructed according to the procedure are also described. Details of the design choices in the procedure are given, including those regarding the component substitution boxes (s-boxes), the overall framework, the key schedule, and the round function. A fully specified example CAST cipher, an output of this design procedure, is presented as an aid to understanding the concepts and to encourage detailed analysis by the cryptologic community.

Constructing Symmetric Ciphers Using the CAST Design Procedure*

Field of the Invention

The invention resides generally in symmetric cryptosystems and their construction procedures. In particular, it is directed to new ciphers which belong in a family of DES-like substitution-permutation network cryptosystems and to methods of cryptographically transforming plaintext into ciphertext using such novel ciphers. The invention relates also to prcedures for constructing such new ciphers.

Background of the Invention

1. Introduction and Motivation

This paper describes the CAST design procedure for a family of encryption algorithms. The ciphers produced, known as CAST ciphers, are provably resistant to differential cryptanalysis [8], linear cryptanalysis [31], and related-key cryptanalysis [13]. Furthermore, they can be shown to posses a number of desirable cryptographic properties such as avalanche [19, 20], Strict Avalanche Criterion (SAC) [49], Bit Independence Criterion (BIC) [49], and an absence of weak and semi-weak keys [25, 16, 36]. CAST ciphers are based on the well-understood and extensively-analyzed framework of the Feistel cipher [19, 20] – the framework used in DES – but with a number of improvements (compared to DES) in both the round function and the key schedule which guarantee good cryptographic properties in fewer rounds than DES. These ciphers therefore have very good encryption / decryption performance (comparing very favourably with many alternatives of similar cryptographic strength) and can be designed with parameters which make them particularly suitable for software implementations on 32-bit machines.

The search for a general-purpose design procedure for symmetric encryption algorithms is motivated by a number of factors, including the following.

---

* Version: Fri., Sept. 29, 1995.

- Despite years of speculation and warning regarding the inevitable limit to the useful lifetime of the Data Encryption Standard (as originally defined in [37]), this algorithm remains firmly entrenched in a number of environments partly because there is no obvious candidate for a DES replacement with acceptable speed and security.

- New and powerful cryptanalytic attacks have forced re-designs of suggested candidates such as FEAL [34, 35, 9], LOKI [14, 10, 15], and IDEA [29, 30]. Thus, such attacks must be accounted for and avoided in the design procedure itself, so that algorithms produced by the procedure are known to be immune to these attacks.

- The continued disparity between "domestic-strength" cryptography and "exportable-strength" cryptography, along with the potential for multiple flavours of exportable-strength cryptography (perhaps depending on "commercial escrow" considerations), means that the paradigm of a single DES replacement algorithm almost certainly has to be abandoned in favour of a design procedure describing a family of algorithms where keysize is at least one parameter defining a specific instance of the family. Recent cipher proposals such as RC-2, RC-4, and RC-5 have recognized and addressed this requirement.

1.1. Background

Some aspects of the CAST design procedure were discussed in [1, 5-7]. Analysis of CAST-like ciphers containing purely randomly-generated s-boxes with respect to both linear and differential cryptanalysis was presented in [24, 28]. As well, cryptanalysis of a 6-round CAST cipher was described in [43]; this statistical attack requires a work factor of roughly 2^{48} operations.

1.2. Outline of the Paper

The remainder of the paper is organized as follows. Section 2 presents an overview of the CAST design procedure, with subsections covering substitution box design, Feistel-type Substitution-Permutation Network (SPN) considerations, the importance of key scheduling, and possibilities for the round function. Section 3 presents a deeper treatment of the design procedure, giving further details, along with assertions and theorems, regarding these four main aspects of CAST cipher design. The fourth section covers design alternatives available for both the SPN framework and the implementation of the round function. Section 5, along with Appendix A, gives the full specification for an example CAST cipher, one produced

using the design procedure described in this paper. Finally, Section 6 closes the paper with some concluding comments.

Objects of the Invention

It is therefore an object of the inventio to provide a novel construction procedure for symmetric ciphers.

It is another object of the invention to provide novel symmetric ciphers which have one or more desirable properties such as resistance to differential cryptanalysis, linear cryptanalysis, and related-key cryptanalysis.

It is a further object of the invention to provide a method of cryptographically transforming palintext into ciphertex using novel round functions.

Summary of the Invention

Briefly stated, according to one aspect, the invention relates to a data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of a predetermined bitlength comprising a plurality of consecutive transformation rounds of half of each data block. Each consecutive transformation round comprises steps of combining the half data block with a first masking key of predetermined length using a first binary operation to generate a first modified half data block and combining the first modified half data block with a second masking key of predetermined length using a second (different) binary operation to generate a second modified half data block. The method fruther includes steps of processing the second modified half data block by a plurality of (m x n) mutually different substitution boxes to generate a third modified half data block and XORing the third modified half data block with the remaining half of the data block to generate a transformed half data block of a transformation round.

Brief Description of the Drawings

Figure 1 is a known SPN (Substitution-Permutation Network) cipher.

Figure 2 shows a round function according to one embodiment of the invention.

Detailed Description of the Preferred Embodiments of the Invention

2. Overview of the CAST Design Procedure

This section gives a brief overview of the concepts and considerations relevant to the CAST design procedure. The four main aspects of a CAST cipher (s-boxes, framework, key schedule, and round function) are covered in separate subsections.

2.1. S-Box Design Overview

An $m \times n$ substitution box is a $2^m \times n$ lookup table, mapping m input bits to n output bits. It substitutes, or replaces, the input with the output in a nonlinear way so that any change to the input vector results in a random-looking change to the output vector which is returned. The substitution layer in an SPN cipher is of critical importance to security since it is the primary source of nonlinearity in the algorithm (note that the permutation layer is a linear mapping from input to output).

The dimensions m and n can be of any size; however, the larger the dimension m, the (exponentially) larger the lookup table. For this reason m is typically chosen to be less than 10. The CAST design procedure makes use of substitution boxes which have fewer input bits than output bits (e.g., 8×32); this is the opposite of DES and many other ciphers which use s-boxes with more input bits than output bits (e.g., 6×4).

Research into cipher design and analysis suggests that s-boxes with specific properties are of great importance in avoiding certain classes of cryptanalytic attacks such as differential and linear cryptanalysis. Furthermore, it is significantly more difficult (and, in some cases, impossible) to satisfy certain properties using "small" s-boxes. The CAST design procedure therefore incorporates a construction algorithm for "large" s-boxes which possess specific cryptographic properties.

2.2. Framework Design Overview

Ciphers designed around a new basis for cryptographic security (most notably RC-5, based upon the conjectured security of data-dependent rotation operations) may prove to be extremely attractive candidates for DES replacement algorithms, but are not yet mature enough to be recommended for widespread use. The CAST

procedure is instead based upon a framework which has been extensively analysed by the cryptologic community for several decades.

The CAST framework is the "Substitution-Permutation Network" (SPN) concept as originally put forward by Shannon [46]. SPNs are schemes which alternate layers of bit substitutions with layers of bit permutations, where the number of layers has a direct impact on the security of the cipher. Furthermore, CAST uses the Feistel structure [19, 20] as opposed to the "tree structure" [22, 23] to implement the SPN. This is because the Feistel structure is well-studied and appears to be free of basic structural weaknesses, whereas the tree structure has some inherent weaknesses [22, 41] unless a significant number of layers are added (which may destroy the one property, "completeness"[1], which tree structures are provably able to achieve).

The following diagram illustrates a general Feistel-structured SPN. Basic operation is as follows. A message block of n bits is input and split into a left half L and a right half R. The right half and a subkey K_i are input to a "round function", f_1, the output of which is used to modify (through XOR addition) the left half. Swapping the left and right halves completes round one. This process continues for as many rounds as are defined for the cipher. After the final round (which does not contain a swap in order to simplify implementation of the decryption process), the left and right halves are concatenated to form the ciphertext.

The parameters which can be selected for the framework are the blocksizes (the number of bits in both the plaintext and ciphertext data blocks) and the number of rounds. For all cases "higher" typically means greater security but (particularly for the number of rounds) reduced encryption / decryption speed. In practice, it is common to choose the plaintext and ciphertext blocksizes to be equal so that the encryption process results in no data expansion (an important consideration in many applications).

As is evident in recent work by Biham [12] and by Knudsen [27], good s-box design is not sufficient to guarantee good SPN cryptosystems (both results show that finding 6×4 s-boxes resistant to differential cryptanalysis in isolation – that is, with relatively flat Output XOR distributions – and putting them directly in DES

---

[1]Completeness states that output bit j can be changed by inverting only input bit i in some input vector, for all i, j [26].

makes the "improved" algorithm much more susceptible to differential cryptanalysis than the original[2]). It is therefore of great importance to design the substitution-permutation network such that it takes advantage of the good properties of the s-boxes without introducing any cryptographic weaknesses.

2.3. Key Schedule Design Overview

Keying in the CAST design procedure is done in the manner typical for Feistel networks. That is, an input key (a "primary key") is used to create a number of subkeys according to a specified key scheduling algorithm; the subkey for a given round is input to the round function for use in modifying the input data for that round.

The design of a good key schedule is a crucial aspect of cipher design. A key schedule should possess a number of properties, including some guarantee of key/ciphertext Strict Avalanche Criterion[3] and Bit Independence Criterion[4] in order to avoid certain key clustering attacks [23, 48]. Furthermore, it should ensure that the primary key bits used in round i to create subkey i are different from those used in round i+1 to create subkey i+1 (this is due to the work of Grossman and Tuckerman [21], who showed that DES-like cryptosystems without a key that varies through successive rounds can be broken). Finally, if any key bit is used in round N (the last round) for the first time then the network fails the key/ciphertext completeness test[5], since complementing that bit can affect at most half the

---

[2]Note that the susceptibility of DES-like cryptosystems employing flat-output-XOR-distribution s-boxes to differential cryptanalysis is due primarily to the fact that these s-boxes have more input bits than output bits [5], and not to the fact that the output XOR distribution is flat, as was implied in [12, 45]. This is one of the main reasons that the CAST design procedure has concentrated on $m \times n$ s-boxes where $m << n$.

[3]The Strict Avalanche Criterion (SAC) states that s-box output bit j should change with probability 1/2 when any single input bit i is inverted, for all i, j (note that for a given i and j the probability is computed over the set of all pairs of input vectors which differ only in bit i) [48, 49].

[4]The (output) Bit Independence Criterion (BIC) states that s-box output bits j and k should change independently when any single input bit i is inverted, for all i, j, k (note that for a given i, j, and k the independence is computed over the set of all pairs of input vectors which differ only in bit i) [48, 49].

[5]Note that completeness tests can involve the input and output of an s-box, the input and output of a round function, the plaintext and ciphertext of a cipher, or the key and ciphertext of a cipher.

ciphertext bits. All key bits must therefore be used by round *N*-1. In fact, the CAST procedure stipulates that they all be used by at most round *N*/2 and reused, after one or more "key transformations" (see detailed key schedule in Section 3.3), in the lower half of the network (this ensures good key avalanche for both encryption and decryption, as well as other properties discussed in Section 3.3).

The critical difference between the key schedule proposed in the CAST design procedure and other schedules described in the open literature is the dependence upon substitution boxes for the creation of the subkeys. Other key schedules (the one in DES, for example) typically use a complex bit-selection algorithm to select bits of the primary key for the subkey for round *i*. As is clear from the work by Biham [13], any weaknesses in this bit selection algorithm can lead to simple cryptanalysis of the cipher, regardless of the number of rounds. The schedule proposed in CAST instead uses a very simple bit-selection algorithm and a set of "key schedule s-boxes" to create the subkey for each round. These s-boxes must possess specific properties to ensure cryptographically good key schedules (see Section 3.3 below).

2.4. Round Function Design Overview

The round function in CAST, as stated above, makes use of s-boxes which have fewer input bits than output bits. This is accomplished as follows. Within the round function the input data half is modified by the subkey for that round and is split into several pieces. Each piece is input to a separate substitution box; the s-box outputs are combined; and the result is the output of the round function. Although each $m \times n$ s-box on its own necessarily causes data expansion (since $m<n$), using the set of s-boxes in this way results in no expansion of the message half, allowing the SPN to have input and output blocksizes which are equal.

2.4.1. Avoiding Certain Attacks

Another aspect of round function design involves a specific proposal to guard against differential and linear attacks. Differential [8, 12] and linear [31] cryptanalysis appear to be fairly general-purpose attacks which may be applied to a variety of substitution-permutation network (DES-like) ciphers. The successful (in

a theoretical sense) attacks on DES have been widely reported, as have attacks on FEAL, LOKI, Snefru, and generalized versions of DES, among others [9-11]. Both methods work on the principle of finding high-probability attacks on a single round and then building up "characteristics" (sets of consecutive rounds which interact in useful ways); characteristics which include a sufficient number of rounds can lead to cryptanalysis of the cipher. The probability of a characteristic is equal to the product of the probabilities of the included rounds; this "characteristic probability" determines the work factor of the attack. If the work factor of the attack is less than the work factor for exhaustive search of the key space, the cipher is theoretically broken.

From the above description it can be seen that adding rounds to a DES-like cipher can always be done to increase the work factor of a differential or linear attack, until the work factor surpasses that of exhaustive key search. This makes the cipher computationally resistant to these attacks. The disadvantage of this approach is that the encryption/decryption speed of the cipher is reduced, perhaps drastically, since each added round slows the cipher down by a factor of $1/N$, where N is the number of rounds in the original cipher.

An alternate approach which has been pursued by a number of researchers is to decrease the attack probability of an individual round by improving the properties of the round s-boxes (see [4, 5, 18, 39, 45, 47], for example). This results in a lower characteristic probability for the same number of rounds and therefore has the potential to make the cipher resistant to these attacks without degrading throughput. However, there is always the possibility that for a given cipher the round probability cannot be made low enough to avoid the need to add rounds. Furthermore, there is always the possibility that for a given cipher the best characteristic has not yet been found (and when it is found, it will render the cipher breakable).

The approach proposed in the CAST design procedure is neither of the above. Instead, a slight alteration is suggested for the typical DES-like round function which renders it "intrinsically immune" (as opposed to computationally immune) to differential and linear cryptanalysis. Such an alteration is generally applicable to all DES-like ciphers and may, in some ciphers, be added with little degradation in encryption / decryption speed.

3. Detailed Design

This section covers the four main aspects of a CAST cipher (s-boxes, framework, key schedule, and round function) in more detail than the previous section and provides a number of proofs regarding the cryptographic properties relevant to each aspect.

3.1. Detailed S-Box Design

For the design of $m \times n$ ($m < n$) s-boxes[6], let n be an integer multiple of m; in particular, let $n=rm$ where $r>1$ and r is chosen such that $m < log_2 C(n,n/2) = log_2(\text{“}n \text{ choose } n/2\text{”})$. Such s-boxes can be constructed as follows. Choose n distinct binary bent (see, for example, [38, 42, 3]) vectors ϕ_i of length 2^m such that linear combinations of these vectors sum (modulo 2) to highly nonlinear vectors (Nyberg's work [39] shows that these linear combinations cannot all be bent since $m<2n$; however, it is important that they be highly nonlinear or the resulting s-box will not come close to satisfying the Output Bit Independence Criterion). Furthermore, choose half the ϕ_i to be of weight $(2^{m-1} + 2^{(m/2)-1})$ and the other half to be of weight $(2^{m-1} - 2^{(m/2)-1})$; these are the two weights possible for binary bent vectors of length 2^m. Set the n vectors ϕ_i to be the columns of the matrix M representing the s-box.

Check that M has 2^m distinct rows and that the Hamming weight of each row and the Hamming distance between pairs of rows is close to $n/2$ (i.e., that the set of weights and the set of distances each have a mean of $n/2$ and some suitably small – but nonzero – variance)[7]. As well, if the i^{th} row of M is denoted by r_i, it should be verified that $(r_i \oplus r_j) \neq (i \oplus j)$ for any $i,j \in \{1, ..., 2^m\}$, $i \neq j$ so that a non-zero input XOR is never equal to its resulting output XOR in the s-box (which may

---

[6]An $m \times n$ s-box is represented as a $2^m \times n$ binary matrix M where each column is a vector which corresponds to a Boolean function of the m input variables and which defines the response of a single output bit to any given input. Row i of M, $1 \leq i \leq 2^m$, is therefore the n-bit output vector which results from the i^{th} input vector.

[7]Note that this is impossible if $m \geq n$ but is quite feasible if $2^m \leq C(n,n/2)$.

greatly facilitate finding a differential characteristic for the cipher). This latter condition will, in general, hold if the Hamming distance condition is met. If these conditions are not all satisfied, continue choosing suitable bent vectors (i.e., candidate ϕ_i) and checking the resulting matrix until the conditions are satisfied. Note that it is not difficult to construct 8×32 s-boxes which meet these conditions.

The following assertions and theorems apply to substitution boxes constructed according to the above procedure.

Assertion 1: Requiring each column of an $m \times n$ s-box (where $m < log_2 C(n,n/2)$) to be bent and each row to have Hamming weight approximately $n/2$ ensures that the s-box will provide good *confusion.*

Discussion: According to Shannon [46], the purpose of "confusion" in a cipher is to make the relation between the statistics of the output and a simple description of the input a very complex and involved one. The conditions on the s-box rows and columns ensure that the s-box outputs behave statistically like a collection of random binary vectors of length n. Therefore, any relation between this set of vectors and a simple description of the input vectors cannot be trivial.

Assertion 2: S-boxes with more output bits than input bits accomplish *diffusion.*

Discussion: The purpose of "diffusion" in a cipher is to dissipate the local statistical structure in the input into long-range statistics (those involving long – or longer – combinations of output bits) [46]. S-boxes of size $m \times n$ ($m < n$) accomplish this because all n output bits depend on the m input bits, so that any statistical structure inherent in the input will be spread out over the output bits.

Assertion 3: Requiring the sum (modulo 2) of any pair of rows in an $m \times n$ s-box to have Hamming weight approximately $n/2$ ensures that the s-box will provide good *avalanche* for any given input vector.

Discussion: As discussed in [19, 20] (and echoed in [26]), "avalanche" refers to the property that approximately half the output bits change when any small change is made to the input (i.e., a small input change causes an "avalanche effect" which results in a large, unpredictable change in the output). The condition that any pair of rows is approximately $n/2$ bits apart in Hamming distance clearly provides good avalanche since it ensures that *any* input change (i.e., including small input changes) results in a change of approximately half the output bits.

Theorem 1: Using bent binary vectors as the columns of the $2^m \times n$ matrix which describes an s-box ensures that the s-box will respond "ideally" in the sense of *highest-order strict avalanche criterion* [2, 4][8] to arbitrary changes in the input vector.

Proof: Highest-order SAC is guaranteed for each output bit – this is a property of bent Boolean functions which was proven in [32]. By definition [49], an s-box satisfies the highest-order SAC if and only if each of its output bits satisfies the highest-order SAC.

Theorem 2: If the columns in the s-box matrix are bent vectors which are highly nonlinearly related, then the s-box will show close proximity to *highest-order (output) bit independence criterion.* That is, any change in the m input bits will cause each of the n output bits to change virtually independently of all the other output bits. Furthermore, such s-boxes aid in *immunity to linear cryptanalysis* [31].

Proof: It can be shown that if columns ϕ_j and ϕ_k sum modulo 2 to a linear vector, then s-box output bits j and k will either always change together or never change together when any input bit i is inverted (i.e., they will have a correlation coefficient of ± 1). At the other extreme, if ϕ_j and ϕ_k sum to a bent vector, then j and k will change independently for any input change. Because it is impossible for all column sums to be bent (since $m<2n$), the CAST design procedure uses s-boxes which are partially bent-function-based, where many of the column sums are bent and the remainder are nonlinear but not bent. Using the highest (non-bent) nonlinearity possible ensures that bits j and k will act "virtually" independently (i.e., will have a correlation coefficient which is nonzero, but as small as possible), for all input changes. In highest-order BIC the sums of all column subsets are considered (not just pairs). Requiring that these sums have the highest nonlinearity possible guarantees that the s-box will have close proximity to highest-order BIC[9]. Such s-boxes aid in immunity to linear cryptanalysis because there is no linear combination of component functions which has a small Hamming distance to an affine Boolean function (see the discussion in Section 8.1 of [45]). ♦

---

[8]This has independently been called the Propagation Criterion of degree n in [42].

[9]Note that highest-order BIC itself (i.e., total independence of output bits over the full set of input changes) cannot be achieved except in Nyberg's "perfect nonlinear" $2n \times n$ s-boxes [39], where all column sums are bent.

Theorem 3: *m*×*n* s-boxes designed according to the above procedure can be made to be *ε-robust against differential cryptanalysis*, where $(1 - 2^{-m/2}) \leq \varepsilon \leq (1 - 2^{1-m})$.

Proof: The definition of robustness against differential cryptanalysis for an s-box is given in terms of ε [45]: $\varepsilon = (1 - A(2^{-m})) (1 - B(2^{-m}))$, where A is the largest value in the difference distribution table of the s-box and B is the number of nonzero entries in the first column of the table (the value 2^m in the first row is not counted in either case). Given that $m < n$ and all input vectors are mapped to unique output vectors, B equals zero (i.e., there are no input XORs which result in output XORs of zero). Thus, it is sufficient to show that s-boxes can be constructed such that $2 \leq A \leq 2^{m/2}$.

Let a CAST s-box be constructed by beginning with Nyberg's "perfect nonlinear" $m \times m/2$ s-box and adding binary bent vectors as matrix columns until the full $2^m \times n$ matrix M is complete (adhering to the design constraints given above). Without loss of generality, assume that the first $m/2$ columns of M correspond to a perfect nonlinear s-box (i.e., these columns are bent and all nonzero linear combinations of these columns (modulo 2) are also bent). Consider the $2^{m-1} \times n$ matrix M' of avalanche vectors corresponding to a given change in the s-box input (see [4] for details). In this matrix all columns are of Hamming weight 2^{m-2} (since the columns of M are bent) and all nonzero linear combinations of the first $m/2$ columns are also of Hamming weight 2^{m-2} (linear combinations of other subsets of columns will be of Hamming weight greater than or less than 2^{m-2} since in M those subsets are nonlinear but not bent). It is not difficult to see that within the first $m/2$ columns of M', therefore, each $m/2$-bit "row" will occur *exactly* $T = 2^{m-1}/2^{m/2}$ times, so that regardless of the remaining columns of M', each full n-bit row can occur a *maximum* of T times. Thus, the largest value in the difference distribution table for this s-box is $A \leq 2T = 2^{m/2}$. Clearly, each additional column in M' (beyond the $m/2$ initial columns) has the ability to reduce T; in the limit (when n is sufficiently large compared with m), every row of M' is unique, so that T=1. Therefore $A \geq 2$.

Remark: Although starting with a perfect s-box provides a guaranteed upper bound on A, in practice the same result can be achieved without the perfect s-box if n is sufficiently large. For example, it is not difficult to construct 8×32 s-boxes with A=2 which do not have four component columns which form a perfect s-box. This is why the use of a perfect s-box has not been made a stipulation of the s-box design procedure given above. ♦

3.2. Detailed Framework Design

As was stated previously, the primary parameter options in framework design are blocksize(s) and number of rounds. It is preferable in many applications if the plaintext and ciphertext blocksizes are equal (we will therefore hereafter refer to *the blocksize* of a cipher), but aside from the constraint that the blocksize be large enough to prevent an exhaustive compilation of plaintext / ciphertext pairs for a single key, the only real blocksize consideration is ease of implementation. On current machines, 64 bits (the blocksize of DES) is an attractive choice because left and right data halves and other variables fit nicely into 32-bit registers. However, in the future another choice may be attractive for other reasons.

The number of rounds in the framework appears to be a much more important and delicate decision. There need to be enough rounds to provide the desired level of security, but not so many that the cipher is unacceptably slow for its intended applications. In an SPN of the Feistel type it is clear that the left half of the input data is modified by the output of the round function in rounds 1, 3, 5, 7, and so on, and the right half is modified in rounds 2, 4, 6, 8, and so on. Thus, it is clear that for equal treatment of both halves the number of rounds must be even. However, it is less obvious how many rounds is "enough".

Differential and linear cryptanalysis, the two most powerful attacks currently known for DES-like ciphers, have helped to quantify this design parameter. It has long been known, for example, that DES with 5 or 6 rounds can be broken, but not until 1990, with the introduction of differential cryptanalysis [8], was it clear why 16 rounds were actually used in its design – fewer rounds could not withstand a differential attack [17]. With subsequent improvements to the differential attack [11] and with the introduction of linear cryptanalysis, it now appears that 18-20 rounds would be necessary for DES to be theoretically as strong as its keysize.

A prudent design guideline, therefore, is to select a number of rounds which has an acceptably high work factor for both differential and linear cryptanalysis and then either add a few more rounds or modify the round function to make these attacks even more difficult (in order to add a "safety margin"). As will be seen in Section 3.4, the CAST design procedure chooses the second approach for both security and performance reasons.

Theorem 4: Ciphers designed according to the CAST procedure are *immune to differential cryptanalysis* for an appropriate choice of parameters (in that the work factor for differential cryptanalysis is higher than exhaustive search of the keyspace).

Proof: Recall from above that the largest value in the difference distribution table of CAST-designed $m \times n$ s-boxes is L, where $L \leq 2^{m/2}$. Therefore, the highest probability in the table is $P = L/2^m \leq 2^{-m/2}$. Consider now the f function of this SPN. If a multi-bit change is made to the vector V which is input to f (so that a change is made to the input of each of x of the component s-boxes used for f), then the differential [30] of f (that is, the most successful differential cryptanalytic attack for that single round) has probability at most $P_f = 2^{-x(m/2)}$. Therefore, the strategy for differential cryptanalysis in this cipher must be to change the inputs of the smallest number of s-boxes possible in f in each round.

Let ΔV be an input XOR for f for which the corresponding output XOR is zero. It is conceivable (although unlikely) that such a ΔV involves only two s-boxes[10]; the probability of this differential could therefore be as high as $P_f = 2^{-2(m/2)} = 2^{-m}$. Hence, assuming an N-2 round characteristic (for an N-round cipher), the probability of the characteristic could be as high as P_f^{N-2}.

For example parameters (m=8, N=10), therefore, the total differential probability would be 2^{-64} (sufficient for a 64-bit key); similarly, a 12-round cipher would be sufficient for an 80-bit key. Even if an improved characteristic is found subsequently, recall that 8×32 CAST s-boxes can be found with a differential probability of 2^{1-m}, bringing the probability of the full cipher to approximately $2^{2(1-m)(N-2)}$. It is therefore not difficult to ensure a higher work factor than exhaustive search of any reasonable keyspace by using CAST with N=12 (or more) rounds. ♦

Theorem 5: Ciphers designed according to the CAST procedure are *immune to linear cryptanalysis* for an appropriate choice of parameters (in that the work factor for linear cryptanalysis is higher than exhaustive search of the keyspace).

Proof: The relationship in a CAST cipher between the minimum nonlinearity of the $m \times n$ substitution boxes in the round function (N_{min}), the number of rounds in

---

[10]It cannot involve only one s-box (due to the s-box design), and is much more likely to involve 3 or 4 s-boxes in f (if such a differential exists at all for the s-boxes chosen in a specific implementation).

the overall cipher (N), and the number of known plaintexts required for the recovery of a single key bit (N_L) has been given by Heys and Tavares [24] (based on the results in [31]):

$$N_L \geq \frac{2^{2-4N}}{\left(\frac{2^{m-1}-N_{\min}}{2^m}\right)^{4N}}$$

Substituting $N_{min} = 74$ and $N = 12$ results in N_L being lower-bounded[11] by approximately 2^{62}, so that recovering 64 bits of key would require approximately 2^{68} known plaintexts (more than adequate security for a 64-bit key). It should be noted that 8×32 s-boxes with minimum nonlinearity $N_{min} = 74$ have been constructed using the CAST procedure; more rounds, higher nonlinearity s-boxes, or additional operations in the round function (see Section 3.4) would all permit CAST ciphers with longer keys to be used with guaranteed resistance to linear cryptanalysis. ♦

3.3. Detailed Key Schedule Design

As indicated in Section 2.3 above, the key schedule used in the CAST design procedure has three main components: a relatively simple bit-selection algorithm mapping primary key bits to "partial key" bits; one or more "key transformation" steps; and a set of "key schedule s-boxes" which are used to create subkeys from partial keys in each round. An example key schedule for an 8-round algorithm employing a 64-bit key is as follows.

Let $KEY = k_1k_2k_3k_4k_5k_6k_7k_8$, where k_i is the i^{th} byte of the primary key. The partial keys K'_i are selected from the primary key according to the following bit-selection algorithm: $K'_1=k_1k_2$, $K'_2=k_3k_4$, $K'_3=k_5k_6$, $K'_4=k_7k_8$, $K'_5=k_4'k_3'$, $K'_6=k_2'k_1'$, $K'_7=k_8'k_7'$, $K'_8=k_6'k_5'$, where KEY is transformed to $KEY' = k_1'k_2'k_3'k_4'k_5'k_6'k_7'k_8'$ between round 4 and round 5. The key transformation step is defined by:

$$k_1'k_2'k_3'k_4' = k_1k_2k_3k_4 \oplus S_1[k_5] \oplus S_2[k_7];$$
$$k_5'k_6'k_7'k_8' = k_5k_6k_7k_8 \oplus S_1[k_2'] \oplus S_2[k_4'].$$

---

[11]Note that it is not currently known how tight this lower bound is for CAST-designed ciphers.

The bytes of *KEY'* are used to construct the final four partial keys, as shown above. The set of partial keys is used to construct the subkeys K_i using key shedule s-boxes S_1 and S_2:

$$K_i = S_1(K'_{i,1}) \oplus S_2(K'_{i,2})$$

where $K_{i,j}$ denotes the j^{th} byte of K_i. Although a similar schedule can be constructed for a 12- or 16-round system or for different block or key sizes, for simplicity of notation and concreteness of explanation, the theorems below apply to the specific example given here.

3.3.1. Definitions Related to Key Scheduling

In a block cipher, an *inverse key* I for a given encryption key K is defined to be a key such that $ENC_I(p) = ENC_K^{-1}(p) = DEC_K(p)$ for any plaintext vector p. Furthermore, a *fixed point of a key K* is a plaintext vector x such that $ENC_K(x) = x$.

From work done on cycling properties and key scheduling in DES [16, 25, 36], the following definitions have been introduced. A key is *weak* if it is its own inverse (such keys generate a palindromic set of subkeys[12] and have 2^{32} fixed points in DES). A key is *semi-weak* if it is not weak but its inverse is easily found – there are two subclasses: a key is *semi-weak, anti-palindromic* if its complement is its inverse (such keys generate an anti-palindromic set of subkeys[13] and have 2^{32} fixed points in DES); a key is *semi-weak, non-anti-palindromic* if its inverse is also semi-weak, non-anti-palindromic (such keys generate a set of subkeys with the property that $K_i \oplus K_{N+1-i} = V$, where N is the number of rounds and $V = 000\ldots0111\ldots1$ or $111\ldots1000\ldots0$ in DES). DES has 4 weak keys, 4 semi-weak anti-palindromic keys, and 8 semi-weak non-anti-palindromic keys.

Let H and K be keys which generate sets of subkeys H_i and K_i, $i = 1, ..., N$, respectively, for an N-round DES-like (Feistel-type SPN) cipher. We define H to be a *subkey reflection inverse key* of K (denoted $inverse_{SR}$) if $K_i = H_{N+1-i}$, $i = 1$,

---

[12]A palindromic set of subkeys is one with the property that $K_i \oplus K_{N+1-i} = \boldsymbol{0}$, where N is the number of rounds in the cipher and $\boldsymbol{0}$ is the all-zero vector.

[13]An anti-palindromic set of subkeys is one with the property that $K_i \oplus K_{N+1-i} = \boldsymbol{1}$, where N is the number of rounds in the cipher and $\boldsymbol{1}$ is the all-one vector.

..., N. It is clear that a subkey reflection inverse key of K is an inverse key of K; whether the converse always holds true for DES-like ciphers is an open question. Thus, for a given key K, $\{H\} \subseteq \{I\}$. In DES the semi-weak key pairs are subkey reflection inverses of each other and the weak keys are subkey reflection inverses of themselves.

3.3.2. Key Schedule Theorems

Theorem 6: Ciphers using the key schedule proposed in Section 3.3 can be shown to have *no inverses$_{SR}$ key* $H \in \{0,1\}^{64}$ for any key $K \in \{0,1\}^{64}$.

Proof: There are two steps to this proof. Let $S_1[k_2'] \oplus S_2[k_4']$ be equal to the 4-byte vector $a_1a_2a_3a_4$ and let $S_1[k_5] \oplus S_2[k_7]$ be equal to the 4-byte vector $b_1b_2b_3b_4$. In the first (general) step, we prove that for the transformation given in the key schedule of Section 3.3, if inverses$_{SR}$ keys exist for the cipher then $a_1=a_2$, $a_3=a_4$, $b_1=b_2$, *and* $b_3=b_4$ all simultaneously hold. The second step, which is specific to each implementation of the CAST design, is to examine the specific s-boxes chosen in the implementation to verify that the equalities do not hold simultaneously (note that s-boxes satisfying this condition do exist).

Step 1:

Theorem: For the transformation given in the key schedule of Section 3.3, if inverses$_{SR}$ keys exist for the cipher then $a_1=a_2$, $a_3=a_4$, $b_1=b_2$, *and* $b_3=b_4$ all simultaneously hold, where a_i and b_i are defined as above.

Proof: Let H and K be cipher keys whose respective key schedules are given by Section 3.3. If H is the inverses$_{SR}$ of K then $h_1=k_6'$, $h_2=k_5'$, $h_3=k_8'$, $h_4=k_7'$, $h_5=k_2'$, $h_6=k_1'$, $h_7=k_4'$, $h_8=k_3'$, and $h_1'=k_6$, $h_2'=k_5$, $h_3'=k_8$, $h_4'=k_7$, $h_5'=k_2$, $h_6'=k_1$, $h_7'=k_4$, $h_8'=k_3$. Substituting these equalities into the key schedule gives:

$$
\begin{aligned}
h_1'h_2'h_3'h_4' &= h_1h_2h_3h_4 \oplus S_1[h_5] \oplus S_2[h_7] \\
or\ \ k_6k_5k_8k_7 &= k_6'k_5'k_8'k_7' \oplus S_1[k_2'] \oplus S_2[k_4'] \\
&= k_6'k_5'k_8'k_7' \oplus k_5k_6k_7k_8 \oplus k_5'k_6'k_7'k_8'
\end{aligned}
$$

$$
\begin{aligned}
h_5'h_6'h_7'h_8' &= h_5h_6h_7h_8 \oplus S_1[h_2'] \oplus S_2[h_4'] \\
or\ \ k_2k_1k_4k_3 &= k_2'k_1'k_4'k_3' \oplus S_1[k_5] \oplus S_2[k_7] \\
&= k_2'k_1'k_4'k_3' \oplus k_1k_2k_3k_4 \oplus k_1'k_2'k_3'k_4'
\end{aligned}
$$

Therefore, $k_6 = k_6' \oplus k_5 \oplus k_5' = k_6' \oplus a_1$, whence $a_1=a_2$. Similarly, the remaining substitutions yield $a_3=a_4$, $b_1=b_2$, *and* $b_3=b_4$. Note that these must hold simultaneously since the equalities given for the h_i and k_i necessarily hold simultaneously. ♦

Step 2:

For any specific implementation of the CAST design, the key schedule s-boxes (S_1 and S_2) can be examined to determine whether $a_1=a_2$, $a_3=a_4$, $b_1=b_2$, *and* $b_3=b_4$ hold simultaneously. If these do not hold simultaneously then the cipher has been shown to have no inverses$_{SR}$ key H for any given key K (otherwise a new S_1 and S_2 can be chosen and Step 2 can be repeated). ♦

Although the proof above applies to an 8-round implementation of a CAST cipher, the result can easily be extended to higher numbers of rounds. This may be done by modifying the proof itself (using essentially the same format and procedure, but with notation based on the new key schedule), or simply by using the eight subkeys above as the first four and last four subkeys in an N-round cipher ($N > 8$). This latter approach works because if the cipher has inverses$_{SR}$ keys, then certain equalities must hold between the first four and last four subkeys. Verifying that the equalities do not hold for these eight subkeys, then, ensures that the N-round cipher has no inverses$_{SR}$ keys.

Theorem 7: Ciphers using the key schedule proposed in this paper are *immune to related-key cryptanalysis.*

Proof: There are no related keys in the key schedule described in Section 3.3 (i.e., the derivation algorithm of a subkey from previous subkeys is not the same in all rounds because of the construction procedure and the transformation step), and so ciphers using this key schedule are not vulnerable to the "chosen-key-chosen-plaintext", "chosen-key-known-plaintext", or "chosen-plaintext-unknown-related-keys" attacks [13]. Furthermore, the CAST procedure has no known *complementation properties* (unlike DES, for example) and so CAST-designed ciphers appear not to be vulnerable to reduced key searches based on this type of weakness. ♦

3.3.3. Discussion of Key Scheduling Theorems

The above properties of the key schedule are due to the fact that s-boxes are employed in the schedule itself (i.e., in the *generation* of the subkeys), rather than simply in the *use* of the subkeys. To the author's knowledge, this is a novel proposal in key scheduling which appears to have some interesting properties.

Conjecture: Ciphers using the key schedule proposed in Section 3.3 have *no easily-found fixed points for any key.*

Discussion: From Theorem 6 above, this key schedule avoids all inverses$_{SR}$ keys. It is therefore guaranteed to avoid the fixed points associated with weak and semi-weak keys in DES (since using this key schedule in DES would guarantee the non-existance of weak and semi-weak keys). It can be shown that a large random mapping from a set of binary vectors onto itself has a non-zero expected number of fixed points [40]. Since a cryptographically strong block cipher should appear, by any statistical analysis, to act like a random mapping for every key, it is not clear that guaranteeing the non-existence of fixed points of encryption is a desirable objective. However, such fixed points should be found no more easily for any given key than for any other. From all evidence available thus far in the open literature, fixed points have only been easily[14] found in DES-like ciphers for weak and semi-weak keys; this leads to the open research question stated above as a conjecture .

3.4. Detailed Round Function Design

The round function given in Section 2.4 for a CAST cipher with a 64-bit blocksize and 8×32 s-boxes can be illustrated as follows. A 32-bit data half is input to the function along with a subkey K_i. These two quantities are combined using operation "*a*" and the 32-bit result is split into four 8-bit pieces. Each piece is input to a different 8×32 sbox (S_1, ..., S_4). S-boxes S_1 and S_2 are combined using operation "*b*"; the result is combined with S_3 using operation "*c*"; this second result is combined with S_4 using operation "*d*". The final 32-bit result is the output of the round function.

---

[14]Requiring a level of effort for an n-bit block cipher of roughly $2^{n/2}$ operations rather than 2^n operations.

A simple way to complete the definition of the CAST round function is to specify that all operations (*a*, *b*, *c*, and *d*) are XOR additions of 32-bit quantities, although other – more complex – operations may be used instead (for example, see the discussion in the following subsection regarding the first operation *a*).

Assertion 4: *Diffusion* is achieved in a CAST cipher by the end of the third round.

Discussion: Because each s-box diffuses its input statistics over the full output vector (see Assertion 2 in Section 3.1), in any given round local statistics in the right half are spread evenly over the modified right half. From the Feistel framework, then (see Fig.1 in Section 2.2), any local statistics in R1 are diffused evenly over R2, and again diffused over R3. In the same way, local statistics in L1 (which is modified to become R2) are diffused over R3, and again diffused over R4. Given that R3 is identical to L4, all local statistics in the plaintext block {L1,R1} are diffused over the full block {L4,R4} which is used as input to round four. Note that diffusion also holds for the subkeys used at each round: the subkey which is input to round *i* is diffused over the full block by the end of round *i*+1. ♦

Assertion 5: *Confusion* is achieved in a CAST cipher by the end of the third round.

Discussion: A similar argument to the one above can be used to show that a measure of confusion is achieved by the end of round three. In this case, however, additional rounds serve to increase the complexity of the mapping from plaintext or key statistics to ciphertext statistics. ♦

Theorem 8: For appropriate design choices, CAST ciphers are guaranteed to exhibit *highest-order SAC* for both plaintext and key changes.

Proof: Given that each s-box satisfies the avalanche property and guarantees highest-order SAC[15] (see Section 3.1), any change to the input of s-box S_i causes approximately half its output bits to change. If operations *b*, *c*, and *d* in the round function *f* are XOR addition (see above), then approximately half the bits in the modified message half will be inverted. Let *V* be the vector of changes to the output of S_i when its input is changed. Then $V = (v_1, v_2, ..., v_n)$, where v_i is a random binary variable with $Prob(v_i=0) = Prob(v_i=1) = 1/2$. Similarly, let $W = (w_1, w_2,$

---

[15]Note that the avalanche property relates to any specific input change; the SAC, on the other hand, is an average calculated over the full input space.

..., w_n) be the vector of changes to s-box S_j when its input is changed. Clearly, if $Z = V \oplus W$, then $Prob(z_i=0) = Prob(v_i=w_i) = 1/2$ if v_i and w_i are independent (that is, have a correlation coefficient of zero over all possible inputs). This is guaranteed for S_i and S_j if columns ϕ_i and ϕ_j in the corresponding s-box matrices sum (modulo 2) to a bent vector. This means that if changes are made to both S_i and S_j, it is still the case that the outputs of f will change with probability 1/2. This argument generalizes to any number of the s-boxes (once the corresponding output bits are independent), which proves that any change to the input of f changes each bit in the output of f with probability 1/2 over all inputs. The limit to the number of $m \times n$ s-boxes with independent corresponding output bits is a direct result of Nyberg's "perfect" s-box theorem: it is $m/2$. Therefore, if $r \leq m/2$ (where r is the number of s-boxes used for the data half in f), the simplest way to achieve the independence is to choose the corresponding columns in the s-box matrices such that they are the columns of an $m \times m/2$ "perfect" s-box. Note that key/ciphertext highest-order SAC imposes no requirement beyond that needed for plaintext/ciphertext highest-order SAC because of the definition of f.

Remark: In practice, close proximity to highest-order SAC appears to be readily achieved after a number of rounds without the requirement that operations b, c, and d be XOR addition and even without the requirement that perfect s-boxes be used as the columns for corresponding output bits. ♦

Assertion 6: For appropriate design choices, CAST ciphers exhibit close proximity to *highest-order BIC* for both plaintext and key changes.

Discussion: A similar argument to the one above can be used to show that close proximity to highest-order BIC can be achieved for both plaintext and key changes when operations b, c, and d are XOR addition. Again, however, in practice it appears that this property is readily achieved after several rounds whether or not XOR addition is used as the binary operation. ♦

3.4.1. Operation "*a*" and Intrinsic Immunity to Attacks

As discussed previously, the number of rounds and the properties of the round function s-boxes can be chosen to guarantee *computational* immunity to differential and linear cryptanalysis. In the following subsections we discuss the proposal that extra work in the round function – specifically, some care in the choice of operation

"*a*" – can conceivably give *intrinsic* immunity to these attacks (in that these attacks can no longer be mounted); see subsection 3.4.2 (and also 4.2).

3.4.1.1. Review of Differential and Linear Cryptanalysis

Differential and linear cryptanalysis (chosen- and known-plaintext attacks, respectively) are similar in flavour in that both rely on s-box properties to formulate an attack on a single s-box. Each then defines a *general approach* to attacking the round function and builds on this approach to formulate a variety of *particular* round function attacks. These particular attacks are then extended to create a number of characteristics for the overall cipher. The most successful characteristic (that is, the one with highest probability) is essentially the principal differential or linear attack on the cipher, and the cipher is theoretically broken if the work factor for this principal attack is less than the work factor for exhaustive search of the key space (even if the attack requires an impractical amount of chosen or known plaintext).

3.4.1.2. A Brief Description of Differential Cryptanalysis

In differential cryptanalysis the s-box property which is exploited is its "input XOR" to "output XOR" mapping. It was noted by Biham and Shamir that for a given pattern of change in the input of the DES s-boxes, some output change patterns are much more likely to occur than others, over the space of all possible inputs. (This analysis was repeated for all possible input change patterns, so that an XOR table could be created for each s-box [8].) Thus, for any specific input, a given change to that input will cause a given output change with a certain probability. The attack takes advantage of the large probabilities (some as high as 0.25) which occur in some s-boxes.

The *general* round attack which uses the above mapping is as follows. In DES the input to the round is the right half of the data, R, and the subkey, K, for that round. After an expansion step for R, the data and key are XOR'ed together, so that the input to the set of s-boxes is $X = E(R) \oplus K$. The output of the set of s-boxes is $Y = S(X)$, which is then permuted to form the output of the round function $R' = P(Y)$. Since the expansion step is linear, two data vectors R_1 and R_2 which differ by ΔR (so that $\Delta R = R_1 \oplus R_2$), produce $\Delta X = X_1 \oplus X_2 = E(R_1) \oplus K \oplus$

$E(R_2) \oplus K = E(R_1) \oplus E(R_2) = E(\Delta R)$ during two encryptions with the same key. It is therefore not difficult to choose ΔR such that ΔX has a desired pattern. Furthermore, since the permutation step is also linear, it is not difficult to find a ΔY such that $\Delta R' = P(Y_1) \oplus P(Y_2) = P(\Delta Y)$ has a desired pattern.

The implication of the above comments is that it is possible to choose a particular ΔR such that a particular ΔX occurs (one which presents a desired set of input XORs to a desired set of s-boxes). With a known probability, a particular ΔY will result (a desired set of output XORs from the s-boxes), which will be reflected in a particular $\Delta R'$. Finding ΔR pairs which result in useful $\Delta R'$ pairs constitutes the set of *particular* round function attacks on the cipher. In this context, a $\Delta R'$ pair is "useful" if it can act as a desired ΔR pair in the following round, so that particular round function attacks can be iterated and concatenated into characteristics with high overall probability.

3.4.1.3. A Brief Description of Linear Cryptanalysis

In linear cryptanalysis the s-box property which is exploited is linearity. It was noted by Matsui that for some of the DES s-boxes, the XOR sum of a subset of the input bits is equal to the XOR sum of a subset of the output bits significantly more or less often than half the time, over the space of all possible inputs. (This analysis was repeated for all possible input and output subsets, so that a distribution table could be created for each s-box [31].) Thus, for any specific input, the input sum will be equal to the output sum with a certain probability. The attack takes advantage of the fact that in some s-boxes the probabilities are significantly different from the ideal of 0.5 (some as low as 0.19).

The *general* round attack which uses the linearity property is as follows. For DES the values R, K, X, Y, and R', and the operations $E(\bullet)$ and $P(\bullet)$ are as discussed above. With a probability given by the distribution tables, a subset of the bits of Y will sum to a value (zero or one) which is equal to the sum of a subset of the bits of X. Thus, $\Sigma(Y) = \Sigma(X)$, where the operation $\Sigma(\bullet)$ is taken to mean the XOR sum of a specific subset of the bits in the argument. This implies that $\Sigma(Y) = \Sigma(E(R) \oplus K) = \Sigma(E(R)) \oplus \Sigma(K)$, and so $\Sigma(K) = \Sigma(E(R)) \oplus \Sigma(Y)$. Now, knowing R immediately yields $\Sigma(E(R))$ and knowing R' immediately yields $\Sigma_p(R') = \Sigma_p(P(Y)) = \Sigma(Y)$, where the operation $\Sigma_p(\bullet)$ is defined to be the XOR sum of the

permuted indices of the subset of bits used in $\Sigma(\bullet)$. Therefore, knowledge of R and R' can be used to solve for the equivalent of one key bit, $\Sigma(K)$, with a known probability.

The *particular* round function attacks on the cipher consist of those $\Sigma(R)$ subsets which result in useful $\Sigma(R')$ subsets. In this context, a $\Sigma(R')$ subset is "useful" if it can be XOR'ed with a desired $\Sigma(L)$ subset from the previous round (where L is the left half of the data) to yield a desired $\Sigma(R)$ subset for the following round, so that particular round function attacks can be iterated and concatenated into characteristics with high overall probability.

3.4.2. Modification of Operation "*a*"

The goal behind modifying the round function is to eliminate the possibility of both differential and linear cryptanalytic attacks against the cipher. This is done by inserting a nonlinear, key-dependent operation before the s-box lookup to effectively mask the inputs to the set of s-boxes. If these inputs are well "hidden", then s-box properties (such as the input XOR to output XOR mapping, or linearity) cannot be exploited in a general round function attack because the actual inputs to the s-boxes will not be known.

More specifically, the following modification to the round function f is proposed:

$$f(R, K) = f(R, K_1, K_2) = S(a(R \oplus K_1, K_2))$$

where $a(\bullet, \bullet)$ is an operation with properties as defined below. For DES, the expansion operation can be placed either around R or around $(R \oplus K_1)$ – that is, $f(R, K) = S(a(E(R) \oplus K_1, K_2))$ or $f(R, K) = S(a(E(R \oplus K_1), K_2))$ – depending on whether K_1 is 32 or 48 bits in length. As well, the permutation operation can be placed around $S(\bullet)$ as is done in the current round definition.

Several properties are required of the function $a(\bullet, \bullet)$. These will be discussed below, but they are enumerated here for reference.

(1) The subset sum operation must not be distributive over $a(\bullet, \bullet)$.

(2) $a(\bullet, \bullet)$ must represent a nonlinear mapping from its input to its output, so that any linear change in either input leads to a nonlinear change in the output vector.

(3) $a(\bullet, \bullet)$ must be relatively simple to implement in software (in terms of code size and complexity).

(4) $a(\bullet, \bullet)$ must execute efficiently (no more slowly than the remainder of the round function, for example).

(5) $a(\bullet, \bullet)$ must effectively "hide" its R (or $E(R)$) input if K_1 and K_2 are unknown (in the sense that there must be no way to cancel the effect of the keys in the round function using an operation on a single R value or a pair of R values).

A function which appears to encompass all the properties listed above is modular multiplication, for an appropriate choice of modulus. If R, K_1, and K_2 are 32 bits in length, two candidate moduli are $(2^{32} - 1)$ and $(2^{32} + 1)$. Meijer [33] has given a simple algorithm to carry out multiplication modulo $(2^{32} - 1)$ in a high-level language using only 32-bit registers, and has shown that multiplication with this modulus is a "complete" operation (in that every input bit has the potential to modify every output bit [26]), so that this modulus appears to satisfy nonlinearity, simplicity, and data hiding. However, this modulus does not satisfy the fifth property ideally, since zero always maps to zero, and $(2^{32} - 1)$ always maps to either $(2^{32} - 1)$ or zero (depending on the implementation), regardless of the key in use. (Note, however, that in a practical implementation it is a simple matter to ensure that the computed subkey K_2 is never equal to 0 or to $(2^{32} - 1)$, and masking R with K_1 ensures that it is not easy for the cryptanalyst to choose R such that $(R \oplus K_1)$ is equal to 0 or to $(2^{32} - 1)$.)

The modulus $(2^{32} + 1)$ may be a better choice with respect to property five than $(2^{32} - 1)$ if either of two simple manipulations are performed. Firstly, each input can be incremented by one, so that the computation is actually done with $(R+1)$ and $(K+1)$. Thus the arguments belong to the set $[1, 2^{32}]$ rather than $[0, 2^{32} - 1]$, avoiding both the zero and the $(2^{32} + 1)$ "fixed point" inputs. Alternatively, the inputs can be left as is (so that the computation is done with R and K), with only the zero input mapped to the value 2^{32} (and the 2^{32} output mapped back to zero). Implementation of multiplication using this modulus is thus only slightly more difficult using a high-level language with 32-bit registers than for the modulus $(2^{32} - 1)$, and on platforms where the assembly language instructions give access to the full 64-bit result of a 32-bit multiply operation, the modular reduction can be accomplished quite simply and efficiently. Furthermore, as for $(2^{32} - 1)$, multiplication with this modulus represents a nonlinear mapping from input to output.

In order to ensure that the modular multiplication does not perform badly with respect to property five, it is necessary that the subkey K_2 be relatively prime to the modulus. Thus, when the subkeys are being generated, the K_2 used in each round must not have 3, 5, 17, 257, or 65537 as factors if the modulus $n = (2^{32} - 1)$, and must not have 641 or 6700417 as factors if $n = (2^{32} + 1)$.

Finally, it appears that either modulus can be used to satisfy property one, since the subset sum operation is not distributive over modular multiplication.

3.4.2.1. Making the Round Function Intrinsically Immune to Differential Cryptanalysis

Property five listed above prevents a differential attack as described by Biham and Shamir, and property two prevents a simple modification to their description. Recall the equation given in Section 3.4.1.2:

$$\Delta X = X_1 \oplus X_2 = E(R_1) \oplus K \oplus E(R_2) \oplus K = E(R_1) \oplus E(R_2) = E(\Delta R)$$

during two encryptions with the same key. This is the critical component of the differential attack because it shows that the XOR sum of two data inputs (R_1 and R_2) completely determines the input XOR for the round s-boxes. This is why this attack would ideally be mounted using chosen plaintext (so that the cryptanalyst can select the input XORs which will construct the highest-probability characteristic). Property five prevents such an attack by the requirement that no operation on a pair of R values can cancel the effect of the key. Modular multiplication appears to achieve property five in the modified equation

$$\begin{aligned}\Delta X &= X_1 \oplus X_2 \\ &= a(R_1 \oplus K_1, K_2) \ \oplus \ a(R_2 \oplus K_1, K_2) \\ &= (((R_1 \oplus K_1) * K_2) \bmod n) \ \oplus \ (((R_2 \oplus K_1) * K_2) \bmod n)\end{aligned}$$

since knowledge of R_1 and R_2 does not seem to reveal ΔX if K_1 and K_2 are not known. Thus, the input XOR to output XOR mapping of the round s-boxes cannot be exploited through knowledge/choice of R_1 and R_2.

Modular multiplication also appears to satisfy property two because it is not obvious that any simple modification to the differential attack will cause knowledge of R_1 and R_2 to reveal information about ΔX if K_1 and K_2 are not known. This is not true of arbitrary operations which may be proposed for $a(\bullet, \bullet)$. For example, if

$a(\bullet, \bullet)$ is real addition (modulo n), then re-defining ΔX to be subtraction (modulo n) yields

$$\begin{aligned}\Delta X &= (X_1 - X_2) \bmod n \\ &= (a(R_1 \oplus K_1, K_2) - a(R_2 \oplus K_1, K_2)) \bmod n \\ &= ((((R_1 \oplus K_1) + K_2) \bmod n) - (((R_2 \oplus K_1) + K_2) \bmod n)) \bmod n \\ &= ((R_1 \oplus K_1) - (R_2 \oplus K_1)) \bmod n\end{aligned}$$

In such a situation the difference between R_1 and R_2 (XOR or real subtraction) reveals a significant amount of information about ΔX which may be used in subsequent rounds to construct a characteristic.

3.4.2.2. Making the Round Function Intrinsically Immune to Linear Cryptanalysis

Property one given above prevents a linear attack as described by Matsui. Recall the equation given in Section 3.4.1.3:

$$\Sigma(Y) = \Sigma(X) = \Sigma(E(R) \oplus K) = \Sigma(E(R)) \oplus \Sigma(K)$$

$$\text{Therefore, } \Sigma(K) = \Sigma(E(R)) \oplus \Sigma(Y)$$

This is the critical component of the linear attack because the distributive nature of the subset sum operation $\Sigma(\bullet)$ over the XOR operation allows the equivalent of one key bit to be computed using only knowledge of $\Sigma(E(R))$ and $\Sigma(Y)$. This is why this attack would typically be mounted using known plaintext (so that the cryptanalyst can use knowledge of $\Sigma(plaintext)$ and $\Sigma(ciphertext)$ to work through intermediate rounds to solve for various key bits). Property one prevents such an attack by the requirement that $\Sigma(\bullet)$ not be distributive over $a(\bullet, \bullet)$. Modular multiplication appears to achieve this requirement, as seen in the modified equation

$$\Sigma(Y) = \Sigma(X) = \Sigma(((R \oplus K_1) * K_2) \bmod n)$$

since it appears that this equation cannot be rearranged in any way to solve for subset sums of K_1 and K_2 given only subset sums of R and Y. (Note that either $E(R)$ or $E(R \oplus K_1)$ may be substituted in the above equation, if required.)

3.4.3. Implementing Operation "*a*" in a CAST Cipher

A CAST cipher implemented with a blocksize and keysize of 64 bits, four 8×32 s-boxes $S_1...S_4$ in the round function, and 32-bit subkeys in each round, has been shown to have a work factor for differential and linear attacks which is greater than exhaustive search of the key space if 12 or more rounds are used. If operations *a*, *b*, *c*, and *d* are all XOR addition, the round function *f* may be computed simply as:

$$f(R, K) = S_1(B^{(1)}) \oplus ... \oplus S_4(B^{(4)})$$

where $B = R \oplus K$ and $B^{(j)}$ is the j^{th} byte of *B*. Application of the technique described in this section yields the modified computation of operation "*a*", where *f* remains identical but *B* is now computed as

$$B = ((R \oplus K_1) * K_2) \bmod n.$$

Examination of the assembly language instructions required for the modular multiplication step alone (using either $(2^{32} - 1)$ or $(2^{32} + 1)$ as the modulus) shows that multiplication takes approximately the same amount of time as the remainder of the round on a Pentium-class PC, so that there is a performance impact of about a factor of two, compared with a version of CAST where operation "*a*" is simple XOR addition.

4. Alternative Operations and Design Choices

A number of options are available both for the round function operations and for the framework design which do not appear to compromise security and do not degrade encryption / decryption perfomance of the resulting cipher. In fact, for some choices it appears that security or performance may be enhanced, thus motivating the use of these alternatives in practice.

4.1. Binary Operations in the Round Function

Throughout this paper the operations *a*, *b*, *c*, and *d* in the round function have been given as the eXclusive-OR of two binary quantities (i.e., addition modulo 2). It should be clear, however, that other binary operations may be used instead.

Particularly attractive are addition and subtraction modulo 2^{32}, since these operations take no more time than XOR and so will not degrade encryption / decryption performance in any way. Experimental evidence suggests that using such alternative operations may significantly increase security against linear cryptanalysis [51], but in any case appears to make both linear and differential characteristics more difficult to find.

4.2. Extension to Operation "*a*"

Discussed in the previous section was the proposal to add extra computation (using extra key bits) to the operation "*a*" in the round function. The specific computation suggested was multiplication with another 32-bit subkey using a modulus of either $(2^{32} - 1)$ or $(2^{32} + 1)$. However, it was noted that this suggestion can degrade performance by as much as a factor of two. An alternative operation which appears to be quite attractive is rotation (i.e., circular shifting) by a given number of bits. This operation is similar to the central operation of the cipher RC-5, except that this is a key-dependent rotate (controlled by a 5-bit subkey) rather than a data-dependent rotate, since data-dependent rotation may be less appropriate for a Feistel-type structure.

The extended "*a*" operation for a CAST cipher with a 64-bit blocksize is then

$$a(R, K) = a(R, K_1, K_2) = ((R \bullet K_1) <<< K_2),$$

where "•" is any binary operation (such as XOR or addition modulo 2^{32}), "<<<" is the circular left shift operator, K_1 is a 32-bit subkey, and K_2 is a 5-bit subkey. The primary advantage of the rotation operation over modular multiplication is speed: on typical computing platforms an n-bit rotation ($0 \le n \le 31$) can be accomplished in a small number of clock cycles, thus causing minor performance degradation in the overall cipher. Rotation satisfies property (1) from Section 3.4.2 because it prevents a linear attack as described by Matsui for all cases except the extreme case where the input subset considered consists of the full set of input bits. It is highly unlikely that this extreme case applied in every round of an N-round cipher will describe a successful linear characteristic for the cipher.

4.3. Non-Uniformity within the Round Function

The discussion thus far implies that the binary operation in *a*, *b*, *c*, and *d* must be the same in all four instances (e.g., XOR). However, there is no reason that this needs to be the case. For example, it would be perfectly acceptable for *b* and *d* to use addition modulo 2^{32} while *c* uses XOR (this is precisely the combination used in the Blowfish cipher [44]). Certainly many variations are possible, and while it is not clear that any one variation is significantly better than any other, it does appear to be the case that the use of different operations within *a*, *b*, *c*, and *d* can add to the security of the overall cipher (note that the IDEA cipher has long advanced the conviction that operations over different groups contribute to cipher security [29]).

4.4. Non-Uniformity From Round to Round

Another design option is to vary the definition of the round function itself from round to round. Thus, in an *N*-round cipher there may be as many as *N* distinct rounds, or there may be a smaller number of distinct rounds with each type of round being used a certain number of times. The variations in the round definitions may be due to the kinds of options mentioned in the previous subsection or may be more complex in nature.

Whether the idea of a number of distinct rounds [50] in a cipher adds in any significant way to its cryptographic security is an open question. However, it certainly appears to complicate the analysis of the cipher, which is sometimes effective in complicating the cryptanalysis of the cipher.

5. An Example CAST Cipher

In order to facilitate detailed analysis of the CAST design procedure, and as an aid to understanding the procedure itself, an example CAST cipher (an output of the design procedure described in this paper) is provided in this section (with further details given in the Appendix). This 12-round cipher has a blocksize of 64 bits and a keysize of 80 bits; it uses the rotation operation to provide intrinsic immunity to linear and differential attacks; it uses a mixture of XOR, addition and subtraction

(modulo 2^{32}) for the operations *a*, *b*, *c*, and *d* in the round function; and it uses three variations of the round function itself throughout the cipher. Finally, the 8×32 s-boxes used in the round function each have a minimum nonlinearity of 74 and a maximum entry of 2 in the difference distribution table.

This example cipher appears to have cryptographic strength in accordance with its keysize (80 bits) and has very good encryption / decryption performance: over 1 MByte/sec on a 486-DX2 66MHz PC, and over 2.5 MBytes/sec on a 90 MHz Pentium.

5.1. Pairs of Round Keys

This instance of a CAST cipher uses a pair of subkeys per round; a 32-bit quantity K_m is used as a "masking" key and a 5-bit quantity K_r is used as a "rotation" key.

5.2. Non-Identical Rounds

Three different round functions are used in this example CAST cipher. The rounds are as follows (where "D" is the original input to the *f* function and "I_a" – "I_d" are the most significant byte through least significant byte of I, respectively). Note that "+" and "-" are addition and subtraction modulo 2^{32}, "^" is bitwise XOR, and "<<<" is the circular left-shift operation.

```
Type 1:  I = ((Kmi + D) <<< Kri)
         f = ((S1[Ia] ^ S2[Ib]) - S3[Ic]) + S4[Id]

Type 2:  I = ((Kmi ^ D) <<< Kri)
         f = ((S1[Ia] - S2[Ib]) + S3[Ic]) ^ S4[Id]

Type 3:  I = ((Kmi - D) <<< Kri)
         f = ((S1[Ia] + S2[Ib]) ^ S3[Ic]) - S4[Id]
```

Rounds 1, 4, 7, and 10 use *f* function Type 1.
Rounds 2, 5, 8, and 11 use *f* function Type 2.
Rounds 3, 6, 9, and 12 use *f* function Type 3.

5.3. Key Schedule

Let the primary 80-bit key be $A = a1a2a3a4a5a6a7a8a9a0$, where each a_i is a byte.

Let $K_{m1}, \ldots, K_{m12}$ be twelve 32-bit masking subkeys (one per round).
Let $K_{r1}, \ldots, K_{r12}$ be twelve 32-bit rotate subkeys (one per round); only the least significant 5 bits are used in each round.

See Appendix A for a detailed description of how to generate K_{mi} and K_{ri} from A.

5.4. Substitution Boxes

This example CAST cipher uses eight substitution boxes: s-boxes S1, S2, S3, and S4 are round function s-boxes; S5, S6, S7, and S8 are key schedule s-boxes.

See Appendix A for the s-box contents.

6. Conclusions

The CAST design procedure can be used to produce a family of encryption algorithms which are provably resistant to differential cryptanalysis, linear cryptanalysis, and related-key cryptanalysis. CAST ciphers also possess a number of other desirable cryptographic properties and have good encryption / decryption speed on common computing platforms.

Analysis of the procedure described in this paper by members of the cryptologic community is strongly encouraged so as to increase confidence in the various aspects of the design presented.

7. References

[1] C. M. Adams, *A Formal and Practical Design Procedure for Substitution-Permutation Network Cryptosystems*, Ph.D. Thesis, Department of Electrical Engineering, Queen's University, 1990.

[2] C. M. Adams and S. E. Tavares, *The Use of Bent Sequences to Achieve Higher-Order Strict Avalanche Criterion in S-Box Design*, Technical Report TR 90-013, Dept. of Electrical Engineering, Queen's University, Kingston, Ontario, Jan., 1990.

[3] C. M. Adams and S. E. Tavares, *Generating and Counting Binary Bent Sequences*, IEEE Transactions on Information Theory, vol. IT-36, 1990, pp.1170-1173.

[4] C. M. Adams, *On immunity against Biham and Shamir's "differential cryptanalysis"*, Information Processing Letters, vol.41, Feb.14, 1992, pp.77-80.

[5] C. M. Adams and S. E. Tavares, *Designing S-Boxes for Ciphers Resistant to Differential Cryptanalysis*, Proceedings of the 3rd Symposium on the State and Progress of Research in Cryptography, Rome, Italy, Feb., 1993, pp.181-190.

[6] C. M. Adams, *Simple and Effective Key Scheduling for Symmetric Ciphers*, in the Workshop Record of the Workshop on Selected Areas in Cryptography (SAC 94), May 5-6, 1994, pp.129-133.

[7] C. M. Adams, *Designing DES-Like Ciphers with Guaranteed Resistance to Differential and Linear Attacks*, in the Workshop Record of the Workshop on Selected Areas in Cryptography (SAC 95), May 18-19, 1995, pp.133-144.

[8] E. Biham and A. Shamir, *Differential Cryptanalysis of DES-Like Cryptosystems*, Journal of Cryptology, vol.4, 1991, pp.3-72.

[9] E. Biham and A. Shamir, *Differential Cryptanalysis of FEAL and N-Hash*, in Advances in Cryptology: Proc. of Eurocrypt '91, Springer-Verlag, 1992, pp.1-16.

[10] E. Biham and A. Shamir, *Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI, and Lucifer*, Advances in Cryptology: Proc. of CRYPTO '91, Springer-Verlag, 1992, pp.156-171.

[11] E. Biham and A. Shamir, *Differential Cryptanalysis of the full 16-round DES*, in Advances in Cryptology: Proc. of CRYPTO '92, Springer-Verlag, 1993, pp.487-496.

[12] E. Biham, *Differential Cryptanalysis of Iterated Cryptosystems*, Ph.D. Thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.

[13] E. Biham, *New Types of Cryptanalytic Attacks Using Related Keys*, in Advances in Cryptology: Proc. of Eurocrypt '93, Springer-Verlag, 1994, pp.398-409.

[14] L. Brown, J. Pieprzyk, and J. Seberry, *LOKI – A Cryptographic Primitive for Authentication and Secrecy Applications*, Advances in Cryptology: Proc. of AUSCRYPT 90, 1990, pp.229-236.

[15] L. Brown, M. Kwan, J. Pieprzyk, and J. Seberry, *Improving Resistance to Differential Cryptanalysis and the Redesign of LOKI*, Advances in Cryptology: Proc. of ASIACRYPT 91.

[16] D. Coppersmith, *The Real Reason for Rivest's Phenomenon,* in Advances in Cryptology: Proc. of CRYPTO '85, Springer-Verlag, New York, 1986, pp.535-536.

[17] D. Coppersmith, *The Data Encryption Standard (DES) and its Strength Against Attacks*, IBM Report RC 18613.

[18] M. Dawson and S. E. Tavares, *An Expanded Set of S-Box Design Criteria Based on Information Theory and its Relation to Differential-Like Attacks*, in Advances in Cryptology: Proc. of Eurocrypt '91, Springer-Verlag, 1992, pp.352-367.

[19] H. Feistel, *Cryptography and Computer Privacy*, Scientific American, vol.228, 1973, pp.15-23.

[20] H. Feistel, W. Notz, and J. L. Smith, *Some Cryptographic Techniques for Machine-to-Machine Data Communications*, Proceedings of the IEEE, vol.63, 1975, pp.1545-1554.

[21] E. Grossman and B. Tuckerman, *Analysis of a Feistel-Like Cipher Weakened by Having No Rotating Key*, Technical Report RC 6375, IBM, 1977.

[22] H. M. Heys and S. E. Tavares, *Cryptanalysis of tree-structured substitution-permutation networks*, IEE Electronics Letters, vol.29, #1, 1993, pp.40-1.

[23] H. M. Heys, *The Design of Substitution-Permutation Network Ciphers Resistant to Cryptanalysis*, Ph.D. Thesis, Department of Electrical and Computer Engineering, Queen's University, 1994.

[24] H. M. Heys and S. E. Tavares, *On the security of the CAST encryption algorithm*, in Canadian Conference on Electrical and Computer Engineering, Halifax, Nova Scotia, Canada, Sept., 1994, pp.332-335.

[25] B. S. Kaliski Jr., R. L. Rivest, and A. T. Sherman, *Is the Data Encryption Standard a Group? (Results of Cycling Experiments on DES),* Journal of Cryptology, vol.1-1, 1988, pp.3-36.

[26] J. B. Kam and G. I. Davida, *Structured Design of Substitution-Permutation Encryption Networks*, IEEE Transactions on Computers, vol. C-28, 1979, pp.747-753.

[27] L. R. Knudsen, *Iterative Characteristics of DES and s^2-DES*, in Advances in Cryptology: Proc. of CRYPTO '92, Springer-Verlag, 1993, pp.497-511.

[28] J. Lee, H. M. Heys, and S. E. Tavares, *On the Resistance of the CAST Encryption Algorithm to Differential Cryptanalysis*, in the Workshop Record of the Workshop on Selected Areas in Cryptography (SAC 95), May 18-19, 1995, pp.107-120.

[29] X. Lai and J. L. Massey, *A Proposal for a New Block Encryption Standard*, Advances in Cryptology: Proc. of EUROCRYPT 90, Springer-Verlag, 1991, pp.389-404.

[30] X. Lai, J. L. Massey, and S. Murphy, *Markov Ciphers and Differential Cryptanalysis*, in Advances in Cryptology: Proc. of Eurocrypt '91, Springer-Verlag, 1991, pp.17-38.

[31] M. Matsui, *Linear Cryptanalysis Method for DES Cipher*, in Advances in Cryptology: Proc. of Eurocrypt '93, Springer-Verlag, 1994, pp.386-397, 1994.

[32] W. Meier and O. Staffelbach, *Nonlinearity Criteria for Cryptographic Functions*, in Advances in Cryptology: Proc. of Eurocrypt '89, Springer-Verlag, 1990, pp.549-562.

[33] H. Meijer, Multiplication-Permutation Encryption Networks, Technical Report #85-171, Queen's University, Department of Computing and Information Science, 1985.

[34] S. Miyaguchi, A. Shiraishi, and A. Shimizu, *Fast Data Encryption Algorithm Feal-8*, Review of electrical communications laboratories, vol.36, #4, 1988.

[35] S. Miyaguchi, *FEAL-N specifications.*

[36] J. H. Moore and G. J. Simmons, *Cycle Structure of the DES with Weak and Semi-Weak Keys,* in Advances in Cryptology: Proc. of CRYPTO '86, Springer-Verlag, New York, 1987, pp.9-32.

[37] National Bureau of Standards (U.S.), *Data Encryption Standard (DES)*, Federal Information Processing Standards Publication 46, Jan. 15, 1977.

[38] K. Nyberg, *Constructions of bent functions and difference sets*, in Advances in Cryptology: Proc. of Eurocrypt '90, Springer-Verlag, 1991, pp.151-160.

[39] K. Nyberg, *Perfect nonlinear S-boxes*, in Advances in Cryptology: Proc. of Eurocrypt '91, Springer-Verlag, 1991, pp.378-386.

[40] L. O'Connor (personal communication).

[41] L. O'Connor, *An average case analysis of a differential attack on a class of SP-networks*, in the Workshop Record of the Workshop on Selected Areas in Cryptography (SAC 95), May 18-19, 1995, pp.121-130.

[42] B. Preneel, W. Van Leekwijck, L. Van Linden, R. Govaerts, and J. Vandewalle, *Propagation characteristics of boolean functions*, in Advances in Cryptology: Proc. of Eurocrypt '90, Springer-Verlag, Berlin, 1991, pp.161-173.

[43] V. Rijmen and B. Preneel, *On Weaknesses of Non-surjective Round Functions*, in the Workshop Record of the Workshop on Selected Areas in Cryptography (SAC 95), May 18-19, 1995, pp.100-106.

[44] B. Schneier, *The Blowfish Encryption Algorithm*, Dr. Dobb's Journal, April, 1994, pp.38-40,98-99.

[45] J. Seberry, X.-M. Zhang, and Y. Zheng, *Systematic Generation of Cryptographically Robust S-Boxes (Extended Abstract)*, in Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, Nov. 3-5, 1993, pp.171-182.

[46] C. E. Shannon, *Communication Theory of Secrecy Systems*, Bell Systems Technical Journal, vol. 28, 1949, pp.656-715.

[47] M. Sivabalan, S. E. Tavares, and L. E. Peppard, *On the Design of SP Networks from an Information Theoretic Point of View*, in Advances in Cryptology: Proc. of CRYPTO '92, Springer-Verlag, 1993, pp.260-279.

[48] A. F. Webster, *Plaintext/Ciphertext Bit Dependencies in Cryptographic Systems*, M.Sc. Thesis, Department of Electrical Engineering, Queen's University, Kingston, Ont., 1985.

[49] A. F. Webster and S. E. Tavares, *On the Design of S-Boxes*, in Advances in Cryptology: Proc. of CRYPTO '85, Springer-Verlag, New York, 1986, pp.523-534.

[50] M. Wiener (personal communication).

[51] A. Youssef (personal communication).

A. Appendix

This appendix provides full details of the example CAST cipher given in Section 5.

A.1. Key Schedule

Let the primary 80-bit key be `A = a1a2a3a4a5a6a7a8a9a0,` where each a_i is a byte.

Let $K_{m1}, \ldots, K_{m12}$ be twelve 32-bit masking subkeys (one per round).
Let $K_{r1}, \ldots, K_{r12}$ be twelve 32-bit rotate subkeys (one per round); only the least significant 5 bits are used in each round.

Let `b1...b8,` through `g1...g8` and `bw, bx, by, bz` through `gw, gx, gy, gz` be intermediate (temporary) bytes.

Let `Si[]` represent s-box *i* and let `"^"` represent XOR addition.

The subkeys are formed from the primary key as follows.

Masking subkeys:

```
b1b2b3b4 = a1a2a3a4 ^ S5[a6] ^ S6[a8] ^ S7[a5] ^ S8[a7] ^ S7[a9]
b5b6b7b8 = a5a6a7a8 ^ S5[b1] ^ S6[b3] ^ S7[b2] ^ S8[b4] ^ S8[a0]
bwbxbybz = b5b6b7b8 ^ S5[a9] ^ S6[a0] ^ S7[a9] ^ S8[a0]

Km1  = S5[b1] ^ S6[b2] ^ S7[b8] ^ S8[b7] ^ S5[bw]
Km2  = S5[b3] ^ S6[b4] ^ S7[b6] ^ S8[b5] ^ S6[bx]
Km3  = S5[b5] ^ S6[b6] ^ S7[b4] ^ S8[b3] ^ S7[by]
Km4  = S5[b7] ^ S6[b8] ^ S7[b2] ^ S8[b1] ^ S8[bz]

c1c2c3c4 = b1b2b3b4 ^ S5[b5] ^ S6[b7] ^ S7[b6] ^ S8[b8] ^ S7[bw]
c5c6c7c8 = b5b6b7b8 ^ S5[c2] ^ S6[c4] ^ S7[c1] ^ S8[c3] ^ S8[bx]
cwcxcycz = c5c6c7c8 ^ S5[bw] ^ S6[bx] ^ S7[by] ^ S8[bz]

Km5  = S5[c4] ^ S6[c3] ^ S7[c5] ^ S8[c6] ^ S5[cw]
Km6  = S5[c2] ^ S6[c1] ^ S7[c7] ^ S8[c8] ^ S6[cx]
Km7  = S5[c8] ^ S6[c7] ^ S7[c1] ^ S8[c2] ^ S7[cy]
Km8  = S5[c6] ^ S6[c5] ^ S7[c3] ^ S8[c4] ^ S8[cz]

d1d2d3d4 = c1c2c3c4 ^ S5[c5] ^ S6[c7] ^ S7[c6] ^ S8[c8] ^ S7[cw]
d5d6d7d8 = c5c6c7c8 ^ S5[d2] ^ S6[d4] ^ S7[d1] ^ S8[d3] ^ S8[cx]
dwdxdydz = d5d6d7d8 ^ S5[cw] ^ S6[cx] ^ S7[cy] ^ S8[cz]

Km9  = S5[d4] ^ S6[d3] ^ S7[d5] ^ S8[d6] ^ S5[dw]
Km10 = S5[d2] ^ S6[d1] ^ S7[d7] ^ S8[d8] ^ S6[dx]
Km11 = S5[d8] ^ S6[d7] ^ S7[d1] ^ S8[d2] ^ S7[dy]
```

```
Km12 = S5[d6] ^ S6[d5] ^ S7[d3] ^ S8[d4] ^ S8[dz]
```

Rotate subkeys:

```
e1e2e3e4 = d1d2d3d4 ^ S5[d6] ^ S6[d8] ^ S7[d5] ^ S8[d7] ^ S7[dw]
e5e6e7e8 = d5d6d7d8 ^ S5[e1] ^ S6[e3] ^ S7[e2] ^ S8[e4] ^ S8[dx]
ewexeyez = e5e6e7e8 ^ S5[dw] ^ S6[dx] ^ S7[dy] ^ S8[dz]

Kr1  = S5[e1] ^ S6[e2] ^ S7[e8] ^ S8[e7] ^ S5[ew]
Kr2  = S5[e3] ^ S6[e4] ^ S7[e6] ^ S8[e5] ^ S6[ex]
Kr3  = S5[e5] ^ S6[e6] ^ S7[e4] ^ S8[e3] ^ S7[ey]
Kr4  = S5[e7] ^ S6[e8] ^ S7[e2] ^ S8[e1] ^ S8[ez]

f1f2f3f4 = e1e2e3e4 ^ S5[e5] ^ S6[e7] ^ S7[e6] ^ S8[e8] ^ S7[ew]
f5f6f7f8 = e5e6e7e8 ^ S5[f2] ^ S6[f4] ^ S7[f1] ^ S8[f3] ^ S8[ex]
fwfxfyfz = f5f6f7f8 ^ S5[ew] ^ S6[ex] ^ S7[ey] ^ S8[ez]

Kr5  = S5[f4] ^ S6[f3] ^ S7[f5] ^ S8[f6] ^ S5[fw]
Kr6  = S5[f2] ^ S6[f1] ^ S7[f7] ^ S8[f8] ^ S6[fx]
Kr7  = S5[f8] ^ S6[f7] ^ S7[f1] ^ S8[f2] ^ S7[fy]
Kr8  = S5[f6] ^ S6[f5] ^ S7[f3] ^ S8[f4] ^ S8[fz]

g1g2g3g4 = f1f2f3f4 ^ S5[f5] ^ S6[f7] ^ S7[f6] ^ S8[f8] ^ S7[fw]
g5g6g7g8 = f5f6f7f8 ^ S5[g2] ^ S6[g4] ^ S7[g1] ^ S8[g3] ^ S8[fx]
gwgxgygz = g5g6g7g8 ^ S5[fw] ^ S6[fx] ^ S7[fy] ^ S8[fz]

Kr9  = S5[g4] ^ S6[g3] ^ S7[g5] ^ S8[g6] ^ S5[gw]
Kr10 = S5[g2] ^ S6[g1] ^ S7[g7] ^ S8[g8] ^ S6[gx]
Kr11 = S5[g8] ^ S6[g7] ^ S7[g1] ^ S8[g2] ^ S7[gy]
Kr12 = S5[g6] ^ S6[g5] ^ S7[g3] ^ S8[g4] ^ S8[gz]
```

A.2. Substitution Boxes

This example CAST cipher uses eight substitution boxes: s-boxes S1, S2, S3, and S4 are round function s-boxes; S5, S6, S7, and S8 are key schedule s-boxes. The s-box contents are written in hexadecimal and are to be read left to right, top to bottom.

A.2.1. S-Box S1

```
0xc6b00b1e, 0xd08d094d, 0x959cb449, 0x8d531db4, 0x4be173c6, 0x5768439b,
0x128a2452, 0x0f3ff37a, 0xd13e2600, 0xcd088c51, 0x8e296754, 0x9f7f55ff,
0x5faef124, 0x4ed3e8bd, 0x08a43a43, 0x1b77f7fb, 0xc0a9ed79, 0x7281c4b7,
0x4b776caa, 0xff75ab5d, 0xf91a4cf9, 0x4a7a7a4d, 0x71514583, 0xcbd5d1d5,
0xcaa98800, 0x7576516c, 0x4150fdfb, 0xfb37f9fa, 0xf657b43f, 0x4f3ff3ef,
0x7c612b9d, 0xcf7ffdfb, 0x80ea38a2, 0x68922405, 0xda4fa8f7, 0x3c8a46c1,
0xd21cdbce, 0x3194b822, 0x8498a509, 0x666378af, 0xb05d8ac2, 0x5264708c,
0xe8dff3ba, 0x07c9c831, 0xe359af3f, 0x0d6fabfc, 0xb5a05a25, 0x5977d3f8,
0x890c5e39, 0x84a28601, 0x8b67ff16, 0x80ac9028, 0x88e79bf2, 0x854511f5,
```

```
0x8f7fb425, 0x8b76c5ff, 0x6bca5bc1, 0x6bd93db0, 0x679a19e8, 0x627efbcb,
0x6d5f6ad8, 0x6abfdf50, 0x66a6c4f9, 0x6b7fb9f8, 0x851ac1cd, 0x539ee5ca,
0x3fa7791e, 0xee4adaa3, 0xb0da1081, 0x64722b5c, 0x0180ed45, 0xdd7d3aa9,
0x03286987, 0xd66be246, 0xbdc2aa87, 0x6cc198bc, 0x3c263a67, 0xec8925ba,
0x83bed710, 0x586d1abc, 0x8ab38c7e, 0x71898970, 0xe87ca369, 0x1d254b1e,
0x0b7d85ba, 0xf92f979a, 0x6b618a40, 0x986c1e92, 0xc99ac587, 0x3e1e14ab,
0xa2aa30b8, 0x586432ad, 0x44497b78, 0xbd6536bc, 0x273fc5ca, 0xdc6530bc,
0xc3ee7b71, 0x8b904102, 0x005567f4, 0x4f918356, 0xf8abbe8d, 0xb2ded2e5,
0x36926a4e, 0x7461b37c, 0xc2f5ce45, 0x8946951b, 0x0b15be3d, 0x443505f2,
0xf14de078, 0xbf3566fb, 0x3fba3326, 0x7a6d3ebf, 0xc3b8b63e, 0xe7bb4246,
0x384d3281, 0x12fe72eb, 0x8b0c54b5, 0xa640fe22, 0x744f7db6, 0x588f08b8,
0x28471d46, 0x00f0f3f7, 0xd44154ff, 0xf16f301c, 0x6edd219b, 0x48e51a03,
0x9cf8aefe, 0xb86574bf, 0x8d06d47d, 0x491bf432, 0xa62a7926, 0x64c7daf8,
0x5a574491, 0x9cfe7ee5, 0x7b1cf91c, 0xb6a92e10, 0x78ac797c, 0xbebeb314,
0x559ffa1e, 0x96cd88a4, 0xa63a2e77, 0x654535e1, 0x8932c728, 0x42e10a85,
0x813f9826, 0x8b1599e9, 0x72e1a3d3, 0x76e14e07, 0x40a8918b, 0x43eac70a,
0xb2e5daf9, 0xb2610e8a, 0x131fd57e, 0x16c20411, 0xeae620a3, 0xe2a12694,
0xddc56b61, 0xd6a92685, 0x2df394d0, 0x26e96085, 0xc97a6fc3, 0xa104515b,
0xebd9278e, 0x851c83ad, 0xa3aeae9c, 0xca028754, 0x8d0e7a7f, 0xeff5a7e5,
0xfbe9df9c, 0x91d2c5e8, 0xd149aec6, 0xbe79154a, 0x9ac9b069, 0xf6f97688,
0xb4362776, 0xd0e12e86, 0xcabca364, 0x1d36537e, 0x20d17279, 0xf93a7752,
0x21510485, 0xfcd5aeba, 0xcccb2967, 0x12c21880, 0x305a0c9d, 0xe86de2ec,
0xde0c0486, 0x0bea2485, 0xd6c97583, 0x03290a12, 0x3734fb8c, 0xe2e92486,
0xc153e151, 0x75f7e74a, 0xf3ee4b8e, 0x4821e210, 0xae93d829, 0x101be9dd,
0x97e9cff4, 0x2b14183c, 0x0545c904, 0xba2222fc, 0x398bc832, 0x8a8cba18,
0x6a4d1acb, 0xd8e00719, 0x5dd7d584, 0xee04183d, 0xc4da06de, 0xd7e22bd1,
0x0e1593ef, 0x1b4e51bb, 0x7d30a737, 0x6f40d522, 0xb72aeac5, 0xae273e32,
0xbfd36706, 0xa255be29, 0x76e3121f, 0x6e0d123c, 0x00245bd9, 0x1b0c1c29,
0xc9528578, 0xda0c523d, 0x8581d3ef, 0x2df9cba3, 0x1e3c1772, 0xb9f8b9c5,
0x56e63420, 0xf6b7126c, 0xc0fb4ac7, 0x620893c9, 0x86be64e4, 0x250fdf80,
0x1d7c1cbe, 0xb27a2776, 0x572200d1, 0xfb5c4470, 0xc1d3b18a, 0x6c043c3e,
0x8ff7b49c, 0xf1d068c6, 0x9c044011, 0xe4976a6e, 0x3d45343d, 0x402fbe92,
0x28041f0b, 0x5ec42a38, 0x7c28b5e5, 0x0cbbd354, 0x6228b66e, 0x170410ad,
0xcab4c13f, 0xbe8c30be, 0xd2956e74, 0xae0c163e
```

A.2.2. S-Box S2

```
0xc2ad2c5e, 0x194d87b3, 0x82c127ce, 0x56bb0629, 0x688a0d7a, 0xb726d1c2,
0x21f3de43, 0xf4cd6b22, 0x88c5295c, 0x5eba71a7, 0xcb14e9b7, 0x16b6d157,
0x24dc2fa1, 0xfa5958f5, 0x6b72774c, 0xbb2283d0, 0x069d6a56, 0x83fbdc6d,
0x11c78559, 0x9d32faea, 0x16239af7, 0x9f7254a5, 0x0618cfa8, 0x83d22ba1,
0x508f6b36, 0xde7acca9, 0x447b55a7, 0xc75ad1a1, 0x41475489, 0xcf33d385,
0x57b890e7, 0xd110d7a1, 0x61152b5e, 0x38882409, 0xd9ac7bdd, 0x88258fbd,
0x2659d784, 0x75568779, 0x9616f2b3, 0xc805baaa, 0x01c014da, 0x55670ac4,
0xbfc6df20, 0xe732d018, 0x4ff1eb6d, 0x1d72f223, 0xfc6b158c, 0xa7b3fe37,
0xa4aaa246, 0x549d7b56, 0x6d3a1124, 0x93107081, 0x597c701a, 0xaba2f99c,
0x966ed435, 0x6fa506d5, 0xc805995f, 0x3304b3b4, 0x0739aba1, 0xf662d3ee,
0x3be6a0a9, 0xceb8822e, 0xf732c5b5, 0x0d3e5645, 0xcb70db1c, 0x40d450f7,
0xb358b096, 0x3ea2f1f9, 0xa1479a2c, 0x2fbb2e31, 0xd8ee0991, 0x55d05cf0,
0x29d8fe9e, 0xa777e6e6, 0x53591efc, 0xdf2b260a, 0x4d05f86b, 0xc3c407a3,
0x326fc08c, 0xbbef3432, 0x0e505514, 0x12262b00, 0xb99a5217, 0xa4af05ab,
```

```
0xc7faa510, 0xd73f23c7, 0x7e513060, 0x6a4bb4e7, 0xd1123474, 0xcfe33bff,
0x65e6eae9, 0x7ed726c0, 0x115a0beb, 0x062e24e5, 0xafa567b7, 0xb88da0e3,
0x684cdc1c, 0x5911bb51, 0x0035c48d, 0x303c784f, 0x4fd460d6, 0x7dcb70aa,
0x2f8b2d63, 0x195ccd68, 0x9059cb18, 0xa4aad589, 0xff8b0863, 0xceaf2777,
0xb66814a3, 0x84efa545, 0xd5f6a24e, 0xe73a09d5, 0xace39d04, 0xfd40c427,
0x3d67ee62, 0x6a8d8fe2, 0x78e56ff9, 0x23ef86de, 0xeea723ff, 0xb678d983,
0x891cce1d, 0xda9d0cee, 0x1ea434e7, 0x4fef24bd, 0x5bbbd7cf, 0x07a57d7e,
0xcfaf32e7, 0x94e76107, 0x69f7a056, 0x427b29bd, 0x69fd88cc, 0x4f858823,
0x37e3a276, 0x181c52ca, 0x38e1754b, 0x19d5e52a, 0x63fe8254, 0x45d8ddad,
0x645e65b7, 0x4d8c5f11, 0x3da381a1, 0x1541fcf9, 0x3248f846, 0x16610998,
0xaffeed5e, 0x4c89506b, 0x5a9d2a5d, 0xb6027ee0, 0x695f19fb, 0x8430dead,
0x9d564aaa, 0x7cee8dad, 0x1bb5cf3e, 0xf54445a5, 0xef49d2a3, 0x0c505deb,
0xde54f681, 0x340a5d8f, 0x2c811eed, 0xc80859e9, 0xca43a756, 0xc3b6870b,
0xa292fcd7, 0xa11b01b5, 0x6f505c8c, 0x6e4c0c71, 0x0f2450b9, 0x077b34a2,
0xcaf7b6d2, 0xce2eabc2, 0xa0ad7228, 0xac085e5c, 0x60ee6869, 0x66695e2f,
0x05529a84, 0x0896747f, 0x0dc5254e, 0xfbe7fa5c, 0x96629628, 0x6820f489,
0x1060d712, 0xe4c07b94, 0x8d005835, 0x72ffa0d9, 0x33333457, 0xc81137b4,
0xac2a0dad, 0x5d685fa6, 0x22952fa5, 0xd1820424, 0xbc084bbd, 0x4640d80d,
0x6fd5a8eb, 0x241d0106, 0xa79be06b, 0xe863800c, 0x51d1cadf, 0x1f7e52c6,
0x9e035d66, 0xd7372d07, 0xdd1caa69, 0x93eab513, 0x13ec6d03, 0x5bee57b3,
0xeb85a994, 0xa3235c50, 0x24aab079, 0x69534185, 0xa8cc2de3, 0x42ab58f9,
0x3d3f02ec, 0xd0607e5e, 0x9779d9e3, 0x73825630, 0x0ae04a9d, 0xea88ed14,
0x65d76f83, 0x8b224d0c, 0xf12b9212, 0x1a225575, 0x51b6561c, 0xb2e85510,
0xcb631642, 0x2e6ad154, 0xcce5afeb, 0x7dd0e7ac, 0xa4f4bc78, 0x16fd09b8,
0x39221421, 0x892e045d, 0x59467096, 0xe9ddbc9f, 0x549196ef, 0xe01c8b70,
0x3f1f5a94, 0x8a6a56cc, 0xa6886858, 0x100bf6b6, 0xc330d2b9, 0x77e07c62,
0x0a73e5f3, 0x2dc5bad2, 0x29c09691, 0x0e42f415, 0x8e06370e, 0xa372fb29,
0xaa365000, 0x84dd8070, 0x9dd59cea, 0xbe777711, 0xba486d14, 0x9b1a570a,
0x1d37a73c, 0x3760048b, 0x3b6a4310, 0x106610b0
```

A.2.3. S-Box S3

```
0x86f5c342, 0xc231da03, 0x64140aed, 0x129ec99e, 0x3ef407ec, 0x6fcb995f,
0xe0382359, 0xb9ba0244, 0x72524815, 0x3a759e48, 0xb3491e6d, 0xcb8e4b5e,
0xe61bfda2, 0x91ec2964, 0x27dee3ca, 0x5a3ad1fe, 0x22bdbec0, 0x2dc09f7a,
0x0cdf5081, 0x12ea514f, 0x99f9ae94, 0x980a4411, 0x8fc26e5a, 0xa58ac137,
0x47aa9b46, 0x6b132788, 0x60e5aa94, 0x5fba6f72, 0xd95ebb20, 0xd14249b8,
0xfabb0177, 0xdbb2ef5b, 0x07e182db, 0x73c6cdd8, 0x6777b8e8, 0x0b91adb8,
0xf457a25b, 0x842d2285, 0x8998d5b3, 0xdbd06aa2, 0xbd2bd4d0, 0xe438849d,
0xde35b50b, 0x9d49e649, 0x59a24077, 0x2ec75a8e, 0x1b16c97d, 0x4cefb517,
0xa8560728, 0x94c66e8e, 0x7c4d1ac9, 0x7b1ad37a, 0xdea5f3ad, 0xfd1d191b,
0x13936002, 0x311f4f3d, 0x232cfff2, 0x1dafdb72, 0xde68f9bf, 0xd9822476,
0x4b18fe7b, 0x4098dc82, 0x9382d372, 0x9b986d5e, 0x02bc173a, 0x56211e7b,
0x208c9e97, 0x465edfe6, 0x7a2c93d6, 0x2b8a9d67, 0x44b125e3, 0x3d7ad47c,
0x7352cced, 0x3ac4caa2, 0x52908857, 0x0b475f24, 0x32837b58, 0x553cafdc,
0x33dff722, 0x5e034584, 0xa72d38b8, 0xd9a959d0, 0x680684bb, 0x37738535,
0xd9286aae, 0xb8d21069, 0x2f5af870, 0x7573554f, 0xe6e30fbe, 0x8a4331f2,
0x30343cae, 0x5f6bfb08, 0x8dd72f98, 0xd51bdf40, 0x5e6ac50d, 0x1f7b7b21,
0xa23856a3, 0x265f49a0, 0xb3ffbcd2, 0x1f517bc0, 0x704f34a1, 0xc064a63d,
0x4d514109, 0xcf91bc9a, 0x5dbb5028, 0xe5801277, 0x6ffc3171, 0xdd003233,
0x8dfac44d, 0x2a9fdeb6, 0xaf57cd95, 0x18d7216d, 0x0c568150, 0xa126e824,
```

```
0x48845eb3, 0xde83c700, 0x1eb43557, 0x9dcdcde3, 0x534b6428, 0xf167db45,
0x63f56b0a, 0xfdf60f08, 0x1ea97dc5, 0x99d3700c, 0x5f516803, 0xc44948fa,
0x17130708, 0x8f50f924, 0x3e6c3e77, 0xb2fc2237, 0x9c5fdeda, 0x2283b18f,
0x84fffbfa, 0x1f54c56f, 0x10617eea, 0x89a3d770, 0x8b8bb8a0, 0x021ae3ca,
0x0a40eb7b, 0xb9931a68, 0x1c520414, 0xa9e5d4d1, 0x9f07974e, 0x20d709e8,
0x9bf44ad1, 0xdd7b42dc, 0xbcd4acb6, 0xe3a3ad7d, 0xe3f45286, 0xa0011424,
0xff8b9678, 0x95a33d06, 0x363747d2, 0x589a529b, 0x1ae856a6, 0x65b39345,
0x61014390, 0x21cbb10c, 0x42b2d541, 0x23ab136d, 0xb6ea7bef, 0x828935cc,
0x572cc4da, 0x7b8c55ad, 0x06985ecd, 0x3cb6de11, 0xf9810104, 0xeb49ba96,
0x8d602160, 0x9d55583b, 0x672c491c, 0x67d4367f, 0x232fb845, 0x164aa69b,
0xe38f95f9, 0xf6026905, 0x188df738, 0xadffb308, 0xc4564efb, 0x4253af4c,
0x646c0f3f, 0xcd12e98a, 0xab9a1124, 0x01b6b60c, 0xdb232663, 0x67240f45,
0x24750c8c, 0xa30b5941, 0xbb83074f, 0x30952596, 0x63cb2f44, 0xe381956c,
0x85da15f0, 0x591319b0, 0xe738b55f, 0x09bc5808, 0xbfd8903f, 0x64ea3ea8,
0xcb1787af, 0x329cfeb7, 0x3574c3a7, 0xfd5448df, 0x546682be, 0x86a5f1ed,
0x37357d11, 0xd2caad96, 0x74f97c59, 0x9b11626d, 0x219b3356, 0x96ed7b89,
0xa7f28773, 0x39c586f8, 0xdcda6943, 0x7f26bfa3, 0x60ecffad, 0xfaa55681,
0x28812cd5, 0x8635bb1e, 0xb5c63f63, 0x1a9df8c0, 0xca7728d7, 0x5a6dd80b,
0x599ceec4, 0xd89d78e8, 0x2ccc5068, 0xe8ef4e4b, 0x3c5b3f1f, 0xd0b37c2a,
0xbd7f37c8, 0x4700a556, 0x82b76a41, 0x40f79351, 0x120f5a67, 0xe312312e,
0x291a3299, 0xd8621dfa, 0x8888c380, 0x6deddd5c, 0xa8316eee, 0x5dc50280,
0x83728ebf, 0x67e0ca5d, 0x0f60f57e, 0xd83544c9, 0x9b82363a, 0x523dc28d,
0x14bdeaf1, 0xfe31dd8b, 0xe4054d64, 0x388224c0, 0x5b4b7709, 0x9ca5f2c4,
0xd0356ec8, 0x0bbb4e11, 0x58a504c1, 0x88b6fee9
```

A.2.4. S-Box S4

```
0x154b0bc2, 0x9e92acd6, 0xe8d3562e, 0x607b3270, 0xe148e878, 0x7f97f0d6,
0x18af89ad, 0x8cb5df89, 0x4a28e9c0, 0xcf75d66f, 0xaab7d57a, 0x3cb2462b,
0xb5503fbb, 0x3db35e39, 0x558ba589, 0xc784e535, 0xf190ac77, 0x278b7320,
0xae647e1f, 0x7f8f5d12, 0x6814f368, 0xbfea6e26, 0x264d12e3, 0xfff7fe37,
0x204229c2, 0xf8175a3f, 0x7eb95eff, 0xb135beec, 0xafbd2e64, 0x6eb5fe17,
0xe4d0f00b, 0x3bb53e56, 0x804a28e9, 0xe74801c9, 0xd535853d, 0xa34026c6,
0xd16d31a0, 0xb20c9b68, 0x9bd07dfc, 0xe072d02f, 0x68cc31a2, 0x016ccf1d,
0x3e970216, 0x566919c6, 0x2f8c62ff, 0x5af7fe3f, 0x622e2a72, 0x1fbfde1d,
0x636e0fa0, 0x49aa21c0, 0xd7bed210, 0xe88c49bb, 0xe7f155bc, 0xddb50597,
0x5931665e, 0x7f370e7e, 0x7559715c, 0x51f5bcb2, 0xdea5f68f, 0xffd6e11e,
0xfaa10c2c, 0xd6355e1e, 0x5f76ff1c, 0x7f89fa91, 0x9a14227a, 0x90de812a,
0xd70faf92, 0xd7a79f88, 0x1e94d1c1, 0x0150192e, 0x4ef2a450, 0x53eaf634,
0x3df59079, 0x20b8ff96, 0x6563fcc6, 0x6b7d7fd7, 0xa39e8607, 0xa36f7785,
0xf357c830, 0xe84bdc8d, 0x7fd485cf, 0x78579ad9, 0xc0b2d7a7, 0xc15b64ea,
0xfe4b8a91, 0xe02f47de, 0x509bfb5e, 0x402b938b, 0xa714903a, 0xbf437783,
0x016d7343, 0x06614715, 0x386b179d, 0x3169d7af, 0x9a860db2, 0x846117ae,
0x0e960151, 0xf9042c75, 0x8aebfc85, 0x7c948b3a, 0xb7b25818, 0x44c3f6d0,
0x250f0044, 0xdf2dfdd6, 0xbe90485b, 0x5ea962a4, 0x314b2fae, 0xc9ae207e,
0x00438b42, 0xec2bd783, 0x8c7a93ce, 0x7070e3e1, 0xeca92618, 0x8676c879,
0x996afbac, 0xfe507047, 0xa8ad7c44, 0xca78a82f, 0xc765dfa6, 0xa0eb6786,
0xd38ec8e4, 0xa6a9154e, 0xa179df37, 0xc08a18a3, 0x947665d4, 0xe1e977a6,
0xe9a8d6a0, 0x805dd72d, 0x67170782, 0x29cc85b6, 0xfa09f946, 0xb685b898,
0x17b2eeb9, 0x4c437d1e, 0x8fd5a2ce, 0xdae9d860, 0x1ed6e781, 0x59abd226,
0x9841d998, 0xc86e604b, 0x648cb1d9, 0x28695051, 0xe655a748, 0xb178e057,
```

```
0x86e6a39f, 0xb775dfe3, 0x19b0f07f, 0x2c595352, 0xff48fd0b, 0xc9386946,
0x77b99522, 0x4929f4dd, 0x5416afa2, 0x6e41505f, 0xc86b543f, 0xe3433b0d,
0x396d2205, 0x1a6bf075, 0xb380786a, 0x8d631036, 0xf5942429, 0x10160b29,
0x87edaa55, 0x75b6ae0e, 0x42953fc0, 0xa3d0b2a8, 0x2c287f9e, 0xd62ef34e,
0xfbb23fe3, 0x17ba4cd4, 0x8c690a74, 0x62bd1d86, 0x5b50e4bc, 0xad2df05f,
0x31788532, 0xc943dff7, 0x139a00c8, 0x1954ae83, 0x6068dc70, 0x7b5245db,
0xd5af537e, 0xc96fa6ff, 0xa847c89e, 0xa9e9201c, 0x268df73c, 0x27ab3152,
0x487ff8cf, 0x4da846df, 0xe971086c, 0xe0eb5074, 0x88aedafc, 0x895fd079,
0xf7b7d1c5, 0x587f57b5, 0xba2aff05, 0x1ea6ea9f, 0xf79128ff, 0x4d7b6b19,
0xa67770cc, 0x1a490e22, 0x36f461c7, 0x89990420, 0x686a0fdb, 0xc05ea648,
0x2dbdf79a, 0x894a8612, 0x7f76350e, 0xc1482610, 0x175d75d8, 0xd756c9e5,
0x8899a638, 0x4d729555, 0x76e87b0d, 0xa902bf41, 0xfe908360, 0x290a669e,
0x0cbfe9a5, 0xd6ea821c, 0x8840867c, 0x4be83d0b, 0x7144e403, 0xba482632,
0xf2297a2c, 0x2d48c631, 0x64b7f26e, 0xb1a5d96a, 0xc7cc2c12, 0x159dfc0d,
0x5bb5a987, 0x8ae020ef, 0xed08fdd9, 0x368e2148, 0x9211b9e5, 0x57801e92,
0x3c4ad833, 0xe285dbc1, 0xab60f2fe, 0x640e261c, 0x00d3c371, 0xd9731df4,
0x83a2d68f, 0x2977b8c5, 0xb1430a33, 0x127183d8, 0xe50c8579, 0x415df4b8,
0xc9ec8e99, 0x69cab61b, 0xbfa5b17b, 0x0f086751, 0x885c2e88, 0x2d0b409d,
0xd8599e6b, 0x68c88633, 0xe18f0cbf, 0x4974023a
```

A.2.5. S-Box S5

```
0x2bb1ce76, 0xa24f25c4, 0x831431d4, 0x0303db1a, 0x08db19f4, 0x8f32c2d9,
0xa9f21d00, 0x35f432ef, 0x2cb25fbf, 0xa02e5aad, 0x8d95f281, 0x1aed8191,
0x17176c34, 0x915730db, 0xb2ea75b1, 0x3d48aae2, 0x0f031db4, 0xd4ecade3,
0xcb717039, 0x196a7209, 0x91aa7df8, 0x48d732bb, 0x52c7ea11, 0x9048e28b,
0xef394063, 0x347ee412, 0x38cac74d, 0xf874a625, 0x69d5909e, 0xb4cba299,
0xb46fa239, 0x608ea099, 0x3d73d51a, 0x98daade4, 0xe6088fb9, 0x4e93582a,
0xc82ceda5, 0x7f8b52cf, 0x108ca1ee, 0xb81f4457, 0x1f973404, 0xb7cb9a2a,
0xd20508da, 0x65667ee7, 0xf6d1e7c3, 0x4848e71b, 0x35ce8218, 0x866eaf6d,
0x06dd7926, 0x3d86adbc, 0xf14dce25, 0xc73906c9, 0x12460656, 0x3796dad3,
0xe89ea68f, 0xd26793c2, 0x23005bd9, 0x1064a315, 0xd87a3298, 0xf43ca15c,
0x2b089b69, 0x022c02d8, 0xd06caa1f, 0xf46cad18, 0x070810b0, 0x18f2d342,
0x3fa9ef55, 0x3ebea5dc, 0x84626376, 0x958f1c4d, 0xb54fcb80, 0xb84dc829,
0x470fa1fe, 0x5d9324a8, 0x702cac04, 0x64505b17, 0xdeae3ae2, 0xc6eeeedd,
0xfc530776, 0xe8f154e5, 0x33bee3b2, 0x59515b74, 0xc8cca6f9, 0xadd78c8f,
0x0d178b38, 0x766aec3d, 0xe87a1c51, 0x9ef11c0d, 0x9e84b6f5, 0xf8c31e95,
0x65779d4c, 0x1cc95ca2, 0xa36cc61d, 0xd8725c1e, 0x5ad258fb, 0x3c375e9f,
0x04cea398, 0x86675360, 0xefb1513a, 0x632ea6ec, 0x14959373, 0x8732ac5b,
0xee31736e, 0x71a2be81, 0xe12eca01, 0x7a766c2d, 0x1ab87e5d, 0x9fdb2461,
0xef681d41, 0x69f5991d, 0x1977dcdf, 0x83d3517a, 0x2f642f64, 0x943b5329,
0x07f418e7, 0xa384784f, 0xfefbf4c2, 0x4b2f2455, 0xd02354cf, 0x78da6d54,
0x27b9ad0b, 0x8cd9d190, 0x10c3409b, 0xb481dbdb, 0xf1b161be, 0x58915c5f,
0xdcd17cdd, 0x78d1530e, 0x18e9ba0a, 0x579621f8, 0xb08d05ec, 0xf0b35e46,
0xbb4bc9ec, 0xfa6bf6d5, 0x1a6a2918, 0x46a56293, 0x796a5b67, 0x33b79e31,
0xdee4579d, 0x8b3cb08d, 0xc08f9958, 0x80860547, 0x6332a4cd, 0x2e90ae7e,
0x3c735928, 0xc73ca9ef, 0x3ea05561, 0xcad32615, 0x62323980, 0x99460687,
0x673faec9, 0x81d9a697, 0x38e9456f, 0xc7aee50e, 0x2b9b66d5, 0xcba4a719,
0x7c6d34a6, 0x879aa685, 0x65b7e261, 0x9396a505, 0x0a2b4122, 0x4943a9d8,
0x11d9fa81, 0x5d6b5d76, 0x3bb468e9, 0x689a56c3, 0x215491f6, 0x6f0e543b,
0xae4f7098, 0xe452deb6, 0xa57c8cc6, 0xf0ff8ffb, 0x8141e7fb, 0xdf916287,
```

```
0x86163764, 0xd5f6abe1, 0x31adddfe, 0x2a16a9b0, 0x20d4aa7d, 0x34c8d2d5,
0xe1d6177a, 0xe44fdeef, 0xff26e657, 0xe7b697ce, 0xf0d01e91, 0xe3b42209,
0xef23b200, 0xe3a46060, 0x3eb8ca05, 0x3775b6c4, 0x23b48e47, 0x2734a894,
0x2baf9b33, 0xd2d42881, 0x73cf2492, 0x92f1df7f, 0xc80d4c91, 0x3f29d7be,
0x99280067, 0x74e367aa, 0xed285ad9, 0x11f51fcb, 0xbca2f6e7, 0x4a7e95f4,
0x16c93071, 0xe8c024be, 0x527429f5, 0xa4d6af86, 0x1f3158d1, 0x157ea087,
0xc2e27c5e, 0xc191276c, 0xe17030bf, 0xf80427fe, 0x227da776, 0x309fa7ee,
0xd6ab4c06, 0xd4ece076, 0x09d9c32b, 0x10e6a261, 0x292b9dda, 0x34dca7fd,
0xf4f5e75c, 0xf0d0a4fc, 0x2c69c85f, 0x4801a8a3, 0x279fdbfd, 0x4f295c4f,
0x78f2e9c0, 0x0fdc57a8, 0x6016bc89, 0x194c5112, 0x4f097162, 0x3610d74e,
0x523e05be, 0x35bd2a82, 0x0707e286, 0x61d3e37e, 0x1550965c, 0x6fb4aa09,
0x07eb7443, 0xfc54a8da, 0x69928340, 0x8f8a53ac, 0x72941a11, 0x8709df96,
0x1864ebe8, 0xf2f496a7, 0x8b9617bc, 0x60f6af73, 0xf8653ffc, 0x1ce6e518,
0xfbfecf2d, 0x123717bc, 0x90f6a77a, 0x7476a97d
```

A.2.6. S-Box S6

```
0x4301e0aa, 0x934959bd, 0x08a01c00, 0x55804207, 0x8d3d33d7, 0xdd6e22b0,
0xc1361255, 0x156ebfea, 0x07add6f6, 0xd0ceabd1, 0x4a49a184, 0x93ca9f6b,
0x4598280b, 0x9f02c8c4, 0x02c707a1, 0xdfdeff6e, 0xc8c70ddf, 0x993a8ed4,
0x58f14a38, 0x87757d83, 0x48e24911, 0x13cdc99a, 0xd75bd08e, 0x0b6ebf7d,
0xf5421b0d, 0xac26f7fe, 0x6a7ac372, 0xbeef3fe9, 0x71beed43, 0xafea37a8,
0xec186eb4, 0xb6deffff, 0x47822eaa, 0x31310013, 0x21976592, 0xdf3bc04b,
0xe3d39772, 0x1811450b, 0x8440fd32, 0xfefeffe3, 0x7bcd8915, 0x8c05aa0c,
0x142e1875, 0xe87114d4, 0x509514bd, 0xa7cefefc, 0x33e12155, 0xc64ebf64,
0x4e7cb3ef, 0x2b4934fa, 0xee0dbfea, 0x0036e03f, 0xbe0c8274, 0xdc815251,
0x17deac59, 0xf73eff74, 0xd4da34de, 0x3f2615a3, 0x7416f6c3, 0x176caba6,
0x2373be35, 0x42d5fde0, 0x870ddbf0, 0x674ebff5, 0xd09f1a57, 0xd0d1f563,
0xeb3170f8, 0x6318aefc, 0x4babcf28, 0xce7e1c5c, 0x722afc8f, 0x76f68313,
0x60397e0e, 0x635fd70a, 0x5cd4dd79, 0x545233a5, 0x7201c8d9, 0x7d96b60d,
0x415f6746, 0x4c46c392, 0x5b58e52a, 0xcaa2600a, 0xfa6cfef8, 0xe4e4d148,
0xae797fce, 0x30d12f76, 0x04c2e474, 0x1c768384, 0x76d667ed, 0x6fb6cb05,
0xdce2b387, 0x4977c317, 0x83a211a1, 0x1cfa8341, 0x2f851043, 0x3546c303,
0xd4101457, 0x42a13cfd, 0xa20b414a, 0xbdab3cb0, 0x65c4b98d, 0x7f81fbf7,
0x175c49e8, 0x8d66c31a, 0x5c55e1ed, 0x4f9c46e7, 0x22be2ca8, 0x3be1a81a,
0x678d266f, 0x71da0025, 0x10791bb2, 0x05d68398, 0xddef9b1a, 0x38d94a14,
0x7c9d430a, 0x17af5cf4, 0x481666ab, 0xab1d34ad, 0xe447c2a3, 0x8026c38d,
0x3742883e, 0xdcbeb968, 0x9283ce16, 0xf4fc4758, 0xa1ee90d7, 0xc545c919,
0x0490ff07, 0xe4d68309, 0xc588a876, 0xc7d42384, 0xd60de7da, 0x5e2d3958,
0xe6b4788e, 0xe303cbfc, 0xff7f6a6a, 0x79e354b0, 0x35748d28, 0xbe03402a,
0xa9f04a1c, 0xa977e506, 0x170553f0, 0x10cb00ea, 0x843afdec, 0x01531436,
0x465e57c3, 0x3dc7f6ed, 0x8d382192, 0xf9a8072c, 0xf32fa0e8, 0x8fc4b096,
0x3b961b11, 0x41235427, 0x33cbf143, 0x48bb3ca5, 0x739729aa, 0x866254b4,
0x86a78788, 0x71f75da6, 0xc0d1a6a9, 0xb05314a7, 0x491fc7b6, 0xb5aceb5a,
0x772e9e28, 0x0286ab54, 0x18ca6fab, 0xe2ac2d17, 0x2859974d, 0x527314b9,
0x1110720b, 0x62c8d0c7, 0xaf93f2cd, 0x54dc7ef9, 0x4288ddc6, 0xbcd7b7c2,
0xf75cc818, 0x88c3543c, 0xc8f14833, 0x6fb4ddf3, 0x13d0d460, 0xb8fb8ad0,
0x0551d80d, 0xa458aa4d, 0xd94375c6, 0x7d73142e, 0x6a577e50, 0xcbbb4f88,
0x35ff1d7b, 0x19f1d0bb, 0xa4ea667e, 0x881816be, 0xf98400ad, 0x51c354ad,
0xd9e9ad73, 0xdbb370a2, 0x8a6374da, 0x074a2a5b, 0xefdd7b89, 0xefec0ae8,
0xb39c7b48, 0x358497b1, 0x9d1fda28, 0x126dc309, 0x4092c919, 0x4110b630,
```

```
0xaf634cda, 0xada081db, 0x785d62f3, 0xfd34d732, 0x5a3e40ce, 0x71a0e7cb,
0xd05a6aaa, 0xf5c6541f, 0xba4b69cf, 0x9327a982, 0x37f0d013, 0x19c49726,
0x0fa0725b, 0xa4d4ffa6, 0x0af0a6a7, 0x2e0557b2, 0xeb448492, 0xcd1816b7,
0xecb327a6, 0x4c34d7a3, 0x557202b3, 0xc9c3284c, 0x2b4d4508, 0x3fe9a857,
0x1122beac, 0x8ac36c13, 0x64badc6f, 0x7e14d7b8, 0x9977e50b, 0x8eaec3d4,
0x66fc39e8, 0xf8b33dcf, 0xda6f10ec, 0x453cb6e3, 0x2b3b0d07, 0x34a49738,
0xd49d9f3e, 0x33db5ce5, 0x1ebfd778, 0xf09dc9e3, 0x1cb4c32a, 0xfc3b3349,
0xd525e4c4, 0x3594d72f, 0x56303d48, 0x37dc1cbb, 0x1c95da56, 0xf59ec3bd,
0x9988b764, 0x7077ddbf, 0xd5e6dba2, 0xbda497a9
```

A.2.7. S-Box S7

```
0x813c06d4, 0x67033b80, 0x88a17185, 0x7d7ba354, 0xace95364, 0x5aa2776a,
0xa4139684, 0x57e084fb, 0x15408c02, 0xfa6ec746, 0x1907265c, 0xed2fc499,
0x2ca939fb, 0xd53d3805, 0x3b31ffd0, 0xcffe8e5b, 0x3267bfc2, 0xae1182e9,
0x417891eb, 0xca159f43, 0x6ae64885, 0xe5bfdd34, 0x1341a399, 0x9d5edc5f,
0x3cbe068c, 0xba7a97df, 0x5ee36a2a, 0xc7df8efe, 0x7c02fdf2, 0xef2797db,
0x0e5f8aa5, 0x94df4cff, 0x9f97093b, 0xc080ec10, 0xf05c3a86, 0xb3237184,
0xb8512108, 0xffd49029, 0xd64f2f23, 0x9bffe9df, 0xdd7e41ad, 0x81a568ba,
0xb3618fff, 0xfb314ec5, 0xe37cee5b, 0xa1fbc0a6, 0x960ec35b, 0xd65fdc7f,
0x3ecb36db, 0x4e29aa7b, 0xffb9a411, 0x97cd4a9e, 0xe9a64412, 0x828d3d78,
0x22de9cca, 0x473e4e7b, 0x3978b2d5, 0x59f53821, 0xed464270, 0x9dbefcaf,
0xfbd0aba9, 0x865a9777, 0x3f5ccfda, 0x44fe1edb, 0x17cfb151, 0x77f08429,
0x2ed64a2c, 0x4f585cfd, 0x569aece9, 0x39908cc7, 0x7ef5a109, 0x1587f376,
0x86a3778f, 0xedddb0c7, 0xbf3095dd, 0xd94d331c, 0xdb4e4672, 0xa3cf4f88,
0xe946885d, 0x990df1d6, 0xb7d18843, 0xa263b560, 0x910be66e, 0x98e6e0e6,
0x2495f708, 0x34dc2ab1, 0x00b2d438, 0x096d23d2, 0x4b997905, 0x558cac5a,
0x7d145da7, 0x73ecf173, 0xde31067f, 0xd915a85e, 0xf0acf520, 0xe2ac3372,
0x02b47abe, 0xda7213bd, 0x666b4d27, 0xa500ce2d, 0xb2325e85, 0x7ce76380,
0xc92918a6, 0x1998de52, 0x214d3e20, 0xf4171f3f, 0x41163076, 0x8353b940,
0x808b11d2, 0x5f08ff2f, 0xe5f9f4de, 0x28aca3f2, 0xb0adc55a, 0xc85addf6,
0xdf8a9f9c, 0xa13e753b, 0x57c53b9f, 0x33efc2f9, 0x34adeb63, 0x530df1f6,
0xc18f495c, 0xb40203a0, 0xbaf179f5, 0xc58d8322, 0x2df3d024, 0x5869e0f6,
0x402ff057, 0x3a8d6156, 0x8ac190b3, 0x2ad4aca3, 0x335a6736, 0x92e45475,
0x8316c491, 0x241d83dd, 0x2be081b1, 0x8831d04e, 0x3b9f5bb5, 0x90f1b0f5,
0x82b4b3ff, 0x24d8332c, 0x26526c4a, 0x9eea4d32, 0x94c28ae5, 0x2409d9ee,
0x2af4a963, 0x77c6b548, 0xac8fe4dc, 0xe5caca77, 0x3111dfb1, 0x69400a03,
0xbdbef538, 0xf4e90bea, 0x760d503d, 0x20a980ea, 0xe0105fcd, 0xae6ad95a,
0x63972846, 0x2498806c, 0xe588fcc0, 0xbf281b4a, 0x97087e4c, 0x8f771b35,
0x5be76c24, 0x58bee6a5, 0x07ae55fc, 0x016b6618, 0xdcbc3896, 0xc42efd6a,
0x34cb170a, 0x29123f0f, 0xfc923a4d, 0xf6c49970, 0xbd8f38eb, 0xaa24f717,
0x78fdd6ee, 0x7da88bca, 0x253ac16a, 0x15defddc, 0x820e9137, 0xbc107faa,
0x42513027, 0x6e72e8c9, 0xe921caeb, 0xce89d9ce, 0xb4a16574, 0x810e0f12,
0x07f55786, 0x30098b0b, 0xd04dfd1c, 0xededc246, 0x758bd93f, 0x4f09496e,
0x03cdd8c9, 0xa5d8ecf5, 0xcad2a360, 0x7f385423, 0x669a84e3, 0xd8d0878f,
0xaef949c3, 0x15a9583c, 0x77835fc7, 0xd8bd388b, 0xbb7cff81, 0x0f453b56,
0x0e4aec3c, 0xb7e7c540, 0xd94a0297, 0x6d05599c, 0xb0bd611d, 0x644b7d3e,
0xa3036ca6, 0x68c64a2d, 0x409d9fc3, 0x87dc0279, 0x51b27d66, 0x9f250b98,
0x7ed5d04b, 0xb0a04490, 0x7c1897bf, 0xa5a65928, 0x9e5b2c34, 0x4d554016,
0x84847cba, 0x56a49b38, 0x15d4f236, 0xaa7a1b67, 0x922fe47a, 0x3162a6f3,
0xd232d58e, 0x7da76a4e, 0x5c25f0ec, 0xf9b63518, 0x77079778, 0xc35fb775,
```

```
0xf11a7a3b, 0x5159910a, 0xa187389d, 0x0b283761, 0x34f51e94, 0x9ca40bb8,
0xb4a3cd14, 0x2c5275ae, 0x7dc25545, 0xf51cbff0, 0x03cdb055, 0x80efe8b7,
0xc0ad42bd, 0x454599bc, 0x53a96102, 0xd306cb6c, 0x8fbd93fc, 0x17c50b79,
0xf991796e, 0x6c214a38, 0x3507194d, 0xae85c91c
```

A.2.8. S-Box S8

```
0x03755908, 0x8130788b, 0x60b34458, 0xf5a713fb, 0x1b3e6039, 0x84ce8770,
0x62da564e, 0xfad6beb4, 0x10e90337, 0x8e7bb301, 0x687c5e1a, 0xe4c286dd,
0x0470b043, 0x9d7f8fed, 0x672370b0, 0xfcad5e7c, 0x8a7833e0, 0x19203447,
0x1600c2a6, 0x9ffb87ac, 0xadf01f95, 0x3cc7c2ec, 0x2e36e31e, 0xb50f0559,
0xf527ecc1, 0x62dc821e, 0x7ccafc4b, 0xf4efcee8, 0xcbf3b9bf, 0x444f44ec,
0x5c441201, 0xdc8bcabd, 0x0f01e193, 0x0e403103, 0x933c20ef, 0x976db906,
0x5aaf1212, 0x5ad1188c, 0xd9d9f5b3, 0xcc1acfc7, 0x1fe6961f, 0x0b562563,
0x85e12831, 0x913ccce5, 0x5748c7ed, 0x58efe7cd, 0xd3e0e9e7, 0xcc87e7fc,
0x8dbab47f, 0xcbd07112, 0x6a4796f4, 0x24cf2e6d, 0xa427ad45, 0xfca8ae32,
0x590d8ff9, 0x19cd88e9, 0x319e46ed, 0x7a7118a1, 0xde9fba85, 0x98ef87ec,
0x108d0eea, 0x5fafdfee, 0xe4bf444c, 0xb6af8ffe, 0xbb0c22ed, 0x1d4c4774,
0x5ac87aad, 0xefdb2d0c, 0x8f401cc4, 0x24b4b09f, 0x78a42da1, 0xdceac743,
0x6c9138da, 0xd20381f6, 0x940561ff, 0x20babd2a, 0x4a0acbbc, 0xe703b51a,
0xa71c4a55, 0x1cd1258b, 0x34054217, 0x015c03b0, 0x0a7afc51, 0x2583f85b,
0x398b2362, 0x04b9fd09, 0x16489fe9, 0x2d3370ae, 0x215e953c, 0x0aa4b0e9,
0x04b7c0a6, 0x2c93f71f, 0x318dc948, 0x0c337d19, 0x067a2ff6, 0x24f3fd4a,
0xbb7d9a64, 0xfa3c0afc, 0xb9451c02, 0xed1585f1, 0x22d62be7, 0x60ab6163,
0x23a7c95c, 0x6822fe32, 0xcd9ba7e0, 0x9f2e1b94, 0xcb9a17cc, 0x9540f512,
0x5735f71a, 0x18939f3a, 0x51df9202, 0x0effd009, 0x3fc2c59a, 0x1bac42e5,
0xa63faa1b, 0x9eb3539a, 0xfc5bd4ba, 0xded6dfd7, 0x6173b40e, 0x43f5b51c,
0xebe23502, 0xda092656, 0x74e08670, 0x4097bc1b, 0x24f43515, 0x0dd3a41b,
0xbc8138bb, 0x8cd3f40b, 0x9cd38f74, 0x7a168aab, 0xff37b654, 0x0aa1e1d3,
0xe80a8279, 0x13e26d24, 0x9dfe103e, 0x69b058bc, 0x2be77547, 0xd55d5d4d,
0x53da9c66, 0xb76470b5, 0x5f445603, 0xa2d95981, 0x30e706cc, 0xcb8b3810,
0x117edd8c, 0xa686ce6f, 0x4d2430aa, 0xe0dd7580, 0xfec46dbd, 0x43eb90d0,
0xb1122152, 0x0ac97f71, 0xc6998ae1, 0x7d7a7c56, 0x837c0e7f, 0x2bc93880,
0x34d7e7b3, 0x9be93280, 0x6330d06d, 0xc3ad3091, 0x9c2727fb, 0x7d66c767,
0x4c3af29b, 0xa84b4b2e, 0x05b9f47e, 0xe5fd7edc, 0xc6fd8783, 0x3f7ca18f,
0x6868f83b, 0x9870c76b, 0xbec5ca71, 0x42ba3a89, 0xe05e29c5, 0x0f49b1a1,
0x2424af9b, 0xdba10dd0, 0x1a3c4a03, 0x7c768f7e, 0x31e14484, 0x5bc9dc41,
0xdb31db45, 0xab84700a, 0xe62979f1, 0x860b7a81, 0xaea0389d, 0xcdd7eaad,
0x81ab6889, 0xe7e97180, 0x638b588a, 0x08092982, 0x5bcbb624, 0x2989f992,
0x3b550b6e, 0xf9954aab, 0x3ab3775e, 0xef2220db, 0x838b017b, 0x4c67a534,
0x987f942e, 0x5073deb4, 0x2860b155, 0xf6da9045, 0x305c5c7c, 0xece3b4bd,
0x8ec1d203, 0x475a9c89, 0x8f27c3d6, 0x5408bc18, 0xb0fc5384, 0x41050667,
0x2ea1f1a2, 0xc55af588, 0x5540aeb5, 0xa46a50ca, 0xd693a25a, 0x2d0af579,
0xcd1f0ce3, 0x2afdb15e, 0x44fecd6d, 0xac4afe88, 0x315668bb, 0xcc6af48a,
0xa6f11265, 0x442af899, 0x37a4a3f3, 0x76e50367, 0x59bc3189, 0x0dcc8826,
0x623f3274, 0x2078f8cc, 0x037c4493, 0x44bb6f85, 0xc5ea363b, 0x93f70663,
0xaf410a73, 0xf939fc81, 0x9fdce6cd, 0xd0ca36a9, 0xf9e42b81, 0xa626c5da,
0xb7bbc419, 0xb3f54376, 0x42668794, 0x5e4a5e49, 0x5cb25d45, 0x5605fe10,
0xa1a8bdf9, 0xa3ccb88b, 0xab23b48d, 0xb2502ba5, 0x542bab83, 0x406eb588,
0x480d9c8a, 0x458aad88, 0xbc0a352c, 0xac0a7d98
```

WHAT IS CLAIMED IS:

1. In a data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of a predetermined bitlength comprising a plurality of consecutive transformation rounds of half of each data block, each consecutive transformation round comprising steps of

combining the half data block with a first masking key of predetermined length using a first binary operation to generate a first modified half data block,

combining the first modified half data block with a second masking key of predetermined length using a second (different) binary operation to generate a second modified half data block,

processing the second modified half data block by a plurality of (m x n) mutually different substitution boxes to generate a third modified half data block, and

XORing the third modified half data block with the remaining half of the data block to generate a transformed half data block of a transformation round.

2. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 1, wherein the first binary operation is addition modulo 2^n, or subtraction modulo 2^n, or bitwise XOR, and the second binary operation is multiplication modulo $(2^n - 1)$, or multiplication modulo $(2^n + 1)$

3. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 1, wherein the first binary operation is addition modulo 2^n, or subtraction modulo 2^n, or bitwise XOR, and the second binary operation is a circular shift (i.e., rotation) by a number of bits specified by the second masking key.

4. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 1, wherein all

first masking keys and all second masking keys for all the transformation rounds are generated before the first transformation round is performed.

5. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 4, wherein all first masking keys and all second masking keys for all the transformation rounds are generated by a plurality of partially bent-function-based (m x n) substitution boxes from the key bits, where the key bits comprise a key pattern of z bytes in the following order: k1, k2, k3, ..., k(z-1), kz.

6. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 5, wherein each data block contains 64 bits, the substitution boxes are eight partially bent-function-based 8x32 s-boxes, S1, S2, S3, ..., S8, and the key bits comprise a key pattern of 10 bytes in the following order: k1, k2, k3, ..., k9, k0.

7. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 6, wherein the transformation round function means has a first plurality of partially bent-function-based (m x n) s-boxes for processing key bits to generate a first masking key and a second masking key, and a second plurality of partially bent-function-based (m x n) s-boxes for processing the second modified data half.

8. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 7, wherein the first plurality of s-boxes comprises four partially bent-function-based 8x32 s-boxes and the second plurality of s-boxes comprises four partially bent-function-based 8x32 s-boxes.

9. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 1, wherein consecutive transformation rounds may be computed differently due to the presence of different binary operations within each transformation round.

10. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 9, wherein the choice of which particular binary operations to use in any particular round is dependent upon the value of certain predetermined bits of the first masking key or the second masking key, or upon the value of certain predetermined bits of the half data block being operated upon.

11. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 9, wherein the choice of which particular binary operations to use in each transformation round is fully specified for all implementations of the method and is independent of any key bits or data bits.

12. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 11, wherein the binary operations addition modulo 2^n, subtraction modulo 2^n, and bitwise XOR can be used to combine the half data block with the first masking key and to combine the s-box outputs which result from the processing of the second modified half data block.

13. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 12, wherein three different transformation rounds are used:

Type 1: I = ((Kai + D) <<< Kbi)
O = ((S1[Ia] ^ S2[Ib]) - S3[Ic]) + S4[Id]

Type 2: I = ((Kai ^ D) <<< Kbi)
O = ((S1[Ia] - S2[Ib]) + S3[Ic]) ^ S4[Id]

Type 3: I = ((Kai - D) <<< Kbi)
O = ((S1[Ia] + S2[Ib]) ^ S3[Ic]) - S4[Id]

where "D" is the original input to the transformation round, "Ia" - "Id" are the most significant byte through least significant byte of I, respectively, and "O" is the output of

the transformation round. In this notation "+" and "-" are addition and subtraction modulo 2**32, "^" is bitwise XOR, and "<<<" is the circular left-shift operation.

14. The data encryption method of cryptographically transforming plaintext into ciphertext in data blocks of predetermined bitlength according to claim 13, wherein twelve transformation rounds are used in total and

rounds 1, 4, 7, and 10 use transformation round Type 1,

rounds 2, 5, 8, and 11 use transformation round Type 2, and

rounds 3, 6, 9, and 12 use transformation round Type 3.

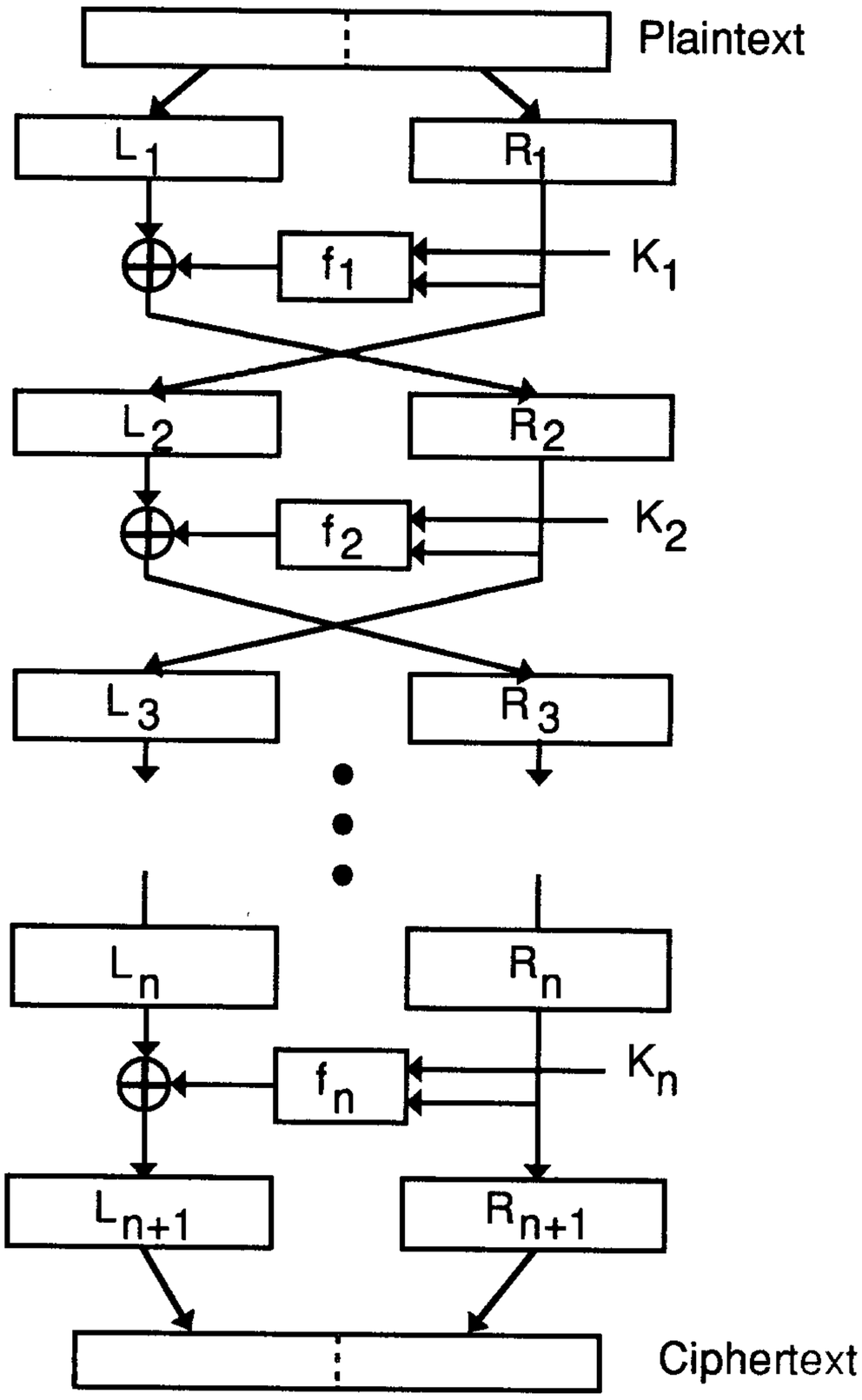


Fig 1

Yakubaru Tayeebally
PATENT AGENT

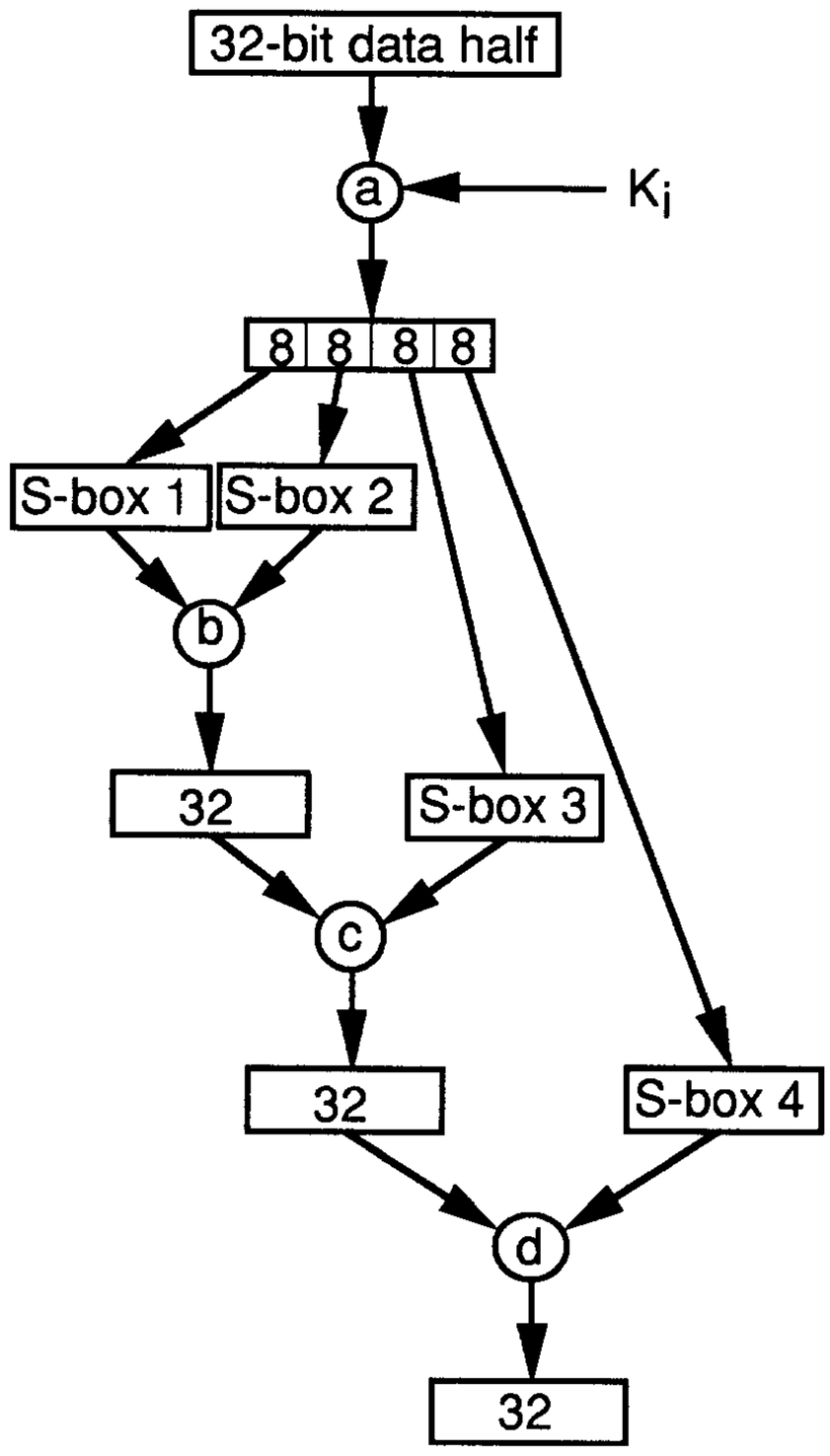
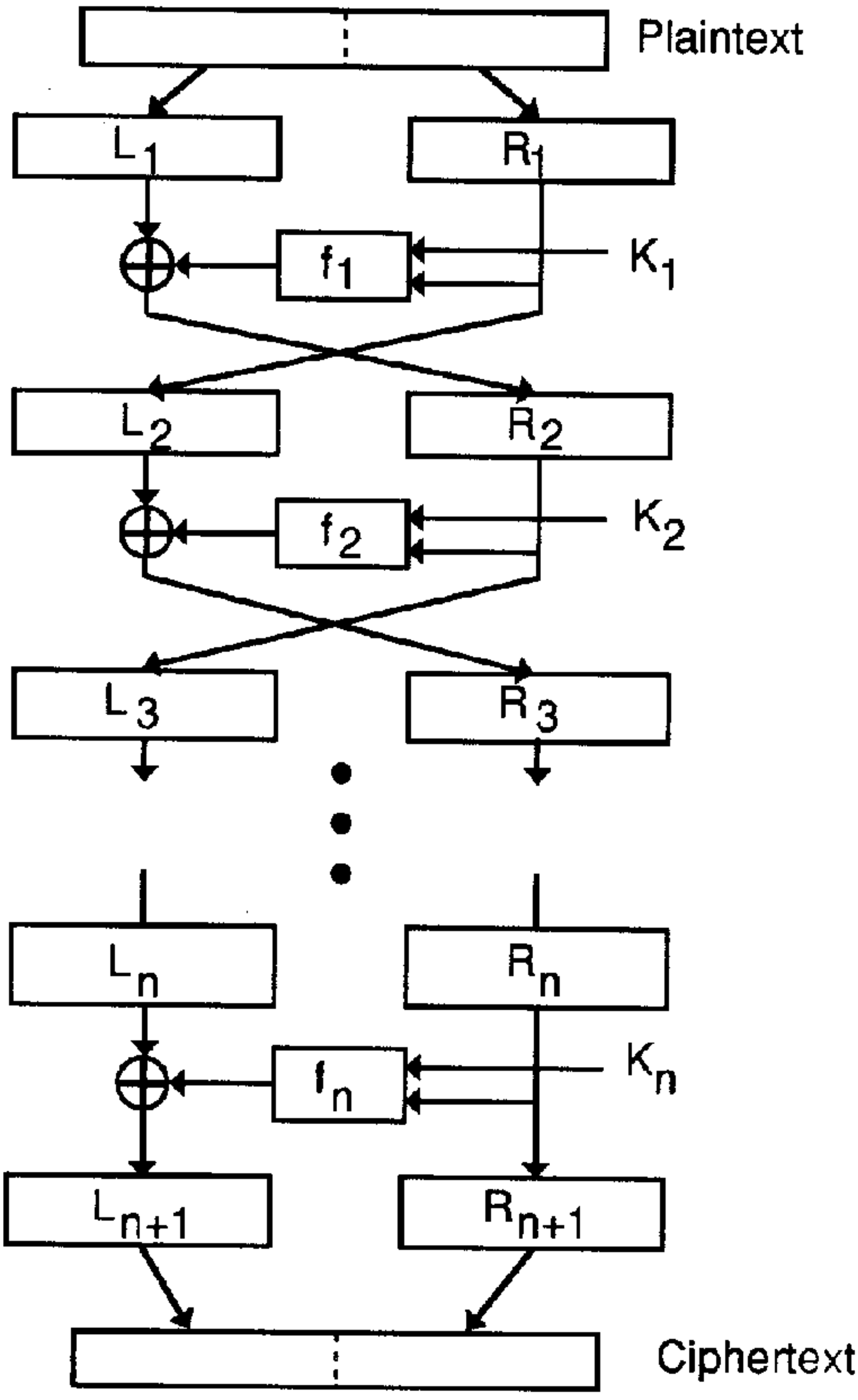


Fig 2

Yeshwant Tayeebally
PATENT AGENT


Plaintext
L_1
R_1
f_1
K_1
L_2
R_2
f_2
K_2
L_3
R_3
L_n
R_n
f_n
K_n
L_{n+1}
R_{n+1}
Ciphertext