Constructing Symmetric Ciphers Using the CAST Design Procedure

CARLISLE M. ADAMS
Entrust Technologies, P.O. Box 3511 Station C, Ottawa, Canada, K1Y 4H7
cadams@entrust.com

Keywords: Design of Encryption Algorithms, Block Ciphers, Substitution Boxes, Key Scheduling, Differential Cryptanalysis, Linear Cryptanalysis

Abstract. This paper describes the CAST design procedure for constructing a family of DES-like Substitution-Permutation Network (SPN) cryptosystems which appear to have good resistance to differential cryptanalysis, linear cryptanalysis, and related-key cryptanalysis, along with a number of other desirable cryptographic properties. 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. An 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.

1. Introduction and Motivation

This paper describes the CAST design procedure for a family of encryption algorithms. The ciphers produced, known as CAST ciphers, appear to have good resistance to differential cryptanalysis [8], linear cryptanalysis [33], and related-key cryptanalysis [9]. Furthermore, they can be shown to possess a number of desirable cryptographic properties such as avalanche [18, 19], Strict Avalanche Criterion (SAC) [54], Bit Independence Criterion (BIC) [54], and an absence of weak and semi-weak keys [25, 12, 40]. CAST ciphers are based on the well-understood and extensively-analyzed framework of the Feistel cipher [18, 19] – the framework used in DES – but with a number of improvements (compared to DES) in both the round function and the key schedule which provide 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.

- Despite years of speculation and warning regarding the inevitable limit to the useful lifetime of the Data Encryption Standard (as originally defined in [41]), 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 [38, 39, 8], LOKI [10, 8, 11], 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 SAFER [32], Blowfish [49], and RC5 [48] 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, 31]. As well, cryptanalysis of a 6-round CAST cipher was described in [47]; this statistical attack requires a work factor of roughly 2^{48} operations and requires 82 known plaintexts.

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 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.

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 separately.

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 such 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)[1].

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. However, it can be very difficult (and, in some cases, impossible) to satisfy some of these properties using "small" s-boxes. The CAST design procedure therefore incorporates a construction algorithm for "large" (e.g., 8×32) s-boxes which possess several important cryptographic properties.

2.2. Framework Design Overview

Ciphers designed around a new basis for cryptographic security (most notably RC5 [48], 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 analyzed by the cryptologic community for well over 20 years.

The CAST framework is the "Substitution-Permutation Network" (SPN) concept as originally put forward by Shannon [51]. 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 [18, 19]

---

[1]Note that the use of 8×32 s-boxes was first suggested by Ralph Merkle for the hash function Snefru [36] and for the block ciphers Khufu and Khafre [37].

to implement the SPN. This is because the Feistel structure is well-studied and appears to be free of basic structural weaknesses, whereas some other forms of the SPN, such as the "tree structure" [22, 23] have some inherent weaknesses [22, 45] unless a significant number of layers are added (which may destroy the one property, "completeness"[2], which tree structures are provably able to achieve). Note that some other forms of SPN, such as that employed in SAFER [32], also appear currently to be free of basic structural weaknesses, but have not been subject to intense analysis for nearly as long as the Feistel structure.

The following diagram illustrates a general Feistel-structured SPN. Basic operation is as follows. A message block of $2n$ bits is input and split into a left half L_1 and a right half R_1. The right half and a subkey K_1 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.

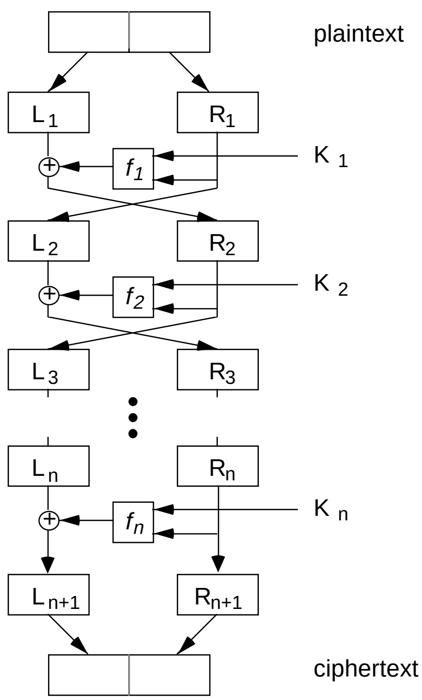

Fig.1: SPN (Feistel) Cipher

---

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

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. Except for the use of randomized encryption, the plaintext and ciphertext blocksizes are chosen to be equal so that the encryption process results in no data expansion (an important consideration in many applications).

As is evident in the work by Biham [8] 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 makes the "improved" algorithm much more susceptible to differential cryptanalysis than the original). 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[5] attacks [17, 23, 53]. Furthermore, it should ensure that the primary key bits

---

[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*) [53, 54].

[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*) [53, 54].

[5]If keys which are close to each other in Hamming distance result in ciphertexts which are likely also to be close in Hamming distance, then it may be possible to find a key faster than exhaustive search in a known

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 [20], who showed that DES-like cryptosystems without a key that varies through successive rounds can be broken). Finally, all key bits should be used by round $N/2$ (in an N-round cipher) and then reused in the remaining rounds (to ensure good key avalanche for both encryption and decryption).

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 Knudsen [28] and by Biham [9], 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 using XOR or other binary operations; 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] and linear [33] cryptanalysis are general-purpose attacks which may be applied to a variety of substitution-permutation network (DES-like) ciphers. Both methods work on the principle of finding high-probability attacks

---

plaintext attack by searching for the correct key cluster and then searching for the correct key within that cluster.

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[6]; this "characteristic probability" determines the work factor[7] 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.

Resistance to these attacks can be achieved either by adding rounds (which reduces the speed of the cipher) or by improving the properties of the round s-boxes (which may or may not make the round probability low enough to avoid the need to add rounds in a given cipher). The latter approach has been pursued by a number of researchers (see [4, 5, 16, 43, 50, 52], for example).

The approach proposed in the CAST design procedure presented below includes both of the above. More importantly, however, it also includes a slight alteration to the typical DES-like round function which renders it "intrinsically immune" (as opposed to computationally immune) to differential and linear cryptanalysis as described in [8, 33]. 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 assertions, theorems, and remarks regarding the cryptographic properties relevant to each aspect.

---

[6]Assuming independent round keys (a reasonable assumption (i.e., a good approximation) for most known ciphers).

[7]The number of operations required for the attack, which may or may not be directly related to the number of chosen plaintexts required.

3.1. Detailed S-Box Design

For the design of $m \times n$ ($m < n$) s-boxes[8], let n be an integer multiple of m (where $2n$ is the blocksize of the cipher); in particular, let $n = rm$ where r is an integer greater than 1 (note that then $m \le \log_2 C(n,n/2) = \log_2(\text{“}n \text{ choose } n/2\text{”})$). Such s-boxes can be constructed as follows. Choose n distinct binary bent (see, for example, [42, 46, 3]) vectors ϕ_i of length 2^m such that linear combinations of these vectors sum (modulo 2) to highly nonlinear, near-SAC-fulfilling vectors (Nyberg's work [43] shows that these linear combinations cannot all be bent since $m<2n$; however, it is important that they be highly nonlinear and close to SAC-fulfilling so as to satisfy the Output Bit Independence Criterion and aid in resistance to linear cryptanalysis). 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. Note that each new s-box should be generated from an independent “pool” of bent vectors to ensure that columns in different s-boxes are distinct and not linearly related.

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)[9]. If these conditions are not 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 possible to construct 8×32 s-boxes which meet these conditions within a few weeks of running time on common computing platforms.

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

Assertion 1: S-boxes constructed as described above have good *confusion*, *diffusion*, and *avalanche*.

---

[8]An $m \times n$ s-box is represented as a $2^m \times n$ binary matrix M where each of the n columns 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, $1 \le i \le 2^m$, is therefore the n-bit output vector which results from the i^{th} input vector.

[9]Note that this is impossible if $m \ge n$ but is quite feasible if $n = rm$, since then $2^m \le C(n,n/2)$.

Discussion: It is not difficult to see that the given requirements on the s-box rows and columns lead to good s-box confusion and diffusion properties (as described by Shannon [51]) and also ensure good avalanche (as discussed in [18, 19] and echoed in [26]).

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][10] 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 [34]. By definition [54], an s-box satisfies the highest-order SAC if and only if each of its output bits satisfies the highest-order SAC. ☐

Assertion 2: If the columns in the s-box matrix are bent vectors whose linear combinations are highly nonlinearly related and near SAC-fulfilling, then the s-box will show close proximity to *highest-order (output) bit independence criterion*. That is, small changes in the m input bits will cause each of the n output bits to change virtually independently of all other output bits. Furthermore, such s-boxes aid in *immunity to linear cryptanalysis* [33].

Discussion: 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 in which the column sums are highly nonlinear and near SAC-fulfilling but not necessarily bent. Proximity to BIC is defined in terms of proximity to SAC: if columns ϕ_j and ϕ_k sum to a vector which comes close to satisfying the SAC (i.e., over all single-bit input changes, the output changes with probability γ, where $(0.5-\omega) \le \gamma \le (0.5+\omega)$ and ω is "small"), then output bits j and k will act "virtually" independently (i.e., will have a correlation coefficient which is nonzero, but "small", as determined by ω), for all single-bit input changes. In highest-order BIC the sums of all column subsets are considered (not just pairs). Requiring that these sums are near-SAC-fulfilling means (by definition) that the s-box will have close proximity to highest-order BIC[11]. Such s-boxes aid in immunity to linear cryptanalysis because there is no linear

---

[10]This has independently been called the Propagation Criterion of degree n in [46].

[11]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 [43], where all column sums are bent.

combination of component functions which has a small Hamming distance to an affine Boolean function (see the discussion in Section 8.1 of [50]).

Lemma 1: $m \times n$ s-boxes designed according to the above procedure can be made to have a largest value, L, in the difference distribution table such that $2 \leq L \leq 2^{m/2}$.

Proof: 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[12] corresponding to a given change in the s-box input (see [4, 54] 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}. 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 $L \leq 2T = 2^{m/2}$. 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 $L \geq 2$. □

Remark 1: Although starting with a perfect s-box provides a guaranteed upper bound on L, 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 L=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 and number of rounds. Aside from the constraint that the blocksize be large

---

[12]Let $c = c_1c_2...c_m$ be a fixed m-bit vector of nonzero Hamming weight and let $f(x) = f(x_1x_2...x_m)$ be a Boolean function of m input variables. Divide the 2^m possible inputs of f into 2^{m-1} pairs x and $(x \oplus c)$ and sort the pairs into increasing values of x. Label the i^{th} pair $[x, (x \oplus c)]_i$. Then the 2^{m-1}-bit vector v is called the "avalanche vector" of f with respect to c if the i^{th} bit of $v = g([x, (x \oplus c)]_i) = f(x) \oplus f(x \oplus c)$ for $i = 0...2^{m-1}-1$.

enough to preclude birthday-attack-derived analysis of the plaintext data, the only real blocksize consideration is ease of implementation. On current machines and for many typical environments, 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 a larger choice may be warranted for environments wherein significantly more than 2^{32} data blocks (i.e., 2^{33} or more) may be encrypted using a single key.

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 [13]. With subsequent improvements to the differential attack [8] 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 2: With respect to *differential cryptanalysis*, *N*-round ciphers designed according to the CAST procedure can be constructed with *N*-2 round characteristics which have probability significantly smaller than the inverse of the size of the keyspace.

Proof: Recall from Lemma 1 that the largest value in the difference distribution table of CAST-designed $m \times n$ s-boxes is L, where $2 \le L \le 2^{m/2}$. Select for the round function only s-boxes for which L=2. Therefore the highest probability in each table is $P = L/2^m \le 2^{1-m}$. 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 characteristic [30] of f (that is, the most successful differential cryptanalytic attack for that single round) has probability at most $P_f = 2^{x(1-m)+y}$ (because the s-box outputs are combined (e.g., using XOR) rather than simply concatenated (as in DES)). Note that the y in the exponent accounts for the possibility that there may be as many as 2^y sets of the r component s-box output XORs which combine to produce a desired output XOR of f; randomness arguments suggest that y is expected to be less than 4. Given P_f, 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. To ensure that such a ΔV must involve 3 or more s-boxes, the following condition is stipulated: for all pairs of s-boxes in the round function, ensure that $S_i(a) \oplus S_j(b) \neq S_i(c) \oplus S_j(d)$ except when a=c and b=d (in which case, of course, they must be equal). The probability of the characteristic for a single round could therefore be as high as $P_f = 2^{3(1-m)+y}$. Hence, assuming an N-2 round characteristic (for an N-round cipher), the probability of the characteristic could be as high as $P_f^{(N-2)/2} = 2^{(3(1-m)+y)(N-2)/2}$, since ΔV is only used on every other round and an input XOR of zero is used otherwise[13].

For parameters m=8, and N=12, and with a conservative estimate of y=5, the characteristic probability is $\le 2^{-80}$. This value can be decreased dramatically, if desired, by doing extra checking during the s-box construction / selection process to ensure that $y < 5$, or that ΔV must involve all 4 s-boxes. □

Remark 2: It has been shown [30, 44] that immunity against differential attacks can only be proven through the use of differentials, not characteristics. However, since the probability of an r-round differential with input difference A and output difference B is the sum of the probabilities of all r-round characteristics with input difference A and output difference B [44], it would be necessary that there exist significantly more than 2^{16} such maximum-probability characteristics in order for a differential to exist which would

---

[13] Although it is recognized that multiplying the P_f values in an iterated cipher with additive keys (with respect to differential attacks where the difference is addition) is only strictly correct if the round keys are independent and uniformly random, this product appears to be a good approximation of the characteristic probability for most known ciphers.

threaten a cipher with a 64-bit blocksize. We therefore conjecture immunity to differential cryptanalysis for CAST-designed ciphers with this blocksize.

Theorem 3: With respect to *linear cryptanalysis*, *N*-round ciphers designed according to the CAST procedure can be constructed with linear relations requiring a number of known plaintexts approximately equal to the total number of possible plaintexts.

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 the overall cipher (N), and the number of known plaintexts required for the recovery of a single key bit with 97.7% confidence (N_L) has been given by Heys and Tavares [24]:

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

This relationship was derived by substituting α (the number of s-box linear approximations involved in the overall linear approximation) into the "piling-up lemma" of [33] to get $\left|p_L - \frac{1}{2}\right| \leq 2^{\alpha-1}\left|p - \frac{1}{2}\right|^{\alpha}$ and noting that $N_L = \left|p_L - \frac{1}{2}\right|^{-2}$ for 97.7% confidence in the suggested key. The value α was estimated at $2N$, assuming 4 s-boxes per CAST round function (thus 4 s-boxes involved in the best 2-round approximation), and $N/2$ iterations of the best 2-round approximation. Finally, $\left|p - \frac{1}{2}\right|$ depends on the nonlinearity of the component s-boxes: $\left|p - \frac{1}{2}\right| = \left(\frac{2^{m-1} - N_{\min}}{2^m}\right)$.

Substituting $N_{min} = 74$ and $N = 12$ results in N_L being lower-bounded[14] by approximately 2^{62} (which appears to be adequate security for a 64-bit blocksize since there are only 2^{64} possible plaintexts and since it is not currently known how tight this lower bound is for CAST-designed ciphers). As another example, for a cipher with a 96-bit blocksize, α may be estimated at $3N$ (that is, the cipher may be constructed with 6 s-boxes per round); thus, for the same N_{min} and N, $N_L \geq 4 \times \left(\frac{1}{1 - N_{\min}/2^{m-1}}\right)^{6N} \approx 2^{96.6}$. □

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

[14] Like differential cryptanalysis, formal results in this area require round keys which are independent and uniformly random. However, most equations derived using this assumption appear to be good approximations for most known ciphers.

additional operations in the round function (see Section 3.4) should all permit CAST ciphers with longer keys to be used with sufficient resistance to linear cryptanalysis.

Remark 3: Like the situation in differential cryptanalysis with characteristics and differentials, immunity to linear cryptanalysis can only be proved using "total linear relations", not "linear relations" (as used in the theorem above). However, a number of factors suggest that CAST ciphers are immune to this attack. Firstly, the lower bound for linear relations appears to be acceptably high and is not known to be tight. Secondly, the structure of the CAST round function (e.g., the XOR sum of a number of s-boxes) is such that any subset of output bits must involve data bits and key bits from each component s-box (thus, finding "useful" multi-round linear relations appears to be more difficult for CAST than for DES). Finally, the goal of linear cryptanalysis is to derive, with reasonable probability, the XOR sum of a subset of subkey bits. In DES and some other ciphers, these subkey bits correspond directly to bits of the primary key and so exhaustive search on primary key bits not deduced by the attack recovers the entire key. In CAST, however, the subkey bits do not correspond directly to primary key bits (see Section 3.3 below or the example key schedule in Appendix A) and so it is not clear that knowing a subset of these bits will aid in any significant way in recovering the primary key.

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. A simple key schedule for an 8-round algorithm employing a 64-bit key is as follows (this schedule is for illustrative purposes, using a relatively small number of rounds and little complexity in order to show how an absence of $inverse_{SR}$ keys can be proven; in practice, a more involved schedule (with more entropy per subkey [47]) would be used – see Appendix A, which provides a schedule for a 16-round algorithm with a 128-bit key).

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:

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

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 schedule 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 more involved 12- or 16-round system or for different block or key sizes, for simplicity of notation and concreteness of explanation, the theorem and remarks 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 $\mathrm{ENC}_I(p) = \mathrm{ENC}_{K^{-1}}(p) = \mathrm{DEC}_K(p)$ for any plaintext vector p. Furthermore, a *fixed point of a key K* is a plaintext vector x such that $\mathrm{ENC}_K(x) = x$ and an *anti-fixed point of a key K* is a plaintext vector x such that $\mathrm{ENC}_K(x)$ is the complement of x.

From work done on cycling properties and key scheduling in DES [12, 14, 25, 40], the following definitions have been introduced. A key is *weak* if it is its own inverse (such keys generate a palindromic set of subkeys[15] 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[16] and have 2^{32} anti-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 $K_i = H_{N+1-i}$, $i = 1, ..., N$. It is clear that a subkey

---

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

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

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 Theorem and Remarks

Theorem 4: Ciphers using the key schedule proposed in Section 3.3 can be shown to have *no inverse*$_{SR}$ *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 inverse$_{SR}$ 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 inverse$_{SR}$ keys exist for the cipher then the subkeys $K_i = H_{N+1-i}$ (by definition) and the partial keys $K'_i = H'_{N+1-i}$ (by construction of the key schedule s-boxes; see Section 3.1). Therefore, $a_1=a_2$, $a_3=a_4$, $b_1=b_2$, *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 inverse$_{SR}$ 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 transformation step gives:

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

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

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. $\square$

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 inverse$_{SR}$ key H for any given key K (otherwise a new S_1 and S_2 can be chosen and Step 2 can be repeated). $\square$

Although the proof above applies to an 8-round implementation of a CAST cipher, the result can 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 inverse$_{SR}$ 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 inverse$_{SR}$ keys.

Assertion 3: Ciphers using the key schedule proposed in this paper are *immune to related-key cryptanalysis* as described in [9].

Discussion: There are no related keys [27, 9] 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 as described in [9].

Remark 4: From Theorem 4 above, this key schedule avoids all inverse$_{SR}$ 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-existence of weak and semi-weak keys). From all evidence available thus far in the open literature, fixed points have only been easily[17] found in DES-like ciphers for weak and semi-weak keys; we

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

therefore conjecture that ciphers using the key schedule proposed in Section 3.3 have *no easily-found fixed points for any key*.

Remark 5: 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.

Theorem 4 and the above remarks regarding 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.

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 s-box (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.

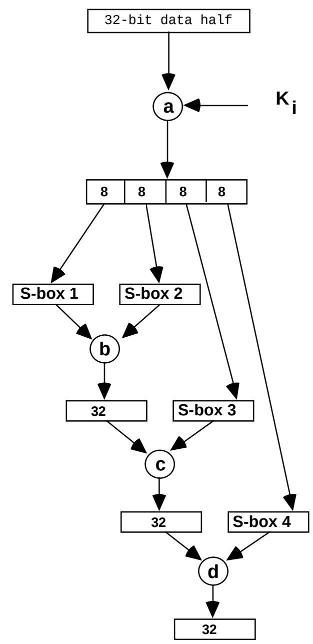

Fig. 2:
CAST Round Function

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: The CAST round function exhibits good *confusion, diffusion*, and *avalanche*.
Discussion: It is not difficult to see that the round function possesses these properties due to the fact that the component s-boxes possess these properties (Assertion 1).

Remark 6: Although confusion, diffusion, and avalanche are somewhat vague terms and cannot be proven formally, they can be argued on an intuitive level for the CAST s-boxes and round function. Note that a round function which achieves all three properties simultaneously should lead to a faster buildup of complexity and data / key interdependency in a Feistel network than a round function which does not. This appears to be the case for CAST ciphers, which show very good statistical properties after only 2-3 rounds whereas DES, for example, requires 5-6 rounds to display similar properties[18].

---

[18]Note that in the DES round function a single bit change in the input can change a maximum of 8 of the 32 output bits. It therefore does not satisfy the avalanche property.

Theorem 5: For appropriate design choices, the CAST round function is 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[19] (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 $W = (w_1, w_2, ..., w_n)$ 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 $t \leq m/2$ (where t 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 7: In practice, close proximity to highest-order SAC appears to be readily achieved for the CAST round function 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 5: For appropriate design choices, the CAST round function exhibits 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

---

[19]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.

readily achieved for the CAST round function whether or not XOR addition is used as the binary operation.

Remark 8: Although this seems to be difficult to prove theoretically, the above properties of the round function (confusion, diffusion, avalanche, highest-order SAC, and highest-order BIC) lend evidence to the conjecture that an *N*-round CAST cipher employing such a round function will behave as a random permutation for arbitrary input bit changes.

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 provide *computational* immunity to differential and linear cryptanalysis. We now 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 the attacks as described in [8, 33] can no longer be mounted); see also Section 4.2.

3.4.1.1. 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 generalizes this to attack the round function and extends the round function attack to create a number of characteristics for the overall cipher. The most successful characteristic (that is, the one with highest probability) theoretically breaks the cipher if its work factor 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). In terms of notation, for the DES round function let R be the data input, K be the subkey, $E(\bullet)$ be the expansion step, $S(\bullet)$ be the s-box step, $P(\bullet)$ be the permutation step, and R' be the function output. Furthermore, let $X = E(R) \oplus K$ and $Y = S(X)$, so that $R' = P(Y)$. Finally, let L be the left half of the data which is not input to the round function.

In differential cryptanalysis the s-box property which is exploited is its "input XOR" to "output XOR" mapping, where a specific ΔX leads to a specific ΔY with high probability. Due to the linearity in the $E(\bullet)$ and $P(\bullet)$ operations with respect to XOR, $\Delta X = X_1 \oplus X_2 = E(R_1) \oplus K \oplus E(R_2) \oplus K = E(R_1) \oplus E(R_2) = E(\Delta R)$ during two encryptions with the same key, and $\Delta R' = P(Y_1) \oplus P(Y_2) = P(\Delta Y)$. Thus ΔR pairs can be found which result in "useful" $\Delta R'$ pairs, where a $\Delta R'$ pair is "useful" in this context if it can act as a desired ΔR

pair in the following round, so that round function attacks can be iterated and concatenated into characteristics with high overall probability.

In linear cryptanalysis the s-box property which is exploited is linearity. Let $\Sigma(\bullet)$ be the XOR sum of a specific subset of the bits in the argument and let $\Sigma_p(\bullet)$ be the XOR sum of the permuted indices of the subset of bits used in $\Sigma(\bullet)$ with respect to the permutation $P(\bullet)$. Then $\Sigma(Y) = \Sigma(X)$ with high probability. Again due to linearity, $\Sigma(Y) = \Sigma(E(R) \oplus K) = \Sigma(E(R)) \oplus \Sigma(K)$, and so $\Sigma(K) = \Sigma(E(R)) \oplus \Sigma(Y)$. Since knowing R immediately yields $\Sigma(E(R))$ and knowing R' immediately yields $\Sigma_p(R') = \Sigma_p(P(Y)) = \Sigma(Y)$, various R can be found which result in "useful" R', where an R' is "useful" in this context if it can be XOR'ed with a desired $\Sigma(L)$ from the previous round to yield a desired $\Sigma(R)$ for the following round, so that round function attacks can be iterated and concatenated into characteristics with high overall probability.

3.4.1.2. Modification of Operation "*a*"

The goal behind modifying the round function is to eliminate the possibility of both differential and linear cryptanalytic attacks (as described in [8, 33]) 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 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).

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

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

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[20] are $(2^{32} - 1)$ and $(2^{32} + 1)$. Meijer [35] describes 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 third 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 three 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 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.

---

[20] Note that multiplication modulo 2^{32}-1 was first used in a cryptographic setting by Donald Davies in MAA [15] and that multiplication modulo 2^{16}+1 was first used in IDEA [29].

In order to ensure that the modular multiplication does not perform badly with respect to property three, 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.

3.4.1.3. Making the Round Function Intrinsically Immune to Differential Cryptanalysis

Property three 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 3.4.1.1:

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

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 three prevents such an attack with the requirement that no operation on a pair of R values can cancel the effect of the key. Modular multiplication appears to achieve property three 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

$$\Delta X = (X_1 - X_2) \bmod n$$

$= (a(R_1 \oplus K_1, K_2) \; - \; a(R_2 \oplus K_1, K_2)) \bmod n$

$= ((((R_1 \oplus K_1) + K_2) \bmod n) \; - \; (((R_2 \oplus K_1) + K_2) \bmod n)) \bmod n$

$= ((R_1 \oplus K_1) - (R_2 \oplus K_1)) \bmod n$

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.

3.4.1.4. 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 3.4.1.1:

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

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

This is the critical component of the linear attack because the distributive nature of the subset sum operation $\Sigma(\bullet)$ over the XOR operation may allow the equivalent of one key bit to be computed[21] 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 Σ(*plaintext*) and Σ(*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[22], 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.)

---

[21]Note that if two linear approximations exist involving the same bits and with the same bias, but with opposite sign, no information can be found on the single key bit. The reason this attack works on DES is that one approximation has a higher probability than the others in the DES round function. This situation may or may not exist in other round functions, including the one proposed for CAST ciphers.

[22]Note that Harpes, et al, have found that ciphers using modular addition or multiplication (with large moduli) to insert the key into the round function tend to be immune not only to Matsui's linear cryptanalysis, but also to their generalization of linear cryptanalysis using I/O sums [21].

3.4.1.5. 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, appears to require more chosen/known plaintexts for differential and linear attacks than exist for that blocksize 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 performance 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 and encouraging further research into a proof of security for each alternative. If such proofs become available, the corresponding options will be formally incorporated into the CAST design procedure. Note that all alternatives have been included in the example cipher given in Section 5, primarily to stimulate analysis of these options in the context of a real cipher, but also because the author believes these to be good design choices.

4.1. Binary Operations in the Round Function

Throughout this paper the operations *b*, *c*, and *d* in the round function (as well as at least part of operation *a*) have been specified as the XOR of two binary quantities. 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 [56], but this is yet to be proven formally.

4.2. Extension to Operation "*a*"

Discussed in Section 3.4.1 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 RC5 [48], except that here we suggest a key-dependent rotation (controlled by a 5-bit subkey) rather than a data-dependent rotation, since data-dependent rotation appears to be less appropriate for a Feistel-type structure.

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

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

where "$\bullet$" 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 the n-bit rotation ($0 \leq n \leq 31$) specified by K_2 can be accomplished in a small number of clock cycles, thus causing very minor performance degradation in the overall cipher. Rotation satisfies property (1) from Section 3.4.1.2 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 *b*, *c*, and *d* (and at least part of *a*) 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 [49]). 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, 30]).

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 [55] in a cipher adds in any significant way to its cryptographic security is an open question. However, there is no evidence thus far that variations resulting from mixed operations (as suggested in Section 4.3) can in any way weaken the cipher and lead to its cryptanalysis.

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 Appendices A, B, and C). This 16-round cipher has a blocksize of 64 bits and a keysize of 128 bits; it uses rotation in operation *a* to provide intrinsic immunity to linear and differential attacks; it uses a mixture of XOR, addition and subtraction (modulo 2^{32}) in 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 (128 bits) and has very good encryption / decryption performance: 3.3 MBytes/sec on a 150 MHz Pentium processor.

In order to simplify future reference (i.e., to disambiguate this example from any other CAST-designed cipher discussed elsewhere), this example cipher will be referred to as CAST-128.

5.1. Pairs of Round Keys

CAST-128 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 CAST-128. The rounds are as follows (where "D" is the data 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, 10, 13, and 16 use *f* function Type 1.
Rounds 2, 5, 8, 11, and 14 use *f* function Type 2.
Rounds 3, 6, 9, 12, and 15 use *f* function Type 3.

5.3. Key Schedule

Let the 128-bit key be `x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF`, where `x0` represents the most significant byte and `xF` represents the least significant byte.

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

5.4. Substitution Boxes

CAST-128 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. Although 8 s-boxes require a total of

8 KBytes of storage, note that only 4 KBytes are required during actual encryption/decryption since subkey generation is typically done prior to any data input.

See Appendix B for the contents of s-boxes S1 - S8.

6. Conclusions

The CAST design procedure can be used to produce a family of encryption algorithms which appear to have good resistance to differential cryptanalysis, linear cryptanalysis, and related-key cryptanalysis, as described in the literature. 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 Elec. Eng., Queen's University, Kingston, Ontario, Canada, 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 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 the Data Encryption Standard*, Springer-Verlag, 1993.

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

[10] 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.

[11] 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.

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

[13] D. Coppersmith, *The Data Encryption Standard (DES) and its Strength Against Attacks*, IBM Journal of Research and Development, vol.38, #3, May 1994, pp.243-250.

[14] D. Davies, *Some Regular Properties of the 'Data Encryption Standard' Algorithm*, in Advances in Cryptology: Proc. of Crypto '82, Springer-Verlag, New York, 1983, pp.89-96.

[15] D. Davies, *A Message Authenticator Algorithm Suitable for A Mainframe Computer*, in Advances in Cryptology: Proc. of Crypto '84, Springer-Verlag, New York, 1985, pp.394-400.

[16] 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.

[17] W. Diffie and M. E. Hellman, *Privacy and Authentication: An Introduction to Cryptography*, in Proceedings of the IEEE, vol.67, 1979, pp.397-427.

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

[19] 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.

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

[21] C. Harpes, G. Kramer, and J. Massey, *A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-up Lemma*, Proc. of Eurocrypt '95, Springer-Verlag, 1995, pp.24-38.

[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 Trans. on Computers, vol. C-28, 1979, pp.747-753.

[27] L. R. Knudsen, *Cryptanalysis of LOKI91*, in Advances in Cryptology: Proc. of Auscrypt '92, Springer-Verlag, 1993, pp.196-208.

[28] 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.

[29] X. Lai and J. L. Massey, *A Proposal for a New Block Encryption Standard*, Adv. 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] 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.

[32] J. Massey, *SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm*, Proceedings of the Cambridge Security Workshop on Fast Software Encryption, Cambridge, U.K., Dec. 9-11, 1993, Springer-Verlag, pp.1-17. [See also: *SAFER K-64: One Year Later*, in Proceedings of the Second International Workshop on Fast Software Encryption, Springer-Verlag, 1995, pp.212-241; and *Strengthened Key Schedule for the Cipher SAFER*, posted to the USENET newsgroup sci.crypt, September 9, 1995]

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

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

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

[36] R. Merkle, *A Fast Software One-Way Hash Function*, Journal of Cryptology, vol.3, #1, 1990, pp.43-58.

[37] R. Merkle, *Fast Software Encryption Functions*, in Advances in Cryptology: Proc. of Crypto '90, Springer-Verlag, New York, 1991, pp.477-501.

[38] S. Miyaguchi, A. Shiraishi, and A. Shimizu, *Fast Data Encryption Algorithm Feal-8*, Review of Electrical Communications Laboratories, vol.36, #4, 1988.

[39] S. Miyaguchi, *The FEAL Cipher Family*, in Advances in Cryptology: Proc. of Crypto '90, Springer-Verlag, New York, 1991, pp.627-638.

[40] 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.

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

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

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

[44] K. Nyberg and L. Knudsen, *Provable Security Against Differential Cryptanalysis*, in Advances in Cryptology: Proc. of Crypto ‘92, Springer-Verlag, 1993, pp.566-574.

[45] 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.

[46] 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.

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

[48] R. Rivest, *The RC5 Encryption Algorithm*, Proceedings of the Second International Workshop on Fast Software Encryption, Springer-Verlag, 1995, pp.86-96.

[49] B. Schneier, *The Blowfish Encryption Algorithm*, Proceedings of the Cambridge Security Workshop on Fast Software Encryption, Cambridge, U.K., Dec. 9-11, 1993, Springer-Verlag, pp.191-204.

[50] 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.

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

[52] 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.

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

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

[55] M. Wiener (personal communication).

[56] A. Youssef (personal communication).

Appendix A.

This appendix provides full details of the CAST-128 key schedule (see Section 5).

A.1. Key Schedule

Let the 128-bit key be `x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF`, where `x0` represents the most significant byte and `xF` represents the least significant byte.

Let K_{m_1}, ..., $K_{m_{16}}$ be sixteen 32-bit masking subkeys (one per round).
Let K_{r_1}, , $K_{r_{16}}$ be sixteen 32-bit rotate subkeys (one per round); only the least significant 5 bits are used in each round.

Let `z0..zF` be intermediate (temporary) bytes.
Let `Si[]` represent s-box *i* and let "^" represent XOR addition.

The subkeys are formed from the key `x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF` as follows.

```
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8]
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA]
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9]
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB]
K1  = S5[z8] ^ S6[z9] ^ S7[z7] ^ S8[z6] ^ S5[z2]
K2  = S5[zA] ^ S6[zB] ^ S7[z5] ^ S8[z4] ^ S6[z6]
K3  = S5[zC] ^ S6[zD] ^ S7[z3] ^ S8[z2] ^ S7[z9]
K4  = S5[zE] ^ S6[zF] ^ S7[z1] ^ S8[z0] ^ S8[zC]
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0]
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2]
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1]
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3]
K5  = S5[x3] ^ S6[x2] ^ S7[xC] ^ S8[xD] ^ S5[x8]
K6  = S5[x1] ^ S6[x0] ^ S7[xE] ^ S8[xF] ^ S6[xD]
K7  = S5[x7] ^ S6[x6] ^ S7[x8] ^ S8[x9] ^ S7[x3]
K8  = S5[x5] ^ S6[x4] ^ S7[xA] ^ S8[xB] ^ S8[x7]
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8]
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA]
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9]
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB]
K9  = S5[z3] ^ S6[z2] ^ S7[zC] ^ S8[zD] ^ S5[z9]
K10 = S5[z1] ^ S6[z0] ^ S7[zE] ^ S8[zF] ^ S6[zC]
K11 = S5[z7] ^ S6[z6] ^ S7[z8] ^ S8[z9] ^ S7[z2]
K12 = S5[z5] ^ S6[z4] ^ S7[zA] ^ S8[zB] ^ S8[z6]
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0]
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2]
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1]
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3]
K13 = S5[x8] ^ S6[x9] ^ S7[x7] ^ S8[x6] ^ S5[x3]
K14 = S5[xA] ^ S6[xB] ^ S7[x5] ^ S8[x4] ^ S6[x7]
K15 = S5[xC] ^ S6[xD] ^ S7[x3] ^ S8[x2] ^ S7[x8]
K16 = S5[xE] ^ S6[xF] ^ S7[x1] ^ S8[x0] ^ S8[xD]
```

[The remaining half is identical to what is given above, carrying on from the last created `x0..xF` to generate keys K_{17} - K_{32}.]

```
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8]
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA]
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9]
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB]
K17 = S5[z8] ^ S6[z9] ^ S7[z7] ^ S8[z6] ^ S5[z2]
K18 = S5[zA] ^ S6[zB] ^ S7[z5] ^ S8[z4] ^ S6[z6]
K19 = S5[zC] ^ S6[zD] ^ S7[z3] ^ S8[z2] ^ S7[z9]
K20 = S5[zE] ^ S6[zF] ^ S7[z1] ^ S8[z0] ^ S8[zC]
```

```
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0]
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2]
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1]
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3]
K21 = S5[x3] ^ S6[x2] ^ S7[xC] ^ S8[xD] ^ S5[x8]
K22 = S5[x1] ^ S6[x0] ^ S7[xE] ^ S8[xF] ^ S6[xD]
K23 = S5[x7] ^ S6[x6] ^ S7[x8] ^ S8[x9] ^ S7[x3]
K24 = S5[x5] ^ S6[x4] ^ S7[xA] ^ S8[xB] ^ S8[x7]
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8]
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA]
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9]
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB]
K25 = S5[z3] ^ S6[z2] ^ S7[zC] ^ S8[zD] ^ S5[z9]
K26 = S5[z1] ^ S6[z0] ^ S7[zE] ^ S8[zF] ^ S6[zC]
K27 = S5[z7] ^ S6[z6] ^ S7[z8] ^ S8[z9] ^ S7[z2]
K28 = S5[z5] ^ S6[z4] ^ S7[zA] ^ S8[zB] ^ S8[z6]
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0]
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2]
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1]
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3]
K29 = S5[x8] ^ S6[x9] ^ S7[x7] ^ S8[x6] ^ S5[x3]
K30 = S5[xA] ^ S6[xB] ^ S7[x5] ^ S8[x4] ^ S6[x7]
K31 = S5[xC] ^ S6[xD] ^ S7[x3] ^ S8[x2] ^ S7[x8]
K32 = S5[xE] ^ S6[xF] ^ S7[x1] ^ S8[x0] ^ S8[xD]
```

A.2. Masking Subkeys And Rotate Subkeys

Let $K_{m_1}, \ldots, K_{m_{16}}$ be 32-bit masking subkeys (one per round).
Let K_{r_1}, , $K_{r_{16}}$ be 32-bit rotate subkeys (one per round); only the least significant 5 bits are used in each round.

```
for (i=1; i<=16; i++)  { Kmi = Ki;  Kri = K16+i; }
```

Appendix B.

This appendix provides the contents of the CAST-128 s-boxes (see Section 5).

S-Box S1
```
30fb40d4 9fa0ff0b 6beccd2f 3f258c7a 1e213f2f 9c004dd3 6003e540 cf9fc949 bfd4af27
88bbbdb5 e2034090 98d09675 6e63a0e0 15c361d2 c2e7661d 22d4ff8e 28683b6f c07fd059
ff2379c8 775f50e2 43c340d3 df2f8656 887ca41a a2d2bd2d a1c9e0d6 346c4819 61b76d87
22540f2f 2abe32e1 aa54166b 22568e3a a2d341d0 66db40c8 a784392f 004dff2f 2db9d2de
97943fac 4a97c1d8 527644b7 b5f437a7 b82cbaef d751d159 6ff7f0ed 5a097a1f 827b68d0
90ecf52e 22b0c054 bc8e5935 4b6d2f7f 50bb64a2 d2664910 bee5812d b7332290 e93b159f
b48ee411 4bff345d fd45c240 ad31973f c4f6d02e 55fc8165 d5b1caad a1ac2dae a2d4b76d
c19b0c50 882240f2 0c6e4f38 a4e4bfd7 4f5ba272 564c1d2f c59c5319 b949e354 b04669fe
b1b6ab8a c71358dd 6385c545 110f935d 57538ad5 6a390493 e63d37e0 2a54f6b3 3a787d5f
6276a0b5 19a6fcdf 7a42206a 29f9d4d5 f61b1891 bb72275e aa508167 38901091 c6b505eb
84c7cb8c 2ad75a0f 874a1427 a2d1936b 2ad286af aa56d291 d7894360 425c750d 93b39e26
187184c9 6c00b32d 73e2bb14 a0bebc3c 54623779 64459eab 3f328b82 7718cf82 59a2cea6
04ee002e 89fe78e6 3fab0950 325ff6c2 81383f05 6963c5c8 76cb5ad6 d49974c9 ca180dcf
380782d5 c7fa5cf6 8ac31511 35e79e13 47da91d0 f40f9086 a7e2419e 31366241 051ef495
aa573b04 4a805d8d 548300d0 00322a3c bf64cddf ba57a68e 75c6372b 50afd341 a7c13275
915a0bf5 6b54bfab 2b0b1426 ab4cc9d7 449ccd82 f7fbf265 ab85c5f3 1b55db94 aad4e324
cfa4bd3f 2deaa3e2 9e204d02 c8bd25ac eadf55b3 d5bd9e98 e31231b2 2ad5ad6c 954329de
adbe4528 d8710f69 aa51c90f aa786bf6 22513f1e aa51a79b 2ad344cc 7b5a41f0 d37cfbad
1b069505 41ece491 b4c332e6 032268d4 c9600acc ce387e6d bf6bb16c 6a70fb78 0d03d9c9
d4df39de e01063da 4736f464 5ad328d8 b347cc96 75bb0fc3 98511bfb 4ffbcc35 b58bcf6a
e11f0abc bfc5fe4a a70aec10 ac39570a 3f04442f 6188b153 e0397a2e 5727cb79 9ceb418f
1cacd68d 2ad37c96 0175cb9d c69dff09 c75b65f0 d9db40d8 ec0e7779 4744ead4 b11c3274
dd24cb9e 7e1c54bd f01144f9 d2240eb1 9675b3fd a3ac3755 d47c27af 51c85f4d 56907596
a5bb15e6 580304f0 ca042cf1 011a37ea 8dbfaadb 35ba3e4a 3526ffa0 c37b4d09 bc306ed9
98a52666 5648f725 ff5e569d 0ced63d0 7c63b2cf 700b45e1 d5ea50f1 85a92872 af1fbda7
d4234870 a7870bf3 2d3b4d79 42e04198 0cd0ede7 26470db8 f881814c 474d6ad7 7c0c5e5c
d1231959 381b7298 f5d2f4db ab838653 6e2f1e23 83719c9e bd91e046 9a56456e dc39200c
20c8c571 962bda1c e1e696ff b141ab08 7cca89b9 1a69e783 02cc4843 a2f7c579 429ef47d
427b169c 5ac9f049 dd8f0f00 5c8165bf
```

S-Box S2

```
1f201094 ef0ba75b 69e3cf7e 393f4380 fe61cf7a eec5207a 55889c94 72fc0651 ada7ef79
4e1d7235 d55a63ce de0436ba 99c430ef 5f0c0794 18dcdb7d a1d6eff3 a0b52f7b 59e83605
ee15b094 e9ffd909 dc440086 ef944459 ba83ccb3 e0c3cdfb d1da4181 3b092ab1 f997f1c1
a5e6cf7b 01420ddb e4e7ef5b 25a1ff41 e180f806 1fc41080 179bee7a d37ac6a9 fe5830a4
98de8b7f 77e83f4e 79929269 24fa9f7b e113c85b acc40083 d7503525 f7ea615f 62143154
0d554b63 5d681121 c866c359 3d63cf73 cee234c0 d4d87e87 5c672b21 071f6181 39f7627f
361e3084 e4eb573b 602f64a4 d63acd9c 1bbc4635 9e81032d 2701f50c 99847ab4 a0e3df79
ba6cf38c 10843094 2537a95e f46f6ffe a1ff3b1f 208cfb6a 8f458c74 d9e0a227 4ec73a34
fc884f69 3e4de8df ef0e0088 3559648d 8a45388c 1d804366 721d9bfd a58684bb e8256333
844e8212 128d8098 fed33fb4 ce280ae1 27e19ba5 d5a6c252 e49754bd c5d655dd eb667064
77840b4d a1b6a801 84db26a9 e0b56714 21f043b7 e5d05860 54f03084 066ff472 a31aa153
dadc4755 b5625dbf 68561be6 83ca6b94 2d6ed23b eccf01db a6d3d0ba b6803d5c af77a709
33b4a34c 397bc8d6 5ee22b95 5f0e5304 81ed6f61 20e74364 b45e1378 de18639b 881ca122
b96726d1 8049a7e8 22b7da7b 5e552d25 5272d237 79d2951c c60d894c 488cb402 1ba4fe5b
a4b09f6b 1ca815cf a20c3005 8871df63 b9de2fcb 0cc6c9e9 0beeff53 e3214517 b4542835
9f63293c ee41e729 6e1d2d7c 50045286 1e6685f3 f33401c6 30a22c95 31a70850 60930f13
73f98417 a1269859 ec645c44 52c877a9 cdff33a6 a02b1741 7cbad9a2 2180036f 50d99c08
cb3f4861 c26bd765 64a3f6ab 80342676 25a75e7b e4e6d1fc 20c710e6 cdf0b680 17844d3b
31eef84d 7e0824e4 2ccb49eb 846a3bae 8ff77888 ee5d60f6 7af75673 2fdd5cdb a11631c1
30f66f43 b3faec54 157fd7fa ef8579cc d152de58 db2ffd5e 8f32ce19 306af97a 02f03ef8
99319ad5 c242fa0f a7e3ebb0 c68e4906 b8da230c 80823028 dcdef3c8 d35fb171 088a1bc8
bec0c560 61a3c9e8 bca8f54d c72feffa 22822e99 82c570b4 d8d94e89 8b1c34bc 301e16e6
273be979 b0ffeaa6 61d9b8c6 00b24869 b7ffce3f 08dc283b 43daf65a f7e19798 7619b72f
8f1c9ba4 dc8637a0 16a7d3b1 9fc393b7 a7136eeb c6bcc63e 1a513742 ef6828bc 520365d6
2d6a77ab 3527ed4b 821fd216 095c6e2e db92f2fb 5eea29cb 145892f5 91584f7f 5483697b
2667a8cc 85196048 8c4bacea 833860d4 0d23e0f9 6c387e8a 0ae6d249 b284600c d835731d
dcb1c647 ac4c56ea 3ebd81b3 230eabb0 6438bc87 f0b5b1fa 8f5ea2b3 fc184642 0a036b7a
4fb089bd 649da589 a345415e 5c038323 3e5d3bb9 43d79572 7e6dd07c 06dfdf1e 6c6cc4ef
7160a539 73bfbe70 83877605 4523ecf1
```

S-Box S3

```
8defc240 25fa5d9f eb903dbf e810c907 47607fff 369fe44b 8c1fc644 aececa90 beb1f9bf
eefbcaea e8cf1950 51df07ae 920e8806 f0ad0548 e13c8d83 927010d5 11107d9f 07647db9
b2e3e4d4 3d4f285e b9afa820 fade82e0 a067268b 8272792e 553fb2c0 489ae22b d4ef9794
125e3fbc 21fffcee 825b1bfd 9255c5ed 1257a240 4e1a8302 bae07fff 528246e7 8e57140e
3373f7bf 8c9f8188 a6fc4ee8 c982b5a5 a8c01db7 579fc264 67094f31 f2bd3f5f 40fff7c1
1fb78dfc 8e6bd2c1 437be59b 99b03dbf b5dbc64b 638dc0e6 55819d99 a197c81c 4a012d6e
c5884a28 ccc36f71 b843c213 6c0743f1 8309893c 0feddd5f 2f7fe850 d7c07f7e 02507fbf
5afb9a04 a747d2d0 1651192e af70bf3e 58c31380 5f98302e 727cc3c4 0a0fb402 0f7fef82
8c96fdad 5d2c2aae 8ee99a49 50da88b8 8427f4a0 1eac5790 796fb449 8252dc15 efbd7d9b
a672597d ada840d8 45f54504 fa5d7403 e83ec305 4f91751a 925669c2 23efe941 a903f12e
60270df2 0276e4b6 94fd6574 927985b2 8276dbcb 02778176 f8af918d 4e48f79e 8f616ddf
e29d840e 842f7d83 340ce5c8 96bbb682 93b4b148 ef303cab 984faf28 779faf9b 92dc560d
224d1e20 8437aa88 7d29dc96 2756d3dc 8b907cee b51fd240 e7c07ce3 e566b4a1 c3e9615e
3cf8209d 6094d1e3 cd9ca341 5c76460e 00ea983b d4d67881 fd47572c f76cedd9 bda8229c
127dadaa 438a074e 1f97c090 081bdb8a 93a07ebe b938ca15 97b03cff 3dc2c0f8 8d1ab2ec
64380e51 68cc7bfb d90f2788 12490181 5de5ffd4 dd7ef86a 76a2e214 b9a40368 925d958f
4b39fffa ba39aee9 a4ffd30b faf7933b 6d498623 193cbcfa 27627545 825cf47a 61bd8ba0
d11e42d1 cead04f4 127ea392 10428db7 8272a972 9270c4a8 127de50b 285ba1c8 3c62f44f
35c0eaa5 e805d231 428929fb b4fcdf82 4fb66a53 0e7dc15b 1f081fab 108618ae fcfd086d
f9ff2889 694bcc11 236a5cae 12deca4d 2c3f8cc5 d2d02dfe f8ef5896 e4cf52da 95155b67
494a488c b9b6a80c 5c8f82bc 89d36b45 3a609437 ec00c9a9 44715253 0a874b49 d773bc40
7c34671c 02717ef6 4feb5536 a2d02fff d2bf60c4 d43f03c0 50b4ef6d 07478cd1 006e1888
a2e53f55 b9e6d4bc a2048016 97573833 d7207d67 de0f8f3d 72f87b33 abcc4f33 7688c55d
7b00a6b0 947b0001 570075d2 f9bb88f8 8942019e 4264a5ff 856302e0 72dbd92b ee971b69
6ea22fde 5f08ae2b af7a616d e5c98767 cf1febd2 61efc8c2 f1ac2571 cc8239c2 67214cb8
b1e583d1 b7dc3e62 7f10bdce f90a5c38 0ff0443d 606e6dc6 60543a49 5727c148 2be98a1d
8ab41738 20e1be24 af96da0f 68458425 99833be5 600d457d 282f9350 8334b362 d91d1120
2b6d8da0 642b1e31 9c305a00 52bce688 1b03588a f7baefd5 4142ed9c a4315c11 83323ec5
dfef4636 a133c501 e9d3531c ee353783
```

S-Box S4

```
9db30420 1fb6e9de a7be7bef d273a298 4a4f7bdb 64ad8c57 85510443 fa020ed1 7e287aff
e60fb663 095f35a1 79ebf120 fd059d43 6497b7b1 f3641f63 241e4adf 28147f5f 4fa2b8cd
c9430040 0cc32220 fdd30b30 c0a5374f 1d2d00d9 24147b15 ee4d111a 0fca5167 71ff904c
2d195ffe 1a05645f 0c13fefe 081b08ca 05170121 80530100 e83e5efe ac9af4f8 7fe72701
d2b8ee5f 06df4261 bb9e9b8a 7293ea25 ce84ffdf f5718801 3dd64b04 a26f263b 7ed48400
547eebe6 446d4ca0 6cf3d6f5 2649abdf aea0c7f5 36338cc1 503f7e93 d3772061 11b638e1
72500e03 f80eb2bb abe0502e ec8d77de 57971e81 e14f6746 c9335400 6920318f 081dbb99
ffc304a5 4d351805 7f3d5ce3 a6c866c6 5d5bcca9 daec6fea 9f926f91 9f46222f 3991467d
a5bf6d8e 1143c44f 43958302 d0214eeb 022083b8 3fb6180c 18f8931e 281658e6 26486e3e
8bd78a70 7477e4c1 b506e07c f32d0a25 79098b02 e4eabb81 28123b23 69dead38 1574ca16
df871b62 211c40b7 a51a9ef9 0014377b 041e8ac8 09114003 bd59e4d2 e3d156d5 4fe876d5
2f91a340 557be8de 00eae4a7 0ce5c2ec 4db4bba6 e756bdff dd3369ac ec17b035 06572327
99afc8b0 56c8c391 6b65811c 5e146119 6e85cb75 be07c002 c2325577 893ff4ec 5bbfc92d
d0ec3b25 b7801ab7 8d6d3b24 20c763ef c366a5fc 9c382880 0ace3205 aac9548a eca1d7c7
041afa32 1d16625a 6701902c 9b757a54 31d477f7 9126b031 36cc6fdb c70b8b46 d9e66a48
56e55a79 026a4ceb 52437eff 2f8f76b4 0df980a5 8674cde3 edda04eb 17a9be04 2c18f4df
```

```
b7747f9d ab2af7b4 efc34d20 2e096b7c 1741a254 e5b6a035 213d42f6 2c1c7c26 61c2f50f
6552daf9 d2c231f8 25130f69 d8167fa2 0418f2c8 001a96a6 0d1526ab 63315c21 5e0a72ec
49bafefd 187908d9 8d0dbd86 311170a7 3e9b640c cc3e10d7 d5cad3b6 0caec388 f73001e1
6c728aff 71eae2a1 1f9af36e cfcbd12f c1de8417 ac07be6b cb44a1d8 8b9b0f56 013988c3
b1c52fca b4be31cd d8782806 12a3a4e2 6f7de532 58fd7eb6 d01ee900 24adffc2 f4990fc5
9711aac5 001d7b95 82e5e7d2 109873f6 00613096 c32d9521 ada121ff 29908415 7fbb977f
af9eb3db 29c9ed2a 5ce2a465 a730f32c d0aa3fe8 8a5cc091 d49e2ce7 0ce454a9 d60acd86
015f1919 77079103 dea03af6 78a8565e dee356df 21f05cbe 8b75e387 b3c50651 b8a5c3ef
d8eeb6d2 e523be77 c2154529 2f69efdf afe67afb f470c4b2 f3e0eb5b d6cc9876 39e4460c
1fda8538 1987832f ca007367 a99144f8 296b299e 492fc295 9266beab b5676e69 9bd3ddda
df7e052f db25701c 1b5e51ee f65324e6 6afce36c 0316cc04 8644213e b7dc59d0 7965291f
ccd6fd43 41823979 932bcdf6 b657c34d 4edfd282 7ae5290c 3cb9536b 851e20fe 9833557e
13ecf0b0 d3ffb372 3f85c5c1 0aef7ed2
```

S-Box S5

```
7ec90c04 2c6e74b9 9b0e66df a6337911 b86a7fff 1dd358f5 44dd9d44 1731167f 08fbf1fa
e7f511cc d2051b00 735aba00 2ab722d8 386381cb acf6243a 69befd7a e6a2e77f f0c720cd
c4494816 ccf5c180 38851640 15b0a848 e68b18cb 4caadeff 5f480a01 0412b2aa 259814fc
41d0efe2 4e40b48d 248eb6fb 8dba1cfe 41a99b02 1a550a04 ba8f65cb 7251f4e7 95a51725
c106ecd7 97a5980a c539b9aa 4d79fe6a f2f3f763 68af8040 ed0c9e56 11b4958b e1eb5a88
8709e6b0 d7e07156 4e29fea7 6366e52d 02d1c000 c4ac8e05 9377f571 0c05372a 578535f2
2261be02 d642a0c9 df13a280 74b55bd2 682199c0 d421e5ec 53fb3ce8 c8adedb3 28a87fc9
3d959981 5c1ff900 fe38d399 0c4eff0b 062407ea aa2f4fb1 4fb96976 90c79505 b0a8a774
ef55a1ff e59ca2c2 a6b62d27 e66a4263 df65001f 0ec50966 dfdd55bc 29de0655 911e739a
17af8975 32c7911c 89f89468 0d01e980 524755f4 03b63cc9 0cc844b2 bcf3f0aa 87ac36e9
e53a7426 01b3d82b 1a9e7449 64ee2d7e cddbb1da 01c94910 b868bf80 0d26f3fd 9342ede7
04a5c284 636737b6 50f5b616 f24766e3 8eca36c1 136e05db fef18391 fb887a37 d6e7f7d4
c7fb7dc9 3063fcdf b6f589de ec2941da 26e46695 b7566419 f654efc5 d08d58b7 48925401
c1bacb7f e5ff550f b6083049 5bb5d0e8 87d72e5a ab6a6ee1 223a66ce c62bf3cd 9e0885f9
68cb3e47 086c010f a21de820 d18b69de f3f65777 fa02c3f6 407edac3 cbb3d550 1793084d
b0d70eba 0ab378d5 d951fb0c ded7da56 4124bbe4 94ca0b56 0f5755d1 e0e1e56e 6184b5be
580a249f 94f74bc0 e327888e 9f7b5561 c3dc0280 05687715 646c6bd7 44904db3 66b4f0a3
c0f1648a 697ed5af 49e92ff6 309e374f 2cb6356a 85808573 4991f840 76f0ae02 083be84d
28421c9a 44489406 736e4cb8 c1092910 8bc95fc6 7d869cf4 134f616f 2e77118d b31b2be1
aa90b472 3ca5d717 7d161bba 9cad9010 af462ba2 9fe459d2 45d34559 d9f2da13 dbc65487
f3e4f94e 176d486f 097c13ea 631da5c7 445f7382 175683f4 cdc66a97 70be0288 b3cdcf72
6e5dd2f3 20936079 459b80a5 be60e2db a9c23101 eba5315c 224e42f2 1c5c1572 f6721b2c
1ad2fff3 8c25404e 324ed72f 4067b7fd 0523138e 5ca3bc78 dc0fd66e 75922283 784d6b17
58ebb16e 44094f85 3f481d87 fcfeae7b 77b5ff76 8c2302bf aaf47556 5f46b02a 2b092801
3d38f5f7 0ca81f36 52af4a8a 66d5e7c0 df3b0874 95055110 1b5ad7a8 f61ed5ad 6cf6e479
20758184 d0cefa65 88f7be58 4a046826 0ff6f8f3 a09c7f70 5346aba0 5ce96c28 e176eda3
6bac307f 376829d2 85360fa9 17e3fe2a 24b79767 f5a96b20 d6cd2595 68ff1ebf 7555442c
f19f06be f9e0659a eeb9491d 34010718 bb30cab8 e822fe15 88570983 750e6249 da627e55
5e76ffa8 b1534546 6d47de08 efe9e7d4
```

S-Box S6

```
f6fa8f9d 2cac6ce1 4ca34867 e2337f7c 95db08e7 016843b4 eced5cbc 325553ac bf9f0960
dfa1e2ed 83f0579d 63ed86b9 1ab6a6b8 de5ebe39 f38ff732 8989b138 33f14961 c01937bd
f506c6da e4625e7e a308ea99 4e23e33c 79cbd7cc 48a14367 a3149619 fec94bd5 a114174a
eaa01866 a084db2d 09a8486f a888614a 2900af98 01665991 e1992863 c8f30c60 2e78ef3c
d0d51932 cf0fec14 f7ca07d2 d0a82072 fd41197e 9305a6b0 e86be3da 74bed3cd 372da53c
4c7f4448 dab5d440 6dba0ec3 083919a7 9fbaeed9 49dbcfb0 4e670c53 5c3d9c01 64bdb941
2c0e636a ba7dd9cd ea6f7388 e70bc762 35f29adb 5c4cdd8d f0d48d8c b88153e2 08a19866
1ae2eac8 284caf89 aa928223 9334be53 3b3a21bf 16434be3 9aea3906 efe8c36e f890cdd9
80226dae c340a4a3 df7e9c09 a694a807 5b7c5ecc 221db3a6 9a69a02f 68818a54 ceb2296f
53c0843a fe893655 25bfe68a b4628abc cf222ebf 25ac6f48 a9a99387 53bddb65 e76ffbe7
e967fd78 0ba93563 8e342bc1 e8a11be9 4980740d c8087dfc 8de4bf99 a11101a0 7fd37975
da5a26c0 e81f994f 9528cd89 fd339fed b87834bf 5f04456d 22258698 c9c4c83b 2dc156be
4f628daa 57f55ec5 e2220abe d2916ebf 4ec75b95 24f2c3c0 42d15d99 cd0d7fa0 7b6e27ff
a8dc8af0 7345c106 f41e232f 35162386 e6ea8926 3333b094 157ec6f2 372b74af 692573e4
e9a9d848 f3160289 3a62ef1d a787e238 f3a5f676 74364853 20951063 4576698d b6fad407
592af950 36f73523 4cfb6e87 7da4cec0 6c152daa cb0396a8 c50dfe5d fcd707ab 0921c42f
89dff0bb 5fe2be78 448f4f33 754613c9 2b05d08d 48b9d585 dc049441 c8098f9b 7dede786
c39a3373 42410005 6a091751 0ef3c8a6 890072d6 28207682 a9a9f7be bf32679d d45b5b75
b353fd00 cbb0e358 830f220a 1f8fb214 d372cf08 cc3c4a13 8cf63166 061c87be 88c98f88
6062e397 47cf8e7a b6c85283 3cc2acfb 3fc06976 4e8f0252 64d8314d da3870e3 1e665459
c10908f0 513021a5 6c5b68b7 822f8aa0 3007cd3e 74719eef dc872681 073340d4 7e432fd9
0c5ec241 8809286c f592d891 08a930f6 957ef305 b7fbffbd c266e96f 6fe4ac98 b173ecc0
bc60b42a 953498da fba1ae12 2d4bd736 0f25faab a4f3fceb e2969123 257f0c3d 9348af49
361400bc e8816f4a 3814f200 a3f94043 9c7a54c2 bc704f57 da41e7f9 c25ad33a 54f4a084
b17f5505 59357cbe edbd15c8 7f97c5ab ba5ac7b5 b6f6deaf 3a479c3a 5302da25 653d7e6a
54268d49 51a477ea 5017d55b d7d25d88 44136c76 0404a8c8 b8e5a121 b81a928a 60ed5869
97c55b96 eaec991b 29935913 01fdb7f1 088e8dfa 9ab6f6f5 3b4cbf9f 4a5de3ab e6051d35
a0e1d855 d36b4cf1 f544edeb b0e93524 bebb8fbd a2d762cf 49c92f54 38b5f331 7128a454
48392905 a65b1db8 851c97bd d675cf2f
```

S-Box S7

```
85e04019 332bf567 662dbfff cfc65693 2a8d7f6f ab9bc912 de6008a1 2028da1f 0227bce7
4d642916 18fac300 50f18b82 2cb2cb11 b232e75c 4b3695f2 b28707de a05fbcf6 cd4181e9
e150210c e24ef1bd b168c381 fde4e789 5c79b0d8 1e8bfd43 4d495001 38be4341 913cee1d
```

```
92a79c3f 089766be baeeadf4 1286becf b6eacb19 2660c200 7565bde4 64241f7a 8248dca9
c3b3ad66 28136086 0bd8dfa8 356d1cf2 107789be b3b2e9ce 0502aa8f 0bc0351e 166bf52a
eb12ff82 e3486911 d34d7516 4e7b3aff 5f43671b 9cf6e037 4981ac83 334266ce 8c9341b7
d0d854c0 cb3a6c88 47bc2829 4725ba37 a66ad22b 7ad61f1e 0c5cbafa 4437f107 b6e79962
42d2d816 0a961288 e1a5c06e 13749e67 72fc081a b1d139f7 f9583745 cf19df58 bec3f756
c06eba30 07211b24 45c28829 c95e317f bc8ec511 38bc46e9 c6e6fa14 bae8584a ad4ebc46
468f508b 7829435f f124183b 821dba9f aff60ff4 ea2c4e6d 16e39264 92544a8b 009b4fc3
aba68ced 9ac96f78 06a5b79a b2856e6e 1aec3ca9 be838688 0e0804e9 55f1be56 e7e5363b
b3a1f25d f7debb85 61fe033c 16746233 3c034c28 da6d0c74 79aac56c 3ce4e1ad 51f0c802
98f8f35a 1626a49f eed82b29 1d382fe3 0c4fb99a bb325778 3ec6d97b 6e77a6a9 cb658b5c
d45230c7 2bd1408b 60c03eb7 b9068d78 a33754f4 f430c87d c8a71302 b96d8c32 ebd4e7be
be8b9d2d 7979fb06 e7225308 8b75cf77 11ef8da4 e083c858 8d6b786f 5a6317a6 fa5cf7a0
5dda0033 f28ebfb0 f5b9c310 a0eac280 08b9767a a3d9d2b0 79d34217 021a718d 9ac6336a
2711fd60 438050e3 069908a8 3d7fedc4 826d2bef 4eeb8476 488dcf25 36c9d566 28e74e41
c2610aca 3d49a9cf bae3b9df b65f8de6 92aeaf64 3ac7d5e6 9ea80509 f22b017d a4173f70
dd1e16c3 15e0d7f9 50b1b887 2b9f4fd5 625aba82 6a017962 2ec01b9c 15488aa9 d716e740
40055a2c 93d29a22 e32dbf9a 058745b9 3453dc1e d699296e 496cff6f 1c9f4986 dfe2ed07
b87242d1 19de7eae 053e561a 15ad6f8c 66626c1c 7154c24c ea082b2a 93eb2939 17dcb0f0
58d4f2ae 9ea294fb 52cf564c 9883fe66 2ec40581 763953c3 01d6692e d3a0c108 a1e7160e
e4f2dfa6 693ed285 74904698 4c2b0edd 4f757656 5d393378 a132234f 3d321c5d c3f5e194
4b269301 c79f022f 3c997e7e 5e4f9504 3ffafbbd 76f7ad0e 296693f4 3d1fce6f c61e45be
d3b5ab34 f72bf9b7 1b0434c0 4e72b567 5592a33d b5229301 cfd2a87f 60aeb767 1814386b
30bcc33d 38a0c07d fd1606f2 c363519b 589dd390 5479f8e6 1cb8d647 97fd61a9 ea7759f4
2d57539d 569a58cf e84e63ad 462e1b78 6580f87e f3817914 91da55f4 40a230f3 d1988f35
b6e318d2 3ffa50bc 3d40f021 c3c0bdae 4958c24c 518f36b2 84b1d370 0fedce83 878ddada
f2a279c7 94e01be8 90716f4b 954b8aa3
```

S-Box S8

```
e216300d bbddfffc a7ebdabd 35648095 7789f8b7 e6c1121b 0e241600 052ce8b5 11a9cfb0
e5952f11 ece7990a 9386d174 2a42931c 76e38111 b12def3a 37ddddfc de9adeb1 0a0cc32c
be197029 84a00940 bb243a0f b4d137cf b44e79f0 049eedfd 0b15a15d 480d3168 8bbbde5a
669ded42 c7ece831 3f8f95e7 72df191b 7580330d 94074251 5c7dcdfa abbe6d63 aa402164
b301d40a 02e7d1ca 53571dae 7a3182a2 12a8ddec fdaa335d 176f43e8 71fb46d4 38129022
ce949ad4 b84769ad 965bd862 82f3d055 66fb9767 15b80b4e 1d5b47a0 4cfde06f c28ec4b8
57e8726e 647a78fc 99865d44 608bd593 6c200e03 39dc5ff6 5d0b00a3 ae63aff2 7e8bd632
70108c0c bbd35049 2998df04 980cf42a 9b6df491 9e7edd53 06918548 58cb7e07 3b74ef2e
522fffb1 d24708cc 1c7e27cd a4eb215b 3cf1d2e2 19b47a38 424f7618 35856039 9d17dee7
27eb35e6 c9aff67b 36baf5b8 09c467cd c18910b1 e11dbf7b 06cd1af8 7170c608 2d5e3354
d4de495a 64c6d006 bcc0c62c 3dd00db3 708f8f34 77d51b42 264f620f 24b8d2bf 15c1b79e
46a52564 f8d7e54e 3e378160 7895cda5 859c15a5 e6459788 c37bc75f db07ba0c 0676a3ab
7f229b1e 31842e7b 24259fd7 f8bef472 835ffcb8 6df4c1f2 96f5b195 fd0af0fc b0fe134c
e2506d3d 4f9b12ea f215f225 a223736f 9fb4c428 25d04979 34c713f8 c4618187 ea7a6e98
7cd16efc 1436876c f1544107 bedeee14 56e9af27 a04aa441 3cf7c899 92ecbae6 dd67016d
151682eb a842eedf fdba60b4 f1907b75 20e3030f 24d8c29e e139673b efa63fb8 71873054
b6f2cf3b 9f326442 cb15a4cc b01a4504 f1e47d8d 844a1be5 bae7dfdc 42cbda70 cd7dae0a
57e85b7a d53f5af6 20cf4d8c cea4d428 79d130a4 3486ebfb 33d3cddc 77853b53 37effcb5
c5068778 e580b3e6 4e68b8f4 c5c8b37e 0d809ea2 398feb7c 132a4f94 43b7950e 2fee7d1c
223613bd dd06caa2 37df932b c4248289 acf3ebc3 5715f6b7 ef3478dd f267616f c148cbe4
9052815e 5e410fab b48a2465 2eda7fa4 e87b40e4 e98ea084 5889e9e1 efd390fc dd07d35b
db485694 38d7e5b2 57720101 730edebc 5b643113 94917e4f 503c2fba 646f1282 7523d24a
e0779695 f9c17a8f 7a5b2121 d187b896 29263a4d ba510cdf 81f47c9f ad1163ed ea7b5965
1a00726e 11403092 00da6d77 4a0cdd61 ad1f4603 605bdfb0 9eedc364 22ebe6a8 cee7d28a
a0e736a0 5564a6b9 10853209 c7eb8f37 2de705ca 8951570f df09822b bd691a6c aa12e4f2
87451c0f e0f6a27a 3ada4819 4cf1764f 0d771c2b 67cdb156 350d8384 5938fa0f 42399ef3
36997b07 0e84093d 4aa93e61 8360d87b 1fa98b0c 1149382c e97625a5 0614d1b7 0e25244b
0c768347 589e8d82 0d2059d1 a466bb1e f8da0a82 04f19130 ba6e4ec0 99265164 1ee7230d
50b2ad80 eaee6801 8db2a283 ea8bf59e
```

Appendix C.

This appendix provides test vectors for the CAST-128 cipher described in Section 5 and in Appendices A and B.

C.1. Single Key-Plaintext-Ciphertext Set

```
128-bit key       = 01 23 45 67 12 34 56 78 23 45 67 89 34 56 78 9A (hex)
64-bit plaintext  = 01 23 45 67 89 AB CD EF (hex)
64-bit ciphertext = 23 8B 4F E5 84 7E 44 B2 (hex)
```

C.2. Full Maintenance Test

A maintenance test for CAST-128 has been defined to verify the correctness of implementations. It is defined in pseudo-code as follows, where *a* and *b* are 128-bit vectors, *aL* and *aR* are the leftmost and rightmost halves of *a*, *bL* and *bR* are the leftmost and rightmost halves of *b*, and encrypt(*d*,*k*) is the encryption in ECB mode of block *d* under key *k*.

```
Initial a = 01 23 45 67 12 34 56 78 23 45 67 89 34 56 78 9A (hex)
Initial b = 01 23 45 67 12 34 56 78 23 45 67 89 34 56 78 9A (hex)

do 1,000,000 times
{
    aL = encrypt(aL,b)
    aR = encrypt(aR,b)
    bL = encrypt(bL,a)
    bR = encrypt(bR,a)
}

Verify a == EE A9 D0 A2 49 FD 3B A6 B3 43 6F B8 9D 6D CA 92 (hex)
Verify b == B2 C9 5E B0 0C 31 AD 71 80 AC 05 B8 E8 3D 69 6E (hex)
```