Software Performance Characterization of Block Cipher Structures Using S-boxes and Linear Mappings*

Lu Xiao[1] and Howard M. Heys[2]

[1]QUALCOMM Incorporated, lxiao@qualcomm.com
[2]Electrical and Computer Engineering, Faculty of Engineering and Applied Science, Memorial University of Newfoundland, howard@engr.mun.ca

Abstract

In this paper, we present a new framework for evaluating the performance characteristics of block cipher structures composed of S-boxes and Maximum Distance Separable (MDS) mappings. In particular, a novel performance metric is introduced and applied to nested Substitution-Permutation Networks (SPNs) and Feistel networks with round functions composed of S-boxes and MDS mappings. Within each cipher structure, many cases are considered based on two types of S-boxes (4×4 and 8×8) and parameterized MDS mappings. In our study of each case, the performance is analyzed based on a table lookup implementation. Although this implementation method is the typical approach used for software realization, it may also be applicable to hardware realization in some instances. Cipher security, in the form of resistance to differential and linear attacks, is applied as a basis which is used to normalize the performance in the analysis. Because the discussed structures are similar to many existing ciphers such as AES and Camellia, the analysis provides a meaningful mechanism for seeking efficient ciphers through a wide comparison of security, performance, and implementation methods.

Keywords: encryption, cipher, cryptography, key, security, cryptanalysis, software.

1 Introduction

A *block cipher* is an algorithm to encrypt constant-length blocks of data (*plaintext*) to a random-looking format of the same length (*ciphertext*) by using another block of data called

*This work was undertaken during the first author's graduate study at Memorial University of Newfoundland and funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

a *key* [1]. In a *symmetric key block cipher*, the same key is used for both encryption and decryption. Since decryption is computationally difficult without knowledge of the key, for secure communication applications, encrypted data can be protected from unwanted eavesdropping as long as the key is kept private to only the transmitter and receiver. A block cipher (usually referring to a symmetric key block cipher) is typically composed of several rounds of simple cryptographic operations. The cipher key is expanded to a number of *subkeys* by a key schedule and the subkeys are mixed with data blocks in different rounds typically by bitwise eXclusive-ORs (XORs). Because of its high performance, the block cipher is now widely used in various security applications [2] such as basic data encryption in Internet protocols (IPsec and SSL/TLS), wireless communications, and digital rights management.

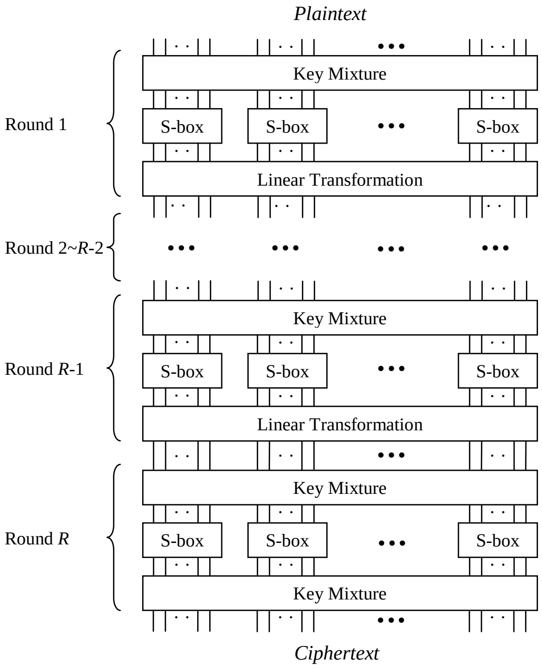


Figure 1: A Substitution-Permutation Network

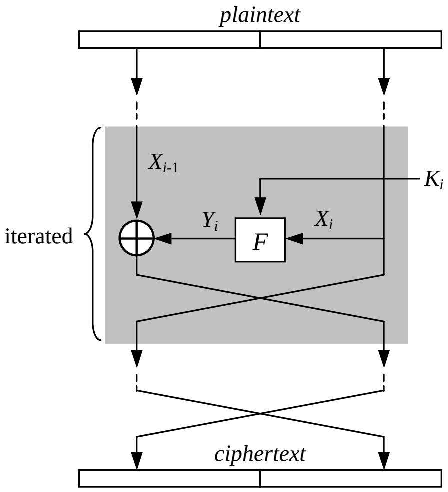


Figure 2: A Feistel Network

In a symmetric key block cipher, the concepts of *confusion* and *diffusion* are vital to security [3]. The Substitution-Permutation Network (SPN)[1] and the Feistel network are two typical architectures used to achieve this [1]. During encryption using an SPN cipher, as

[1] For historical reasons, these ciphers are referred to as Substitution-Permutation Networks although the permutation layer is now typically replaced by an invertible linear transformation layer to improve resistance to differential and linear cryptanalysis [4].

Figure 1 illustrates, typically the input data of each round is mixed with subkey bits before entering the Substitution-boxes (S-boxes). Each S-box performs a nonlinear mapping on small sub-blocks thus creating confusion in the data. An $m{\times}n$ S-box performs a mapping from m input bits to n output bits. The outputs of S-boxes are modified by a linear transformation whose purpose is to generate a diffusion of statistical effects in the data. The decryption is composed of the inverse linear transformations, the inverse S-boxes, and the key mixtures in reverse order. To maintain similar dataflow in encryption and decryption, SPNs typically omit the linear transformation in the last round of encryption.

As the other typical architecture of block ciphers, the Feistel network has been widely used and studied. In each round i of a Feistel network as shown in Figure 2, the right half of the round input (denoted as X_i) goes through function F parameterized by subkey K_i. Also called the *round function*, F often consists of key mixture, S-boxes, and a linear transformation [1]. The output of F, denoted as Y_i, is XORed bit-by-bit with the left half of the round input X_{i-1}. The round output is then derived by swapping X_i and $X_{i-1} \oplus Y_i$, where "$\oplus$" represents bitwise XOR.

As the only nonlinear component in both cipher architectures, the S-box is well designed to meet many security requirements. Different S-boxes in a cipher can have different mappings or one mapping can be used for all S-boxes in a cipher. In many recently proposed block ciphers, the outputs of a layer of parallel S-boxes are passed through a linear transformation based on a Maximum Distance Separable (MDS) code [5].

Recent initiatives in cryptography have focussed on the development of new block cipher standards. As the successor of the Data Encryption Standard (DES) [6], the SPN cipher Rijndael [7] was selected by the U.S. National Institute of Standards and Technology as the Advanced Encryption Standard (AES) [8] in November 2001. As a Feistel network proposed in [9], Camellia was included together with AES into the NESSIE[2] portfolio of 128-bit block ciphers in February 2003 [10]. Consequently, AES and Camellia are two important examples of ciphers that are expected to be widely used in cryptographic applications of the future.

In this paper, a novel performance metric is presented which allows us to consider the

[2] New European Schemes for Signatures, Integrity, and Encryption

performance of a cipher structure as the combination of security and efficiency. Many parameterized cases of 128-bit block ciphers are studied including cases which correspond closely to recently proposed ciphers such as AES and Camellia. The security of each cipher case is evaluated by considering the resistance to differential cryptanalysis [11] and linear cryptanalysis [12]. The software efficiency is mainly evaluated through a table-lookup implementation, where the number of table lookups is used as the time measure and the table size required is the space measure. A table-lookup implementation method is selected because it is usually efficient and such a method makes it possible to compare performance generally across different cipher configurations and different computing platforms. The efficiencies of other implementations (e.g., bitslicing, *xtime* [7], and power-index exchange) are also briefly examined in this paper. The significance and novelty of this work is that it facilitates the performance characterization of AES, Camellia, and many other existing or prospective ciphers from a cryptographic view. Our analysis is focused on software-oriented implementation on typical computer systems. However, similar techniques can be applied to hardware implementations where a table lookup method is used as the basis for the implementation.

2 Background

2.1 Block Ciphers

AES is an SPN with a block size of 128 bits. The key size can be 128, 192, and 256 bits, which requires 10, 12, and 14 iterations of the round structure, respectively. Each round of AES consists of a layer of 16 parallel 8×8 S-boxes, a linear transformation composed of the shifting of bytes and MDS mappings, and key mixture using XORs. The ciphers SHARK [13], Square [14], Anubis [15], Khazad [16], and Hierocrypt (including Type I, Type II, and Hierocrypt-3) [17, 18] are further examples of SPNs based on 8×8 S-boxes. Serpent [19] has a simple round structure based on small 4×4 S-boxes.

Camellia is an 18-round Feistel network for use with 128-bit keys or a 24-round Feistel network for use with 192- or 256-bit keys. The round function of Camellia consists of key mixture using XORs, 8 parallel 8×8 S-boxes, and a linear transformation. Camellia also contains input/output subkey mixtures and two logic functions every 6 rounds denoted as FL and FL^{-1}.

2.2 Differential and Linear Attacks

Differential cryptanalysis [11] and linear cryptanalysis [12] have been recognized as two of the most fundamental security threats to block ciphers. Differential attacks are chosen-plaintext attacks which exploit the statistical relation between the input difference and output difference of any two plaintext-ciphertext pairs. A *difference* is defined to be the bitwise XOR of two vectors. For example, in an SPN cipher of R rounds, an attacker hopes to find a mapping from some plaintext difference to some output difference of the (R–1)-th round, which occurs with a significant probability p. With this mapping (called a *differential*), the attacker can decrypt the corresponding ciphertexts one round with all possible key candidates of the last round. By checking for which key candidate the output difference of the differential holds for the calculated outputs of the $(R-1)$-th round most frequently, the valid key candidate of the last round can be distinguished.

Linear attacks are known-plaintext attacks which exploit linear expressions (XOR sums) consisting of a subset of input bits, output bits, and key bits. If all bit variables of a linear expression are random, the probability that the linear approximation holds is 0.5 for all plaintext-ciphertext pairs associated with the same cipher key. However, based on the cipher structure, an attacker may find some linear approximations exist with a probability quite different from 0.5. A possible attacking method is to find such a linear expression consisting of bit variables of the plaintext and output of the $(R-1)$-th round. Then the corresponding ciphertexts are decrypted one round with all possible candidates of the last round subkey. By checking for which key candidate the linear expression holds true with a probability bias significantly different than 0.5, the attacker can distinguish the valid key candidate from others.

For differential and linear cryptanalysis, once the last subkey is distinguished, it is straightforward for the attacker to use the same technique to determine key bits from the $(R-1)$-th round, the $(R-2)$-th round, etc.. To thwart these two attacks, cipher designers construct ciphers so that there are no large differential probabilities and there are no large biases from 0.5 for the probability that a linear expression holds. To achieve this, no highly likely *characteristics* should exist in the cipher. A *differential characteristic* of γ rounds is a sequence denoted as $(\Delta Z_0, \Delta Z_1, \cdots, \Delta Z_i, \cdots, \Delta Z_\gamma)$, where ΔZ_0 is the input difference of the first round, ΔZ_γ is the output difference of the last round, and ΔZ_i is the output difference of the i-th round and also the input of the $(i+1)$-th round $(0 < i < \gamma)$. Since any linear approximation of a data block or vector can be regarded as its inner product with a masking bit vector over GF(2), a *linear characteristic* is a sequence of masking values denoted as $(\Gamma Z_0, \Gamma Z_1, \cdots, \Gamma Z_i, \cdots, \Gamma Z_\gamma)$, where ΓZ_0 is the masking value for the first round input, ΓZ_γ is that for the last round output, and ΓZ_i is that for the i-th round output and also the $(i+1)$-th round input. An S-box is *active* if it is involved in the differential (respectively, linear) characteristic in a differential (respectively, linear) attack.

It should be noted that differential and linear cryptanalysis are practical approximations of more powerful theoretical attacks. Also depending on the specific cipher design, it is possible for some ciphers to be more vulnerable to other attacks, such as integral attacks [7].

2.3 Properties of S-boxes

The properties of the S-boxes in a cipher are important in the consideration of a cipher's security against differential and linear attacks. An $m \times n$ S-box, S, performs a mapping from an m-bit input X to an n-bit output Y. Considering all S-boxes, $\{S_i\}$, in a cipher, the maximum differential probability p_s is defined [11] as:

$$p_s \triangleq \max_i \max_{\Delta X \neq 0, \Delta Y} prob\{S_i(X) \oplus S_i(X \oplus \Delta X) = \Delta Y\}$$

where "$\oplus$" denotes a bitwise XOR and "ΔX" and "ΔY" denote bitwise XOR differences. The maximum linear probability[3] is defined [12] as:

$$q_s \triangleq \max_i \max_{\Gamma Y \neq 0, \Gamma X} (2 \times prob\{X \cdot \Gamma X = S_i(X) \cdot \Gamma Y\} - 1)^2$$

where "$\cdot$" denotes a bitwise inner product and ΓX and ΓY denote masking variables. In this paper, all 4×4 S-boxes are assumed to satisfy $p_s, q_s \leq 2^{-2}$ and all 8×8 S-boxes are assumed to

[3] The terminology used here is the same as defined in [9, 20, 21] although it should be noted that it does not represent a true probability.

satisfy $p_s, q_s \leq 2^{-6}$. Many proposed ciphers such as Serpent, AES, Hierocrypt, and Camellia have S-boxes satisfying these requirements; others such as Anubis and Khazad have slightly higher p_s and q_s. The probability of a differential (respectively, linear) characteristic is upper bounded by $p_s^{n_a}$ (respectively, $q_s^{n_a}$), where n_a is the number of active S-boxes in the characteristic used in attacking the cipher.

Note that all S-boxes of a cipher can have the same mapping as is the case in AES. Alternatively, different mappings can be chosen for the S-boxes in one round as in DES or for S-boxes in different rounds as in Serpent.

2.4 MDS Mappings

A linear code over Galois field GF(2^n) is denoted as a (k, m, d)-code [5], where k is the symbol length of the encoded message, m is the symbol length of the original message, and d is the minimal symbol distance between any two encoded messages. An (k, m, d)-code is MDS if $d = k - m + 1$. In particular, a $(2m, m, m+1)$-code with generation matrix $\mathcal{G} = [\mathcal{I}|\mathcal{C}]$ where $\mathcal{C}$ is an $m \times m$ matrix and $\mathcal{I}$ is an identity matrix, determines an MDS mapping from the input $\mathcal{X}$ to the output $\mathcal{Y}$ through matrix multiplication over a Galois field as follows:

$$f_M : \mathcal{X} \mapsto \mathcal{Y} = \mathcal{C} \cdot \mathcal{X} \tag{1}$$

where

$$\mathcal{X} = \begin{pmatrix} X_0 \\ \vdots \\ X_{m-1} \end{pmatrix}, \quad \mathcal{Y} = \begin{pmatrix} Y_0 \\ \vdots \\ Y_{m-1} \end{pmatrix}, \quad \mathcal{C} = \begin{pmatrix} C_{0,0} & \ldots & C_{0,m-1} \\ \vdots & \ddots & \vdots \\ C_{m-1,0} & \ldots & C_{m-1,m-1} \end{pmatrix}.$$

Every entry in $\mathcal{X}, \mathcal{Y}$, and $\mathcal{C}$ is an element in GF(2^n).

When an invertible linear transformation $f : \mathcal{X} \mapsto \mathcal{Y}$ is used in a cipher, the avalanche effect which creates resistance to differential and linear attacks may be measured with its branch number B, which is defined [22] as:

$$B = \min_{\mathcal{X} \neq 0} \{H(\mathcal{X}) + H(\mathcal{Y})\}$$

where $H(\mathcal{X})$ and $H(\mathcal{Y})$ denote the number of nonzero elements in $\mathcal{X}$ and $\mathcal{Y}$, respectively. It is proved that an MDS mapping as defined in (1) has an optimal branch number B equal to $m + 1$.

2.5 Nested SPNs

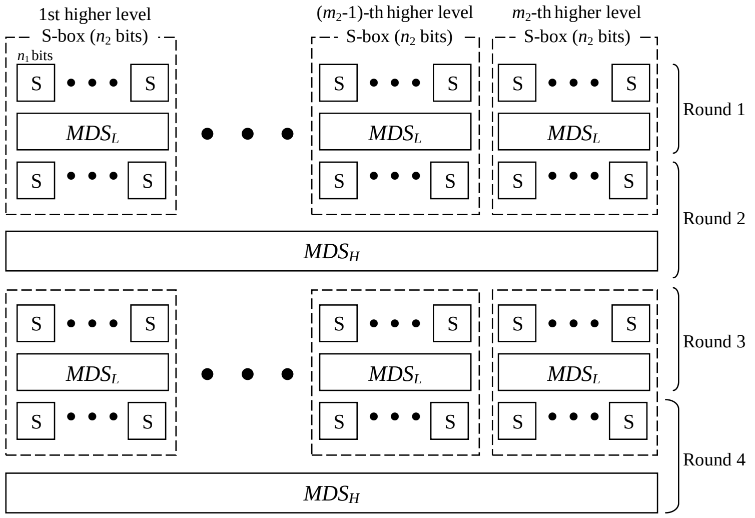
The concept of a nested SPN was first introduced in [17]. In a nested SPN, S-boxes may be viewed at different levels: each S-box at a higher level is actually a small SPN at the lower level. In this paper, we examine nested SPNs which have the following properties:

- The structure contains just two levels of SPNs. A higher level S-box consists of a lower level SPN; a lower level S-box is an actual 4×4 or 8×8 S-box.
- The linear transformation layers in both levels are based on MDS codes, denoted as MDS_H for the higher level and MDS_L for the lower level.
- The subkey mixture occurs directly before each layer of actual (i.e., lower-level) S-boxes. One additional subkey mixture is used to replace the linear transformation at the end of the cipher structure. The subkey bits are mixed with data bits by XOR operations.
- A "round" refers to the combination of the subkey mixture, lower-level S-box layer, and subsequent MDS_L or MDS_H linear transformation.

As Figure 3 shows, MDS_L is an MDS mapping from a $(2m_1, m_1, m_1+1)$-code over GF(2^{n_1}), while MDS_H is an MDS mapping from a $(2m_2, m_2, m_2+1)$-code over GF(2^{n_2}). The variables m_1, m_2, n_1, and n_2 represent parameter choices for a nested SPN.

In the most straightforward case, the output of each S-box forms one source symbol for the MDS mapping, and each encoded symbol forms the input of a subsequent S-box at the same level. So the size of an S-box is n_1 bits at the lower level and n_2 bits at the higher level. This leads to $n_2 = n_1m_1$. Thus, the block size of the SPN is $n_1m_1m_2$. For example, the 128-bit block cipher Hierocrypt (Type I) [17] is described as the iteration of such a 4-round structure where $n_1 = 8$, $n_2 = 32$, and $m_1 = m_2 = 4$.

At each level of a nested SPN, the branch number of the MDS layer determines the minimum number of active S-boxes in differential or linear cryptanalysis. For 4 rounds of a nested SPN, an active S-box at the higher level contains at least $m_1 + 1$ active S-boxes



MDS_L: based on a ($2m_1$, m_1, m_1+1)-code over GF(2^{n_1})
MDS_H: based on a ($2m_2$, m_2, m_2+1)-code over GF(2^{n_2})

Figure 3: Basic 2-level Nested SPN (4 Rounds)

at the lower level. Since there are at least $m_2 + 1$ active S-boxes at the higher level, the minimum number of active lower-level S-boxes is $(m_1 + 1)(m_2 + 1)$. Therefore, the security against differential and linear attacks is evaluated as the following:

Theorem 1 (deduced from [13, 14, 7, 17]): *With the assumption that all S-box approximations involved in linear and differential cryptanalysis are independent, for $4r$ rounds of a nested SPN the maximum differential characteristic probability (denoted by P_d) is upper bounded by $p_s^{r(m_1+1)(m_2+1)}$ and the maximum linear characteristic probability (denoted by P_l) is upper bounded by $q_s^{r(m_1+1)(m_2+1)}$.*

To attack a cipher using differential cryptanalysis, the number of chosen plaintexts is expected to be in the order of $1/P_d$. Similarly, for linear cryptanalysis, the number of known plaintexts is expected to be in the order of $1/P_l$. Hence, the upper bounds of P_d and P_l provided in Theorem 1 indicate the lower bounds of required workload for attacking $4r+1$ rounds of the cipher based on a $4r$ round characteristic.

The basic operations in MDS codes are multiplications and additions in finite fields. When n_2 is large, operations over GF(2^{n_2}) are inefficient and MDS_H can be costly in computation. An alternative method to obtain the same branch number is to concatenate several parallel MDS codes over a smaller finite field. The concatenated codes may be designed to facilitate a bitslice implementation.

Theorem 2 [17]: *An MDS mapping defined by a $(2m, m, m+1)$-code over the nl-bit symbol set can be constructed by concatenating l mappings defined by a $(2m, m, m+1)$-code over the n-bit symbol set, where l can be any positive integer.*

For the example illustrated by Figure 3, the MDS_H mapping is defined as a $(2m_2, m_2, m_2+1)$-code over GF(2^{n_2}) where $n_2 = m_1 n_1$. We refer to this as the basic MDS_H layer. To improve the efficiency of the MDS_H layer, instead of using the basic MDS_H, according to Theorem 2 we may consider equivalent MDS_H layers constructed as l parallel MDS codes with smaller field size of the form denoted as $l \times (2m_2, m_2, m_2 + 1)$ over GF(2^{n_2}) where n_2 is now the smaller field size such that $ln_2 = m_1 n_1$. Obviously, the basic MDS_H layer corresponds to $l = 1$. For improved efficiency, we can let $l > 1$ and $n_2 < m_1 n_1$. For example, $l = m_1$ and $n_2 = n_1$ is equivalent to the basic MDS_H layer but is composed of l parallel $(2m_2, m_2, m_2 + 1)$-codes over the smaller fields GF(2^{n_1}). Hence, the general relation $ln_2 = m_1 n_1$ can be used to determine different cases of MDS_H defined by the values of the symbol size, n_2, or the number of parallel MDS mappings, l. A similar approach can also be applied to the MDS_L layer. However, restrictions on values of n and m must be considered for designing a $(2m, m, m + 1)$-code over GF(2^n) such that $2m \leq 2^n + 1$ in order that it is possible to construct an MDS code [5].

The 128-bit ciphers Square, AES, and Anubis can be regarded as the iterations of 4-round nested SPNs where $n_1 = n_2 = 8$ and $m_1 = m_2 = 4$. The parameters of Hierocrypt (Type II) are selected as $n_1 = 8$, $n_2 = 4$, and $m_1 = m_2 = 4$.

2.6 One Class of Feistel Network

Figure 4 illustrates one particular class of round function F used for Feistel networks (as shown in Figure 2). Such a round function can be regarded as an SPN of one round with a size equal to half of the cipher block size. The round function includes one layer of

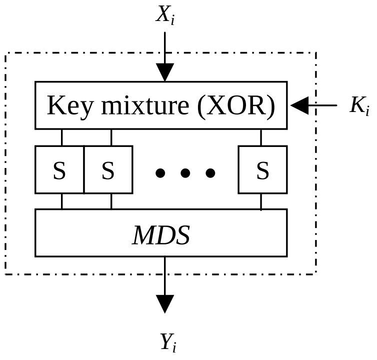


Figure 4: One Class of Round Function

key mixture with K_i (i.e., bitwise XOR of X_i and K_i), one layer of invertible[4] S-boxes for substitution, and an MDS mapping layer as a linear transformation. If the MDS mapping layer is constructed through concatenation of several small MDS mappings, it is necessary to include a permutation of MDS symbols in the linear transformation in order to ensure the avalanche effect.

In a Feistel network whose round function has an invertible linear transformation appended to a layer of S-boxes, it is proved in [21] that the number of active S-boxes in any differential or linear characteristic of $4r$ rounds is lower bounded by $r \times B + \lfloor r/2 \rfloor$, where B is the branch number of the linear transformation and r is an integer. Therefore, we get:

Theorem 3 (deduced from [21]): *For $4r$ rounds of a Feistel cipher with the round function of Figure 4, the maximum differential characteristic probability P_d and maximum linear characteristic probability P_l are upper bounded by $p_s^{r \times B + \lfloor r/2 \rfloor}$ and $q_s^{r \times B + \lfloor r/2 \rfloor}$, respectively.*

3 Methodology, Applicability, and Limitations

It is hard to generally compare software performance among different block ciphers. The main difficulties are: (1) each cipher has its own security margin, (2) each implementation method represents a tradeoff between memory and speed, (3) the number of clock cycles required by one operation is determined by the platform, and (4) one specific instruction set

[4] Invertible S-boxes are used so that a bijective round function can be constructed, which achieves the given upper bounds of maximal differential and linear probabilities faster in rounds than a general round function [23].

may facilitate some operation combinations (e.g., DSP processors can do multiplication and accumulation using one single instruction). Considering the above difficulties, in order to do a general comparison, we need to select a general and efficient method to implement all the cipher cases in this paper. Moreover, a meaningful study of the performance of ciphers should make comparisons between ciphers in consideration of a consistent security level.

The methodology in this paper is summarized by the following points:

- *Table lookup method is selected for implementation.* The number of table lookups is used to evaluate efficiency and the table size is used to evaluate memory requirement. It is assumed that the available registers in the processor are more than enough to store all the table base addresses. (In practice, it is possible to minimize the number of tables by using circulant matrices for MDS codes as is done in AES [7].)

- *Differential and linear cryptanalysis are used for security evaluation.* More specifically, maximum differential and linear characteristic probabilities are deduced for each cipher case. Side-channel attacks are not considered because, as discussed in [24], table lookup implementation is not vulnerable to timing attacks and facilitates resistance against power attacks by software balancing of the lookup address.

- *Security and efficiency are considered together for performance comparison.* As defined in Section 4.2, a performance metric is used to show the security contribution provided by each efficiency unit in a cipher.

The applicability of this methodology includes:

- *Ciphers using S-boxes and linear transformations, such as MDS mappings.* A large set of recently proposed ciphers are covered, including SHARK, Square, AES, Anubis, Khazad, Hierocrypt, Serpent, and Camellia. Actually, although the metric in this paper is broadly applicable to any cipher that can have its security characterized on a per round basis, the analysis is most obviously applicable to SPNs and Feistel ciphers. It is particularly useful to consider these 2 classes to determine the various tradeoffs in selecting cipher parameters such as S-box size and MDS field size.

- *Typical processors such as Pentium series and Alpha machines.* Since parallelism in processors is not considered, the analysis cannot be directly applied to processors with parallel units such as the VLIW CPUs discussed in [25]. Because of the memory requirements of the table lookup approach, the method has limited applicability to a memory-restricted environment such as a smart card.

- *No limitation on MDS codes.* The elements in the generation matrix can be freely selected to meet the MDS property. Some MDS codes (e.g., MixColumn in AES [7]) are optimized for *xtime* realization, while the majority are not (e.g., Hierocrypt's MDS mappings [17]).

- *No limitation on key schedule.* However, it is assumed that the key schedule is done beforehand and all subkeys are stored in memory. Some ciphers can expand subkeys on the fly to save memory but require more processing time during encryption.

- *No limitation on the number of rounds.* Many ciphers, such as AES and Camellia, require different numbers of rounds for encryption when key lengths are changed. Our metric considers the security contribution provided by each lookup, which is roughly independent on the number of rounds as will be demonstrated in Section 4.2. Thus, these ciphers can be analyzed effectively and uniformly whatever key length and number of rounds are selected.

The limitations of this methodology are:

- *Although it is general and fast, the table lookup method might not always be the fastest for specific processors.* Some CPUs can combine data processing instructions with shifts and rotates without a penalty, which may lead to a fast implementation using the *xtime* operation [7] (i.e., one-bit left shifting followed by addition with the irreducible polynomial) when the processor word is small. For example, in Rijndael's AES proposal [7], the *xtime* method is recommended for 8-bit processors while the table lookup method is recommended for 32-bit processors.

- *The discrepancy of efficiency caused by different table sizes is ignored.* A large table causes more cache misses during addressing. On the other hand, a small table needs additional bit manipulations for index preparation and output arrangement [26]. Moreover, it is also difficult to model cache misses across platforms with different cache sizes and structures. The experimental results listed in Section 5 show that the effects of this limitation are tolerable on the tested platforms.

4 Comparison of Software Performance

In this section, we present a detailed discussion of the performance comparison based on table lookup implementations.

4.1 Table Lookup Implementations

The table lookup approach incorporates the S-boxes and the linear transformation into a table that is then accessed to perform both operations. This approach has been used for fast implementations of DES [27], AES [7], and Camellia [28]. Using this approach, the two cipher structures discussed in the former section can be implemented efficiently in software through table lookups, logic operations (e.g., XORs), and rotations. This paper analyzes the efficiency of such fast implementations so that the memory and computational cost for a cipher case can be estimated. Independent of the targeted machine, the space complexity is evaluated as memory used for tables and the time complexity is evaluated by the number of table lookups.

The table lookup approach is chosen for analysis because it is normally faster and more general than other implementation approaches. A table lookup operation involves the reading of data from memory and also encompasses other operations necessary for indexing such as rotation and masking. Although the number of clock cycles to implement different operations is machine dependent, using the number of lookups and the size of the tables is suitable for determining a rough estimate of the time and space complexity of an efficient software implementation.

In software, regardless of the implementation approach, the S-box layer is typically done by table lookups. An MDS mapping based on a $(2m, m, m+1)$-code conceptually performs a matrix multiplication over a Galois field, which requires m^2 modular multiplications and $m(m-1)$ XORs on words that are size of Galois field elements. To bypass costly multiplications, we enlarge the S-box table such that the MDS mapping work is included in the table lookups. This is the essence of the table lookup implementation. According to the size of the S-boxes and the type of MDS mappings, any cipher case may select appropriate methods as follows to generate lookup tables.

4.1.1 Cases with 8×8 S-boxes

The dataflow of a round in these cases involves the keyed input entering 8×8 S-boxes followed by l mappings based on a $(2m, m, m+1)$-code over GF(2^8). (The case of two concatenated mappings over GF(2^4) will be discussed later.) To represent the operations mathematically, we denote the input, output, subkey, and MDS generation matrix as $\{A_i\}$, $\{E_i\}$, $\{K_i\}$, and $\{C_{i,j}\}$, respectively, each containing 8-bit elements. Thus, the key mixture, S-box layer, and MDS mapping can be expressed together as:

$$\begin{pmatrix} E_0 \\ E_1 \\ \vdots \\ E_{m-1} \end{pmatrix} = \begin{pmatrix} C_{0,0} & C_{0,1} & \dots & C_{0,m-1} \\ C_{1,0} & C_{1,1} & \dots & C_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m-1,0} & C_{m-1,1} & \dots & C_{m-1,m-1} \end{pmatrix} \times \begin{pmatrix} S(A_0 \oplus K_0) \\ S(A_1 \oplus K_1) \\ \vdots \\ S(A_{m-1} \oplus K_{m-1}) \end{pmatrix}. \tag{2}$$

Denoting the keyed input as $B_i = A_i \oplus K_i$, (2) is equivalent to:

$$\begin{pmatrix} E_0 \\ E_1 \\ \vdots \\ E_{m-1} \end{pmatrix} = \begin{pmatrix} C_{0,0} \\ C_{1,0} \\ \vdots \\ C_{m-1,0} \end{pmatrix} \times S(B_0) \oplus \cdots \oplus \begin{pmatrix} C_{0,m-1} \\ C_{1,m-1} \\ \vdots \\ C_{m-1,m-1} \end{pmatrix} \times S(B_{m-1}). \tag{3}$$

Hence, we may generate m tables as the following:

$$T_j[\cdot] = \begin{pmatrix} C_{0,j} \times S(\cdot) \\ C_{1,j} \times S(\cdot) \\ \vdots \\ C_{m-1,j} \times S(\cdot) \end{pmatrix} \tag{4}$$

where $0 \leq j \leq m-1$. The output of several S-boxes followed by the MDS mapping may then be generated using:

$$\begin{pmatrix} E_0 \\ E_1 \\ \vdots \\ E_{m-1} \end{pmatrix} = T_0[B_0] \oplus \cdots \oplus T_{m-1}[B_{m-1}]. \tag{5}$$

Each fetch from the table $T_j[\cdot]$ accepts an 8-bit input as the index and produces an $8m$-bit output from the indexed entry. It takes $256m^2$ bytes of memory to store these m tables. Given a processor with a word size of w bits, implementation of (5) needs $m\lceil 8m/w \rceil$ lookups and $(m-1)\lceil 8m/w \rceil$ XORs. In cases where the word size w is larger than the size of a table index, the preparation of a table lookup input will generally need a rotation and masking (bitwise AND) within a word.

When the size of an MDS field is smaller than the size of the S-boxes, we can consider an MDS mapping layer of more than one MDS mapping (i.e., the adjacent S-box output bits may pass through different mappings). The table $T_j[\cdot]$ is then established through concatenation. Each entry of $T_j[\cdot]$ consists of concatenated results from different MDS mappings. The result from one mapping corresponds to a specific subset of the table lookup output. For example, considering 8×8 S-boxes followed by $2 \times (2m, m, m+1)$ over GF(2^4), each coefficient C_{ij} in (2) can be regarded as concatenation of two 4-bit coefficients C'_{ij} and C''_{ij} from two MDS mappings, so that $C_{ij} = C'_{ij} || \, C''_{ij}$, where " $||$ " denotes concatenation. Then we generate m tables as:

$$T_j[\cdot] = \begin{pmatrix} C'_{0,j} \times S'(\cdot) \; || \; C''_{0,j} \times S''(\cdot) \\ C'_{1,j} \times S'(\cdot) \; || \; C''_{1,j} \times S''(\cdot) \\ \vdots \\ C'_{m-1,j} \times S'(\cdot) \; || \; C''_{m-1,j} \times S''(\cdot) \end{pmatrix} \tag{6}$$

where $0 \leq j \leq m-1$, $S'(\cdot)$ and $S''(\cdot)$ represent 4 output bits of an S-box, and $S(\cdot) = S'(\cdot) || \, S''(\cdot)$. When these concatenated tables $\{T_j[\cdot]\}$ are used in (5), the size of tables and the number of lookups and XORs are the same as for the tables required in (4).

4.1.2 Cases with 4×4 S-boxes

In constrained environments such as smart cards, cipher cases using 4×4 S-boxes cost much less memory for table storage than those using 8×8 S-boxes. We can use the same method described by (3) and (5) to generate a set of small tables. Since the variables B_i and E_i in (3) and (5) are now 4 bits, each fetch from the table $T_j[\cdot]$ accepts a 4-bit input as the index and produces a $4m$-bit output from the indexed entry. It takes $8m^2$ bytes of memory to store these m tables since each table requires $16{\cdot}m{\cdot}4/8 = 8m$ bytes. Such an implementation needs $m\lceil 4m/w \rceil$ lookups and $(m-1)\lceil 4m/w \rceil$ XORs.

When memory is not constrained, a modified method can be used to reduce the number of table lookups by a factor of 2. To implement a cipher case with 4×4 S-boxes, each table $T_j[\cdot]$ in (5) has an index of 4 bits. We can combine two tables into one, represented by T'_j, whose index is 8 bits. As a result, (5) is transformed to:

$$\begin{pmatrix} E_0 \\ E_1 \\ \vdots \\ E_{m-1} \end{pmatrix} = T'_0[B_0 || \; B_1] \oplus \cdots \oplus T'_{\frac{m}{2}-1}[B_{m-2} || \; B_{m-1}]. \tag{7}$$

where B_i and E_i are representing 4-bit values. For each 8-bit input $X || \; Y$ composed of 2 concatenated 4-bit values, X and Y, the table performs:

$$T'_j[X || \; Y] = T_{2j}[X] \oplus T_{2j+1}[Y]$$

where $0 \leq j \leq m/2 - 1$. It takes $64m^2$ bytes of memory to store these $m/2$ tables. The implementation of (7) needs $(m/2)\lceil 4m/w \rceil$ lookups and $(m/2-1)\lceil 4m/w \rceil$ XORs.

The method expressed by (7) should also be chosen for the cases where the symbol length of an MDS mapping is larger than the S-box size. For example, the inputs of two adjacent 4×4 S-boxes followed by an MDS mapping over GF(2^8) have to be combined as an 8-bit index to a table of 256 elements.

4.2 Comparison Based on Table Lookup Implementations

4.2.1 Time Performance Metric

In software, the memory used for table storage is independent of the number of rounds. To compare the time used for a given cipher to achieve a certain amount of security, we define the time performance measure $\eta_{(w)}$ with respect to differential and linear attacks, where w

is the processor word size:

$$\eta_{(w)} = \frac{\log_2 1/P}{(\#\ \mathit{of\ rounds}) \times (\#\ \mathit{of\ table\ lookups\ per\ round})}\ . \tag{8}$$

The numerator of (8) is a function of cipher structure while the denominator is a function of both cipher structure and word size w. The value of $\log_2 1/P$ indicates the security of the cipher for a specified number of rounds, where we use a heuristic approach to determine resistance to differential and linear cryptanalysis. For differential cryptanalysis, the number of chosen plaintexts to attack a cipher is expected to be in the order of $1/P$, where P is the maximum differential characteristic probability P_d determined by Theorems 1 and 3; the number of known plaintexts required by linear cryptanalysis is expected to be in the order of $1/P$, where P is the maximum linear characteristic probability P_l of the cipher. For the nested SPNs and Feistel networks discussed in Section 2, $\log_2 1/P$ is a linear function of the number of rounds for both differential and linear cryptanalysis. Therefore, the value of $\eta_{(w)}$ indicates how much security is expected to be obtained within a unit running time (i.e., time for one table lookup), regardless of the number of rounds in a cipher.

Note that one table lookup has associated with it the setup of an index (e.g., one rotation and one masking operation) and a post-lookup XOR. Among these operations, the table lookup would normally require the most clock cycles in most processors. Hence, we use the table lookups as a barometer of the number of operations required to implement the cipher.

4.2.2 Comparison of Nested SPNs

A set of nested SPNs can be generated with appropriate configurations of parameterized MDS_L, MDS_H, and S-boxes. As Theorem 2 illustrates, the MDS mapping defined over a large Galois field can be simplified using several mappings in a smaller Galois field. Table 1 lists the cases of nested SPNs in 12 categories (labelled as N1 to N12) defined by the S-boxes and MDS_L. Thus, the cases within a category only differ in the simplification of MDS_H. Each case can be regarded as $4r$ rounds of a 128-bit cipher where r is an integer, except that no particular key schedule has been defined. Due to the difficulty of finding optimized MDS

Table 1: 128-bit Nested SPNs of $4r$ Rounds

Case	S-box	MDS_L :	MDS_H :	P_d, P_l
	size	$l_1{\times}(2m_1, m_1, m_1{+}1)$ over GF(2^{n_1})	$l_2{\times}(2m_2, m_2, m_2{+}1)$ over GF(2^{n_2})	
N1-a	8×8	8×$(4, 2, 3)$ over GF(2^8)	2×$(16, 8, 9)$ over GF(2^8)	2^{-162r}
N1-b			4×$(16, 8, 9)$ over GF(2^4)	
N2-a	8×8	16×$(4, 2, 3)$ over GF(2^4)	2×$(16, 8, 9)$ over GF(2^8)	2^{-162r}
N2-b			4×$(16, 8, 9)$ over GF(2^4)	
N3-a	8×8	32×$(4, 2, 3)$ over GF(2^2)	2×$(16, 8, 9)$ over GF(2^8)	2^{-162r}
N3-b			4×$(16, 8, 9)$ over GF(2^4)	
N4-a	8×8	4×$(8, 4, 5)$ over GF(2^8)	4×$(8, 4, 5)$ over GF(2^8)	2^{-150r}
N4-b			8×$(8, 4, 5)$ over GF(2^4)	
N5-a	8×8	8×$(8, 4, 5)$ over GF(2^4)	4×$(8, 4, 5)$ over GF(2^8)	2^{-150r}
N5-b			8×$(8, 4, 5)$ over GF(2^4)	
N6-a	8×8	2×$(16, 8, 9)$ over GF(2^8)	8×$(4, 2, 3)$ over GF(2^8)	2^{-162r}
N6-b			16×$(4, 2, 3)$ over GF(2^4)	
N7-a	8×8	4×$(16, 8, 9)$ over GF(2^4)	8×$(4, 2, 3)$ over GF(2^8)	2^{-162r}
N7-b			16×$(4, 2, 3)$ over GF(2^4)	
N7-c			32×$(4, 2, 3)$ over GF(2^2)	
N8	8×8	1×$(32, 16, 17)$ over GF(2^8)	same as MDS_L	2^{-204r}
N9	4×4	16×$(4, 2, 3)$ over GF(2^4)	1×$(32, 16, 17)$ over GF(2^8)	2^{-102r}
N10	4×4	32×$(4, 2, 3)$ over GF(2^2)	1×$(32, 16, 17)$ over GF(2^8)	2^{-102r}
N11-a	4×4	8×$(8, 4, 5)$ over GF(2^4)	2×$(16, 8, 9)$ over GF(2^8)	2^{-90r}
N11-b			4×$(16, 8, 9)$ over GF(2^4)	
N12-a	4×4	4×$(16, 8, 9)$ over GF(2^4)	4×$(8, 4, 5)$ over GF(2^8)	2^{-90r}
N12-b			8×$(8, 4, 5)$ over GF(2^4)	

mappings, the cases with a Galois field larger than GF(2^8) are not considered. The values of P_d and P_l represent the differential and linear characteristic probabilities for $4r$ rounds evaluated by Theorem 1 and used as P in (8) to determine $\eta_{(w)}$.

In relation to real ciphers, case N4-a includes Square, AES, and Anubis. Type II of Hierocrypt belongs to case N4-b with a simplified MDS_H over GF(2^4). Similar to SHARK and Khazad, case N8 is a one-level SPN. However, SHARK and Khazad are 64-bit ciphers because their MDS mappings are based on a (16, 8, 9)-code over GF(2^8).

Table 2 lists the table size (i.e., the sum of sizes of tables required for MDS_L and MDS_H rounds) and the number of table lookups for 4 rounds for each cipher case. The table sizes shown represent a minimum requirement and can only be achieved when an S-box does not differ from the corresponding S-box in another MDS mapping in the same layer (although

Table 2: Software Performance of 128-bit Nested SPNs

Case	Table size (KBytes)	# of table lookups per 4 rounds			$\eta_{(8)}$	$\eta_{(32)}$	$\eta_{(64)}$
		8-bit	32-bit	64-bit			
N1-a,b	17	320	96	64	0.51	1.69	2.53
N2-a,b	17	320	96	64	0.51	1.69	2.53
N3-a,b	17	320	96	64	0.51	1.69	2.53
N4-a,b	8[†]	256	64	64	0.59	2.34	2.34
N5-a,b	8[†]	256	64	64	0.59	2.34	2.34
N6-a,b	17	320	96	64	0.51	1.69	2.53
N7-a,b,c	17	320	96	64	0.51	1.69	2.53
N8	64	1024	256	128	0.20	0.80	1.59
N9	64.03125	576	192	128	0.18	0.53	0.80
N10	64.03125	576	192	128	0.18	0.53	0.80
N11-a	16.125	384	128	96	0.23	0.70	0.94
N11-b	0.625	384	128	128	0.23	0.70	0.70
N12-a	4.5	384	96	96	0.23	0.94	0.94
N12-b	0.625	384	128	128	0.23	0.70	0.70

[†] : By use of the same mapping in MDS_L and MDS_H, half of the table size can be saved.

S-boxes may be different within the domain of one MDS mapping). As a result, only one table as in (5) is required for each of the MDS_L and MDS_H layers.

For each case using 4×4 S-boxes, the tables with 4-bit indices are created as shown in (5) or (6) when the MDS mapping is chosen over GF(2^4) or GF(2^2), respectively. However, as explained in Section 4.1.2, the performance can be improved by using 8-bit indices in the lookup as in (7) at the expense of more memory. For example, case N11-b can be implemented using 8-bit indices thereby doubling the efficiency but requiring 8 times the memory to store the lookup tables for both the MDS_L and MDS_H rounds. When GF(2^8) is chosen for the MDS mapping, the length of table indices has to be 8 bits. The number of table lookups is used in the calculation of the denominator in (8).

The table also includes the performance $\eta_{(w)}$ for each case. The implementation performance on three types of processors (i.e., $w = 8, 32, 64$) are considered. The implementation on an 8-bit processor is suitable for smart cards, where the memory size is constrained. The implementations on 32-bit and 64-bit processors are suitable for applications on general purpose computers and workstations. The values of $\eta_{(w)}$ are also presented in Figure 5. By comparing these measures, it is possible to distinguish the cases which are more efficient in

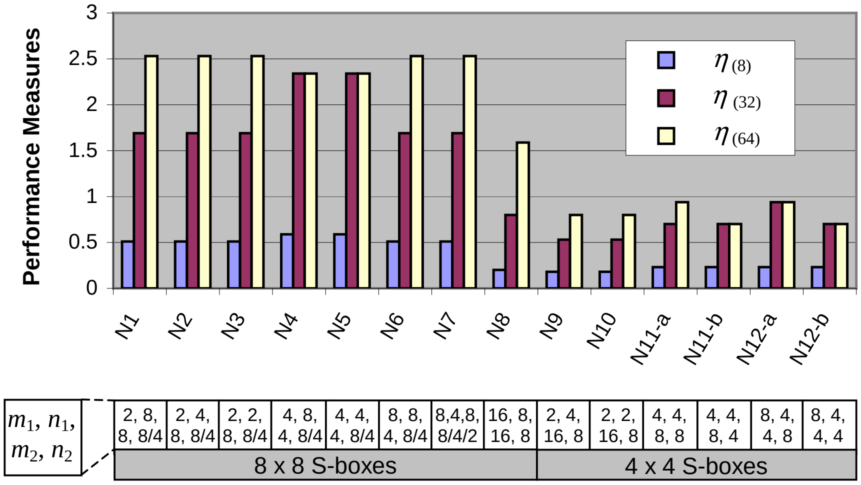


Figure 5: Software Performance Comparison of Nested SPNs

software and the following general conclusions can be made:

- The implementation performance is improved when the word size of the processor increases, although in some cases there is no difference in the performance on a 32-bit or 64-bit processor.
- The cases with larger S-boxes (N1, ···, N8) have better performance but cost more memory to store the lookup tables.
- The cases with the same S-box size (N1, ···, N8 and N9, ···, N12) share similar performance although their memory requirements can vary significantly (as shown in Table 2).
- Cases N4 and N5 have the best, or close to the best, performance for all word sizes.

4.2.3 Comparison of Feistel Networks

The Feistel network discussed in this section is limited to the class described in Section 2.4, which has an SPN-like round function. To construct a typical 128-bit cipher, such a Feistel network has a 64-bit round function which contains sixteen 4×4 or eight 8×8 parallel S-boxes followed by an MDS mapping layer. As listed in Table 3, six categories (labelled as F1 to F6) of these 128-bit Feistel networks can be generated. To ensure a good avalanche effect, an appropriate fixed permutation of MDS symbols after the MDS mapping is expected, which may cost a small amount of additional processing time. The cases of the same category in Table 3 only differ in the simplification of the MDS mapping. The performance comparison details are given in Table 4, where table lookups use 4-bit indices when 4×4 S-boxes are used in a cipher case. A summary of the performance measure $\eta_{(w)}$ is also illustrated in Figure 6. The following conclusions can be drawn:

- An MDS mapping that has a large branch number (i.e., $m+1$) results in good performance for implementations on computers supporting a large word size (e.g., comparing $\eta_{(8)}$ and $\eta_{(64)}$ in cases F3-a and F3-b).
- Although they require more memory, the cases with 8×8 S-boxes demonstrate higher performance.
- For the cases with 4×4 S-boxes, we can tradeoff memory and time requirements by choosing the element size of the MDS mapping. Using small Galois fields for the MDS codes, cases F4-b, F4-c, and F5-b can be used for some memory-constrained applications. However, their performance is not as high as the counterparts using large Galois fields (e.g., F4-a and F5-a) for a word size larger than 8.
- Compared with nested SPN networks, the Feistel networks discussed here need less memory but result in a lower performance.

Camellia uses 8×8 S-boxes and a linear transformation which is not MDS-based with branch number 5. (An MDS-based linear transformation would have a branch number of 9.) Hence, a simplified Camellia structure (without FL/FL^{-1} functions) produces a security level equivalent to F2-a,b in Table 3. A fast Camellia implementation using table lookups was introduced in [28], which incorporates the linear transformation and S-boxes into several tables with 8-bit indices and 64-bit entries. In this method, a simplified Camellia has the equivalent number of table lookups to 18-round F3-a,b. As a result, Camellia uses tables as large as F3-a,b, while its performance is lower than both F2-a,b and F3-a,b on 32-bit processors and equal to F2-a,b but lower than F3-a,b on 64-bit processors.

Table 3: 128-bit Feistel Networks of $4r$ Rounds

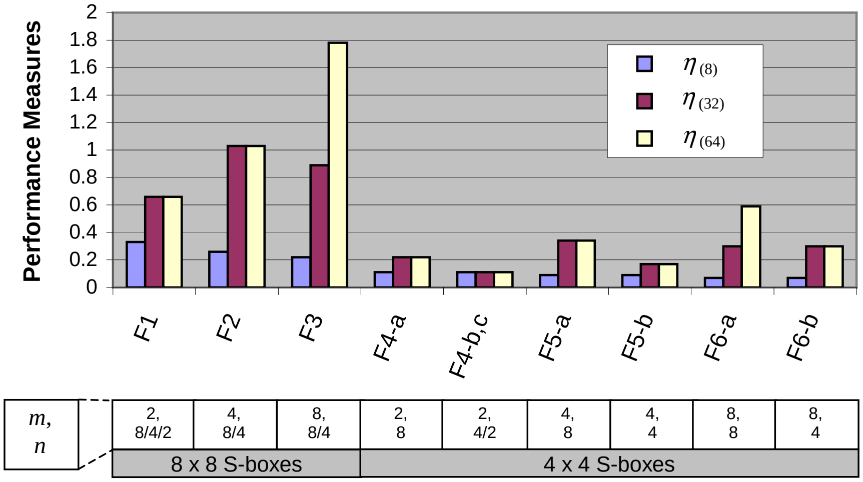
Case	S-box size	MDS	P_d, P_l
		$l \times (2m, m, m{+}1)$ over GF(2^n)	
F1-a	8×8	$4 \times (4, 2, 3)$ over GF(2^8)	$2^{-6(3r+\lfloor \frac{r}{2} \rfloor)}$
F1-b		$8 \times (4, 2, 3)$ over GF(2^4)	
F1-c		$8 \times (4, 2, 3)$ over GF(2^2)	
F2-a	8×8	$2 \times (8, 4, 5)$ over GF(2^8)	$2^{-6(5r+\lfloor \frac{r}{2} \rfloor)}$
F2-b		$4 \times (8, 4, 5)$ over GF(2^4)	
F3-a	8×8	$1 \times (16, 8, 9)$ over GF(2^8)	$2^{-6(9r+\lfloor \frac{r}{2} \rfloor)}$
F3-b		$2 \times (16, 8, 9)$ over GF(2^4)	
F4-a	4×4	$4 \times (4, 2, 3)$ over GF(2^8)	$2^{-2(3r+\lfloor \frac{r}{2} \rfloor)}$
F4-b		$8 \times (4, 2, 3)$ over GF(2^4)	
F4-c		$16 \times (4, 2, 3)$ over GF(2^2)	
F5-a	4×4	$2 \times (8, 4, 5)$ over GF(2^8)	$2^{-2(5r+\lfloor \frac{r}{2} \rfloor)}$
F5-b		$4 \times (8, 4, 5)$ over GF(2^4)	
F6-a	4×4	$1 \times (16, 8, 9)$ over GF(2^8)	$2^{-2(9r+\lfloor \frac{r}{2} \rfloor)}$
F6-b		$2 \times (16, 8, 9)$ over GF(2^4)	

Table 4: Software Performance of 128-bit Feistel Networks

Case	Table size (KBytes)	# of table lookups per round			$\eta_{(8)}$	$\eta_{(32)}$	$\eta_{(64)}$
		8-bit	32-bit	64-bit			
F1-a,b,c	1	16	8	8	0.33	0.66	0.66
F2-a,b	4	32	8	8	0.26	1.03	1.03
F3-a,b	16	64	16	8	0.22	0.89	1.78
F4-a	1	16	8	8	0.11	0.22	0.22
F4-b,c	0.03125	16	16	16	0.11	0.11	0.11
F5-a	4	32	8	8	0.09	0.34	0.34
F5-b	0.125	32	16	16	0.09	0.17	0.17
F6-a	16	64	16	8	0.07	0.30	0.59
F6-b	0.5	64	16	16	0.07	0.30	0.30

5 Experimental Results

The performance comparison above is based on the assumption that the number of lookups is a good time measure for table lookup implementations. We implemented typical SPN cipher cases from Table 3 in “C” and determined the throughput for each implementation on a 933MHz Intel Pentium III computer and a 2GHz Pentium 4 computer, respectively. The codes are compiled using the MS Visual C++ 6.0 compiler and optimized for speed during compilation. It is expected that the throughput will vary inversely to the number of



m, *n*	2, 8/4/2	4, 8/4	8, 8/4	2, 8	2, 4/2	4, 8	4, 4	8, 8	8, 4
	8 x 8 S-boxes			4 x 4 S-boxes					

Figure 6: Software Performance Comparison of Feistel Networks

lookups, considering throughput to be defined as:

$$throughput = \frac{block\ length}{processing\ time\ for\ each\ block}.$$

To mitigate the effect of the multitasking operating system, we only calculated the clocks used for the test process, which repeated encrypting test vectors for millions of times. During the experiments, all unnecessary tasks were removed as much as possible.

When the table index is 8 bits, the byte permutation after the lookup operations (e.g., the concatenation of parallel MDS mappings discussed in Theorem 2) can be easily done by reordering the inputs of table lookups next round. When the table index is 4 bits as in N11-b, bit manipulation within bytes costs more processing time. This cost partly decreases the advantage of using small tables (624 bytes), which can be easily cached during the program runs. Compared with N11-b, N8 requires a much larger table size (64 KB), which causes more cache misses and results in a throughput lower than expected. Even so, as shown in Figure 7, the expected trend of throughputs evaluated by the number of lookups in total can still be clearly observed in the implementations. The bottom two rows of Table 5 list the results of our 10-round AES implementation and reference code in ANSI C2.0 [29], respectively. The AES implementation uses 4 tables as defined in (4).

Table 6 lists the experimental throughput results for AES and Camellia on two 64-bit

Table 5: Experimental Results of 32-bit Implementations of Nested SPNs

Case	# of rounds	Throughput (Mbps)		# of lookups (4 rounds)	Comments
		Pentium III	Pentium 4		
N1-a	32	45.65	77.42	96	similar to N2, N3, N6, N7
N4-a	32	60.86	105.2	64	similar to 32-round AES
N8	32	17.02	22.83	256	with uniform round
N11-b	32	36.21	74.49	128	similar to N12-b
N4-a	10	155.9	262.5	64	10-round AES (our code)
N4-a	10	120.6	182.6	64	reference AES code

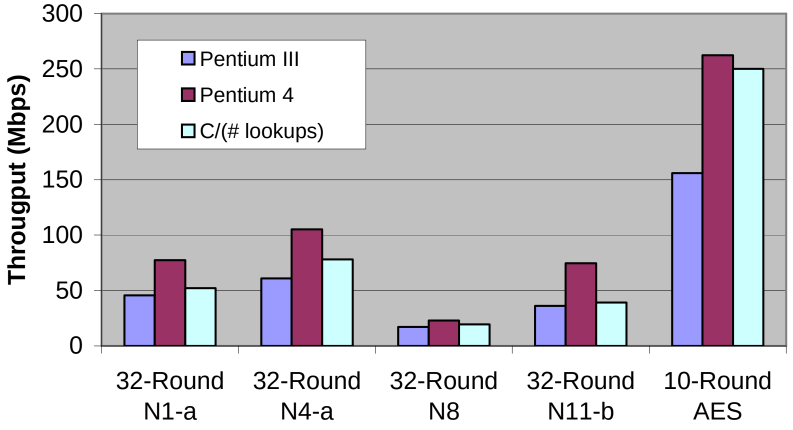


Figure 7: Comparison of Throughputs
(C: a constant, which is set to 40,000 here)

processors. The GNU C++ compiler is used on a 600 MHz Alpha processor (COMPAQ AlphaServer DS10, Tru64 Unix V5.1) while the SUN Workshop C++ Compiler is used on a 500 MHz UltraSPARC IIe processor (SUN Blade 100 workstation, SUN OS V5.8). Both computers have a similar 2-level cache architecture [30], where L1 cache is built in the processor and divided equally for instruction and data. The AlphaServer has 128 KB L1 cache and 2 MB L2 cache; the UltraSPARC IIe has 32 KB L1 cache and 256 KB L2 cache.

All implementations are optimized during compilation using the "-O2" option[5]. The 32-bit implementations of AES and Camellia are tested on the two machines by using 32-bit data type arrays to store lookup tables and 32-bit operations for XORs.

[5] "-O2" turns on all optimization flags specified by "-O". Only loop unrolling and function inlining are not included.

Table 6: Experimental Results of Two Real Ciphers

Cipher	Throughput (Mbps)		# of lookups
	Alpha processor (128KB L1 Cache)	UltraSPARC IIe (32KB L1 Cache)	
AES (10 rounds, 32-bit)	55.3 (32.7†)	69.9 (39.4†)	160
Simplified Camellia (18 rounds, 32-bit)	72.5 (20.1†)	29.0 (14.8†)	288
Simplified Camellia (18 rounds, 64-bit)	87.3 (35.2†)	58.2 (31.4†)	144

† : throughput without optimization during compilation

On the Alpha processor, the throughput and the inverse of estimated number of lookups follow the same trend before optimization. After optimization, the 32-bit Camellia implementation is largely improved and close to its 64-bit implementation. With the smallest number of lookups, the 64-bit simplified Camellia is still fastest after optimization. On the UltraSPARC IIe processor, the 32-bit and 64-bit Camellia implementations follow the expected trend nicely. The L1 cache in the UltraSPARC IIe processor has a limited size of which only half (16 KB) is used to cache data for all running processes. Therefore, the small table size of AES (4 KB) results in a throughput higher than expected in comparison to Camellia, which, with the larger table size of 16 KB, can be expected to experience more cache misses.

It should be noted that the software comparison of Table 6 is purely based on software throughput. When security is considered simultaneously as shown in Figures 5 and 6, AES has a higher performance (as a typical SPN corresponding to N4-a) than Camellia. This reflects that fact that the Camellia cipher has a lower security margin than the AES cipher.

6 Alternative Implementations

Besides the table lookup approach, a block cipher can be implemented in other ways based on its structure. We briefly discuss these alternatives without a full exposé of their characterization because they apply to specialized circumstances rather than having general application.

6.1 Bitslice Implementations

For some cipher cases, a bitslice software implementation derived from the gate level circuit may be more efficient in parallelized applications [31]. A bitslice design is suitable for the cases whose synthesized circuits are compact. A w-bit processor can be regarded as w bit-processors in parallel. The gate level network circuit is described with instructions in software. Each bit in hardware corresponds to a word in software and each word is the concatenation of bits belonging to w separate blocks. Given enough registers in a processor, the memory requirement is negligible since no table lookups are necessary. Typically, the bitslice technique can be applied in three ways:

(a) *Parallel blocks*: This is the classic bitslice implementation. A total of w plaintext blocks are reorganized so that the bits at the same bit positions of different original blocks are now collected in one register. The number of registers required to store these blocks is equal to the block size of the cipher. Then, all registers are used as signals to a gate network deduced from a hardware implementation. The output signals, w bits each, are converted to their original format as w ciphertext blocks. Whether a cipher case discussed in this paper is suitable for bitslice implementation can be determined from its space performance value η_s in hardware, which was investigated in [32]. When η_s is high, a compact gate network can be used. The gate count of the circuit determines the number of instructions used in the bitslice software. Thus, a high η_s indicates a small number of clock cycles in software. For example, because of the algebraic structure of its S-box, AES can be implemented as a very compact gate network, which enables an efficient bitslice realization [33, 34]. Note that, except for Serpent, representation transformation is required at the beginning and end of a bitslice realization.

(b) *Bitslice cipher*: Serpent is an example of an internal bitslice implementation. In Serpent, each 128-bit block is expressed as four 32-bit words after a bit permutation. S-boxes in each round can be regarded as 32 sets of parallel and identical 4×4 gate networks. A word is the collection of 32 bit signals, each corresponding to its own set of gate networks at the same locations. The other cipher operations can also be easily expressed by words. At the end of encryption, the bits of the four output words are permuted to form a 128-bit block.

(c) *Within special linear operations*: It has been shown that several parallel MDS mappings can be concatenated into one big mapping. When the number of parallel MDS mappings in each round, denoted by w', is at least 8, there is a more subtle bitslice method within the round structure, as used in Hierocrypt's MDS mapping [17]. The linear expression of each output bit is extended to the expression of words, whose size is w' bits. The input and output bit variables are replaced with word variables, each including w' adjacent bits. Such a method works for any concatenated linear transformation with a convenient number of parallel sets. This parallel structure within a specific operation avoids the overhead caused by the block representation transformation between the standard form and bitslice parallel form as required in (a).

It should be noted that, when a processor supports a large word size (32-bit or 64-bit), bitslicing is not as popular and effective as the table lookup method since not all 8×8 S-boxes have AES-like algebraic structure for subfield optimization. Moreover, such an algebraic structure has caused some concern about potential security threats as XSL attacks show [35].

6.2 Power Implementations

Although the table lookup method is very efficient, the memory required for table storage is usually too large for a smart card environment. It is convenient to speed up a smart card application without requiring large tables. Because an S-box just requires a small array, the main concern is then how to perform MDS mappings with low memory cost.

Defined as 1-bit left shift followed by bitwise XOR with an appropriate irreducible polynomial in [7], the *xtime* operation can be used to perform multiplications for the MDS mapping. The operation *xtime* has no table lookups and the matrix multiplication is easier when all coefficients meet two requirements: (1) low Hamming weights; (2) low value. It is easy to find an MDS mapping satisfying these two requirements. When an SPN uses this mapping, however, the mapping's inverse used for decryption does not necessary satisfy the two conditions and many more operations are therefore needed.

An alternative way to implement multiplications makes use of transformation between power and polynomial representations over GF(2^n) for an MDS code based on a primitive polynomial. (This approach cannot be used for AES which has its MDS mapping based on an irreducible but not primitive polynomial.) Suppose $poly(\cdot)$ returns the polynomial representation of a GF(2^n) element from the index of power representation and its inverse function is denoted as $pow(\cdot)$. We know that

$$Y = C \cdot X = poly((pow(C) + pow(X)) \bmod (2^n - 1))$$

when $C \neq 0, X \neq 0$, and where Y is in polynomial representation. If the processor records the carry bit c for n-bit addition $pow(C)+pow(X)$, the modulo operation can be bypassed:

$$Y = C \cdot X = poly((pow(C) + pow(X)) + c)\ .$$

Using this method to perform the MDS mapping after substitution, C indicates one coefficient in MDS generation matrix $\mathcal{C}$. Each coefficient C in $\mathcal{C}$ is constant and nonzero. Denote $pow(C)$ as C_{pow}. If X is the output of an $n \times n$ S-box $S(\cdot)$ with input Z, substitution $S(\cdot)$ can be merged into the above operations:

$$Y = C \cdot X = poly(C_{pow} + pow(S(Z)) + c)\ .$$

Therefore, each multiplication over Galois fields costs two additions and two table lookups. The two tables for $poly(\cdot)$ and $pow(S(Z))$ need 2^{n+1} bytes in total. It can be seen that the nature of the coefficients in generation matrix $\mathcal{C}$ does not affect the speed of multiplication. If the coefficients in $\mathcal{C}$ are randomly selected, this method is more efficient than the *xtime* method.

6.3 General Comparison of Methods

Table 7 gives a general comparison of the software implementation methods discussed in this section.

Table 7: Comparison of Software Methods Used in MDS Codes

Method	Speed	Memory	Universal ?	MDS coefficents affect speed ?	MDS coefficents affect memory ?	S-box/MDS operations merged?
Table lookups	fast	large	yes	no	no	yes
Bitslice	parameter-dependent	none	no[†]	yes	no	no
Power	slow	small	yes[‡]	no[*]	no	yes
xtime	slow	none	yes	yes	no	no

† : the number of parallel sets should be compatible with machine operand sizes.
‡ : the polynomial to define the finite field must be primitive.
$*$: it can make a small difference depending on how many coefficients are 1s in $\mathcal{C}$ (i.e., $C_{pow} = 0$).

7 Conclusions

We have considered the software performance of two cipher structures composed of S-boxes and MDS mappings. Various cipher cases are generated from these structures with different component configurations. Table lookup implementations are used to evaluate the software efficiency of the various cases. A performance metric is defined to capture the security and efficiency simultaneously. Generally, cases using 8×8 S-boxes are faster than cases using 4×4 S-boxes and nested SPNs are more efficient in obtaining security than Feistel networks. Specifically, AES and Camellia were analyzed in terms of software performance, and some interesting performance features were noted and confirmed through experimental results. Three other software implementation methods that are applicable in special circumstances were also discussed and their general advantages and disadvantages were listed.

The methodology, while broad in its applicability to many modern ciphers, does not provide a comparison that is universal for all cipher structures across all platforms. However, it does represent a meaningful first step in better characterizing cipher efficiency for software implementations in relation to cipher security against cryptanalytic attacks.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments and suggestions.

References

[1] D. R. Stinson, *Cryptography Theory and Practice (2nd Edition)*. Chapman & Hall/CRC, 2002.

[2] W. Stallings, *Cryptography and Network Security: Principles and Practice (3rd Edition)*. Prentice Hall, 2002.

[3] C. Shannon, "Communication theory of secrecy systems," in *Bell System Technical Journal*, vol. 28, pp. 656–715, 1949.

[4] H. Heys and S. Tavares, "Substitution-permutation networks resistant to differential and linear cryptanalysis," in *Journal of Cryptology*, vol. 9, pp. 1–19, 1996.

[5] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam: North-Holland, 1977.

[6] National Institute of Standards and Technology, "FIPS 46-3 Data Encryption Standard (DES)," Available at `csrc.nist.gov/publications/fips`.

[7] J. Daemen and V. Rijmen, "AES proposal: Rijndael," Available at `csrc.nist.gov/encryption/aes/rijndael`.

[8] National Institute of Standards and Technology, "FIPS 197 Advanced Encryption Standard (AES)," Available at `csrc.nist.gov/publications/fips`.

[9] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: a 128-bit block cipher suitable for multiple platforms - design and analysis," in *Proceedings of Selected Areas in Cryptography - SAC 2000*, vol. 2012 of *Lecture Notes in Computer Science*, pp. 39–56, Springer-Verlag, 2001.

[10] New European Schemes for Signatures Integrity and Encryption (NESSIE) website. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[11] E. Biham and A. Shamir, “Differential cryptanalysis of DES-like cryptosystems,” in *Proceedings of Advances in Cryptology - CRYPTO’90*, vol. 537 of *Lecture Notes in Computer Science*, pp. 2–21, Springer-Verlag, 1991.

[12] M. Matsui, “Linear cryptanalysis method for DES cipher,” in *Proceedings of Advances in Cryptology - Eurocrypt ’93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 386–397, Springer-Verlag, 1994.

[13] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. D. Win, “The cipher SHARK,” in *Proceedings of Fast Software Encryption - FSE’96*, vol. 1039 of *Lecture Notes in Computer Science*, pp. 99–112, Springer-Verlag, 1997.

[14] J. Daemen, L. R. Knudsen, and V. Rijmen, “The block cipher Square,” in *Proceedings of Fast Software Encryption - FSE’97*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Springer-Verlag, 1997.

[15] P. Barreto and V. Rijmen, “The Anubis block cipher,” in *First Open NESSIE Workshop*, Leuven, November 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[16] P. Barreto and V. Rijmen, “The Khazad legacy-level block cipher,” in *First Open NESSIE Workshop*, Leuven, November 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[17] K. Ohkuma, H. Muratani, F. Sano, and S. Kawamura, “The block cipher Hierocrypt,” in *Proceedings of Selected Areas in Cryptography - SAC 2000*, vol. 2012 of *Lecture Notes in Computer Science*, pp. 72–88, Springer-Verlag, 2001.

[18] Toshiba Corporation, “Security evaluation: Hierocrypt-3.” NESSIE Algorithm Submission, 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[19] R. Anderson, E. Biham, and L. Knudsen, “Serpent: A proposal for the advanced encryption standard.” Available at `www.cl.cam.ac.uk/~rja14/serpent.html`.

[20] M. Matsui, “New block encryption algorithm MISTY,” in *Proceedings of Fast Software Encryption - FSE’97*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Springer-Verlag, 1997.

[21] M. Kanda, “Practical security evaluation against differential and linear attacks for Feistel ciphers with SPN round function,” in *Proceedings of Selected Areas in Cryptography - SAC 2000*, vol. 2012 of *Lecture Notes in Computer Science*, pp. 324–338, Springer-Verlag, 2001.

[22] J. Daemen, *Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis.* PhD thesis, Katholieke Universiteit Leuven, 1995.

[23] M. Kanda, Y. Takashima, T. Matsumoto, K. Aoki, and K. Ohta, “Strategy for constructing fast round functions with practical security against differential and linear cryptanalysis,” in *Proceedings of Selected Areas in Cryptography - SAC’98*, vol. 1556 of *Lecture Notes in Computer Science*, pp. 264–279, Springer-Verlag, 1999.

[24] J. Daemen and V. Rijmen, “Resistance against implementation attacks: A comparative study of the AES proposals,” in *Proceedings of the Second AES Candidate Conference*, pp. 122–132, the U.S. National Institute of Standards and Technology, March 1999.

[25] C. S. Clapp, “Instruction-level parallelism in AES candidates,” in *Proceedings of the Second AES Candidate Conference*, pp. 68–84, the U.S. National Institute of Standards and Technology, March 1999.

[26] A. Fiskiran and R. Lee, “Performance impact of addressing modes on encryption algorithms,” in *Proceedings of the International Conference on Computer Design - ICCD 2001*, pp. 542–545, September 2001.

[27] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd Edition).* John Wiley & Sons, 1995.

[28] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, “Camellia: A 128-bit block cipher suitable for multiple platforms (NESSIE submission),” 2000. Available at `www.cosic.esat.kuleuven.ac.be/nessie`.

[29] P. Barreto and V. Rijmen, "AES reference code in ANSI C." Available at `www.esat.kuleuven.ac.be/~rijmen/rijndael`.

[30] W. Stallings, *Computer Organization and Architecture (6th Edition).* Prentice Hall, 2002.

[31] E. Biham, "A fast new DES implementation in software," in *Proceedings of Fast Software Encryption - FSE'97*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 260–272, Springer-Verlag, 1997.

[32] L. Xiao and H. M. Heys, "Hardware performance characterization of block cipher structures," in *Proceedings of Cryptographers' Track RSA Conference 2003*, vol. 2612 of *Lecture Notes in Computer Science*, pp. 176–192, Springer-Verlag, 2003.

[33] A. Rudra, P. Dubey, C. Jutla, V. Kumar, J. Rao, and P. Rohatgi, "Efficient Rijndael encryption implementation with composite field arithmetic," in *Proceedings of Cryptographic Hardware and Embedded Systems - CHES 2001*, vol. 2162 of *Lecture Notes in Computer Science*, pp. 171–184, Springer-Verlag, 2001.

[34] L. Xiao and H. M. Heys, "Hardware design and analysis of block cipher components," in *Proceedings of the 5th International Conference on Information Security and Cryptology - ICISC 2002*, vol. 2587 of *Lecture Notes in Computer Science*, pp. 164–181, Springer-Verlag, 2003.

[35] N. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," in *Proceedings of Advances in Cryptology - ASIACRYPT 2002*, vol. 2501 of *Lecture Notes in Computer Science*, pp. 267–287, Springer-Verlag, 2002.