Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms

Kazumaro Aoki[†] Tetsuya Ichikawa[‡] Masayuki Kanda[†]
Mitsuru Matsui[‡] Shiho Moriai[†] Junko Nakajima[‡] Toshio Tokita[‡]

[†]Nippon Telegraph and Telephone Corporation
1-1 Hikarinooka, Yokosuka, Kanagawa, 239-0847 Japan
{maro,kanda,shiho}@isl.ntt.co.jp

[‡]Mitsubishi Electric Corporation
5-1-1 Ofuna, Kamakura, Kanagawa, 247-8501 Japan
{ichikawa,matsui,june15,tokita}@iss.isl.melco.co.jp

September 26, 2000

Abstract. We present a new 128-bit block cipher called *Camellia*. Camellia supports 128-bit block size and 128-, 192-, and 256-bit key lengths, i.e. the same interface specifications as the Advanced Encryption Standard (AES). Camellia was carefully designed to withstand all known cryptanalytic attacks and even to have a sufficiently large security leeway for use of the next 10-20 years. There are no hidden weakness inserted by the designers. It was also designed to have suitability for both software and hardware implementations and to cover all possible encryption applications that range from low-cost smart cards to high-speed network systems. Compared to the AES finalists, Camellia offers at least comparable encryption speed in software and hardware. An optimized implementation of Camellia in assembly language can encrypt on a Pentium III (800MHz) at the rate of more than 276 Mbits per second, which is much faster than the speed of an optimized DES implementation. In addition, a distinguishing feature is its small hardware design. The hardware design, which includes key schedule, encryption and decryption, occupies approximately 11K gates, which is the smallest among all existing 128-bit block ciphers as far as we know. It perfectly meet current market requirements in wireless cards, for instance, where low power consumption is a mandaroty condition.

Contents

1 Introduction

This paper presents a 128-bit block cipher called *Camellia*, which was jointly developed by NTT and Mitsubishi Electric Corporation. Camellia supports 128-bit block size and 128-, 192-, and 256-bit key lengths, and so offers the same interface specifications as the Advanced Encryption Standard (AES). The design goals of Camellia are as follows.

High level of security. The recent advances in cryptanalytic techniques are remarkable. A quantitative evaluation of security against powerful cryptanalytic techniques such as differential cryptanalysis [BS93] and linear cryptanalysis [M94] is considered to be essential in designing any new block cipher. We evaluated the security of Camellia by utilizing state-of-art cryptanalytic techniques. We have confirmed that Camellia has no differential and linear characteristics that hold with probability more than 2^{-128}. Moreover, Camellia was designed to offer security against other advanced cryptanalytic attacks including higher order differential attacks [K95, JK97], interpolation attacks [JK97, A00], related-key attacks [B94, KSW96], truncated differential attacks [K95, MT99], boomerang attacks [W99], and slide attacks [BW99, BW00].

Efficiency on multiple platforms. As cryptographic systems are needed in various applications, encryption algorithms that can be implemented efficiently on a wide range of platforms are desirable, however, few 128-bit block ciphers are suitable for both software and hardware implementation. Camellia was designed to offer excellent efficiency in software and hardware implementations, including gate count for hardware design, memory requirements in smart card implementations, as well as performance on multiple platforms.

Camellia consists of only 8-by-8-bit substitution tables (s-boxes) and logical operations that can be efficiently implemented on a wide variety of platforms. Therefore, it can be implemented efficiently in software, including the 8-bit processors used in low-end smart cards, 32-bit processors widely used in PCs, and 64-bit processors. Camellia doesn't use 32-bit integer additions and multiplications, which are extensively used in some software-oriented 128-bit block ciphers. Such operations perform well on platforms providing a high degree of support, e.g., Pentium II/III or Athlon, but not as well on others. These operations can cause a longer critical path and larger hardware implementation requirements.

The s-boxes of Camellia are designed to minimize hardware size. The four s-boxes are affine equivalent to the inversion function in the finite field $\mathrm{GF}(2^8)$. Moreover, we reduce the inversion function in $\mathrm{GF}(2^8)$ to a few $\mathrm{GF}(2^4)$ arithmetic operations. It enables us to implement the s-boxes by fewer gate counts.

The key schedule is very simple and shares part of its procedure with encryption. It supports on-the-fly subkey generation and subkeys are computable in any order. The memory requirement for generating subkeys is quite small; an efficient implementation requires about 32-byte RAM for 128-bit keys and about 64-byte RAM for 192- and 256-bit keys.

Future developments. NTT and Mitsubishi Electric Corporation will propose Camellia in response to the call for contributions from ISO/IEC JTC 1/SC 27, aiming at its being adopted as an international standard. We will submit Camellia to NESSIE (New European Schemes for Signature, Integrity, and Encryption) project as a strong cryptographic primitive.

Outline of the paper. This paper is organized as follows: Section 2 describes the rationale behind Camellia's design. Section 3 discusses the performance of Camellia. Section 4 contains the techniques for software implementation. In Section 5 we discuss our hardware evaluations. In Section 6 we evaluated Camellia's strength against known attacks. We conclude in Section 7.

For the specification of Camellia, please see the separate document titled "Specification of Camellia – a 128-bit Block Cipher." We will follow the definitions and notation given in this separate paper.

2 Design Rationale

2.1 *F*-function

The design strategy of the F-function of Camellia follows that of the F-function of E2 [KMA+98]. The main difference between E2 and Camellia is the adoption of the 1-round (conservative) SPN (Substitution-Permutation Network), not the 2-round SPN, i.e. S-P-S. When the 1-round SPN is used as the round function in a Feistel cipher, the theoretical evaluation of the upper bound of differential and linear characteristic probability becomes more complicated, but the speed under the same level of "real" security is expected to be improved. See Section 6 for detailed discussions on security.

2.2 *P*-function

The design rationale of the P-function is similar to that of the P-function of E2. That is, for computational efficiency, it should be represented using only bytewise exclusive-ORs and for security against differential and linear cryptanalysis, its branch number should be optimal [KTM+99]. From among the linear transformations that satisfy these conditions, we chose one considering highly efficient implementation on 32-processors [AU00] and high-end smart cards, as well as 8-bit processors.

2.3 *s*-boxes

As the s-boxes we adopted functions affine equivalent to the inversion function in GF(2^8) for enhanced security and small hardware design.

It is well known that the smallest of the maximum differential probability of functions in GF(2^8) was proven to be 2^{-6}, and the smallest of the maximum linear probability of functions in GF(2^8) is conjectured to be 2^{-6}. There is a function affine equivalent to the inversion function in GF(2^8) that achieves the best known of the maximum differential and linear probabilities, 2^{-6}. We choose this kind of functions as s-boxes. Moreover, the high degree of the Boolean polynomial of every output bit of the s-boxes makes it difficult to attack Camellia by higher order differential attacks. The two affine functions that are performed at the input and output of the inversion function in GF(2^8) complicates the expressions of the s-boxes in GF(2^8), which makes interpolation attacks ineffective. Making the four s-boxes different slightly improves security against truncated differential cryptanalysis [MT99].

For small hardware design, the elements in GF(2^8) can be represented as polynomials with coefficients in the subfield GF(2^4). In other words, we can implement the s-boxes by using a few operations in the subfield GF(2^4) [MIYY88]. Two affine functions at the input and output of the inversion function in GF(2^8) also play a role in complicating the expressions of the s-boxes in GF(2^4).

2.4 *FL*- and FL^{-1}-functions

FL- and FL^{-1}-functions are "inserted" between every 6 rounds of a Feistel network to provide non-regularity across rounds. One of the goals for such a design is to thwart future unknown attacks. It is one of merits of regular Feistel networks that encryption and decryption procedures are the same except for the order of the subkeys. In Camellia, FL/FL^{-1}-function layers are inserted every 6 rounds, but this property is still preserved.

The design criteria of FL- and FL^{-1}-functions are similar to those of the FL-function of MISTY [M97]. The difference between MISTY and Camellia is the addition of 1-bit rotation. This is expected to make bytewise cryptanalysis harder, but it has no negative impact on hardware size or speed. The design criteria are that these functions must be linear for any fixed key and that their forms depend on key values. Since these functions are linear as long as the key is fixed, they do not make the average differential and linear probabilities of the cipher higher. Moreover, these functions are fast in both software and hardware since they are constructed by logical operations such as AND, OR, XOR, and rotations.

2.5 Key Schedule

The design criteria of the key schedule are as follows.

1. It should be simple and share part of its procedure with encryption/decryption.
2. Subkey generation for 128-, 192- and 256-bit keys can be performed by using the same key schedule (circuit). Moreover, the key schedule for 128-bit keys can be performed by using a part of this circuit.
3. Key setup time should be shorter than encryption time.
   In cases where large amounts of data are processed with a single secret key, the setup time for key schedule may be unimportant. On the other hand, in applications in which the key is changed frequently, key agility is a factor. One basic component of key agility is key setup time.
4. It should support on-the-fly subkey generation.
5. On-the-fly subkey generation should be computable in the same way in both encryption and decryption.

   Some ciphers have separate key schedules for encryption and decryption. In other ciphers, e.g., Rijndael [DR98] or Serpent [ABK98], subkeys are computable in the forward direction only and require unwinding for decryption.
6. There should be no equivalent keys.
7. There should be no related-key attacks or slide attacks.

Criteria 1 and 2 mainly address small hardware requirements, Criteria 3, 4, and 5 are advantageous in terms of practical applications, and Criteria 6 and 7 are for security.

The memory requirement for generating subkeys is quite small. An efficient implementation of Camellia for 128-bit keys requires 16 bytes (=128 bits) for the original secret key, K_L, and 16 bytes (=128 bits) for the intermediate key, K_A. Thus the required memory is 32 bytes. Similarly, an efficient implementation of Camellia for 192- and 256-bit keys needs only 64 bytes.

3 Performance Figures

3.1 Software Performance

Table 1 summarizes the current software performance of Camellia. The table shows that Camellia can be efficiently implemented on low-end smart cards, and 32-bit and 64-bit processors. We use the abbreviations M (mega) for 10^6 and m (milli) for 10^{-3} in the table.

Note that the table only shows "point" data. This means that we can implement a program that requires less memory but works less efficiently. Future results may show "continuous" data.

Optimization level. When we coded programs, we tried to use the techniques described in Section 4. However, since we did not have enough time to try all combinations of the techniques, our numbers are probably not the best.

We think that C implementation is not important, since we can normally use assembly language instead if it is available. Moreover, optimizing a C program means reverse engineering the C compiler, since we often perform the following cycle when optimizing C code.

Step 1: Program or modify C code.

Step 2: Produce assembly code using a C compiler.

Step 3: Check the assembly code and go to Step 1 if not appropriate.

Step 4: Measure timing, and go to Step 1 if not satisfied.

When optimizing a program, we try to predict what assembly code the compiler will produce from the C code. We think that this approach is inefficient and so did not try to optimize C code.

How to measure speed. It is difficult to measure speed on modern processors since there are many elements, for example, status of cache, that are beyond the users control and that influence speed. We decided to measure speed under the following conditions and assumptions:

- All codes and data are correctly aligned.
- Input and output texts and codes are preloaded to the first level cache.
- Branch predictions are correct.
- Setup function (except for on-the-fly implementations) generates subkey-dependent constants from the secret key, and the constants are used by encryption or decryption function.
- Encryption (decryption) function except for on-the-fly implementations can encrypt (decrypt) an integral number of blocks.
- We measured the speed many times, and chose the best result to eliminate cache hit misses and other uncontrollable factors.
- We averaged the speed numbers for large block encryption, but the values include all overheads including loop and function calls.

Table 1: Camellia software performance

Processor	Language	Key length [bits]	Timing data Setup[a] [cycles] ([d])	Timing data Enc.[b] [cycles[e]] ([f])	Dynamic memory[c] Setup[a]	Dynamic memory[c] Enc.[b]	Code size [bytes] Setup[a]	Code size [bytes] Enc.[b]	Table size [bytes]
P III[g]	Assembly	128	160 (4.4M)	371 (242M)	28	36	1,046	2,150	8,224
		192	222 (3.2M)	494 (181M)	28	36	1,469	3,323	8,240
		256	226 (3.1M)	494 (181M)	28	36	1,485	3,323	8,240
P II[h]	ANSI C[i]	128	263 (1.1M)	577 (67M)	44	64	1,600	3,733	4,128
Alpha[j]	Assembly	128	118 (5.7M)	339 (252M)	48	48	1,132	3,076	16,528
		192	176 (3.7M)	445 (192M)	48	48	1,668	4,000	16,528
		256	176 (3.7M)	445 (192M)	48	48	1,676	4,000	16,528
		128	158 (4.2M)	326 (262M)	48	48	1,600	2,928	16,512
8051[k]	Assembly	128	0 (0)	10,217 (10m)	0	32	0	702	288
M32Rx/D[l]	Assembly	128	642 (1.6M)	1,236 (10.4M)	44	44	1,392	3,164	4,128

[a] Key schedule may be included.

[b] The data given in this column are the same as those for decryption, since Camellia is symmetric between encryption and decryption.

[c] Dynamically used memory in bytes including stack area, excluding text and key area, which is usually located in RAM.

[d] Setup performance given in seconds for 8051, and keys/s for other processors.

[e] Performance given in cycles per block. The efficiency in cycles per byte is obtained by dividing each data by 16.

[f] Performance given in seconds for 8051, and b/s for other processors.

[g] IBM PC/AT compatible PC, Intel Pentium III (700MHz), 256KB on-die L2 cache, FreeBSD 4.0R, 128MB main memory.

[h] IBM PC/AT compatible PC, Intel Pentium II (300MHz), 512KB L2 cache, MS-Windows 95, 160MB main memory.

[i] Microsoft Visual C++ 6 with the optimization options `/G6 /Zp16 /ML /Ox /Ob2`.

[j] COMPAQ Professional Workstation XP1000, Alpha 21264 (667MHz), Compaq Tru64 UNIX 4.0F, 2GB main memory.

[k] Intel 8051 (12MHz; 1 cycle = 12 oscillator periods) simulator on Unix.

[l] Mitsubishi 32-bit microcomputer M32Rx/D (100MHz) on MSA2310 evaluation board.

3.2 Hardware Performance

Tables 4 through 7 summarize the hardware performance of 128bit-key Camellia on ASIC (Application Specific Integrated Circuit) and FPGA (Field Programmable Gate Array). Table 2 shows the environment of our hardware design and evaluation.

Table 2: Hardware evaluation environment (ASIC, FPGA)

Language	(ASIC, FPGA) Verilog-HDL
Simulator	(ASIC, FPGA) Verilog-XL
Design library	(ASIC) Mitsubishi Electric 0.35μ CMOS ASIC library (FPGA) Xilinx XC4000XL series
Login synthesis	(ASIC) Design Compiler version 1998.08 (FPGA) Synplify version 5.3.1 and ALLIANCE version 2.1i

We evaluated Type 1 through Type 3 logic. Table 3 shows the top priorities of the logic types.

Table 3: Hardware design policies (outline)

Type	Top priority	Outline of logic
Type 1	Fast implementation from the viewpoint of Enc(Dec) speed	Figure 1
Type 2	Small implementation from the viewpoint of total logic size	Figure 2
Type 3	Small implementation (special case for FPGA)	Figure 3

The details of each type are described in Section 5.

Table 4: Hardware performance (Type 1: Fast implementation [ASIC(0.35μ CMOS)])

Algorithm name	Area[Gate] Enc.&Dec.[a]	Key expan.[b]	Total logic[c]	Key setup time[ns]	Critical-path[ns][d]	Throughput [Mb/s]
DES	42,204	12,201	54,405	—	55.11	1161.31
Triple-DES	124,888	23,207	128,147	—	157.09	407.40
MARS	690,654	2,245,096	2,935,754	1740.99	567.49	225.55
RC6	741,641	901,382	1,643,037	2112.26	627.57	203.96
Rijndael	518,508	93,708	612,834	57.39	65.64	1950.03
Serpent	298,533	205,096	503,770	114.07	137.40	931.58
Twofish	200,165	231,682	431,857	16.38	324.80	394.08
Camellia	216,911	55,907	272,819	24.36	109.35	1170.55

[a]including output registers
[b]including subkey registers
[c]including buffers for fan-out adjustment
[d]Critical path of data encryption (or decryption)

Table 5: Hardware performance (Type 2: Small implementation [ASIC(0.35μ CMOS)])

Algorithm name	Area[Gate] Enc.&Dec.[a]	Key sched.[b]	Total logic [c]	Key setup time[ns]	Critical-path[ns][d]	Throughput [Mb/s]
Camellia	6,367	4,979	11,350	110.2	27.67	220.28

[a]including output registers and data selector
[b]including subkey registers and a part of key expansion logic
[c]including buffers for fan-out adjustment
[d]Critical path of data encryption (or decryption)

Table 6: Hardware performance (Type 2: Small implementation [FPGA(XC4000XL series)])

Algorithm name	Area[CLBs] Total	Critical-path[ns][a]	Throughput [Mb/s]
Camellia	1,296	78.815	77.34

[a]Critical path of data encryption (or decryption)

Table 7: Hardware performance (Type 3: Small implementation [FPGA(XC4000XL series)])

Algorithm name	Area[CLBs] Total	Critical-path[ns][a]	Throughput [Mb/s]
Camellia	874	49.957	122.01

[a] Critical path of data encryption (or decryption)

4 Software Implementation Techniques

This section describes how to implement Camellia efficiently in software. In most cases, an implementation can be divided into two parts: *setup* including key schedule and *data randomization*, that is, encryption or decryption. We first describe how to optimize the setup code, and then describe how to optimize the data randomization code.

This section describes specific techniques for 8-, 32-, or 64-bit processors. However, a technique for 8-bit processors may be applicable to 32- or 64-bit processors and a technique for 32-bit processors may be applicable to 64-bit processors. Other word sizes may need to be considered.

We assume that you first implement Camellia using the specification as it is. This section will optimize the resulting code.

Note that in this section "word" means the natural size of the target processor. For example, the words of IA-32 without MMX technology, IA-32 with MMX technology and Alpha are 32-, 64-, and 64-bits long respectively.

4.1 Setup

4.1.1 Store All Subkeys

Store all subkeys into memory once you generate them if you have sufficient memory, and use the stored subkeys for data randomization.

4.1.2 Subkey Generation Order

You do not have to compute subkeys in order. For example, when you compute subkeys for a 128-bit key, first compute the subkeys that only depend on K_L, and then compute subkeys that only depend on K_A. This reduces the number of registers or memory for storing K_A.

4.1.3 XOR Cancellation Property in Key Schedule

The key schedule of Camellia is based on the Feistel structure. Between the 2nd round and the 3rd round, K_L is XORed to an intermediate value. This structure causes cancellations of K_L. More precisely, the input of the 3rd round can be computed by the following equations.

$$\begin{cases} \text{(right half)} &= F(K_{LL}, \Sigma_1) \\ \text{(left half)} &= F(K_{LR} \oplus \text{(right half)}, \Sigma_2) \end{cases} \quad \text{for 128-bit keys}$$

$$\begin{cases} \text{(right half)} &= K_{RR} \oplus F(K_{LL} \oplus K_{RL}, \Sigma_1) \\ \text{(left half)} &= K_{RL} \oplus F(K_{LR} \oplus \text{(right half)}, \Sigma_2) \end{cases} \quad \text{for 192- and 256-bit keys}$$

Using the above equations, we can eliminate 3 and 2 XORs in $\mathbf{L}$ for 128- and 192/256-bit keys, respectively, compared to the straightforward implementation of the specification.

4.1.4 Rotation Bits for K_L, K_R, K_A, and K_B

You do not need to keep K_L, K_R, K_A, and K_B, but you should keep their rotated values when generating subkeys. You can generate subkeys by rotating the kept values by a sum of integral multiples of 16 ± 1 bits.

4.1.5 kl_5 and kl_6 Generation from k_{11} and k_{12}

For 192- and 256-bit keys, you can use word-oriented rotation to generate (kl_5, kl_6) from (k_{11}, k_{12}), since (kl_5, kl_6) equals $(k_{11}, k_{12}) \lll_{32}$. This saves a few instructions compared to general rotation.

4.1.6 On-the-fly Subkey Generation

You can generate subkeys *on-the-fly*. All subkeys are one of the rotated values of K_L, K_R, K_A, and K_B. Thus, you first generate K_L, K_R, K_A, and K_B, and then rotate them to get the subkeys. Refer to Section 4.1.4 for the rotated numbers of bits for K_L, K_R, K_A, and K_B.

4.1.7 128-bit key and 192/256-bit key

If your code does not need to use key sizes larger than 128 bits, you do not need to generate K_B. That is, you can omit the computations for the last two F-functions.

4.1.8 How to Rotate an Element in Q

8-bit processor. As stated in Section 4.1.4, the amount of rotation in bits is a sum of integral multiples of 16 ± 1. Thus, you can rotate an element in $\mathbf{Q}$ by 16 ± 1 bits by rotating 1-bit left or right followed by a 2-byte move.

32-bit processor. Consider the use of a double precision shift instruction: `shrd` or `shld` if you are programming on IA-32.

4.1.9 F-function

Key schedule includes F-functions, but the main usage of the F-function is for data randomization. Refer to Section 4.2.

4.1.10 Keyed Functions

Camellia has three keyed functions: bitwise XOR, bitwise OR, and bitwise AND. Consider the use of a self-modifying code, if possible.

4.2 Data Randomization

4.2.1 Endian Conversion

Camellia prefers big endian. Thus, the code for little endian processors needs additional code for endian conversions.

The most straightforward implementation converts the endian when loading a register from memory and storing a register to memory. Only FL- and FL^{-1}-functions are endian dependent. More precisely, only the 1-bit rotation in FL- or FL^{-1}-function is endian dependent. This means that you can convert endians just before or just after the 1-bit rotation with the appropriate subkey generation scheme. A combination of computing endian conversion and 1-bit rotation may increase the performance of Camellia. Details are described in Section 4.2.2.

Some processors have a special instruction for endian conversion. For example, IA-32 (after 80486) has `bswap` instruction. Use these instructions. However, do not use the byte swap technique described in [C98, Appendix A]. The technique reduces the code size, but it is not fast, since the memory load and store instruction incurs long latency.

As described above, the endian problem only effects the 1-bit rotation of a 32-bit word. Thus, we do not need full 64-bit word endian conversion.

The following are general methods to realize endian conversion for 32-bit register x. In the following techniques, you can use either $\cup$ or $\oplus$ instead of $+$ in the equations, and you can switch the computational order between shifts including rotations and ANDs with an appropriate conversion of masked constants.

Straightforward.

$$x \leftarrow (x \ll_{24}) + ((x \cap \texttt{0xff00}) \ll_8) + ((x \gg_8) \cap \texttt{0xff00}) + (x \gg_{24})$$

The technique has high parallelism.

Minimum operations without rotation.

$$\begin{aligned} x &\leftarrow (x \ll_{16}) + (x \gg_{16}) \\ x &\leftarrow ((x \cap \texttt{0xff00ff}) \ll_8) + ((x \gg_8) \cap \texttt{0xff00ff}) \end{aligned}$$

Using rotations.

$$x \leftarrow ((x \cap \texttt{0xff00ff}) \ggg_8) + ((x \lll_8) \cap \texttt{0xff00ff})$$

Using SSE. New Intel Pentium family processors including Pentium III have a very effective instruction for reordering data, which is called `pshufw` [I99]. 5 instructions including `pshufw` are sufficient to convert endian for 64-bit data.

4.2.2 1-bit Rotation in Little Endian Interpretation

As described in Section 4.2.1, we do not need endian conversion when loading and storing texts if we can efficiently implement 1-bit rotation in FL- and FL^{-1}-functions.

Assuming x to be a 32-bit register that contains little endian data to be rotated by 1-bit, we can compute 1-bit rotation by the following equation.

$$x \leftarrow ((2x) \cap \texttt{0xfefefefe}) + ((x \ggg_{15}) \cap \overline{\texttt{0xfefefefe}}) \tag{1}$$

Of course, this technique requires an appropriate changes to subkey setup and other functions.

Note that $+$ in Equation (1) can be replaced with $\cup$ or $\oplus$, and computing $2x$ can be done by $\ll_1$, $\lll_1$ or addition with x itself, and you can switch the computational order between shifts including rotations and ANDs with an appropriate conversion of masked constants.

Confirm whether your processor has ANDNOT instruction, such as `pandn` in IA-32 and `bic` in Alpha. In this case, you do not need to prepare the constant, $\overline{\texttt{0xfefefefe}}$.

4.2.3 Whitening

The key additions kw_2 and kw_4 can be combined into other keyed operations using the following equations.

$$\begin{aligned}(x \oplus k) \oplus y &= (x \oplus y) \oplus k,\\ (x \oplus k) \oplus l &= x \oplus (k \oplus l),\\ (x \oplus k) \cap l &= (x \cap l) \oplus (k \cap l),\\ (x \oplus k)_{\lll 1} &= (x_{\lll 1}) \oplus (k_{\lll 1}),\\ (x \oplus k) \cup l &= (x \cup l) \oplus (k \cap \bar{l}),\end{aligned} \tag{2}$$

where x, y, k, l are bit strings. Adjust subkeys at setup to eliminate 2 XORs in **L**.

4.2.4 Key XOR

Using Equations (2), you can move key XORs to any place if the movement does not go through the S-function. For example, changing F-function computation $P(S(X \oplus k))$ to $P(S(X)) \oplus k'$ may improve instruction scheduling.

4.2.5 *S*-function

s_1 is defined by the arithmetics in GF(2^8). However, do not compute GF(2^8) arithmetics; instead precompute and hard-code a table in your program, see Table 4 in the specification.

We strongly suggest that you also precompute and hard-code s_2, s_3, and s_4 tables in addition to s_1, if you have sufficient memory and 8-bit rotation is expensive. If you do not have sufficient memory, the data of s_2, s_3, and s_4 can be generated from the table for s_1 using one rotation (See Section 4.5 in the specification).

If you have sufficient memory, and cost of table lookup is heavy, as is true for the current Java virtual machines, consider the use of a two s-box combined table, for example $(s_1(y_1), s_2(y_2))$.

4.2.6 *P*-function

32-bit processor. Let $(Z_L, Z_R) = ((z_1, z_2, z_3, z_4), (z_5, z_6, z_7, z_8))$ be the input of P-function and $(Z'_L, Z'_R) = ((z'_1, z'_2, z'_3, z'_4), (z'_5, z'_6, z'_7, z'_8))$ be the output of P-function.

From Figure 5 in the specification, you can see that P-function can be computed as follows.

$$\begin{aligned}Z_L &\leftarrow Z_L \oplus (Z_{R\lll 8})\\ Z_R &\leftarrow Z_R \oplus (Z_{L\lll 16})\\ Z_L &\leftarrow Z_L \oplus (Z_{R\ggg 8})\\ Z_R &\leftarrow Z_R \oplus (Z_{L\ggg 8})\\ Z'_L &\leftarrow Z_R\\ Z'_R &\leftarrow Z_L\end{aligned}$$

The critical path of this computation is long. We can modify the computation as follows.

$$
\begin{array}{lclclcl}
 & & & Z_R & \leftarrow & Z_R \lll_8 \\
Z_L & \leftarrow & Z_L \oplus Z_R & Z_R & \leftarrow & Z_R \lll_8 \\
Z_L & \leftarrow & Z_L \ggg_8 & Z_R & \leftarrow & Z_R \oplus Z_L \\
Z_L & \leftarrow & Z_L \oplus Z_R & Z_R & \leftarrow & Z_R \lll_{16} \\
Z_L & \leftarrow & Z_L \lll_8 & Z_R & \leftarrow & Z_R \oplus Z_L \\
Z'_L & \leftarrow & Z_R & Z'_R & \leftarrow & Z_L
\end{array}
$$

The critical path of the above computation is decreased. It seems that the technique requires one additional rotation, however, you can probably combine the first step of the above computation and S-function without any additional cost.

8-bit processor (orthogonal mnemonics). If the instruction in your processor can XOR any combination of registers and has sufficient registers, you can compute P-function by using just 16 XORs using Figure 5 in the specification.

8-bit processor (accumulator based). If your processor is accumulator based, minimizing the number of XORs is not always a good idea, since the computation may require register load from memory and store into memory many times. The following computation is optimized for an accumulator based processor.

$$
\begin{array}{lcl}
z'_8 & \leftarrow & z_1 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7 \\
z'_4 & \leftarrow & z'_8 \oplus z_1 \oplus z_2 \oplus z_3 \\
z'_7 & \leftarrow & z'_4 \oplus z_2 \oplus z_7 \oplus z_8 \\
z'_3 & \leftarrow & z'_7 \oplus z_1 \oplus z_2 \oplus z_4 \\
z'_6 & \leftarrow & z'_3 \oplus z_1 \oplus z_6 \oplus z_7 \\
z'_2 & \leftarrow & z'_6 \oplus z_1 \oplus z_3 \oplus z_4 \\
z'_5 & \leftarrow & z'_2 \oplus z_4 \oplus z_5 \oplus z_6 \\
z'_1 & \leftarrow & z'_5 \oplus z_2 \oplus z_3 \oplus z_4
\end{array}
$$

When indexing z'_i costs many operations, the following is useful.

$$
\begin{array}{lcl}
\sigma & \leftarrow & z_1 \oplus z_2 \oplus z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7 \oplus z_8 \\
z'_1 & \leftarrow & \sigma \oplus z_2 \oplus z_5 \\
z'_2 & \leftarrow & \sigma \oplus z_3 \oplus z_6 \\
z'_3 & \leftarrow & \sigma \oplus z_4 \oplus z_7 \\
z'_4 & \leftarrow & \sigma \oplus z_1 \oplus z_8 \\
z'_5 & \leftarrow & \sigma \oplus z_3 \oplus z_4 \oplus z_5 \\
z'_6 & \leftarrow & \sigma \oplus z_1 \oplus z_4 \oplus z_6 \\
z'_7 & \leftarrow & \sigma \oplus z_1 \oplus z_2 \oplus z_7 \\
z'_8 & \leftarrow & \sigma \oplus z_2 \oplus z_3 \oplus z_8
\end{array}
$$

4.2.7 Substitution and Permutation

This section describes how to efficiently compute $P \circ S$ compared to independently computing S and P.

64-bit processor. If your processor has a sufficiently large first level cache, use the technique described in [RDP+96]. The technique prepares the following tables defined by Equations (3).

$$
\begin{array}{rcl}
SP_1(y_1) & = & (s_1(y_1),\ s_1(y_1),\ s_1(y_1),\ 0,\ s_1(y_1),\ 0,\ 0,\ s_1(y_1)) \\
SP_2(y_2) & = & (0,\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ 0,\ 0) \\
SP_3(y_3) & = & (s_3(y_3),\ 0,\ s_3(y_3),\ s_3(y_3),\ 0,\ s_3(y_3),\ s_3(y_3),\ 0) \\
SP_4(y_4) & = & (s_4(y_4),\ s_4(y_4),\ 0,\ s_4(y_4),\ 0,\ 0,\ s_4(y_4),\ s_4(y_4)) \\
SP_5(y_5) & = & (0,\ s_2(y_5),\ s_2(y_5),\ s_2(y_5),\ 0,\ s_2(y_5),\ s_2(y_5),\ s_2(y_5)) \\
SP_6(y_6) & = & (s_3(y_6),\ 0,\ s_3(y_6),\ s_3(y_6),\ s_3(y_6),\ 0,\ s_3(y_6),\ s_3(y_6)) \\
SP_7(y_7) & = & (s_4(y_7),\ s_4(y_7),\ 0,\ s_4(y_7),\ s_4(y_7),\ s_4(y_7),\ 0,\ s_4(y_7)) \\
SP_8(y_8) & = & (s_1(y_8),\ s_1(y_8),\ s_1(y_8),\ 0,\ s_1(y_8),\ s_1(y_8),\ s_1(y_8),\ 0)
\end{array} \tag{3}
$$

Next, compute the following equation:

$$
(z'_1, z'_2, z'_3, z'_4, z'_5, z'_6, z'_7, z'_8) \leftarrow \bigoplus_{i=1}^{8} SP_i(y_i)
$$

This technique requires the following operations.

# of table lookups	8
# of XORs	7
Size of table (KB)	16

If the first cache of the target processor is moderately large, replace a few of the tables defined by Equations (3) with the tables below.

$$
\begin{array}{rcl}
SP_\alpha(y) & = & (s_1(y),\ s_1(y),\ s_1(y),\ s_1(y),\ s_1(y),\ s_1(y),\ s_1(y),\ s_1(y)) \\
SP_\beta(y) & = & (s_2(y),\ s_2(y),\ s_2(y),\ s_2(y),\ s_2(y),\ s_2(y),\ s_2(y),\ s_2(y)) \\
SP_\gamma(y) & = & (s_3(y),\ s_3(y),\ s_3(y),\ s_3(y),\ s_3(y),\ s_3(y),\ s_3(y),\ s_3(y)) \\
SP_\delta(y) & = & (s_4(y),\ s_4(y),\ s_4(y),\ s_4(y),\ s_4(y),\ s_4(y),\ s_4(y),\ s_4(y))
\end{array} \tag{4}
$$

Then, mask the necessary byte positions. This technique requires the following operations if you use just tables of Equations (4).

# of table lookups	8
# of XORs	7
# of ANDs	8
Size of table (KB)	8

When implementing this technique on Alpha architecture [C98], and if the number of registers is insufficient for storing constants for masking operation, use `zap` or `zapnot` instructions.

If your processor can efficiently copy half bits of a register to the other half, for example, `punpckldq`/`punpckhdq` or `pshufw` instructions in IA-32 [I99] which are realized after Pentium with MMX technology and Pentium III, respectively, prepare SP_1, SP_2, SP_3, and SP_4 defined in Equations (3). Then, compute the following equation:

$$(z'_1, z'_2, z'_3, z'_4, z'_5, z'_6, z'_7, z'_8) \leftarrow \bigoplus_{i=1}^{4} SP_i(y_i) \oplus \nu(\bigoplus_{i=5}^{8} SP_{i-4}(y_i)),$$

where ν denotes the operation that copies the first 4 bytes to the last 4 bytes. This technique requires the following operations.

# of table lookups	8
# of XORs	7
# of νs	1
Size of table (KB)	8

32-bit processor. [AU00] shows efficient implementations of Camellia-type substitution and permutation networks. One of the techniques prepares the following tables defined by Equations (5):

$$\begin{array}{rcl} SP_{1110}(y) &=& (s_1(y),\ s_1(y),\ s_1(y),\ 0) \\ SP_{0222}(y) &=& (0,\ s_2(y),\ s_2(y),\ s_2(y)) \\ SP_{3033}(y) &=& (s_3(y),\ 0,\ s_3(y),\ s_3(y)) \\ SP_{4404}(y) &=& (s_4(y),\ s_4(y),\ 0,\ s_4(y)) \end{array} \tag{5}$$

Then, compute as follows:

$$\begin{array}{rcl} D &\leftarrow& SP_{1110}(y_8) \oplus SP_{0222}(y_5) \oplus SP_{3033}(y_6) \oplus SP_{4404}(y_7) \\ U &\leftarrow& SP_{1110}(y_1) \oplus SP_{0222}(y_2) \oplus SP_{3033}(y_3) \oplus SP_{4404}(y_4) \\ (z'_1, z'_2, z'_3, z'_4) &\leftarrow& D \oplus U \\ (z'_5, z'_6, z'_7, z'_8) &\leftarrow& (z'_1, z'_2, z'_3, z'_4) \oplus (U \ggg_8) \end{array}$$

This technique requires the following operations.

# of table lookups	8
# of XORs	8
# of rotations	1
Size of table (KB)	4

[AU00] also shows an implementation that is suitable for a processor in which rotation is very costly. The technique prepares the following tables in addition to tables defined by Equations (5):

$$\begin{array}{rcl} SP_{1001}(y) &=& (s_1(y),\ 0,\ 0,\ s_1(y)) \\ SP_{2200}(y) &=& (s_2(y),\ s_2(y),\ 0,\ 0) \\ SP_{0330}(y) &=& (0,\ s_3(y),\ s_3(y),\ 0) \\ SP_{0044}(y) &=& (0,\ 0,\ s_4(y),\ s_4(y)) \end{array}$$

Then, compute as follows:

$$\begin{aligned} D &\leftarrow SP_{1110}(y_8) \oplus SP_{0222}(y_5) \oplus SP_{3033}(y_6) \oplus SP_{4404}(y_7) \\ (z'_1, z'_2, z'_3, z'_4) &\leftarrow D \oplus SP_{1110}(y_1) \oplus SP_{0222}(y_2) \oplus SP_{3033}(y_3) \oplus SP_{4404}(y_4) \\ (z'_5, z'_6, z'_7, z'_8) &\leftarrow D \oplus SP_{1001}(y_1) \oplus SP_{2200}(y_2) \oplus SP_{0330}(y_3) \oplus SP_{0044}(y_4) \end{aligned}$$

This technique requires the following operations.

# of table lookups	12
# of XORs	11
Size of table (KB)	8

4.2.8 Making Indices for s-box

You can make an index for s-box by simply using shifts and ANDs. However, several processors have special instructions for making an index, for example, `movzx` in IA-32 [I99] and `extbl` in Alpha [C98].

`movzx` is a fast operation in P6, but it can be used only for the two least significant bytes. A straightforward implementation uses `eax`, `ebx`, `ecx`, and `edx` registers for storing (L_r, R_r), and 2 rotations are used for making indices; 2 rotations are used for recovering byte order in the registers every round. However, you can remove 2 rotations for recovering byte order every round if you prepare rotated tables. Note that the byte order in registers returns to a natural order every 4 rounds.

4.3 General Guidelines

This section describes general guidelines. The guidelines are useful to optimize Camellia as well as other block ciphers. Please refer to the optimization manuals for each processor.

Avoid misaligned data accesses. Almost all processors penalize misaligned data access. Align data to the word boundary.

Avoid partial data accesses. Most processors have a function to access a smaller part than word size. However, this function may cause a penalty. Do not access partial data, even if you do not need full size of word and you have sufficient memory.

Be careful of the size of the cache. If the program or its data exceeds the size of the cache, the speed of the program will significantly decrease. Loop unrolling and table expansion are good techniques to speed up the program, but do not exceed the size of the cache.

Use intrinsic functions. Several compilers support intrinsic functions. For example, when you use Microsoft Visual C++ version 6 compiler on IA-32, and declare "`#pragma intrinsic(_lrotl)`" and use "`_lrotl`", the compiler generates rotation instructions in assembly language. Refer to the manual of the compiler that you use for details.

Measuring precise speeds is difficult. The running time of your code depends on many factors: cache hit misses, OS interrupts, and so on. Furthermore, the cryptographic properties, for example, the number of blocks to be encrypted, also effect the running time.

A few processors have an instruction to get the time stamp. For example, IA-32 (after Pentium) has `rdtsc` [I99] and Alpha has `rpcc` [C98]. It is a good idea to use the time stamp counter for measuring speeds, but you should not directly apply these instructions to out-of-order architectures such as P6 and EV6.

If you want to measure speed precisely, consult good guidebooks. For example, if you use Pentium family processors, refer to [F00].

5 Hardware Evaluations

In Section 3, we showed evaluation results of hardware implementations (ASIC, FPGA) for Camellia with 128-bit keys. In this Section, we describe the design policies of the three types of logic evaluated in Section 3. The details of each type are described below.

5.1 Type 1: Fast Implementation

In Type 1, we evaluate the hardware implementation (ASIC) where the goal is to achieve the fastest encryption and decryption speed with no consideration of logic size. Figure 1 outlines the Type 1 logic. Table 8 shows the basic Type 1 components.

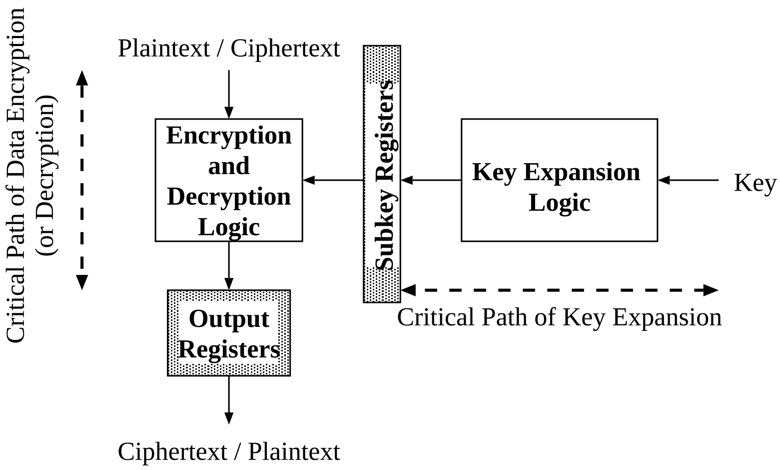

Figure 1: Outline of Type 1 (ASIC)

Table 8: The basic Type 1 components

Encryption and decryption logic	Data randomizing logic for encryption and decryption, which consists of combinational logic.
Output register	Register for the encryption (decryption) data.
Key expansion logic	Logic in which subkeys are generated from key, which consists of combinational logic.
Subkey register	Register for the output data of key expansion logic.

The design policies of these basic components are listed below.

1. "Encryption and decryption logic" and "Key expansion logic"

   (a) Loop architecture is not introduced.

(b) Pipeline architecture is not introduced.

(c) Substitution tables (s-boxes) are designed by logic synthesis tool.

Note that other encryption algorithms (AES finalists, DES and Triple-DES) are evaluated in Section 3, and some of them use arithmetic operations (addition, multiplication, etc). For arithmetic operations, we use faster ones in the library of Synopsys Design Ware Basic Library[Synopsys (1998)].

2. "Output register" and "Subkey register"

   (a) The size of Output register is one block (=128 bits).

   (b) The size of Subkey register is the total length of all subkeys in the algorithm.

Under the above design policies, we evaluated Camellia and other algorithms on ASIC devices. The results are summarized in Table 4 in Section 3. "Throughput" is defined as follows:

$$\text{Throughput[b/s]} = \frac{\text{Block size(128 [bits])}}{\text{Critical path of data encryption(decryption)[sec]}}.$$

5.2 Type 2: Small Implementation

In Type 2, we evaluate the hardware implementations (ASIC, FPGA) with the goal of achieving the smallest logic in encryption (and decryption). Figure 2 outlines the Type 2 logic. Table 9 shows the basic Type 2 components.

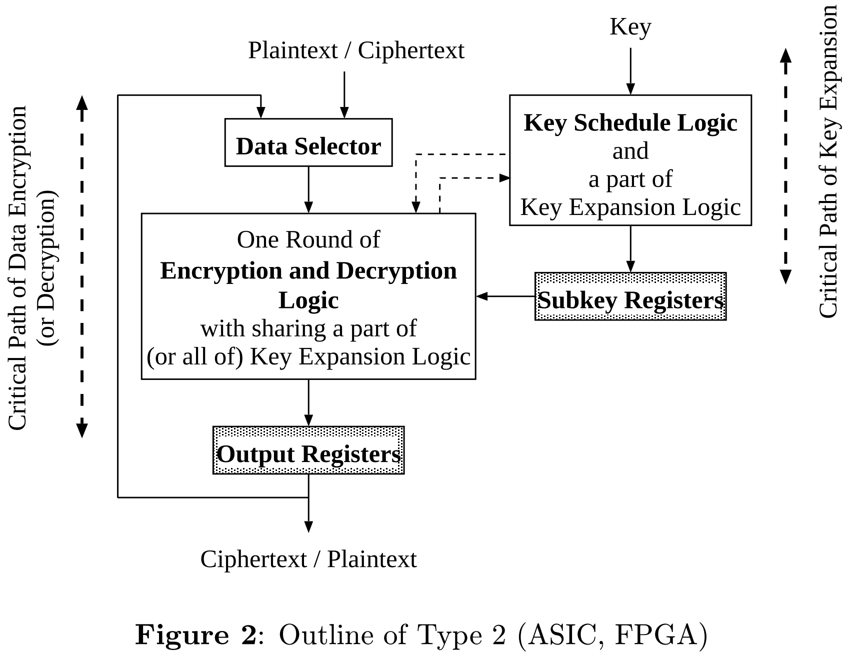

Figure 2: Outline of Type 2 (ASIC, FPGA)

The design policies of these basic components are as follows.

Table 9: The basic Type 2 components

Encryption and decryption logic	Data randomizing logic for one round operation of encryption and decryption, which includes (a part of) key expansion logic, and consists of combinational logics.
Output register	Register for the output (and pending) data.
Data selector	Selector which selects either en(de)cryption data or output data.
Key schedule logic	Logic in which subkeys are generated using (a part of) key expansion logic in encryption and decryption logic and consists of combinational logics.
Subkey register	Register for the output data of key schedule logic.

1. "Encryption and decryption logic" and "Key schedule logic"
   - (a) Loop architecture is introduced.
   - (b) Pipeline architecture is not introduced.
   - (c) Substitution tables (s-boxes) are optimized by hand.
   - (d) Key schedule logic consists (a part of) key expansion logic and control logic.
2. "Output register", "Subkey register" and "Data selector"
   - (a) The size of Output register is one block (=128 bits).
   - (b) The size of Subkey register is that of the subkeys used in Encryption and decryption logic.
   - (c) Data selector is 2-1 selector, whose size is one block (=128 bits).

Under the above design policies, we evaluated Camellia on ASIC and FPGA. The results are summarized in Table 5 (ASIC) and Table 6 (FPGA) in Section 3. "Throughput" is defined as follows:

$$\text{Throughput[b/s]} = \frac{\text{Block size(128 [bits])}}{\text{Critical path of data encryption(decryption)[sec]} \times \text{ round number}}.$$

5.3 Type 3: Small Implementation (Special Case for FPGA)

In Type 3, we evaluated the hardware implementation (FPGA) as a special case of Type 2. In Type 3, we assume that all subkeys are given and are loaded into FPGA internal memory. Figure 3 outlines the Type 3 logic. Table 10 shows the basic Type 3 components.

The design policies of these basic components are as follows.

1. "Encryption and decryption logic"
   - (a) Loop architecture is introduced (which consists of one round operation).
   - (b) Pipeline architecture is not introduced.

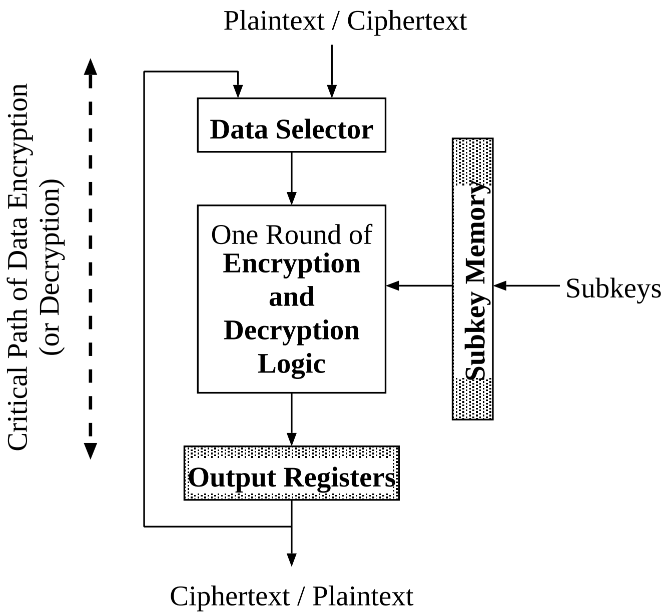

Figure 3: Outline of Type 3 (FPGA)

Table 10: The basic Type 3 components

Encryption and decryption logic	Data randomizing logic for one round operation of encryption and decryption, which includes (a part of) key expansion logic, and consists of combinational logic.
Output register	Register for the output (and pending) data.
Data selector	Selector which selects either en(de)cryption data or output data.
Subkey memory	Memory for the subkeys loaded from outside.

   (c) Substitution tables (s-boxes) are optimized by hand.

2. "Output register", "Subkey memory" and "Data selector"

   (a) The size of Output register is one block (=128 bits).
   (b) The size of Subkey memory is the length of all subkeys in the algorithm.
   (c) Data selector is 2-1 selector whose size is one block (=128 bits).

Under the above design policies, we evaluated Camellia on an FPGA. The results are summarized in Table 7 (FPGA). "Throughput" is defined as follows:

$$\text{Throughput[b/s]} = \frac{\text{Block size(128 [bits])}}{\text{Critical path of data encryption(decryption)[sec]} \times \text{ round number}}.$$

6 Security

6.1 Differential and Linear Cryptanalysis

The most well-known and powerful approaches to attacking many block ciphers are differential cryptanalysis, proposed by Biham and Shamir [BS93], and linear cryptanalysis, introduced by Matsui [M94]. There are several methods of evaluating security against these attacks, where there is a kind of "duality" relation between them [M95, CV95]: in other words, the security against both attacks can be evaluated in similar ways.

It is known that the upper bounds of differential/linear characteristic probabilities can, for several block ciphers, be estimated using the minimum numbers of differential/linear active s-boxes in some consecutive rounds. Kanda [K00] shows the minimum numbers of differential/linear active s-boxes for Feistel ciphers with conservative SPN (S-P) round function. Hereafter, we assume that linear transformation P is bijective.

Definition 1 The branch number $\mathcal{B}$ of linear transformation P is defined by

$$\mathcal{B} = \min_{x \neq 0}(w_{\mathrm{H}}(x) + w_{\mathrm{H}}(P(x))),$$

where $w_{\mathrm{H}}(x)$ denotes the bytewise Hamming weight of x.

Definition 2 A differential active s-box is defined as an s-box given a non-zero input difference. A linear active s-box is defined as an s-box given a non-zero output mask value.

Theorem 1 The minimum number of differential/linear active s-boxes in any eight consecutive rounds is equal or larger than $2\mathcal{B} + 1$.

Definition 3 For any given Δx, Δy, Γx, $\Gamma y \in \mathrm{GF}(2^m)$, the differential/linear probabilities of s_i-box: $\mathrm{GF}(2^m) \to \mathrm{GF}(2^m)$ are defined as:

$$\Pr_x[s_i(x) \oplus s_i(x \oplus \Delta x) = \Delta y] = \frac{\#\{x \in \mathrm{GF}(2^m) | s_i(x) \oplus s_i(x \oplus \Delta x) = \Delta y\}}{2^m}$$

$$\Pr_x[x \cdot \Gamma x = s_i(x) \cdot \Gamma y] = \frac{\#\{x \in \mathrm{GF}(2^m) | x \cdot \Gamma x = s_i(x) \cdot \Gamma y\}}{2^m}$$

Definition 4 Let p_s and q_s be the maximum differential/linear probabilities of all s-boxes $\{s_1, s_2, \ldots\}$.

$$p_s = \max_i \max_{\Delta x \neq 0, \Delta y} \Pr_x[s_i(x) \oplus s_i(x \oplus \Delta x) = \Delta y]$$

$$q_s = \max_i \max_{\Gamma y \neq 0, \Gamma x} (2 \Pr_x[x \cdot \Gamma x = s_i(x) \cdot \Gamma y] - 1)^2$$

Theorem 2 Let $\mathcal{D}$ and $\mathcal{L}$ be the minimum numbers of total differential/linear active s-boxes. Then, the maximum differential/linear characteristic probabilities are bounded by $p_s^{\mathcal{D}}$ and $q_s^{\mathcal{L}}$, respectively.

With the above-mentioned techniques, we prove that Camellia offers immunity to these attacks by showing the upper bounds of maximum differential/linear characteristic probabilities, since Camellia is a Feistel cipher whose round function uses the S-P round function.

In the case of Camellia, the maximum differential/linear probabilities of the s-boxes are

$$p_s = q_s = 2^{-6}.$$

The branch number of the linear transformation (P-function) is 5, i.e.

$$\mathcal{B} = 5.$$

Letting p, q be the maximum differential/linear characteristic probabilities of Camellia reduced to 16-round without FL- and FL^{-1}-functions, respectively, we have

$$p \le p_s^{2(2\mathcal{B}+1)} = (2^{-6})^{22} = 2^{-132} \quad \text{and} \quad q \le q_s^{2(2\mathcal{B}+1)} = (2^{-6})^{22} = 2^{-132}$$

from Theorems 1 and 2. Both probabilities are below the security threshold of 128-bit block ciphers: 2^{-128}. It follows that there is no effective differential characteristic or linear characteristic for Camellia reduced to more than 15 rounds without FL- and FL^{-1}-functions. Since FL- and FL^{-1}-functions are linear for any fixed key, they do not make the average differential/linear probabilities of the cipher higher. Hence, it is proven that Camellia offers enough security against differential and linear attacks.

Note that the result above are based on Theorems 1 and 2. Both theorems deal with general cases of Feistel ciphers with SPN round function, so we expect that Camellia is actually more secure than shown by the result above. As supporting evidence, we counted the number of active s-boxes of Camellia with reduced rounds. The counting algorithm is similar to that described in [M99] except following three items.

- Prepare the table for the number of active s-boxes instead of transition probability table.
- Count the number of active s-boxes instead of computing transition probability.
- FL- and FL^{-1}-functions set all elements to the minimum number of active s-boxes in the table. This means that the algorithm gives consideration to existence of weak subkeys inserted to FL- and FL^{-1}-functions, since there may be some possibility of connecting every later differential/linear characteristic with the previous one with the highest probability, which is equivalent to the minimum number of active s-boxes.

As a result, we confirmed that 12-round Camellia with FL- and FL-functions has no differential/linear characteristic with probability higher than 2^{-128} (see Tables 11 and 12).

6.2 Truncated Differential Cryptanalysis

The attacks using truncated differentials were introduced by Knudsen [K95]. He defined them as differentials where only a part of the difference can be predicted. The notion of truncated differentials introduced by him is wide, but with a byte-oriented cipher it is natural to study bytewise differentials as truncated differentials [MT99].

# of rounds	1	2	3	4	5	6	7	8	9	10	11	12
Estimation based			2^{-12}	2^{-30}		2^{-42}		2^{-66}				2^{-96}
on Th. 1 and 2			(2)	(5)		(7)		(11)				(16)
Camellia	1	2^{-6}	2^{-12}	2^{-42}	2^{-54}	2^{-66}	2^{-72}	2^{-72}	2^{-78}	2^{-108}	2^{-120}	2^{-132}
	(0)	(1)	(2)	(7)	(9)	(11)	(12)	(12)	(13)	(18)	(20)	(22)
without FL/FL^{-1}-	1	2^{-6}	2^{-12}	2^{-36}	2^{-54}	2^{-66}	2^{-78}	2^{-90}	2^{-108}	2^{-126}	2^{-132}	
functions	(0)	(1)	(2)	(6)	(9)	(11)	(13)	(15)	(18)	(21)	(22)	

Note: The numbers in brackets are the number of active s-boxes.

Table 11: Upper bounds of differential characteristic probability of Camellia

# of rounds	1	2	3	4	5	6	7	8	9	10	11	12
Estimation based			2^{-12}	2^{-30}		2^{-42}		2^{-66}				2^{-96}
on Th. 1 and 2			(2)	(5)		(7)		(11)				(16)
Camellia	1	2^{-6}	2^{-12}	2^{-36}	2^{-54}	2^{-66}	2^{-72}	2^{-72}	2^{-78}	2^{-102}	2^{-120}	2^{-132}
	(0)	(1)	(2)	(6)	(9)	(11)	(12)	(12)	(13)	(17)	(20)	(22)
without FL/FL^{-1}-	1	2^{-6}	2^{-12}	2^{-36}	2^{-54}	2^{-66}	2^{-78}	2^{-84}	2^{-108}	2^{-120}	2^{-132}	
functions	(0)	(1)	(2)	(6)	(9)	(11)	(13)	(14)	(18)	(20)	(22)	

Note: The numbers in brackets are the number of active s-boxes.

Table 12: Upper bounds of linear characteristic probability of Camellia

The maximum differential probability is considered to provide the strict evaluation of security against differential cryptanalysis, but computing its value is impossible in general, since a differential is a set of all differential characteristics with the same input difference and the same output difference for a Markov cipher [LMM91]. On the other hand, a truncated differential can be regarded as a subset of the differential characteristics which are exploitable in cryptanalysis. For some ciphers, e.g., byte-oriented ciphers, the probability of truncated differential can be computed easily and correctly, and it gives a more strict evaluation than the maximum differential characteristic probability.

A truncated differential cryptanalysis of reduced-round variants of E2 was presented by Matsui and Tokita at FSE'99 [MT99]. The analysis was based on the "byte characteristic," where the values to the difference in a byte are distinguished between non-zero and zero. Matsui et al. found a 7-round byte characteristic, which leads to a possible attack on an 8-round variant of E2 without IT-Function (the initial transformation) and FT-Function (the final transformation). The best attack of E2 shown in [MSAK00] breaks an 8-round variant of E2 with either IT-Function or FT-Function using 2^{94} chosen plaintexts. In [MSAK00] Moriai et al. also show the attack which distinguishes a 7-round variant of E2 with IT- and FT-Functions from a random permutation using 2^{91} chosen plaintexts.

Camellia is a byte-oriented cipher similar to E2, and it is important to evaluate its security against truncated differential cryptanalysis. We searched for truncated differentials using an algorithm similar to the one described in [MT99, MSAK00]. The main difference of the round function between E2 and Camellia is the adoption of the 1-round SPN not the 2-round SPN, i.e. S-P-S. In the search for truncated differentials of E2, we used about 2^{-8} as the probability of difference cancellation in byte at the XOR of Feistel network. On the other hand, the round function of Camellia doesn't have the second s-boxes-layer and the difference cancellation in byte sometimes occurs with probability 1. Therefore, we also consider the above case. As a result, more than 10-round Camellia is indistinguishable from a random permutation both with/without FL-/FL^{-1}-function layers.

6.3 Truncated Linear Cryptanalysis

We introduce a new cryptanalysis called truncated linear cryptanalysis.

Due to the duality between differential and linear cryptanalysis, we can evaluate security against truncated linear cryptanalysis by using a similar algorithm to that above. To put it concretely, we can perform the search by replacing the matrix of P-function with the transposed matrix. As a result, more than 10-round Camellia is indistinguishable from a random permutation without FL-/FL^{-1}-function layers.

6.4 Cryptanalysis with Impossible Differential

The impossible differential means the differential which holds with probability 0, or the differential which never exists. Using such an impossible differential, it is possible to narrow down the candidates of the subkey. It is known that there is at least one 5-round impossible differential in any Feistel network with a bijective round function. Since Camellia has the Feistel network (with FL- and FL^{-1}-functions inserted between every 6 rounds) and the round function is bijective, Camellia has 5-round impossible differentials. We have not found impossible differentials with more than 5 rounds. Moreover, we expect FL- and FL^{-1}-functions make attacking Camellia using impossible differentials difficult, since the functions change differential paths depending on key values. In consequent, Camellia with full rounds will not be broken by cryptanalysis using impossible differentials.

6.5 Boomerang Attack

Boomerang attack [W99] requires 2 differentials. Let the probability of the differentials be p_Δ and p_∇. An boomerang attack that is superior than exhaustive key search requires

$$p_\Delta p_\nabla \geq 2^{-64}. \tag{6}$$

Using Table 11, there is no combination that satisfies Inequality (6) for Camellia without FL- and FL^{-1}-functions. The best boomerang probability for Camellia without FL- and FL^{-1}-functions reduced to 8-round is bounded by 2^{-66} that is obtained by $p_\Delta = 2^{-12}$ (3 rounds) and $p_\nabla = 2^{-54}$ (5 rounds). Since attackable rounds for Camellia without FL- and FL^{-1}-functions is bounded by much shorter than the specification of Camellia, 18 (for 128-bit keys), or 24 (for 192- or 256-bit keys), Camellia seems secure against a boomerang attack.

6.6 Higher Order Differential Attack

Higher order differential attack is generally applicable to ciphers that can be represented as Boolean polynomials of low degree. All intermediate bits in the encryption process can be represented as Boolean polynomials, i.e. polynomials $\mathrm{GF}(2)[x_1, x_2, \ldots, x_n]$ in the bits of the plaintext: $\{x_1, x_2, \ldots, x_n\}$. In the higher order differential attack described in [JK97, Theorem 1], if the intermediate bits are represented by Boolean polynomials of degree at least d, the $(d+1)$-th order differential of the Boolean polynomial becomes 0.

For high degrees of Boolean polynomials of the s-boxes of Camellia, the functions affine equivalent to the inversion function in $\mathrm{GF}(2^8)$ are adopted as the s-boxes. It is known that the degree of the Boolean polynomial of every output bit of the inversion function in $\mathrm{GF}(2^8)$ is 7, but the degree for the s-boxes of Camellia is not trivial, since affine functions are added at the input and output. We confirmed that the degree of the Boolean polynomial of every output bit of the s-boxes is 7 by finding Boolean polynomial for every output bit of the s-boxes. In Camellia, it is expected that the degree of an intermediate bit in the encryption process increases as the data pass through many s-boxes. For example, the degree becomes $7^3 > 128$ after passing through three s-boxes. Therefore, we expect that higher order differential attacks fail against Camellia with full rounds.

Hori and Kaneko showed that 5-round E2 without IT- and FT-functions can be broken by using the first order differential [HK98]. However, we consider their attack to be similar to the traditional differential attack rather than the above-mentioned higher order differential attack. That is, the main point of their technique is to find the pairs of plaintext and ciphertext which yield a nontrivial constant at some intermediate round. In other words, this means that an attacker traces differential paths. In consequence, we believe that the immunity of Camellia to Hori et al.'s attack is comparable to the immunity to traditional differential attack described in Section 6.1.

6.7 Interpolation Attack and Linear Sum Attack

The interpolation attack proposed in [JK97] is typically applicable to attacking ciphers that use simple algebraic functions. The principle of interpolation attack is that, roughly speaking, if the ciphertext is represented as a polynomial or rational expression of the plaintext whose number of unknown coefficients is N, the polynomial or rational expression can be constructed using N pairs of plaintexts and ciphertexts. Once the attacker constructs the polynomial or rational expression, he can encrypt any plaintext into the corresponding ciphertext or decrypt any ciphertext into the corresponding plaintext for the key without knowing the key. Since N determines the complexity and the number of pairs required for the attack, it is important to make N as large as possible. If N is so large that it is impractical for the attackers to gather N plaintext-ciphertext pairs, the cipher is secure against interpolation attack.

Linear sum attack [A00] is a generalization of the interpolation attack [JK97]. A practical algorithm that evaluates the security against the linear sum attack was proposed in [A00]. We searched for linear relations between any plaintext byte and any ciphertext byte over $\mathrm{GF}(2^8)$ using the algorithm. Table 13 summarizes the results.

Table 13 shows that Camellia is secure against linear sum attack including interpolation attack [JK97]. It also implies that Camellia is secure against SQUARE attack [DKR97] followed

Table 13: Smallest number of unknown coefficients for 128-, 192-, and 256-bit keys

whitening×1 + round×r ($r < 4$)	1
whitening×1 + round×4	255
More rounds	256

by [A00, Theorem 3].

6.8 Weak Keys

Generally speaking, the resistance of block ciphers to cryptanalysis depends on the key. We call the keys which provide lower security than the average "*weak keys*". Various weak keys are known for several cryptanalysis including differential cryptanalysis, linear cryptanalysis, and higher order differential cryptanalysis.

In FL- and FL^{-1}-functions of Camellia, logical AND and OR operations are used as keyed operation. When these irregular functions are inserted, the differential/linear paths as well as their associated probabilities vary from one key to another. On one hand, this could make the analysis more difficult since the differential/linear paths are key dependent. On the other hand, one might be able to take advantage of such key dependencies and try to deduce the round keys used in the irregular rounds by detecting which paths are valid. However, such a technique has not been found for Camellia, and weak keys in this respect are not found.

6.9 Equivalent Keys

Since the set of subkeys generated by the key schedule contain the original secret key, there is no equivalent set of subkeys generated from distinct secret keys. Therefore, we expect that there are no distinct secret keys both of which encrypt each of many plaintexts into the same ciphertext.

6.10 Slide Attack

In [BW99, BW00] the slide attacks were introduced, based on earlier work in [B94, K93]. In particular it was shown that iterated ciphers with identical round functions, that is, equal structures and equal subkeys in the round functions, are susceptible to slide attacks.

In Camellia, FL- and FL^{-1}-functions are "inserted" between every 6 rounds of a Feistel network to provide non-regularity across rounds. Moreover, from the viewpoint of the key schedule, slide attacks seems to be very unlikely to succeed (See Section 6.11).

6.11 Related-key Attack

We are convinced that the key schedule of Camellia makes related-key attacks [B94, KSW96] very difficult. In these attacks, an attacker must be able to get encryptions using several related keys. If the relation between, say, two keys, is known then if the corresponding relations between the subkeys can be predetermined, it might become possible to predict how the keys would encrypt a pair of different plaintexts. In the case of Camellia, the subkeys depend on K_A and K_B, which are the results of encryption of a secret key, and if an attacker wants to change the secret key, he can't get the desired K_A and K_B, and vice versa. These subkey relations will be very hard to control and predict.

6.12 Statistical Tests

Most of statistical characteristics depends on the differential attack and other cryptanalytic attacks. For example, it is frequently discussed how many ciphertext bits are complemented when one plaintext bit is complemented. According to the definition and the property of the differential distribution table, the resistance to differential attacks implies that the number of complemented bits is about a half. Of course, we may find a statistical weakness, if we have enough computational resource. However, none in the world has an efficient resource to compute such a statistical measure for a 128-bit block cipher.

Note that the following. It is frequently tested for a round function, because of the limited computational resource. However, we think that it is not significant, because we can construct a cipher that does not show good statistical properties for the round function but shows good statistical properties for a cipher and we can also construct a cipher that shows good statistical properties for the round function but does not show good statistical properties for a cipher.

6.13 Implementation Attacks

It is well known that a poor implementation can leak information by timing attacks [K96] or power analysis attacks [KJJ99]. Using the classification proposed in [DR99], Camellia is in the group of "favorable" algorithms in respect of the ability to be implemented in a resistant way to these attacks, since Camellia uses only logical operations, table-lookups and fixed rotations.

On the other hand, Chari et al. [CJRR99] claims that all AES candidates are susceptible to power analysis attacks. Daemen and Rijmen [DR99] suggested software countermeasures against Differential Power Analysis (DPA) attacks, which include the insertion of dummy code, power consumption randomization, and the balancing of data. However, Chari et al. suggested that ad hoc countermeasures will not suffice. They proposed that a better approach: to split all intermediate data results using a secret sharing scheme, thereby forcing attackers to analyze joint distribution functions on multiple points in the power signal. Goubin et al. [GP99] proposed a similar strategy, called the duplication method. Messerges [M00] also proposed a similar countermeasure, which is to mask all intermediate data results.

For example, Messerges' random masking technique [M00] is a good countermeasure of Camellia against power analysis attacks. Camellia is well suited algorithm for the random masking. There is no extra overhead for arithmetic operations such as addition and multiplication, and the lookup table is small enough to be masked and stored in RAM.

6.14 Brute Force Attacks

Most brute force attacks are applicable to any block cipher, and the corresponding complexity depends on only the block size or key size*, regardless of its design. Camellia has a block size of 128-bit and allows for the three key sizes of 128-, 192-, and 256-bit. In the discussions below, k denotes the key size in bits.

Exhaustive key search. In exhaustive key search, if an attacker gets one pair of plaintext and ciphertext encrypted in ECB mode, he can find the correct key by encrypting the plaintext with all 2^k possible keys.

A weakness in the key scheduling of the cipher can help improve the efficiency of exhaustive key search attack [K94], but we have not found such a weakness in Camellia. The complexity of the exhaustive key search is estimated to be about 2^{k-1} encryptions on average. Thus, the required complexity for exhaustive key search is 2^{127}, 2^{191}, and 2^{255} encryptions for Camellia with 128-, 192-, and 256-bit keys, respectively. Therefore, Camellia's security against exhaustive key search is adequate.

Time-memory trade-off attack. There are some phrases that are often used in plaintexts. If an attacker encrypts a plaintext block contained in such a phrase using 2^k keys and store them in space for 2^k ciphertexts, then after he gets the corresponding ciphertext, he only has to look it up to find the corresponding key. This attack is called table attack. In this attack, after 2^k encryption is done, the attack complexity is much smaller than is true for exhaustive key search.

Time-memory trade-off attack [H80, KM96] can drastically reduce both time complexity on intercepted ciphertexts of exhaustive key search and space complexity of table attack.

However, both attacks require precomputation equivalent to the time complexity of exhaustive key search. The key sizes supported by Camellia are long enough for security against exhaustive key search by today's technology.

Dictionary attack. In dictionary attack, an attacker collects plaintext-ciphertext pairs under the same key and put them in a "dictionary". When the attacker can see only a ciphertext encrypted by the key, he can check if it is in the dictionary. If it is, he has already the plaintext. Since the block size of Camellia is 128 bits, dictionary attack would require the space for 2^{128} different plaintext blocks to allow the attackers to encrypt or decrypt arbitrary messages under an unknown key. The success probability depends on the space for the dictionary, and as the block size is larger, the required space to achieve the same success probability increases exponentially. The 128-bit block cipher Camellia has enough security against this attack.

Matching ciphertext attack. In matching ciphertext attack [K98, Theorem 2], when about the square root of all ciphertexts are available identical ciphertext blocks can be expected with probability more than $\frac{1}{2}$ by the "birthday paradox" for some modes of operations such as ECB,

*Strictly speaking, the computation time required for the attack depends on the performance of the block cipher. However, the performance only affects the encryption time and only changes the time complexity by negligible factor.

CBC, and CFB modes. Then, valuable information about the plaintexts can be derived. Note that this attack is independent of the key size. Since the block size of Camellia is 128 bits, the threat to this attack is small, if encryption of as many as 2^{64} blocks under the same key is not performed.

7 Strengths and Advantages

Camellia is a block cipher of theory and practice. It was carefully designed to withstand all known cryptanalytic attacks and even to have a sufficiently large security leeway for use of the next 10-20 years. It was also designed to have suitability for both software and hardware implementations and to cover all possible encryption applications that range from low-cost smart cards to high-speed network systems.

Camellia was jointly developed by experts of research and developments of cryptography. In real applications, an encryption function is often an only small component in the entire system, even though it is critically important. The target processor or hardware process might be overwhelmingly fast by adopting the latest technology. However the system architecture could give only small time and memory slots to the encryption function. This is why heavy resource constraints are still a nightmare for developers of encryption systems.

Do not trust performance claims shown in a catalog too much. Encryption is just one of many. Keeping these lessons in mind, the designers of Camellia concentrated on applicability to low cost applications, particularly in hardware, as well as achievement of high encryption performance in software. DES and Triple DES have very small hardware (within 10Kgates), which perfectly meet current market requirements in wireless cards, for instance, where low power consumption is a mandatory condition.

As far as the designers know, Camellia is an only 128-bit block cipher that can be an alternative for Triple DES from hardware viewpoint: a small number of gate counts and low power consumption. Also Camellia establishes high software performance in modern PC processors in the same level as that of AES finalists. We hence believe that Camellia is worthy of being standardized a standard as a new generation of a block cipher.

8 Conclusion

We have presented Camellia, the rationale behind its design, its suitability for both software and hardware implementation, and the results of our cryptanalyses.

Camellia concentrated on applicability to low cost applications, particularly in hardware, as well as achievement of high encryption performance in software. The performances shown in this paper leave room for further optimizations. The updated performance results will be posted on the Camellia home page: `http://info.isl.ntt.co.jp/camellia/`.

We have analyzed Camellia and found no important weakness. Also, there are no hidden weakness inserted by the designers. The cipher has a conservative design and any practical attacks against Camellia would require a major breakthrough in the area of cryptanalysis.

We think that Camellia is a very strong and efficient block cipher, which matches market requirements.

References

[A00] K. Aoki. Practical Evaluation of Security against Generalized Interpolation Attack. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*, Vol. E83-A, No. 1, pp. 33–38, 2000. (A preliminary version was presented at SAC'99).

[ABK98] R. Anderson, E. Biham, and L. Knudsen. Serpent: A Flexible Block Cipher With Maximum Assurance. In *The First AES Candidate Conference*, 1998.

[AU00] K. Aoki and H. Ueda. Optimized Software Implementations of E2. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*, Vol. E83-A, No. 1, pp. 101–105, 2000. (The full paper is available on `http://info.isl.ntt.co.% linebreak[3]jp/e2/RelDocs/`).

[B94] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. *Journal of Cryptology*, Vol. 7, No. 4, pp. 229–246, 1994. (The extended abstract was appeared at EUROCRYPT'93).

[BS93] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, Berlin, Heidelberg, New York, 1993.

[BW99] A. Biryukov and D. Wagner. Slide Attacks. In L. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 245–259, Berlin, Heidelberg, New York, 1999. Springer-Verlag.

[BW00] A. Biryukov and D. Wagner. Advanced Slide Attacks. In S. Vaudenay, editor, *Advances in Cryptology — EUROCRYPT2000*, Volume 1807 of *Lecture Notes in Computer Science*, pp. 589–606, Berlin, Heidelberg, New York, 2000. Springer-Verlag.

[C98] Compaq Computer Corporation. *Alpha Architecture Handbook (Version 4)*, 1998. (You can download the manual from Compaq's technical documentation library: `http://www.support.compaq.com/alpha-tools/documentation/current/chip-docs.html`).

[CJRR99] S. Chari, C. Jutla, J. R. Rao, and P. Rohatgi. A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. In *Second Advanced Encryption Standard Candidate Conference*, pp. 133–147, Hotel Quirinale, Rome, Italy, 1999. Information Technology Laboratory, National Institute of Standards and Technology.

[CV95] F. Chabaud and S. Vaudenay. Links Between Differential and Linear Cryptanalysis. In A. D. Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, Volume 950 of *Lecture Notes in Computer Science*, pp. 356–365. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[DKR97] J. Daemen, L. R. Knudsen, and V. Rijmen. The Block Cipher SQUARE. In E. Biham, editor, *Fast Software Encryption — 4th International Workshop, FSE'97*, Volume 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Berlin, Heidelberg, New York, 1997. Springer-Verlag.

[DR98] J. Daemen and V. Rijmen. *AES Proposal: Rijndael*, 1998. (`http://www.esat.kuleuven.ac.be/~rijmen/rijndael/`).

[DR99] J. Daemen and V. Rijmen. Resistance Against Implementation Attacks. A Comparative Study of the AES Proposals. In *The Second AES Candidate Conference*, 1999.

[F00] A. Fog. *How to optimize for the Pentium microprocessors*, 2000. (`http://www.agner.org/assem/`).

[GP99] L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In *Cryptographic Hardware and Embedded Systems – First International Workshop, CHES'99*, Volume 1717 of *Lecture Notes in Computer Science*, pp. 158–172, Berlin, Heidelberg, New York, 1999. Springer-Verlag.

[H80] M. Hellman. A Cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, Vol. IT-26, No. 4, pp. 401–406, 1980.

[HK98] Y. Hori and T. Kaneko. A study of E2 by higher order differential attack. Technical Report ISEC98-38, The Institute of Electronics, Information and Communication Engineers, 1998. (in Japanese).

[I99] Intel Corporation. *Intel Architecture Software Developer's Manual (Volume 2: Instruction Set Reference)*, 1999. (You can download the manual from Intel's developer site: `http://developer.intel.com/`).

[JK97] T. Jakobsen and L. R. Knudsen. The Interpolation Attack on Block Cipher. In E. Biham, editor, *Fast Software Encryption — 4th International Workshop, FSE'97*, Volume 1267 of *Lecture Notes in Computer Science*, pp. 28–40, Berlin, Heidelberg, New York, 1997. Springer-Verlag.

[K93] L. R. Knudsen. Cryptanalysis of LOKI91. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology — AUSCRYPT'92*, Volume 718 of *Lecture Notes in Computer Science*, pp. 196–208. Springer-Verlag, Berlin, Heidelberg, New York, 1993.

[K94] L. R. Knudsen. Practically secure Feistel ciphers. In R. Anderson, editor, *Fast Software Encryption 1993 — Cambridge Security Workshop (FSE1)*, Volume 809 of *Lecture Notes in Computer Science*, pp. 211–221, Berlin, Heidelberg, New York, 1994. Springer-Verlag.

[K95] L. R. Knudsen. Truncated and Higher Order Differentials. In B. Preneel, editor, *Fast Software Encryption — Second International Workshop*, Volume 1008 of *Lecture Notes in Computer Science*, pp. 196–211. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[K96] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO'96*, Volume 1109 of *Lecture Notes in Computer Science*, pp. 104–113. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[K98] L. R. Knudsen. Block Ciphers — A Survey. In B. Preneel and V. Rijmen, editors, *State of the Art in Applied Cryptography*, Volume 1528 of *Lecture Notes in Computer Science*, pp. 18–48, Berlin, Heidelberg, New York, 1998. Springer-Verlag.

[K00] M. Kanda. Practical Security Evaluation against Differential and Linear Attacks for Feistel Ciphers with SPN Round Function. In *SAC2000, Seventh Annual Workshop on Selected Areas in Cryptography, 14-15 August 2000, Workshop Record*, 2000.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology — CRYPTO'99*, Volume 1666 of *Lecture Notes in Computer Science*, pp. 388–397. Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[KM96] K. Kusuda and T. Matsumoto. Optimization of Time-Memory Trade-Off Cryptanalysis and Its Application to DES, FEAL-32, and Skipjack. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*, Vol. E79-A, No. 1, pp. 35–48, 1996.

[KMA+98] M. Kanda, S. Moriai, K. Aoki, H. Ueda, M. Ohkubo, Y. Takashima, K. Ohta, and T. Matsumoto. A New 128-bit Block Cipher ***E2***. Technical Report ISEC98-12, The Institute of Electronics, Information and Communication Engineers, 1998. (in Japanese).

[KSW96] J. Kelsey, B. Schneier, and D. Wagner. Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO'96*, Volume 1109 of *Lecture Notes in Computer Science*, pp. 237–251. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[KTM+99] M. Kanda, Y. Takashima, T. Matsumoto, K. Aoki, and K. Ohta. A Strategy for Constructing Fast Round Functions with Practical Security against Differential and Linear Cryptanalysis. In S. Tavares and H. Meijer, editors, *Selected Areas in Cryptography — 5th Annual International Workshop, SAC'98*, Volume 1556 of *Lecture Notes in Computer Science*, pp. 264–279, Berlin, Heidelberg, New York, 1999. Springer-Verlag.

[LMM91] X. Lai, J. L. Massey, and S. Murphy. Markov Ciphers and Differential Cryptanalysis. In D. W. Davies, editor, *Advances in Cryptology — EUROCRYPT'91*, Volume 547 of *Lecture Notes in Computer Science*, pp. 17–38. Springer-Verlag, Berlin, Heidelberg, New York, 1991.

[M94] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In T. Helleseth, editor, *Advances in Cryptology — EUROCRYPT'93*, Volume 765 of *Lecture Notes in Computer Science*, pp. 386–397. Springer-Verlag, Berlin, Heidelberg, New York, 1994. (A preliminary version written in Japanese was presented at SCIS93-3C).

[M95] M. Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In A. D. Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, Volume 950 of *Lecture Notes in Computer Science*, pp. 366–375. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[M97] M. Matsui. New Block Encryption Algorithm MISTY. In E. Biham, editor, *Fast Software Encryption — 4th International Workshop, FSE'97*, Volume 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Berlin, Heidelberg, New York, 1997. Springer-Verlag. (A preliminary version written in Japanese was presented at ISEC96-11).

[M99] M. Matsui. Differential Path Search of the Block Cipher E2. Technical Report ISEC99-19, The Institute of Electronics, Information and Communication Engineers, 1999. (in Japanese).

[M00] T. S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In *Fast Software Encryption Workshop 2000*, 2000.

[MIYY88] M. Matsui, T. Inoue, A. Yamagishi, and H. Yoshida. A note on calculation circuits over GF(2^{2n}). Technical Report IT88-14, The Institute of Electronics, Information and Communication Engineers, 1988. (in Japanese).

[MSAK00] S. Moriai, M. Sugita, K. Aoki, and M. Kanda. Security of E2 against Truncated Differential Cryptanalysis. In H. Heys and C. Adams, editors, *Selected Areas in Cryptography — 6th Annual International Workshop, SAC'99*, Volume 1758 of *Lecture Notes in Computer Science*, pp. 106–117, Berlin, Heidelberg, New York, 2000. Springer-Verlag.

[MT99] M. Matsui and T. Tokita. Cryptanalysis of a Reduced Version of the Block Cipher E2. In L. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 71–80, Berlin, Heidelberg, New York, 1999. Springer-Verlag. (Japanese version was presented at SCIS99.).

[RDP+96] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win. The Cipher SHARK. In D. Gollmann, editor, *Fast Software Encryption — Third International Workshop*, Volume 1039 of *Lecture Notes in Computer Science*, pp. 99–111. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[W99] D. Wagner. The Boomerang Attack. In L. R. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 156–170, Berlin, Heidelberg, New York, 1999. Springer-Verlag.