Unified Hardware Architecture for 128-Bit Block Ciphers AES and Camellia

Akashi Satoh and Sumio Morioka

IBM Research, Tokyo Research Laboratory, IBM Japan Ltd., 1623-14,
Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan
{akashi, e02716}@jp.ibm.com

Abstract. We proposed unified hardware architecture for the two 128-bit block ciphers AES and Camellia, and evaluated its performance using a 0.13-μm CMOS standard cell library. S-Boxes are the biggest hardware components in block ciphers, and some times they consume more than half of the design area. The S-Boxes in AES and Camellia are the combination of affine transformations and multiplicative inversions on a Galois fields. The size of the fields is same, but their structures are different. Therefore we converted the basis between the fields by using isomorphism transformations, and shared the inverter between AES and Camellia. The affine transformations were also merged by factoring common terms. In addition to the S-Box sharing, many other components such as permutation layers and key whiting are also merged. As a result, a compact hardware of 14.9K gates with throughputs of 469 Mbps for AES and of 661 Mbps for Camellia was achieved. The hardware synthesized with speed optimization obtained throughputs of 794 Mbps and 1.12 Gbps for each algorithm with 24.4K gates.

1 Introduction

The AES (Advanced Encryption Standard) project [1] for the new US federal standard block cipher algorithm replacing DES (Data Encryption Standard) [2] was started in 1997, and Rijndael [3] was standardized as FIPS PUB 197 [4] in 2001. After then, many block ciphers that have the AES compatible interface, supporting a 128-bit data block and 128/192/256-bit keys, have been proposed for other organizations [5-9]. Camellia [9, 10] was developed by NTT (Nippon Telegraph and Telephone Corp.) and Mitsubishi Electric is the one of them. In February 2003, the NESSIE (New European Schemes for Signatures, Integrity and Encryption) project [6] chose it as a recommended algorithm and decided to input it to ISO and IETF.

Camellia has good performance in both software and hardware implementations, and a promising alternative of AES. However, supporting multiple algorithms simply multiplies the hardware costs, while it is not a big issue in software implementation. The algorithm structures of AES and Camellia are completely different; the former uses SPN (Substitution Permutation Network) and the latter does a Feistel network.

C.D. Walter et al. (Eds.): CHES 2003, LNCS 2779, pp. 304–318, 2003.

However, they use very similar basic components, such as multiplicative inversion on Galois field GF(2^8) and affine transformation on GF(2). Therefore, it is possible to reduce hardware cost by reusing these common components between two algorithms.

In this paper, we first propose an unified S-Box architecture sharing a GF inverter and merging affine transformations between AES and Camellia, and factoring techniques for the permutation layers are also shown. Then entire data path architecture including a key scheduler is described. Finally ASIC hardware performance in size and speed of the proposed architecture is compared with the discrete implementations of the two algorithms.

2 S-Box Structures

2.1 AES S-Box

The AES S-Boxes are combinations of a multiplicative inversion on GF(2^8) and affine transformations. The irreducible polynomial of Equation (1) is used to define the field.

$$m(x) = x^8 + x^4 + x^3 + x + 1 \tag{1}$$

Following the inversion, an affine transformation A defined by Equation (2) is executed in the S-Box for encryption. In the equation, an operator $\oplus$ means XOR (Exclusive-OR). In the decryption S-Box, the multiplicative inversion follows the inverse affine transformation A^{-1} defined by Equation (3) that is not shown in the AES specification [4].

$$A: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{2}$$

$$A^{-1}: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \oplus 1 \\ a_1 \oplus 1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \oplus 1 \\ a_6 \oplus 1 \\ a_7 \end{pmatrix} \tag{3}$$

A 128-bit nonlinear function that contains 16 encryption S-Boxes is called SubBytes, and that contains 16 decryption S-Boxes is called InvSubBytes. Example hardware implementations of each S-Box is shown in Fig. 1. An XOR operation with '1' in Equations (2) and (3) equals to a NOT operation. XOR followed by NOT and XOR

following NOT can be replaced by XNORs (Exclusive NORs). Circuit cost (transistor counts and operation delay) is basically same between XOR and XNOR gates, so hardware performance can be improved by using XNOR instead of XOR with NOT. However the circuits shown in Fig. 1 do not use XNOR and are straightforward implementations of Equations (2) and (3), because they are much suitable for examples.

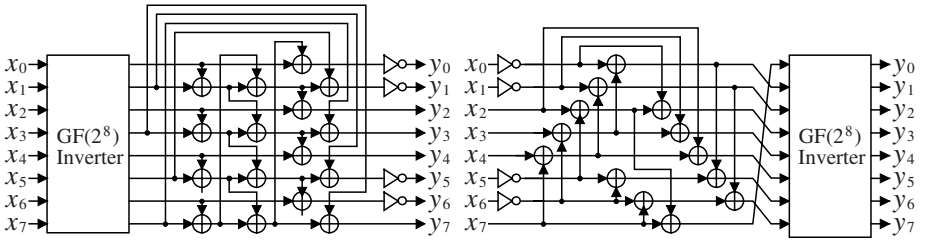

Fig. 1. Straightforward hardware Implementation of AES S-Boxes

2.2 Camellia S-Box

S-Boxes of Camellia use multiplicative inversion on a Galois field and affine transformations in similar fashion of AES. The Camellia description [9, 10] only shows a truth table of the inversion, but its field structure is not clearly described. So we investigated a lot of fields, and found that the field extended by using irreducible polynomials in Equation (4) satisfies the truth table.

$$\begin{cases} \mathrm{GF}(2^4): \quad g_0(x) = x^4 + x + 1 \\ \mathrm{GF}((2^4)^2): g_1(x) = x^2 + x + \omega \quad (\omega = \{1001\}) \end{cases} \tag{4}$$

Two sets of four S-Boxes (S1~S4) are used every iteration round. In the S-Box S1, affine transformations F and H defined by Equations (5) and (6) are executed before and after the multiplicative inversion respectively.

$$F: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_0 \oplus 1 \\ a_1 \oplus 1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \oplus 1 \\ a_6 \\ a_7 \oplus 1 \end{pmatrix} \tag{5}$$

$$H: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{6}$$

The S-Boxes S2 and S3 are defined as S1 followed by 1-bit right rotation (b_7, b_0, b_1, b_2, b_3, b_4, b_5, b_6) and 1-bit left rotation (b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_0) respectively. The input bits of S1 are rotated as (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) for the S-Box S4. Fig. 2 shows an example circuit of the Camellia S-Boxes. Also here, it is possible to combine XOR with NOT into XNOR.

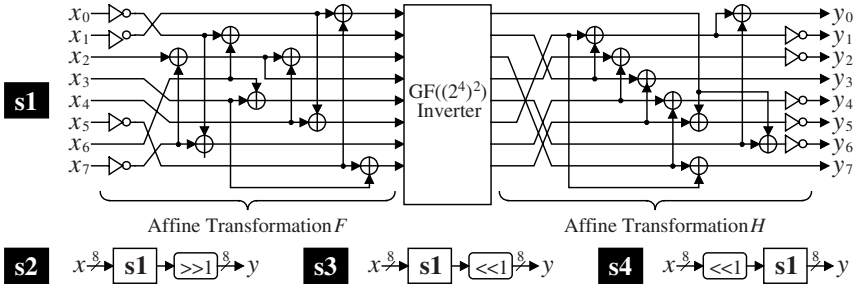

Fig. 2. Camellia S-Boxes

3 Unified S-Box

3.1 Construct Unified S-Box

In this section, we propose the shared S-Box architecture where a multiplicative inverter is reused and affine transformations are merged between SubBytes, InvSubBytes and S1~S4. Fig. 3 shows the process of S-Box sharing. The right arrows in the figure are all 8-bit data buses. In Fig. 3 (1), two S-Boxes SubBytes and InvSubBytes are independently implemented, and have the same GF(2^8) inverter. In (2), the inverter is shared between the S-Boxes by switching affine transformations A and A^{-1} using 2:1 selectors. We also want to share the inverter with Camellia, but AES and Camellia use different Galois field for their S-Boxes. However, all fields who have same size are isomorphic, so we map all elements on the AES's field to the Camellia's composite field, and use the GF($(2^4)^2$) inverter for all S-Boxes. It is possible to use the AES's GF(2^8) inverter in the Camellia S-Boxes, but the GF(2^8) inverter generated from a look-up table is much bigger than the GF($(2^4)^2$) inverter where sub-field arithmetic for compact implementation can be applied. The structure of the GF($(2^4)^2$) inverter is

detailed in Fig. 4, where the width of all data buses is 4 bits. The box [x^{-1}] shows a GF(2^4) inverter, and is designed as SOP (Sum of Products) logic in our implementations.

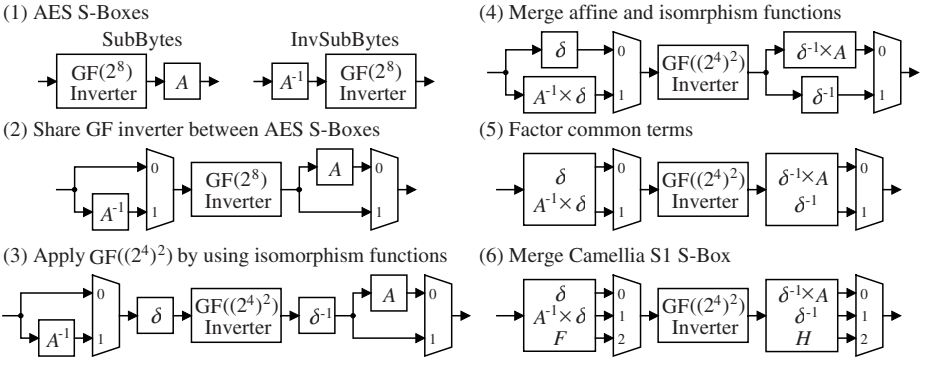

Fig. 3. Unified S-Box architecture

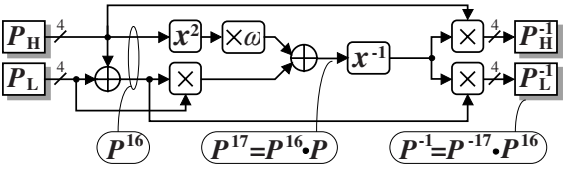

Fig. 4. GF($(2^4)^2$) inverter

δ and δ^{-1} in Equations (7) and (8) are isomorphism functions from GF(2^8) to GF($(2^4)^2$) and from GF($(2^4)^2$) to GF(2^8) respectively. The functions are defined as 8×8 XOR matrices as same as the affine transformations used in the S-Boxes. In Fig. 3(3), the isomorphism functions δ and δ^{-1} placed before and after the GF($(2^4)^2$) inverter expand the critical path. In order to shorten the path, the isomorphism functions δ and δ^{-1} are combined with affine transformations A and A^{-1} in Fig. 3(4). The combined functions $A^{-1}\times\delta$ and $\delta^{-1}\times A$ defined by Equations (9) and (10) require 43 2-input XOR gates, while 48 gates are used for A and A^{-1}. Therefore, circuit size is slightly reduced.

By comparing the matrices between Equations (7) and (9), and between (8) and (10), many common terms (1s in same columns) can be found. Therefore, hardware size can be reduced by sharing XOR gates corresponding to these common terms. However, the half of input bits of Equation (9), (a_1, a_2, a_6, a_7), are XORed with '1', and thus these values are reversed before the matrix operation. Therefore the common terms for these bits cannot be shared between Equations (7) and (9). In order to share these bits, replace these reverse operations on the input bits with those on the output bits as shown in Equation (11). This is possible because these matrix operations are all linear functions. The AES S-Box circuit after merging the matrices becomes Fig. 3(5).

$$\delta: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \tag{7}$$

$$\delta^{-1}: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \tag{8}$$

$$\begin{matrix} A^{-1} \\ \times\delta \end{matrix}: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \oplus 1 \\ a_2 \oplus 1 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \oplus 1 \\ a_7 \oplus 1 \end{pmatrix} \tag{9}$$

$$\begin{matrix} \delta^{-1} \\ \times A \end{matrix}: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{10}$$

$$\begin{matrix} A^{-1} \\ \times\delta \end{matrix}: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{11}$$

Finally, we also merged the Camellia affine transformations F and H according to the same manner, and obtained the shared S-Box shown in Fig. 3(6). Before merging F defined by Equation (5), here we also transform it to Equation (12). In the Camellia

S-Boxes S1~S3, only the order of the output bits is different. Therefore, the circuit shown in Fig. 3(6) can be used for all of them by only twisting the output wires. On the other hand, the input bits are twisted in the S4 S-Box. Therefore, we use the affine transformation F ' defined by Equation (13) instead of F, where the columns of the matrix is rotated to the right by one bit

$$F: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \tag{12}$$

$$F': \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \tag{13}$$

3.2 Hardware Performance of Unified S-Boxes

In this section, hardware performance of the unified S-Box is compared with the AES and Camellia S-Boxes that are independently implemented.

Table 1 shows the number of XOR gates required for each matrix operation described in the previous section. Common terms are not shared between the matrices for the numbers in "original matrices", and are shared for the number in "sharing common terms." While the original matrices require 102 XOR gates in total, the number is reduced by more than 40% for the shared S-Boxes (60 XORs or 56 XORs).

The S-Box performances in size and speed are shown in Table 2, where a 0.13-μm CMOS standard cell library is used. One gate is equivalent to 2-input NAND gate, and the speed is estimated under the worst case conditions. The GF($(2^4)^2$) inverter shown in the Fig. 4 is used in all S-Boxes. Two discrete S-Boxes (SubBytes and InvSubBytes) shown in Fig. 3(1), and one unified S-Box (SubBytes + InvSubBytes) in Fig 3(5) are implemented for AES. The performances between four Camellia S-Boxes are all same, and those between three shared S-Boxes (AES+S1~S3) are also same. When number of merged S-Boxes is increased (SubBytes with InvSubBytes, then with S1~S4), critical path becomes longer, because number of selectors is increased. The shared S-Box uses 411~414 gates that is almost half of 816 (=280+280+256) gates required for discrete implementation of two AES S-Boxes and one Camellia S-Box. In the actual use, a 3:1 selector is additionally needed for discrete implementation to switch three S-Boxes.

Table 1. Numbers of XOR gates required for each matrix operation

Original matrices							Sharing common terms	
δ	δ^{-1}	$A^{-1}\times\delta$	$\delta^{-1}\times A$	F	H	Total	AES+ S1~S3	AES+ S4
20	21	22	21	9	9	102	60	56

Table 2. ASIC Performance of each S-Box circuit

	S-Box type	Gate counts	Delay (ns)
AES	SubBytes	280	3.65
	InvSubBytes	280	3.56
	Merged	349	3.99
Camellia	S1~S4	256	3.45
AES+Canellia	AES + S1~S3	411	4.29
	AES + S4	414	4.65

(0.13-μm CMOS, 1gate = 2-input NAND, worst condition)

4 Unified Permutation Layer

AES uses permutation layers MixColumns and InvMixColumns in encryption and decryption respectively. MixColumns and InvMixColumns are inverse functions each other, and each function is defined as four 4-byte (4×4×8 bits = 128 bits) matrix operations. On the other hand, Camellia has only one permutation layer called P-function that is single 8-byte (8×8 bits = 64 bits) matrix operation. In order to merge these functions, compose two 8-byte matrices by gathering two 4-byte MixColumns and two 4-byte InvMixColimns respectively, and factorize them into a few 8-byte matrices as shown in Equations (14) and (15). Multiplications with the constant valued {8, 4, 3, 2, 1} in the matrices are defined over modulo $m(x)$ of Equation (1). By comparing Equations (14) for MixColumns and (15) for InvMixColumns, it is found that MixColumns is completely included in InvMixColumns [11]. Equation (16) is the matrix representation of P-function whose elements are '0' and '1', and thus modular arithmetic is not required. By breaking P-function into two matrices, many common terms with MixColumns are found. After the factorization and common term sharing, the permutation functions are represented as Equations (17)~(20). The basic structure of shared permutation circuit is shown in Fig. 5.

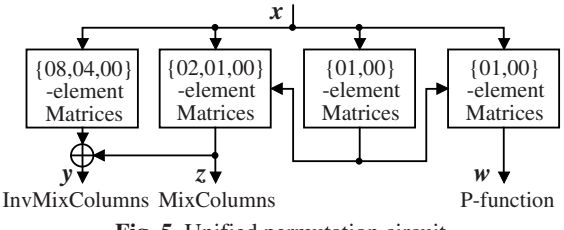

Fig. 5. Unified permutation circuit

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} = \begin{pmatrix} 2\ 3\ 1\ 1 & & 0 & \\ 1\ 2\ 3\ 1 & & & \\ 1\ 1\ 2\ 3 & & & \\ 3\ 1\ 1\ 2 & & & \\ & 2\ 3\ 1\ 1 & & \\ 0 & 1\ 2\ 3\ 1 & & \\ & 1\ 1\ 2\ 3 & & \\ & 3\ 1\ 1\ 2 & & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 2\ 2\ 0\ 0 & & 0 & \\ 0\ 2\ 2\ 0 & & & \\ 0\ 0\ 2\ 2 & & & \\ 2\ 0\ 0\ 2 & & & \\ & 2\ 2\ 0\ 0 & & \\ 0 & 0\ 2\ 2\ 0 & & \\ & 0\ 0\ 2\ 2 & & \\ & 2\ 0\ 0\ 2 & & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 0\ 1\ 1\ 1 & & 0 & \\ 1\ 0\ 1\ 1 & & & \\ 1\ 1\ 0\ 1 & & & \\ 1\ 1\ 1\ 0 & & & \\ & 0\ 1\ 1\ 1 & & \\ 0 & 1\ 0\ 1\ 1 & & \\ & 1\ 1\ 0\ 1 & & \\ & 1\ 1\ 1\ 0 & & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \tag{14}$$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} \mathrm{E}\ \mathrm{B}\ \mathrm{D}\ 9 & & 0 & \\ 9\ \mathrm{E}\ \mathrm{B}\ \mathrm{D} & & & \\ \mathrm{D}\ 9\ \mathrm{E}\ \mathrm{B} & & & \\ \mathrm{B}\ \mathrm{D}\ 9\ \mathrm{E} & & & \\ & \mathrm{E}\ \mathrm{B}\ \mathrm{D}\ 9 & & \\ 0 & 9\ \mathrm{E}\ \mathrm{B}\ \mathrm{D} & & \\ & \mathrm{D}\ 9\ \mathrm{E}\ \mathrm{B} & & \\ & \mathrm{B}\ \mathrm{D}\ 9\ \mathrm{E} & & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \tag{15}$$

$$= \begin{pmatrix} 8\ 8\ 8\ 8 & & 0 & \\ 8\ 8\ 8\ 8 & & & \\ 8\ 8\ 8\ 8 & & & \\ 8\ 8\ 8\ 8 & & & \\ & 8\ 8\ 8\ 8 & & \\ 0 & 8\ 8\ 8\ 8 & & \\ & 8\ 8\ 8\ 8 & & \\ & 8\ 8\ 8\ 8 & & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 4\ 0\ 4\ 0 & & 0 & \\ 0\ 4\ 0\ 4 & & & \\ 4\ 0\ 4\ 0 & & & \\ 0\ 4\ 0\ 4 & & & \\ & 4\ 0\ 4\ 0 & & \\ 0 & 0\ 4\ 0\ 4 & & \\ & 4\ 0\ 4\ 0 & & \\ & 0\ 4\ 0\ 4 & & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 2\ 3\ 1\ 1 & & 0 & \\ 1\ 2\ 3\ 1 & & & \\ 1\ 1\ 2\ 3 & & & \\ 3\ 1\ 1\ 2 & & & \\ & 2\ 3\ 1\ 1 & & \\ 0 & 1\ 2\ 3\ 1 & & \\ & 1\ 1\ 2\ 3 & & \\ & 3\ 1\ 1\ 2 & & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \end{pmatrix} = \begin{pmatrix} 1\ 0\ 1\ 1 & 0\ 1\ 1\ 1 \\ 1\ 1\ 0\ 1 & 1\ 0\ 1\ 1 \\ 1\ 1\ 1\ 0 & 1\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 & 1\ 1\ 1\ 0 \\ 1\ 1\ 0\ 0 & 0\ 1\ 1\ 1 \\ 0\ 1\ 1\ 0 & 1\ 0\ 1\ 1 \\ 0\ 0\ 1\ 1 & 1\ 1\ 0\ 1 \\ 1\ 0\ 0\ 1 & 1\ 1\ 1\ 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 1\ 0\ 0\ 0 & 0\ 1\ 1\ 1 \\ 0\ 1\ 0\ 0 & 1\ 0\ 1\ 1 \\ 0\ 0\ 1\ 0 & 1\ 1\ 0\ 1 \\ 0\ 0\ 0\ 1 & 1\ 1\ 1\ 0 \\ 1\ 1\ 0\ 0 & \\ 0\ 1\ 1\ 0 & 0 \\ 0\ 0\ 1\ 1 & \\ 1\ 0\ 0\ 1 & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 0\ 0\ 1\ 1 & & 0 & \\ 1\ 0\ 0\ 1 & & & \\ 1\ 1\ 0\ 0 & & & \\ 0\ 1\ 1\ 0 & & & \\ & 0\ 1\ 1\ 1 & & \\ 0 & 1\ 0\ 1\ 1 & & \\ & 1\ 1\ 0\ 1 & & \\ & 1\ 1\ 1\ 0 & & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \tag{16}$$

$$\begin{cases} y_0 = E_0 + z_0 \\ y_1 = E_1 + z_1 \\ y_2 = E_0 + z_2 \\ y_3 = E_1 + z_3 \end{cases} \quad \begin{cases} y_4 = E_2 + z_4 \\ y_5 = E_3 + z_5 \\ y_6 = E_2 + z_6 \\ y_7 = E_3 + z_7 \end{cases} \tag{17}$$

$$\begin{cases} z_0 = 2A_0 + A_2 + x_1 \\ z_1 = 2A_1 + A_3 + x_2 \\ z_2 = 2A_2 + A_0 + x_3 \\ z_3 = 2A_3 + A_1 + x_0 \end{cases} \quad \begin{cases} z_4 = 2A_4 + B_0 \\ z_5 = 2A_5 + B_1 \\ z_6 = 2A_6 + B_2 \\ z_7 = 2A_7 + B_3 \end{cases} \tag{18}$$

$$\begin{cases} w_0 = A_2 + B_0 + x_0 \\ w_1 = A_3 + B_1 + x_1 \\ w_2 = A_0 + B_2 + x_2 \\ w_3 = A_1 + B_3 + x_3 \end{cases} \quad \begin{cases} w_4 = A_0 + B_0 \\ w_5 = A_1 + B_1 \\ w_6 = A_2 + B_2 \\ w_7 = A_3 + B_3 \end{cases} \tag{19}$$

$$\begin{cases} A_0 = x_0 + x_1 \\ A_1 = x_1 + x_2 \\ A_2 = x_2 + x_3 \\ A_3 = x_3 + x_4 \end{cases} \quad \begin{cases} A_4 = x_4 + x_5 \\ A_5 = x_5 + x_6 \\ A_6 = x_6 + x_7 \\ A_7 = x_7 + x_4 \end{cases} \quad \begin{cases} B_0 = A_6 + x_5 \\ B_1 = A_7 + x_6 \\ B_2 = A_4 + x_7 \\ B_3 = A_5 + x_4 \end{cases} \tag{20}$$

$$\begin{cases} C_0 = 4(x_0 + x_2) \\ C_1 = 4(x_1 + x_3) \\ C_2 = 4(x_4 + x_6) \\ C_3 = 4(x_5 + x_7) \end{cases} \quad \begin{cases} D_0 = 2(C_0 + C_1) \\ D_1 = 2(C_2 + C_3) \end{cases} \quad \begin{cases} E_0 = D_0 + C_0 \\ E_1 = D_0 + C_1 \\ E_2 = D_1 + C_2 \\ E_3 = D_1 + C_3 \end{cases}$$

Table 3 indicates the hardware size and number of stages of critical path in XOR gates for the permutation functions. The total size of our unified permutation circuit are only 476 XORs, while the original functions require 1,482 XORs. Therefore, the hardware cost is reduced down to less than 1/3 with only additional 2 XOR-gate delay.

Table 3. Performance of permutation functions

	Original matrices				Sharing common terms
	2 Mix-Columns	2 InvMix-Columns	P-func	Total	
XORs	304	880	288	1,482	476
Delay (gates)	3	5	3	5	7

5 Unified Data Path Architecture

Fig. 6 shows the unified data path architecture of the data randomization block. In addition to sharing S-Boxes and permutation, FL/FL^{-1} and key whitening functions are also merged. Only 128-bit key is supported in the current design, but 192- and 256-bit keys can be easily supported by modifying the key scheduler shown in Fig. 7. Many components are shared in the data randomization block, but only registers can be re-used in the key scheduler, because the key scheduling methods of two algorithms are much different.

128 bits are processed in one clock cycle by FL/FL^{-1} and key whitening in Camellia and the first AddRoundKey (using key whitening path) in AES. The other function blocks handle a 64 bits at one time. A straightforward implementation of AES that has 128-bit data path takes 11 cycles for the one block encryption or decryption. But the unified hardware of Fig. 6 processes 64 bits / cycle for the 2~11 rounds, and thus, it takes 20 cycles for these rounds. The key scheduler reusing the S-Boxes in the main data path requires additional 10 cycles. Therefore, our unified hardware takes

1+20+10=31 cycles for AES. If four 8-bit S-Boxes dedicated for the key scheduling are attached, the number of cycle is reduced to 21. On the other hand, Camellia takes 2 cycles for FL/FL^{-1}, 2 cycles for key whitening, and 18 cycles for the Feistel rounds (F function), therefore 22 cycles in total.

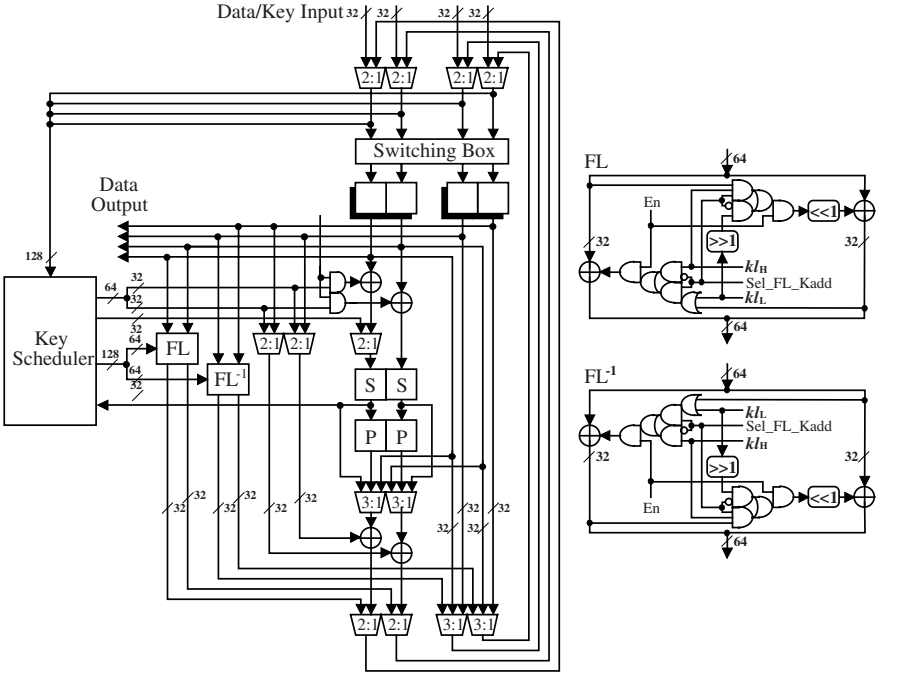

Fig. 6. Data randomization block of unified architecture

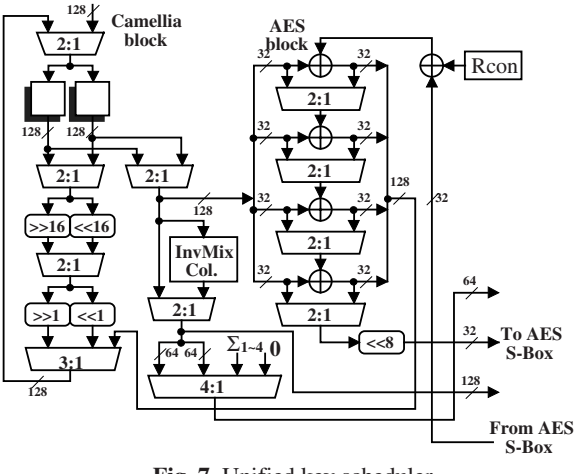

Fig. 7. Unified key scheduler

6 ASIC Performance Comparison

Table 4 shows the performance comparison between our unified hardware and independent implementations of the two algorithms [11, 12], where same 0.13-μm CMOS standard cell library is used for all. Two circuits were generated from each design source by indicating area and speed optimizations to a synthesis tool. The number of S-Boxes are eight (64 bits) in all designs.

In comparison with the AES hardware of the reference [11], the number of cycles of the unified hardware is one cycle fewer. As mentioned before, this is because a 128-bit block is processed at once in the first AddRoundKey while it is executed by 64 bits and takes two cycles in [11]. On the other hand, the Camellia operation takes 22 cycles in the unified hardware, while it is 18 cycles in [12]. Because the Camellia hardware in [12] executes the FL/FL^{-1} functions or the key whitening, and the F function in a same cycle. This approach is suitable for high-speed implementation, but requires additional hardware. Therefore we did not use it for the unified hardware whose priority is compactness. The maximum operation frequency of the unified hardware is lower than that of references [11, 12]. This is because the critical path became longer due to the additional hardware such as selectors to merge two data paths of different algorithms.

Discreet implementations require 21.6K (=8.0K+13.6K) gates for compact versions of two algorithms and 34.6K (=14.8K+19.8K) gates for high-speed versions. On the other hand, our unified hardware is 30% smaller, 14.9K gates and 24.4K gates respectively. The throughputs of the unified hardware are 9~14% lower for AES and 31~40% lower for Camellia. Therefore, the proposed architecture is much suitable for the application such as embedded use where hardware resource is more critical than speed.

Table 4. Hardware performance comparison

	Algorithms	Cycles	Gate counts	Max. frequency (MHz)	Throughput (Mbps)	Synthesis optimization
This work	AES	31	14,918	113.64	469.22	Area
	Camellia	22			661.18	
	AES	31	24,424	192.31	794.05	Speed
	Camellia	22			1,118.89	
Reference [11]	AES	32	7,998	137.17	548.68	Area
			14,777	218.82	875.28	Speed
Reference [12]	Camellia	18	13,557	153.85	1,094.04	Area
			19,783	227.27	1,616.14	Speed

(0.13-μm CMOS, 1gate = 2-input NAND, worst condition)

7 Conclusion

Unified hardware architecture for the 128-bit block ciphers AES and Camellia was proposed and its performance was evaluated in comparison with non-unified implementations. To merge the biggest hardware component S-Box between two algorithms, a multiplicative inverter on $GF((2^4)^2)$ was shared by using isomorphism transformation, and factoring technique was applied on affine transformations. The permutation layers were also merged by sharing common terms of the operator matrix. Our architecture was synthesized by using a 0.13-μm CMOS standard cell library, and compact implementations of 14.9K~24.4K gates were obtained with throughputs of 469M~794Mbps and 661M~1,119Mbps for AES and Camellia respectively. The gate counts were 30% smaller than the conventional implementations where two algorithms were discreetly designed.

References

1. National Institute of Standards and Technology (NIST), "AES Home Page," http://csrc.nist.gov/encryption/aes.
2. National Institute of Standards and Technology (NIST), "Data Encryption Standard (DES)," FIPS Publication 46-3, http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, Oct. 1999.
3. "The Block Cipher Rijndael," http://www.esat.kuleuven.ac.be/~rijmen/rijndael.
4. National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES) FIPS Publication 197," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Nov. 2001.
5. "NESSIE (New European Scheme for Signatures, Integrity and Encryption)", https://www.cosic.esat.kuleuven.ac.be/nessie.
6. ISO/IEC JTC 1/SC27, "Information technology – Security techniques," http://www.din.de/ni/sc27.
7. TV-Anytime Forum, "WG Rights Managements and Protection (RMP)," http://www.tv-anytime.org.
8. CRYPTREC, http://www.ipa.go.jp/security/enc/CRYPTREC.
9. S. Moriai, "Proposal of addition of new cipher suites to TLS to support Camellia, EPOC, and PSEC," Proc. the Forty-Eighth IETF, http://www.ietf.org/proceedings/00jul/SLIDES/tls-cep/index.html, 2000.
10. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S.Moriai, J. Makajima, and T. Tokita, "Specification of Camellia – a 128-bit Block Cipher Version 2.0," http://info.isl.ntt.co.jp/camellia/CRYPTREC/2001/01espec.pdf, 2001.
11. A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-box Optimization," ASIACRYPT 2001, LNCS 2248, pp.239–254, 2001.
12. A. Satoh and S. Morioka, "Compact Hardware Architecture for 128-bit Block Cipher Camellia," Proc. third NESSIE workshop, 2002.

Appendix 1 AES Algorithm

Fig. A1 shows an AES encryption process under a 128-bit secret key. 11 sets of round keys are generated from the secret key, and are fed to each round of the SPN block. The round operation is combination of four primitive functions, SubBytes (sixteen 8-bit S-Boxes), ShiftRows (byte boundary rotations), MixColumns (4-byte × 4-byte matrix operation), and AddRoundKeys (bit-wise XOR). In decryption, the inverse functions (AddRoundKey is identical) are executed in reverse order

The key scheduler uses four S-Boxes and 4-byte constant values Rcon[i] (i=1~10). The highest byte of Rcon[i] is the bit representation of the polynomial x^i mod $m(x)$, and the other three bytes are all zeros. In decryption, these sets of keys are used in reverse order.

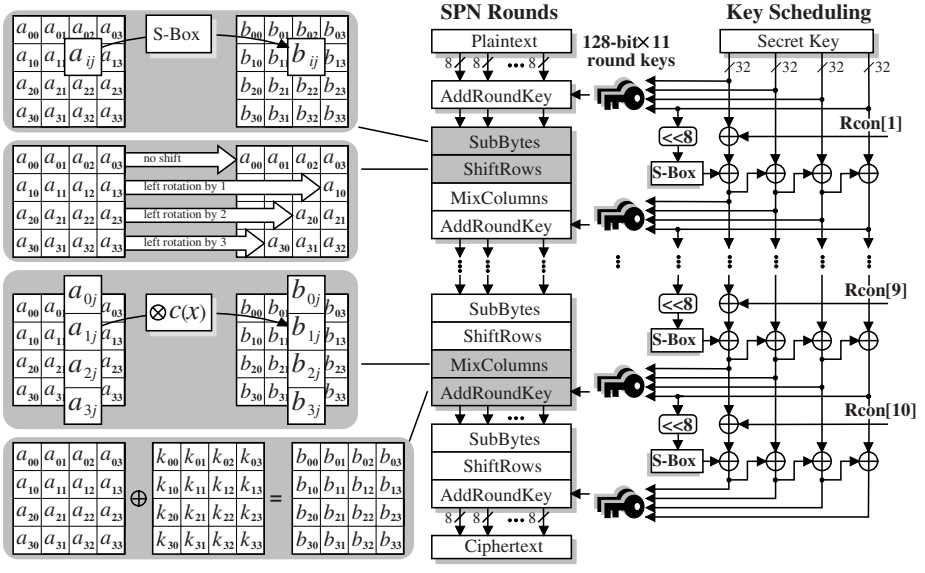

Fig. A1. Encryption process of AES algorithm

Appendix 2 Camellia Algorithm

Fig. A2 shows the encryption process of Camellia for a 128-bit secret key. At the initial and final stages, 128-bit data is XORed with 128-bit round keys. A 22-round data randomization part consists of three 6-round Feistel networks, and two FL/FL^{-1} functions placed between the networks. The 128-bit data input to the Feistel network is divided into two 64-bit data blocks, and the left half is fed into the F function with a 64-bit round key, and its output is XORed with the right half. The left and right half are swapped every round. 64-bit data input to the F function is XORed with the 64-bit

round key. The result is divided into eight 8-bit blocks, and they are fed to eight S-Boxes (S1~S4) followed by the P-function. Same data path can be used in decryption by just changing order of round keys.

As shown in Fig. A3, a 128-bit intermediate key K_A is generated from the 128-bit secret key K_L by using the F function 4 times. The round keys are generated from K_A and K_L by bit rotations.

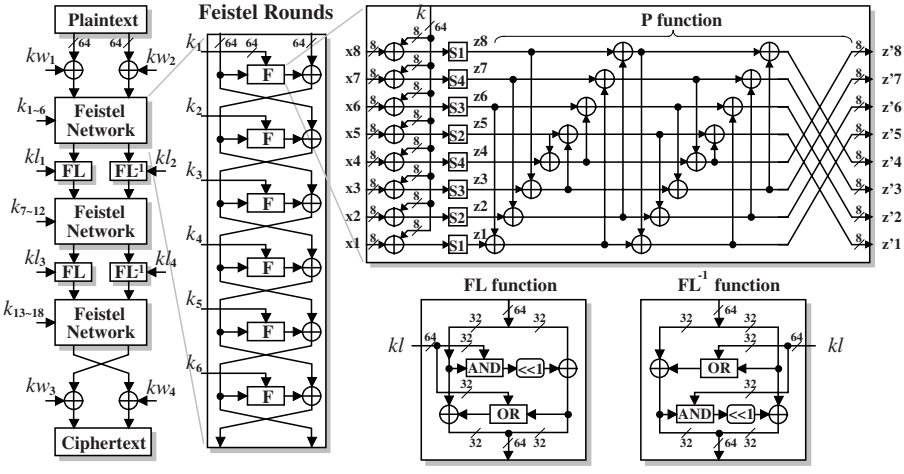

Fig. A2. Encryption process of Camellia for a 128-bit key

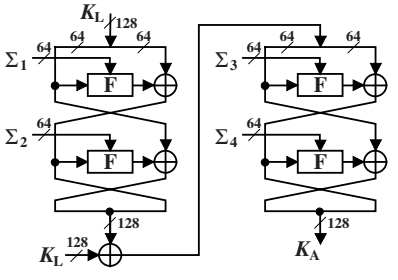

Fig. A3. Intermediate key K_A generating process for a 128-bit key