

An Architecture for S-Box Computation in the AES

S. Chantarawong, P. Noo-intara, and S. Choomchuay

Department of Electronics, Faculty of Engineering, and
Research Center for Communications and Information Technology (ReCCIT)
King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok 10520, Thailand

ABSTRACT

This paper describes an efficient S-box computation developed for the Rijndael ciphering system. The mathematical manipulation lies on composite field computation where the element inversion is performed in the ground field. This development is, on one hand, suitable for applications where table lookup is not applicable or restricted. On the other hand, the development is suitable for a highly-parallel system where data flow can be arranged in pipeline manner. For the compact implementation, the same inversion block could be shared by both encryption and decryption.

KEYWORDS

AES, Rijndael, Encryption, S-box computation, Data Security, VLSI

1. INTRODUCTION

Developed by Joan Daemen and Vincent Rijmen, Rijndael [1,2] was selected (upon security, performance, efficiency, flexibility, and implementability) by NIST as the Advanced Encryption Standard (AES) and published as FIPS 197 in November 2001 [2]. This encryption system also has drawn a good attention of its implementation in both software and hardware since the selection steps [3]. However, most current implementations of AES are done in software. This approach seems to be too slow for fast application such as routers and many other modern communication systems. For some applications, this approach can be considered to be too costly according to its overhead hardware and software. In contrast, in the pure hardware implementation, the higher data rate could be obtained by parallel processing and pipelining [4]. The implementations are physically secure since tempering by an outside attacker is difficult. It's also a cost-effective solution for many application specific systems. The data permutation or S-box computation is an important operation among several others of the EAS system according to its resource consumption. Its efficient implementation can dramatically increase system performance since the algorithm has to perform this operation every round. Moreover, the S-box computation is also required in the KeyScheduling. With these realizations, we are looking into the S-box operation in more fine details, in particular when its conventional ROM approach is limited. As such, the major concern of this paper is the S-box computation.

The implementation is based on finite field mathematic manipulations.

The basic structure of AES is given firstly in section 2. The system comprises of several transformations, used concurrently in sequence call "round". The more detailed BytesSub transformation, the operation that the S-box computation is greatly involved is elaborated in section 3. This step is considered to be both area and power consume one. In section 4, the mathematical manipulations for obtaining an efficient S-box computation are detailed. Inversion and multiplication of elements in $GF(2^k)$ and $GF((2^n)^m)$ are particularly of interest. Implementations of those operations outlined in section 4 are given in section 5 before the conclusion of our paper.

2. THE BASIC STRUCTURE OF THE AES

A full description of the AES is detailed in FIPS 197 [2]. However for sake of understanding, we again outline the AES's structure in this section.

AES is a block cipher developed in effort to address threatened key size of Data Encryption Standard (DES). It allows the data length of 128 bits while supporting three different key lengths, 128, 192, and 256 bits. As such, a mathematical description of the AES is given in Galois Field (2^8). Each round of the whole operation is divided into four basic blocks where data are treated at either byte or bit level. The byte structure seems to be natural for low profile microprocessor (such as 8-bit CPU and microcontrollers). The array of bytes organized as a 4×4 matrix is also called "state". Those four basic steps that describe one round of the AES; BytesSub, ShiftRow, MixColumn, and AddRoundKey are also known as layers.

BytesSub Transformation:

This operation is a non-linear byte substitution. It composes of two sub-transformations; multiplicative inverse and affine transformation. In typical implementations, these two sub-steps are combined into a single table lookup called substitution box or S-box.

ShiftRow Transformation:

This transformation is a linear diffusion process, operates on individual row, i.e. each row of the array is rotated by a certain number of byte positions.

MixColumn Transformation:

This is also a linear diffusion process. A column vector is multiplied (in $GF(2^8)$) with a fixed matrix

where bytes are treated as a polynomial of degree less than 4.

AddRoundKey:

In each round of the AES process, each byte of the array is added (respect to GF(2)) to a byte of the corresponding array of the round subkeys. Round keys are generated by a procedure called "Round Key Expansion" or "KeyScheduling". Those sub-keys are derived from the original keys by EXORing of two previous columns. For columns that are in multiples of four, the process involves round constants addition, byte substitution and shift operations.

All four layers described above have corresponding inverse operations. Excluding the first and the last round, the AES with 128-bit round key proceed for nine iterations. First round of the encryption performs EXOR with the original key and the last round skips MixColumn transform. The deciphering is the reverse order of the ciphering process. Operation steps are similar and at the comparable complexity. As such the same set of hardware can be shared by both processes. However, the inverse MixColumn operation requires matrix elements that are quite complicated compared with {01}, {02} or {03} of the forward one. This results in the slight complicated deciphering hardware.

The operations discussed above are illustrated in Fig. 1 below.

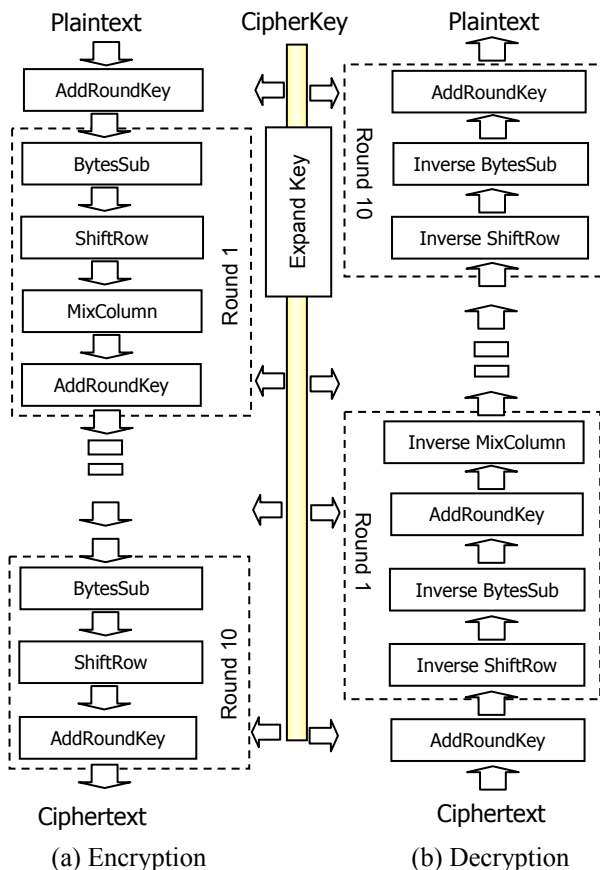


Fig. 1: Block Diagram of AES System

3. BytesSub TRANSFORMATION

This data permutation step operates independently on each byte of the state. The operation comprises of 2 sub-steps:

1. Inversion: Multiplicative inverse of each byte is taken (in GF(2⁸) and {00} is mapped to itself).
2. Affine Transformation: This sub-step is performed in GF(2) and defined by:

$$d(x) = \{1F\}b(x)_{mod(x^8+1)} \oplus c(x) \tag{1}$$

where $c(x) = \{63\} = x^6 + x^5 + x + 1$. This constant has been added in order that the S-box has no fixed point (a map to a) and no opposite fixed point (a map to \bar{a}). The affine transformation can be also written in the matrix form as:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{2}$$

The required addition of {63} can be made vanish by incorporating it within a modified KeyScheduling [5]. In many AES implementations two sub-steps required in the BytesSub transformation are typically combined into a single lookup table. The table size is 16 by 16 with the content 8 bits in length. The ROM size of 256x8 bit is not big for current technology and can be implemented in a fairly simple manner with modern design tools. However when the area is restricted or a ROM cannot be incorporated, the inversion hardware becomes necessary. Within this scenario, the efficient S-box implementation is the major concern. The affine transformation, however requires small number of gates and introduces small delays.

Several techniques for S-box computation have been developed. These are, for instances; (1) The mentioned above table look up where step 2 is usually combined to be a single table, (2) Synthesis and optimized logic function of S-box using CAD tools, and (3) Compute the inversion of element in GF(2⁸) and optimize the logic functions. In the computation of element inversion in GF(2^k) one can use either extended Euclid algorithm [6,7] or composite field technique [8,9, 11,13,14]. The use of composite field in the S-box computation has been reported in literatures [11,13,14]. Rudra *et al.* [10,11] mapped all the operation (except ShiftRow) into the composite field of GF(2⁴)². Multiplication, squaring and inversion are borrowed from those detailed in [12]. Morioka and Satoh [13] also have exploited the used of composite field in the design of a low power S-

box transform. Elements in $GF(2^8)$ are mapped to those defined in $GF((2^2)^2)^2$. Multiplication and inversion are optimized in the ground field. Rijmen [14] also mentioned the computation of inversion in $GF(2^4)$. Our approach has drawn many useful ideas reported in [8], [11] and [13]. More details are elaborated in the next section. To reduce the unnecessary overhead, field transformation is applied to the S-box computation only. We show that it is not necessary to further breakdown the composite field to the lowest ground field. With this finding we can avoid complex design task.

4. AN EFFICIENT S-BOX COMPUTATION

In the case of Rijndael, the field polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ has been chosen. It is an irreducible polynomial in $GF(2^8)$ but not a primitive one. We first have to map elements in $GF(2^k)$ into $GF(2^n)^m$ where $k = mn$. A method elaborated in [11] is again summarized here.

0. Let α be a primitive element of $GF(2^m)^n$, and γ be a primitive element of $GF(2^k)$, such that field isomorphism holds.
1. Map α^i to γ^i for $i \in \{0, \dots, (2^k - 1)\}$.
2. Check whether $\forall i \in \{0, \dots, (2^k - 1)\}$, if $\alpha^r = \alpha^i + 1$ then $\gamma^r = \gamma^i + 1$. If so, we then have the required mapping, otherwise we have to search for the next primitive element.
3. The inverse mapping can be easily found by matrix inversion, i.e., if [T] is a mapping matrix, $[T]^{-1}$ is an inverse mapping matrix.

With above procedure, we select the polynomial $p(x) = x^2 + x + \beta^{14}$ where β^{14} denotes the element in $GF(2^4)$ of which $I(x) = x^4 + x + 1$ is the primitive irreducible polynomial. As a result, the transform matrices that map an element in $GF(2^8)$ to the corresponding element in the composite field $GF(2^4)^2$ or vice versa are obtained as follow.

$$T = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (3)$$

and

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (4)$$

The upper-left element in the above matrices denotes the least significant bit. An advantage of mapping elements from $GF(2^8)$ to $GF((2^4)^2)$ is the simpler multiplicative inverse computation since inversion is performed in $GF(2^4)$. For such a small field size, inversion using either the direct truth-table mapping or table look up consumes small area. Moreover, in Rijndael system data are treated naturally in byte format. Let data (byte) be expressed as $A = \{bc\} = bx + c$, the inversion of A , say $B = A^{-1} = \{pq\} = px + q$. For the field polynomial $p(x) = x^2 + Cx + D$, one can have

$$p = b\Delta^{-1} \quad (5)$$

$$q = (Cb \oplus c)\Delta^{-1} \quad (6)$$

where

$$\Delta = c(Cb \oplus c) \oplus b^2D. \quad (7)$$

or

$$\Delta = bcC \oplus c^2 \oplus b^2D \quad (8)$$

For $GF((2^n)^2)$, the polynomial in the form of $p(x) = x^2 + x + \lambda$ always exists [8]. As such, C and D can be set to $\{1\}$ and $\{9\}$ (in $GF(2^4)$) respectively. Fixed-coefficient multiplication (i.e., b^2D) as well as squaring units are relatively simple according to their small field size. The multiplications required in computing (5), (6) and (7) can be done straight away in $GF(2^4)$ or can be further simplified by making use of composite field $GF((2^2)^2)$ [13]. Borrowed from [8], in our implementation, we use polynomials $p(x) = x^2 + x + \sigma^2$ and $I(x) = x^2 + x + 1$ for the computations in $GF((2^2)^2)$ and $GF(2^2)$ respectively.

In $GF((2^2)^2)$, let $U(x) = u_0 + u_1x$ and $V(x) = v_0 + v_1x$, then

$$Z(x) = U(x)V(x)_{\text{mod } p(x)} = z_0 + z_1x \quad (9)$$

where

$$z_0 = u_0v_0 \oplus \sigma^2u_1v_1 \quad (10)$$

$$z_1 = u_0v_1 \oplus u_1v_0 \oplus u_1v_1. \quad (11)$$

Here $u_i, v_i, \sigma^2 \in GF(2^2)$.

In the (forward) BytesSub transformation, the inversion is followed by the affine transformation given previously in (1). This step can be combined with the inverse mapping and a single logic block is obtained. The resulted matrix is noted in (12) below. Regardless of the hardware reusable, the resulted matrix cannot be shared by the inverse BytesSub transformation.

$$T^{-1}D = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (12)$$

In the decryption process, the inverse affine transform can be expressed as:

$$b(x) = \{4A\}d(x)_{\text{mod}(x^8+1)} \oplus c(x) \quad (13)$$

where $c(x) = \{05\} = x^2 + 1$. This process has to be performed in prior to the (inverse) BytesSub transformation. Similarly, the affine transformation can be merged with the (forward) mapping. The resulted matrix noted in (14) is obtained. The combined matrices given in (12) and (14) are individual. The combined scheme can result in a slightly compact hardware but not applicable to the restricted hardware size application (such as in smart card) where a single inversion circuit is utilized by the ciphering and the deciphering procedures.

$$TB = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (14)$$

5. IMPLEMENTATIONS

The implementation of the BytesSub transformation formulated earlier in section 4 is detailed in this section. Gate count and delay time of the designs with different architectures are investigated. Many related mathematic equations are given as appendix for convenience of referring. The inversion of elements in the S-box operation has been reported of its most power consuming compared to others (i.e., 75% [13]). We thus look at this operation in quite details. The S-box

computation can be divided into 3 blocks as shown in Fig. 2a. The affine transform and the inverse mapping could be combined as mentioned earlier.

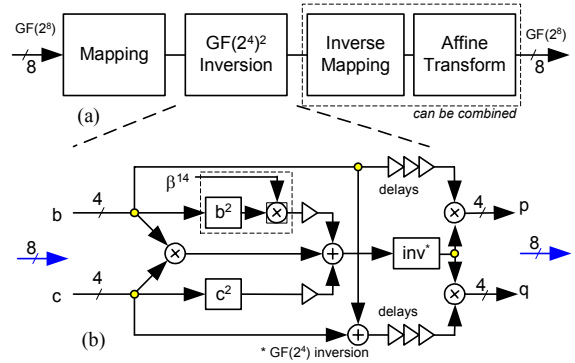


Fig 2: S-box Computation and Inversion in $GF((2^4)^2)$

Shown in Fig. 2b, the inversion in $GF(2^8)$ are performed in the composite field, $GF((2^4)^2)$. Three GP multiplications, two squarings, a fixed-coefficient multiplication and an inversion are involved. There are slight differences of the implementation of Fig.2. One can choose either (7) or (8). The differences are gate count, wiring complexity and critical data path. We chose (8) because of its smaller delays compared to that offered by (7). It also should be noted that the operation $b^2\beta^{14}$ could be combined into a single logic block (see Appendix -5).

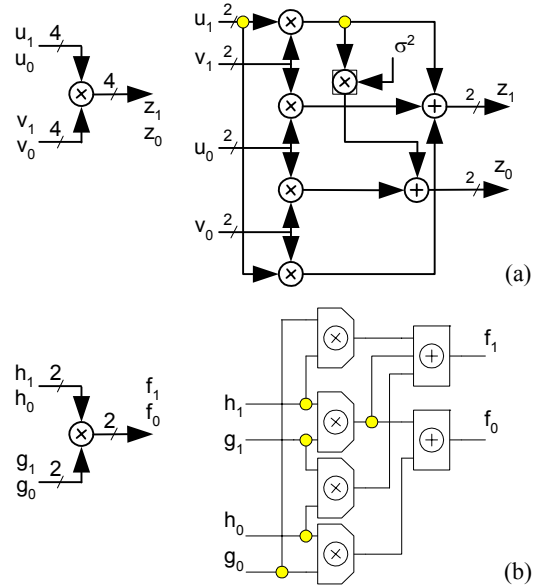


Fig 3: Multiplication in $GF((2^4)^2)$

A direct implementation of a general purpose multiplier (see Appendix -2.1) can result in a complexity of 16 AND and 10 OR, and with the delay time of 3τ . The same multiplier can be also implemented using the composite field technique. This is shown in Fig. 3 above (see also Appendix -2.2). The complexity is 16 AND, 14 OR, and with the delay time of 3τ . Similarly the

inversion in $GF(2^4)$ can be implemented directly as given in Appendix –5 [12]. The complexity is 11 AND, 11 OR, 5 INV, and with the delay time of 3τ . This inversion can be implemented in the composite field $GF((2^2)^2)$. The similar scheme as that shown in Fig. 2b is re-applied. The complexity is 12 AND, 9 OR and with the delay time of 4τ . Regardless the inserted delays, the above discussed inversion in $GF(2^8)$ is summarized in table 1 below.

Table 1 Complexity of $(GF(2^4)^2)$ inversion

Inversion $GF(2^4)^2$	Complexity*				Unit ** required
	AND	XOR	INV	τ	
Squaring	-	2	-	1	2
Fixed-Coeff. multiplication	-	1	-	1	1
Inversion $(GF(2^4))$	11	11	5	3	1
Inversion $(GF(2^4)^2)$	12	9	-	4	(1)
Multiplication $(GF(2^4))$	16	10	-	3	3
Multiplication $(GF(2^4)^2)$	16	14	-	3	(3)

* 1) 2 input gate and 3 input gate are equivalent
 2) Gate delay of 1, 2 and 3 input are assumed to be equal
 ** () could be chosen as an alternative option

The field mapping denoted by (3) and (4) above can be easily implemented with about 16 XORs. This shown in Fig. 4 below. If one needs the combined mapping noted by (12) and (13) can be implemented similarly.

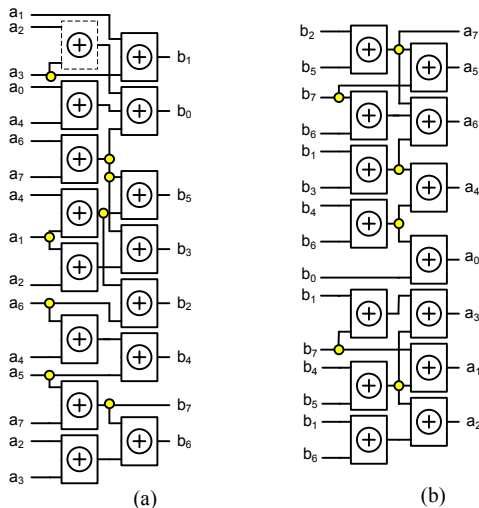


Fig 4: Field Mapping (a) and Inverse Mapping (b)

6. CONCLUSIONS

S-box computation is required in both BytesSub transformation and KeyScheduling process. The efficient implementation results in the desirable performance of the AES system. For the application that the table lookup is restricted, the field inversion computation is

necessary. Inversion computation in the composite field is a key issue investigated in this paper. An obvious overhead is the field mapping since Rijndael is not built up with a primitive irreducible polynomial. Fortunately, field isomorphism holds. The affine transform can be easily combined with the mapping (or inverse mapping) matrix. This results in more compact hardware in some design criteria. However, the resulted S-box is individual and cannot be shared by the ciphering and the inverse ciphering procedure.

Computation of inversion in $GF((2^4)^2)$ is practically accepted since $GF(2^8)$ is a rather large field size. Regarding the design complexity, there is not much gain (in terms of gate count, wiring complexity and design time) of further breaking down the inversion in $GF(2^4)$ into that of $GF((2^2)^2)$. In many cases one can do the inversion in $GF(2^4)$ strait away.

If the high speed is a case, the architecture proposed in this paper can be applied easily. Each byte can have its own byte substitution circuit that operates in a pipeline fashion. The speed of a byte per one clock cycle per one circuit (or as a whole 128×8 bit/clock cycle) could be achieved. For the extreme case the minimum clock period can be made to that of the delay time of a 3 input XOR gate.

Architectures given in this paper are fairly at low level and fairly easy to be realized with either standard cell or other modern VLSI design tools such as HDL and FPGA. The high level language (C programming) has confirmed the validity and correctness of the algorithms.

REFERENCES

- [1] J. Daemen and V. Rijmen, AES Proposal: Rijndael (Version 2). NIST AES Website; <http://csrc.nist.gov/publications/> and <http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf>
- [2] NIST, Advanced Encryption Standard (AES), (FIP PUB 197), November 26, 2001, <http://csrc.nist.gov/publications/>
- [3] J. Nechvatal et. al., Report on the development of Advanced Encryption Standard, NIST publication, October 2, 2000. <http://csrc.nist.gov/CryptoToolkit/aes/>
- [4] I.M. Verbauwhede, P.R. Schaumont, and, H. Kuo, "Deign and Performance Testing of A 2.29 Gb/s Rijndael Processor," *IEEE J. of Solid State-Circuit*, Vol. 38, No. 3, March 2003, pp. 569 – 572.
- [5] S. Murphy and M.J.B. Robshaw, "Essential Algebraic Structure Within the AES," *Proc. CRYPTO 2002*, Vol. 2442, Springer-Verlag, pp. 1-16, 2002.
- [6] K. Araki, I. Fujita and M. Morisue, "Fast Inverter Over Finite Fields Based on Euclid's Algorithm," *Trans. IEICE*, Vol. E-72, No. 11, pp. 1230–1234, Nov. 1989.

- [7] H. Brunner, A. Curiger, and M. Hofstetter, "On Computing Multiplicative Inverse in $GF(2^m)$," *IEEE Trans. on Computer*, Vol. 42., No. 8, pp. 1010–1015, Aug. 1993.
- [8] C. Paar, "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields," *IEEE Trans. on Comp.*, Vol. 45, No. 7, pp 856–861, 1996.
- [9] C. Paar, "Fast Arithmetic Architecture for Public-Key Algorithms over Galois Fields $GF((2^n)^m)$," *Proc. EUROCRYPT'97*, LNCS Vol. 1233, Springer-Verlag, pp. 363-378, 1997.
- [10] C. Jutla, V. Kumar and A. Rudra, "On the Complexity of Isomorphic Galois Field Transformations," *IBM Research Report*, Vol. RC22652 (W0211-243), November 2002.
- [11] A. Rudra et. al., "Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic," *Proc. CHES 2001*, LNCS Vol. 2162, pp. 175-188, 2001.
- [12] E.D. Mastrovito, "VLSI Architecture for Computations in Galois Fields," Ph.D. Thesis, Dept of EE, Linköping University, Linköping, Sweden 1991.
- [13] S. Morioka and A. Satoh, "An Optimized S-box Circuit Architecture for Low Power AES Design," *Proc. CHES 2002*, LNCS Vol. 2523, pp. 172-186, 2003.
- [14] V. Rijmen, "Efficient Implementation of the Rijndael S-box," <http://csrc.nist.gov/CryptoToolkit/aes/>

APPENDIX

Bit-parallel architecture of standard and composite field operations

(A) $GF(2^8)$ Computations

1. Inversion in $GF((2^4)^2)$; $p(x) = x^2 + x + \beta^{14}$

b, c, p, q are elements in $GF(2^4)$

$$A(x) = bx + c; B(x) = A^{-1}(x) = px + q$$

$$p = b\Delta^{-1}, q = (b \oplus c)\Delta^{-1}$$

$$\Delta = \beta^{14}b^2 \oplus bc \oplus c^2.$$

(B) $GF(2^4)$ Computations

2.1 Multiplication in $GF(2^4)$, $I(x) = x^4 + x + 1$

To compute $C(x) = A(x)B(x)_{\text{mod}I(x)}$ where

$$a_i, b_i, c_i \in GF(2^4)$$

$$c_0 = a_0b_0 \oplus a_3b_1 \oplus a_2b_2 \oplus a_1b_3$$

$$c_1 = a_1b_0 \oplus a_0b_1 \oplus a_3b_1 \oplus a_3b_2 \oplus a_2b_2 \oplus a_2b_3 \oplus a_1b_3$$

$$= a_1b_0 \oplus (a_0 \oplus a_3)b_1 \oplus (a_2 \oplus a_3)b_2 \oplus (a_1 \oplus a_2)b_3$$

$$c_2 = a_2b_0 \oplus a_1b_1 \oplus a_0b_2 \oplus a_3b_2 \oplus a_3b_3 \oplus a_2b_3$$

$$= a_2b_0 \oplus a_1b_1 \oplus (a_0 \oplus a_3)b_2 \oplus (a_2 \oplus a_3)b_3$$

$$c_3 = a_3b_0 \oplus a_2b_1 \oplus a_1b_2 \oplus a_0b_3 \oplus a_3b_3$$

$$= a_3b_0 \oplus a_2b_1 \oplus a_1b_2 \oplus (a_0 \oplus a_3)b_3$$

2.2 Multiplication in the composite field $GF((2^2)^2)$,

where $p(x) = x^2 + x + \sigma^2$

$$U(x) = u_0 + u_1x, V(x) = v_0 + v_1x \text{ and}$$

$$Z(x) = U(x)V(x)_{\text{mod}p(x)} = z_0 + z_1x, \text{ where}$$

$$z_i, u_i, v_i \in GF(2^2).$$

$$z_0 = u_0v_0 \oplus \sigma^2u_1v_1 \text{ and } z_1 = u_0v_1 \oplus u_1v_0 \oplus u_1v_1$$

3) Squaring; $B(x) = A(x)A(x)_{\text{mod}I(x)}$, $I(x) = x^4 + x + 1$

$$b_0 = (a_0 \oplus a_2), b_1 = a_2, b_2 = (a_1 \oplus a_3), b_3 = a_3$$

4) Fixed-coefficient multiplication;

To compute $C(x) = \beta^{14}B(x)_{\text{mod}I(x)}$ where

$$I(x) = x^4 + x + 1$$

$$c_0 = (b_0 \oplus b_1), c_1 = b_2, c_2 = b_3, c_3 = b_0$$

5) To compute $C(x) = A^2(x)\beta^{14}_{\text{mod}I(x)}$;

Combine 3) and 4) above

$$c_0 = a_0, c_1 = (a_1 \oplus a_3), c_2 = a_3, c_3 = (a_0 \oplus a_2)$$

6) Inversion; $C(x) = A^{-1}(x)$, $I(x) = x^4 + x + 1$

$$c_0 = a_2 \oplus a_3 \oplus \bar{a}_2(a_0 \oplus a_1) \oplus a_1a_2(a_0 \oplus a_3)$$

$$c_1 = a_0(a_1 \oplus a_2) \oplus a_1(a_2 \oplus a_3) \oplus a_3(\bar{a}_0 \oplus \bar{a}_1)$$

$$c_2 = a_0a_2a_3 \oplus a_0a_1 \oplus \bar{a}_0(a_2 \oplus a_3)$$

$$c_3 = a_1 \oplus \bar{a}_0(a_2 \oplus a_3) \oplus a_1(a_0a_2 \oplus a_3)$$

(C) $GF(2^2)$ Computations

$$I(x) = x^2 + x + 1;$$

7) Multiplication in $GF(2^2)$

To compute $F(x) = g(x)h(x)_{\text{mod}I(x)}$

$$f_0 = g_0h_0 \oplus g_1h_1$$

$$f_1 = g_0h_1 \oplus g_1h_0 \oplus g_1h_1$$

8) Fixed-coefficient multiplication;

$$Z(x) = \sigma^2U(x)_{\text{mod}I(x)}$$

$$z_0 = (u_0 \oplus u_1), z_1 = u_0$$

9) Squaring; $C(x) = A^2(x)_{\text{mod}I(x)}$

$$c_0 = (a_0 \oplus a_1), c_1 = a_1$$

10) Inversion; $C(x) = A^{-1}(x)$

$$c_0 = (a_0 \oplus a_1), c_1 = a_1$$