A Compact 32-bit Architecture for an AES System

Somsak Choomchuay[1], Member,
Surapong Pongyupinpanich[2], and **Somsanouk Pathumvanh**[3], Non-members

ABSTRACT

This paper describes a compact 32-bit architecture developed for the Rijndael ciphering/decyphering system. The implementation is complied with NIST Advanced Encryption Standard (AES). The design processes any 128-bit block data with 128-bit key. For the compact hardware, the field inversion circuit and the key scheduling circuits are shared by both the encryption and decryption process. The on-the-fly KeyScheduling implementation offers fast processing speed but with core size trade-off. According to the evaluation made on the targeted FPGA, the design can offer the throughput of 768 mbps at 264 MHz clock speed.

1. INTRODUCTION

Since the announcement of FIP-197 by NIST [1], the Advanced Encryption Standard (AES) has become more and more involvement in data security issues such as IPSec [2], IEEE802.11i (RSN) [3]. Most current implementations of AES are done in software. This approach seems to be too slow for fast applications such as routers, gateways and some wireless communication systems. It is also vulnerable to attacks. For some applications, this approach can be costly according to its overhead hardware and software. In contrast, in the pure hardware implementation, the higher data rate (Gbits/second) could be obtained by parallelization 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 AES IP cores are also available commercially in the ASIC and FPGAs [5-7]. Such high data rates of about 1.16 Gbit/sec. (for 32-bit data path, 400 MHz, ASIC, [5]) are made available and claimed.

04PSI19: Manuscript received on January 4, 2005 ; revised on August 20, 2005.

[1]The author is with 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. E-mail: kchsomsa@ kmitl.ac.th

[2]The author is with the Faculty of Computer Engineering, Ramkhamhaeng University, Thailand. E-mail: p_surapong2000@ yahoo.com

[3]The author is with the Faculty of Engineering, National University of Laos, P.O. Box 3166, Vientiane, Lao P.D.R. Email: somsanouk@nuol.edu.la

In our implementation we are aiming at the AES hardware with cipher & decipher capability, comprehensive (key expansion is included), supporting various cipher modes (such as CBC, CFB and OFB), and with moderate throughput. Our paper is organized as follows. The basic structure of AES is firstly given in section 2. The concept of a 32-bit architecture is then given in section 3. The system comprises of several transformations, used concurrently in sequence called "round". "BytesSub" transformation or S-box computation, the operation that both area and power consume one, is elaborated in section 3.1. Sections 3.2, 3.3, and 3.4 are dedicated to the implementations of "ShiftRow" transform, "MixColumn" transform, and "AddRoundkey" processes, respectively. Section 4, describes the FPGA implementation and its performance evaluations, while section 5 concludes the work.

Terminology: In our paper, text or character represented data are treated either in hex (byte format) as $\{mm\}$ or in the matrix form, or in the polynomial repre-sentation i.e., $\lambda = \{mn\} = [C] = C(x) = c_7x^7 + c_6x^6 + \ldots + c_1x + c_0$, here $c_i \in \{0, 1\}$.

2. THE BASIC STRUCTURE OF THE AES

A full description of the AES is described in the Rijndael proposal [8] and FIPS 197 [1]. It 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 the Galois Field GF(2^8). The whole operation is divided into four basic blocks where data are treated at either byte or bit level. The array of bytes organized as a 4×4 matrix is also called "state" and those four basic steps (or basic blocks); BytesSub, ShiftRow, MixColumn, and AddRoundKey are also known as layers.

The four layer steps mentioned above describe one round of the AES. Number of rounds is made vary according to the key size. The AES with 128-bit key size operates iteratively on those four basic steps for ten rounds. However, the first and the final rounds are arranged in a slightly different manner compared to others. All four layers have their corresponding inverse operations. The deciphering is, therefore, the reverse order of the ciphering process. Operation steps are similar and at the comparable complexity.

Moreover, both processes can share same set of designed hardware. In particular, the S-box computation that occupies the major chip area since it is required in BytesSub transform and in KeyScheduling processes.

This paper details the design of the AES system based on an iterative loop architecture. The architecture is made versatile by offering both ciphering and dechiphering functions. It has been focused on the design compactivity, good speed, and reasonable gate counts. The 32-bit architecture is aimed. Although the encrypter and the decrypter utilize the similar sub-processes, those sub-processes have their own specification. However, a multiplicative inversion and AddRoundKey operation can be perfectly shared by both. Additional multiplexers were employed in order to save some data holding registers. The encrypter and the decrypter can perfectly utilize the same set of hardware. The block diagram of the system operation is shown in Fig. 1 below.

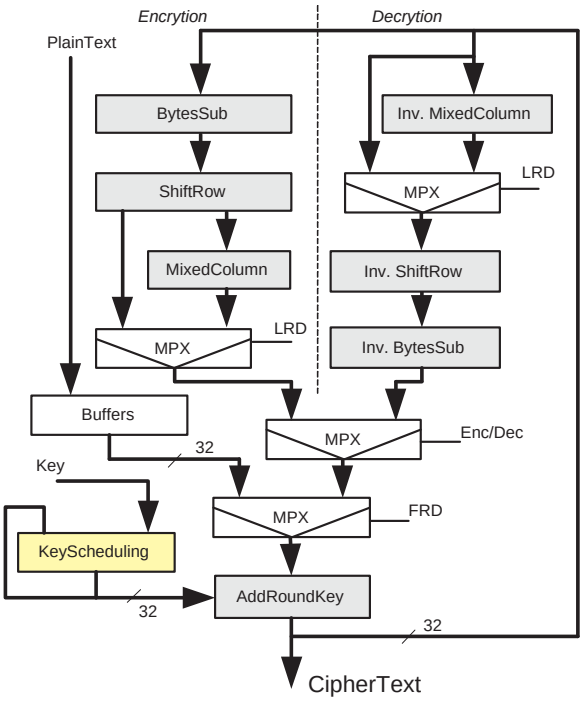

Fig.1: *Block Diagram of the AES system (arrowed-dash lines denote the deciphering data flow)*

3. A 32 BITS ARCHITECTURE

The four rows or four columns data matrix can naturally define the data path of 32 bits. Together with the compact hardware criteria, the architecture can be easily optimized. The AES core can process the data at one column (or one row) at a time. However, the ShiftRow operation requires the availability of all 128 bits data before it can start. In this case, a longer register (128 bits) is needed. The same hardware is used for all round (10 rounds). The throughput can be increased (about 10 times) by un-rolling those round operations. Pipeline registers can be inserted. However, what one has to pay for is the hardware size and cost.

3.1 BytesSub Transformation

The BytesSub transformation 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(2^8), and {00} is mapped to itself.

2. Affine Transformation: This sub-step is performed in GF(2).

To implement the BytesSub transformation, many techniques have been reported. Those are, for instances; (1) The table lookup technique where step 2 is usually combined into a single table known as S-box. For the current technology and design, the table size of 256×8 bit is not considerably big. This technique has been adopted in many realizations. (2) Synthesis and optimized logic function of S-box using CAD tools, and (3) Compute the inversion of element in GF(2^8) and optimize the logic functions. The efficiency of the third technique is much depended on the mathematical theory of field element inversion. This approach is highly considered when the table lookup is not applicable or when the compact design is a case. It also provides desirable features for the highly-paralleled computation.

In our implementation we have chosen the option (3) since the field inversion hardware can be easily shared by both the encryption process and decryption process. The BytesSub (and similarly, the inverse BytesSub) transform of a byte is defined mathematically as:

$$D(x) = \delta A^{-1}(x)_{mod(x^8+1)} \oplus C(x) \qquad (1)$$

where $C(x) = x^6 + x^5 + x + 1 = \{63\}$ and $\delta = \{1F\} = x^4 + x^3 + x^2 + x + 1$ for the encryption process. The constant $C(x)$ 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 $\bar{a}$). Besides the field inversion, the implementation of such a transformation is fairly simple as the circuit can be built up form an array of XOR gates.

The use of composite field in the S-box computation, in particular to simplify the field inversion, has been reported in literatures [9-11]. Rudra et al. [10] mapped all the operation (except ShiftRow) into the composite field of $GF(2^4)^2$. Multiplication, squaring and inversion are borrowed form those detailed in [11]. Morioka and Satoh [12] also have exploited the used of composite field in the design of a low power S-box transform. With the composite field technique, field elements of $GF(2^k)$ are mapped to those of $GF(2^n)^m$ and try to optimize the multiplication and inversion in the lowest field. Our approach

has drawn many useful ideas reported in [9] and [11]. However to reduce the unnecessary overhead, field transformation is applied to the S-box computation only. It is also not necessary to further breakdown the composite field to the lowest ground field, i.e. $GF(2^2)^2$ as proposed in [12].

AES has adopted $m(x) = x^8+x^4+x^3+x+1$ as its field polynomial. Although such a polynomial is an irreducible but it is not a primitive one. Fortunately, with the field isomorphism property, we can map elements in $GF(2^8)$ to the composite field $GF((2^4)^2)$ based on the polynomial $w(x) = x^2 + x + \beta^{14}$, where β^{14} denotes the element in $GF(2^4)$ of which $I(x) = x^4 + x + 1$ is the primitive irreducible polynomial. Let D be an element in $GF(2^8)$ and A be an element in $GF(2^4)^2$, then $A = [T]D$ and $D = [T]^{-1}A$ where

$$T = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (2)$$

and

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

Here $[T]$ and $[T]^{-1}$ are the field transformation matrices. The upper-left element in the above matrices denotes the least significant bit. In the composite field, let a byte-format data be expressed as

$$A = \{pq\} = px + q \quad (4)$$

The inversion of A, say

$$B = A^{-1} = sx + t \quad (5)$$

For the composite field polynomial $w(x) = x^2 + x + \beta^{14}$, one can have

$$s = p\Delta^{-1}, \text{ and} \quad (6)$$

and

$$t = (p \oplus q)\Delta^{-1}, \quad (7)$$

where $\Delta = pq \oplus q^2 \oplus p^2\beta^{14}$.

To perform an inversion in the composite field noted in (5), three general purpose multipliers, two squaring circuits, a fixed-coefficient multiplier and an inversion (in GF(16)) are required. An obvious advantage of mapping elements in $GF(2^8)$ to those of $GF((2^4)^2)$ is the simpler multiplicative inverse computation since inversion is performed in $GF(2^4)$. For such a small field size inversion, we used truth-table directly.

The S-box computation can be divided into 4 sub-blocks namely; field mapping, multiplicative inversion, inverse mapping and affine transform. Affine and inverse-affine transformation circuits are actually combination logics that can be implemented easily. A multiplexer is required whether this block is used as an S-box or S^{-1}-box for the forward or inverse transformation respectively. As shown in Fig. 2, for one column computation (32-bit width) there requires 4 circuit sets. The same S-box is also required in KeyScheduling computation. The detailed circuitry of the S-box computation with composite field inversion can be found in [13] and [14].

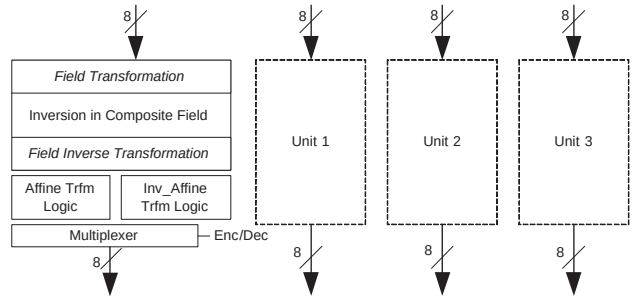

Fig.2: *$S-box$ and $S-box^{-1}$ Computation in $GF((2^4)^2)$*

3.2 ShiftRow Transformation

This process operates on an individual row with individual offset byte. In the state arrangement, data are fed into a square matrix in row order. As such, to operate the ShiftRow transformation we need registers to store the whole data before byte swapping. This can result in the unsmooth data flow. However, implementation is easy. We used the matrix switches for byte swapping (see Fig. 3 below) such that data are arranged in order and ready for the following operation; MixColumn (or Inverse MixColumn) transform. The transform throughput is 32 bits per clock cycle and can be made pipelined for col-umn order. For a wider data path (128-bit) or the higher throughput such as 128 bits per clock cycle, the switches c_0, c_1, c_3, c_4 and multiplexers are not necessary.

3.3 MixColumn Transformation

The MixColumn transform of an AES can be expressed as

$$\acute{C}(x) = C(x)a(x)_{mod(x^4+1)}. \quad (8)$$

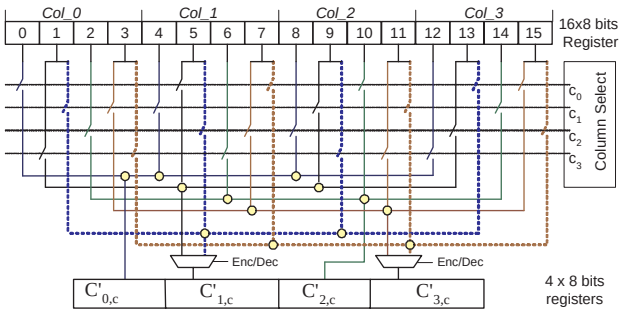

Fig.3: *ShiftRow and Inverse ShiftRow (dash line) Switches with MixColumn-Transform Ready*

Each column is considered as a polynomial with coefficients $C_{i,c}$ defines in $GF(2^8)$. The multiplication is modulo x^4+1 and $a(x)$ is given by

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3, \qquad (9)$$

where $a_0 = \{02\}$, $a_1 = a_2 = \{01\}$ and $a_3 = \{03\}$ respectively.

The inverse MixColumn matrix can be written similarly: $b(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$ is defined as the inverse transform polynomial (i.e. $a(x)^{-1}$) where $b_0 = \{0E\}, b_1 = \{09\}, b_2 = \{0D\}$ and $b_3 = \{0B\}$ respectively.

$C_{i,c}$ is computed in one clock cycle (or one column or 32 bits per one clock cycle) with four parallel fixed-coefficient multipliers, followed by a summing operation as shown in Fig 4. A fix-coefficient multiplier is simple and compact.

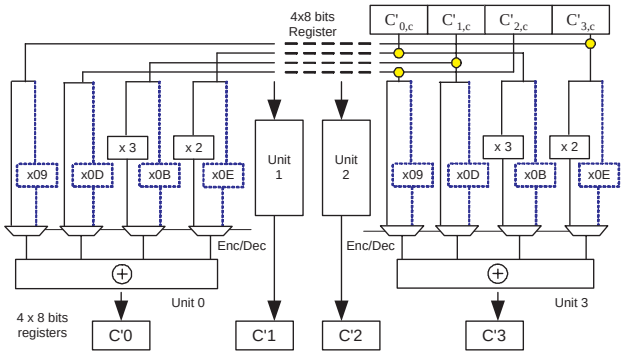

Fig.4: *MixColumn and Inverse MixColumn Transform*

3.4 AddRoundkey and KeyScheduling Transformation

Excluding the first and the last round, the AES with 128 bit round key proceeds for nine iterations. First round of the encryption performs EXORing with the original key and the last round skips MixColumn transform. Round keys are generated by a procedure called "RoundKey Expansion" or "KeyScheduling". Those subkeys are derived from the original key by EXORing of two previous columns. For columns that are in multiples of four, the process involves shift operations (RotWord, ROT), byte substitution (S-box) and round constants (Rcon) addition. In the decrypter mode the operation is reverted.

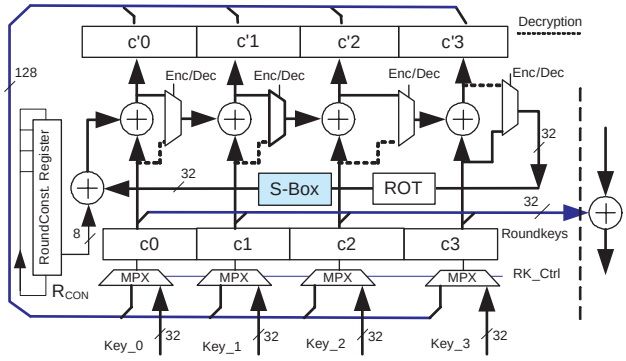

Fig.5: *KeyScheduling Arrangement*

The circuit for implementing the round key expansion operation (and its inverse) is shown in Fig.5. We need two 128-bit registers to store Roundkeys. The Sub-Byte transformation or S-Box can be computed as that detailed in section 3. ROTword register is a circular word-shift register whilst Rcon is feedback word-shift register.

4. FPGA IMPLEMENTATION

The throughput of the design based on the loop architecture given in Fig. 1 is quite proportional to the data path and processing sub-module used. For instance, to compute the BytesSub transformation, an 8-bit module (8-bit data path width) must be used for 16 times. Similarly, 16 hardware units can process the BytesSub in a single clock cycle. This estimation could also be applied to the system with the data path of 16, 32, 64 and 128 bits. However, only slight change can be made to Key-Schudling and ShiftRow circuits.

4.1 A 32-bit Architecture

The entity of the 32 bit FPGA implementation is shown in Fig. 6.

The interface as well as many internal communication data paths are 32-bits width. However, some internal registers, such as those in the shift row operation and key scheduling are 128 bits. The preliminary investigation of an iterative loop architecture (Fig.1 with 10 rounds) resulted that three data blocks of 128 bits each could be achieved in 132 clock cycles with 128 clock cycles latency.

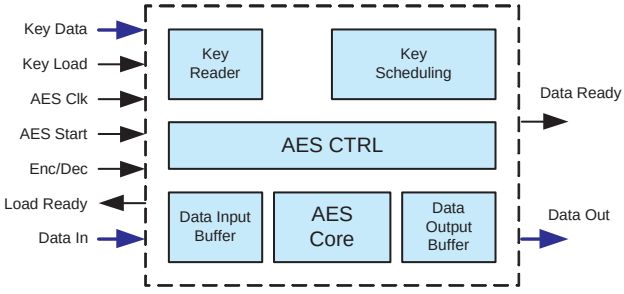

Fig.6: *AES's Chip Entity*

Description:
AES Clk; Input: AES System Clock
Load Ready; Output: Input Data Request Signal
Data Ready; Output: Output Data (Block) Ready and Valid
AES Start; Input: Require to start the operation
Enc/Dec; Input: Require to select the operation
Key Load; Input: Require to Load Key Data
Key Data [31:0]; Input: Encryption/Decryption Key
Data In [31:0]; Input: Input Plain Text Data
Data out [31:0]; Output: Output Cipher Text Data

4.2 Performance Evaluation

Resource required and data rate resulted from the synthesis for targeted Xilinx FPGAs are shown in table 1, below. The throughput (TP) varies form 180 to 500 megabit per second (mpbs) depends on targeted device. As shown in table 2, it should be noted that [5-7] do provide AES cores and Key expansion unit separately. In contrast, the design detailed in this paper didn't utilize the block RAM (BRAM), but included the on-the-fly KeyScheduling unit. With the same path width (32-bit), our design has more CLB slices when compared to [5]. Although we can obtain higher throughput, our obtained bit per clock cycle is similar to that of [5] (about 2.9 bits per clock cycle). The 128-bit data path was estimated upon the 32-bit one. The obtained throughput is higher compared to others. However, an obvious drawback is its high LUT counts. Using BRAM Blocks can lower LUT counts but at the cost of lowering the throughput.

Table 1: *Result for 128-bit key, 32-bit data path, and with Key Expansion Unit (ECB mode)*

Xilinx Devices	CLB Slice	IOs	fmax (MHz)	TP (mbps)
Spartan II (2S200fg456)	866	228	94.5	275
Vertex (V200fg456)	866	228	115	335
Vertex II (XC2V1000)	866	228	264	768

Table 2: *Result of comparison to others (ECB mode)*

Xilinx Devices	Width (bits)	IOs /BRAM	CLB Slice	fmax (MHz)	TP (mbps)
Core Only [5] (XC2V80-6)	32	105/2	115	149	432
Core Only [5] (XC2V1000-6)	128	393/2	416	122	1,415
Core Only [7] Vertex II-5	128	NA/2	181	114	331
Key-Exp. [7] Vertex II-5	128	NA/2	181	149	432
This paper* (XC2V1000)	32	228/no	866	264	768
This paper ** (XC2V1000)	128	NA/no	1,724	264	2,413

*, ** AES Core + Key Expansion, ** Based on estimation

5. CONCLUSION

A 32-bit architecture implementation of the AES system is addressed in this paper. With one column (or one row) processing approach, the design is fairly compact and with throughput trade-off. The inversion, a part of S-box computation, is computed in the composite field. This operation is required in both BytesSub transformation and KeyScheduling process (as well as their inverses). One drawback is the overhead field mapping since Rijndael is not built up with a primitive irreducible polynomial. Hard-wire ShiftRow (and Inverse ShiftRow) operation and Fixed-coefficient multiplications employed in the MixColumn (and Inverse MixColumn) transform lead to both good speed and area saving. The architecture proposed in this paper is thus very suitable for full-custom realization. If the higher throughput is a case, several sub-architectures proposed in this paper can be modified easily to suit the need. Many transforms must be paralleled and subpipelined. The 128-bit (data path) architecture can then be achieved with much higher speed, but not so much cost of hardware. For a very fast speed one, the pipelined loop-unrolled architecture is desirable, however with the increasing of hardware cost. However, one should note that the pipeline structure is suitable for ECB (Electronic Code Book) mode of operation, but not very useful for other three modes (BCB, CFB, and OFB mode) where feedbacks are employed. Alternatively, processing speed could be made higher by employing gate array or standard cell technology.

References

[1] NIST, Advanced Encryption Standard (AES), (FIP PUB 197), November 26, 2001, http://csrc.nist.gov/publications/
[2] IP Security Protocol (IPSec) Charter- Latest

RFCs and Internet Drafts for IPSec, Available at http://ietf.org/html.charters/ipsec-charter.html

[3] Cox, Philip. "Robust Security Network: The future of wire-less security". System Experts Corporation. http://www.systemexperts.com/win2k/SecureWorldExpo-RSN.ppt

[4] I. M. Verbauwhede, P.R. Schaumont, and H. Kuo, "Design 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] CAST, Advanced Encryption Standard Core, available at; http://www.cast-inc.com/cores/aes/index.shtml.

[6] IP Cores, Ultra-Compact, Advanced Encryption Standard Core, available at; http://www.ipcores.com/AES1.pdf.

[7] Ocean-Logic, OL-AES Core Family, Available at http://www.ocean-logic.com/pub/OL_AES.pdf.

[8] J. Daemen and V. Rijmen, AES Proposal: Rijndael (Ver. 2). NIST AES Website; http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf.

[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] A. Rudra et al., "Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic," *Proc. CHES* 2001, LNCS Vol. 2162, pp. 175-188, 2001.

[11] E. D. Mastrovito, "*VLSI Architecture for Computations in Galois Fields*," Ph.D. Thesis, Dept of EE, Linköping Univ., Lingköping, Sweden 1991.

[12] 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.

[13] S. Chantarawong, P. Noo-intara, and S. Choomchuay, "An Architecture for S-Box Computation in the AES," *Proc of Information and Computer Engineering Workshop 2004 (ICEP2004)*, Prince of Songkla University (Phuket Campus), January 2004, pp.157-162.

[14] S. Chantarawong and S. Choomchuay, "An Architecture for a Compact AES System," *Proc. of Electrical Eng./Electronics, Communications*, Computer and Information Technology Conference 2004 (ECTI-CON. 2004), ECTI Association, Thailand, May 2004, pp. 121-124.

Somsak Choomchuay was born in 1959, Thailand. He received B.Eng, and M.Eng. in electronic engineering in 1983 and 1986 respectively. He also has obtained Ph.D. and DIC. from Imperial College, University of London in 1994, the same year that he joined the department of Electronic, Faculty of Engineering, KMITL. He has been the associate professor in electronic engineering since 2000. His research interest includes VLSI system and digital signal processing. Dr.Somsak is an active member of many societies and international conferences.

Surapong Pongyupinpanich was born in 1976, Thailand. He has obtained B.Eng. (Industrial Engineering) and M.Eng. (Electronic Engineering) from KMITL in 1998 and 2003 respectively. He is currently a lecturer of the faculty of computer engineering, Ramkhamhaeng University. His interest concerns crypto system, digital signal processing, and FPGA realization.

Somsanouk Pathumvanh was born in 1978, at Luangprabang, Lao P.D.R. Has possessed bachelor's degree in Electronic Engineering from King Monkut's Institute of Technology Ladkrabang, Bangkok Thailand in 2001, he is now a member of the Faculty of Engineering, National University of Laos (NUOL). During the period of his worked at NUOL, he has gained strong knowledge and experience in Internetworking and network security systems. His research interested was covered Internetworking and related field such as: Sensor network, Routing Techniques, QoS, and Network Security.