High-Speed Hardware Architectures for ARIA with Composite Field Arithmetic and Area-Throughput Trade-Offs

Sang-Woo Lee, Sang-Jae Moon, and Jeong-Nyeo Kim

This paper presents two types of high-speed hardware architectures for the block cipher ARIA. First, the loop architectures for feedback modes are presented. Area-throughput trade-offs are evaluated depending on the S-box implementation by using look-up tables or combinational logic which involves composite field arithmetic. The sub-pipelined architectures for non-feedback modes are also described. With loop unrolling, inner and outer round pipelining techniques, and S-box implementation using composite field arithmetic over $GF(2^4)^2$, throughputs of 16 Gbps to 43 Gbps are achievable in a 0.25 μm CMOS technology. This is the first sub-pipelined architecture of ARIA for high throughput to date.

Keywords: ARIA, block cipher, cryptography, hardware architecture.

I. Introduction

The importance of cryptography is constantly increasing with the increasing amount of sensitive data that is being transmitted over open environments. Compared to software implementations, hardware implementations of cryptographic algorithms provide higher throughput and greater physical security.

The ARIA algorithm was announced as the Korean standard block cipher algorithm in December of 2004. ARIA is a 128-bit block cipher with an involution substitution permutation network (SPN) that supports 128-bit, 192-bit, or 256-bit key lengths [1], [2]. The block size of the data and the key lengths are identical to those of Advanced Encryption Standard (AES), which was specified in 2001 by the National Institute of Standards and Technology [3]. It is important to note that Public Key Cryptography Standard (PKCS) #11, which is the Cryptographic Token Interface Standard published by RSA Security, has recently incorporated the mechanisms of ARIA as well as those of AES [4].

To provide different security features, [5] defines five confidentiality modes of operation for use with an underlying symmetric key block cipher algorithm: electronic codebook (ECB), cipher block chaining (CBC), cipher feedback (CFB), output feedback (OFB), and counter (CTR). These modes can provide cryptographic protection for sensitive, but unclassified, computer data. Some of these, such as the CBC mode, restrict use of the pipeline technique, which processes multiple blocks simultaneously to achieve better performance. However, the ECB and CTR modes, which do not require feedback, can be

Manuscript received Apr. 1, 2008; revised Aug. 18, 2008; accepted Aug. 22, 2008.

This work was supported by the IT R&D program of MKE/IITA, Rep. of Korea [2008-F036-01, Development of Anonymity-based u-Knowledge Security Technology].

Sang-Woo Lee (phone: +82 42 860 1097, email: ttomlee@etri.re.kr) and Jeong-Nyeo Kim (email: jnkim@etri.re.kr) are with S/W & Content Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Sang-Jae Moon (email: sjmoon@knu.ac.kr) is with the Department of Electrical Engineering, Kyungpook National University, Daegu, Rep. of Korea.

pipelined for high throughput. In addition, some of these modes such as CFB, OFB, and CTR do not need decryption of an underlying block cipher. Therefore, the hardware architecture of ARIA should use a different strategy depending on the mode.

1. Related Work

ARIA is known to be relatively strong against differential crypto-analysis and linear crypto-analysis [6]. Investigations of power analysis attacks and countermeasures for ARIA can be found in [7]-[9]. However, only a small number of papers have been published on the hardware architecture for the ARIA algorithm in contrast to AES. A compact design of ARIA was proposed in two recent studies [10], [11]. In particular, [11] suggested a 16-bit architecture of ARIA and proposed a substitution block that used composite field arithmetic. An implementation of a unified hardware architecture for ARIA and AES was reported in [12].

On the other hand, numerous studies of the efficient hardware implementation of AES have been presented [13]-[30]. Again, as ARIA (despite its involutional features) is an SPN block cipher similar to AES, previous work on hardware implementation of the AES algorithm is helpful to the implementation of ARIA in hardware.

Related previous works can be categorized into area-efficient architectures for resource-restricted environments [16]-[21] and high-speed designs using a pipelining technique for server-side applications [23]-[25], [27]-[29]. SubBytes, a substitution step of AES, can be implemented using a look-up-table (LUT) scheme or combinational logic based on composite field arithmetic. Traditionally, an S-box is implemented in an LUT-based approach [13], [14]. Several studies [18]-[21] have proposed alternative methods to implement the S-box in computational form for compact hardware architecture. These references illustrate a method to transform the original field of $GF(2^8)$ to a composite field of $GF(2^4)^2$ or $GF((2^2)^2)^2$ in order to replace the S-box tables. This method was originally suggested by Rijmen, one of the inventors of AES [22]. In the composite field scheme, hardware resources of the S-box can be decreased at the expense of increasing the delay.

Another research area involves architectural optimization for high-speed design. Some of the earlier implementations are based on loop architecture and the LUT-based S-box approach [13], [14]. Thus, they can only provide a throughput of 2 to 3 Gbps. The LUT-based S-box has an undivided delay to access memory. This feature prevents each round unit from being divided into sub-stages for further speed-up. Several studies [23]-[25] suggest a hardware architecture that combines a pipelining technique with a composite-field-based S-box implementation to achieve high throughput.

2. Contribution

This paper presents high-speed hardware architectures for the block cipher ARIA in the form of loop architecture for feedback modes and sub-pipelined architecture for non-feedback modes. In addition, the proper key schedulers for these architectures are proposed. The implementation trade-offs, which depend on the type of S-box, are evaluated to optimize the hardware size and the throughput. The novelty of this study is the application of the hardware optimization techniques found for AES to ARIA. However, the optimization techniques for AES cannot be applied to ARIA directly because ARIA uses two S-boxes. One is equivalent to the AES S-box, but the other is different. A method to represent S_2, the second S-box of ARIA, is described as an affine transformation of the multiplicative inversion over $GF(2^4)^2$ with detailed transformation matrices. An effort was made to explain the detailed matrix conversion, following the affine transformation equations, and the merged matrix of the affine transformation with an isomorphic function between $GF(2^8)$ and $GF(2^4)^2$. Such an elaborate description was omitted in two earlier studies [11], [12], although it is very important for further improvements.

The proposed loop architecture was designed to complete one round of encryption and decryption in one clock cycle. The S-boxes and their inverses were shared for even and odd round substitution tables. The design trade-offs between the LUT and the composite field schemes were evaluated. Furthermore, a round unit was implemented to process the key initialization step in order to reduce the area cost.

There has been no known hardware architecture of ARIA for high throughput proposed to date. This work is the first sub-pipelined hardware architecture of ARIA that uses a method to implement substitution tables based on a composite field operation over $GF(2^4)^2$. The architectures that are addressed in this paper can achieve throughputs in a range from 16 Gbps to 43 Gbps depending on the number of sub-stages. The proposed approach involves unrolling the round loop and inserting pipeline registers both inside and between each round unit. To achieve higher throughput, deep pipelining can be used in inner round units. In this case, it is necessary to design a substitution layer using composite field arithmetic, which allows a deeper level of pipelining leading to an improvement in the throughput.

The remainder of this paper is organized as follows. In section II, the ARIA algorithm is briefly described. The detailed hardware architecture of the primitives is explained in section III. Section IV describes the architectural optimization, which includes the loop and sub-pipelined architectures along with their design trade-offs. In section V, the implementation results and analysis are discussed. Finally, conclusions are given in section VI.

II. ARIA Algorithm

The ARIA algorithm is a symmetric block cipher in which the data is encrypted and decrypted in blocks of 128 bits. ARIA has a variable key length which can be 128, 192, or 256 bits, and the corresponding number of rounds for these lengths is 12, 14, or 16, respectively [1], [2].

ARIA consists of two procedures: a round operation for encryption or decryption and a key scheduler including key initialization and round key generation. Each data block is modified by several rounds of processing in which each round involves the following:

- AddRoundKey: This performs a bitwise exclusive-OR operation between the 128-bit round key and the incoming 128-bit data.
- Substitution layer: Two types of substitution layer, which use four S-boxes (S_1, S_2, S_1^{-1}, and S_2^{-1}) are processed. S_1^{-1} and S_2^{-1} are the inverses of S_1 and S_2, respectively.
- Diffusion layer: This computes an involutional 16×16 binary matrix.

There are two types of substitution layers: one for even rounds and the other for odd rounds. A substitution layer is composed of sixteen S-boxes that are byte-oriented substitution tables. In the case of the AES algorithm, the encryption process only uses the S-box, and the decryption process looks up the inverse of the S-box instead of the S-box. However, the substitution layer of ARIA makes use of both S-boxes and their inverses during encryption and decryption processes.

The key scheduling unit generates round keys from an original master key. The key scheduling operation performs two processes: key initialization and round key generation. In the key initialization process, four 128-bit values, W_0, W_1, W_2, and W_3, are generated from the master key using a three-round 256-bit Feistel cipher based on the ARIA round operation. The 128-bit round keys are generated by the initialization values W_0, W_1, W_2, and W_3 with the cyclic shifts and XOR operations. Here, the last round of ARIA requires two round keys. Therefore, ARIA requires 13, 15, or 17 round keys according to the key lengths of 128 bits, 192 bits, or 256 bits, respectively. The round keys used for decryption are different from the encryption round keys. The order of the round keys is reversed followed by the output of the diffusion to all encryption round keys except during the first and last rounds.

III. Detailed Hardware Architecture of the Primitives

1. Substitution Layer

The substitution layer is the most time-consuming step in the ARIA algorithm. There are two well-known approaches to implementation of the S-box. The first is the LUT method using ROMs. With the LUT scheme, sixteen ROMs (256 entries × 1 byte) are required for each round. Therefore, the LUT approach requires significant hardware resources when multiple round units are implemented. On the other hand, it is possible to implement the S-box with combinational logic using a Galois field operation without tables. While S_1 is defined as an affine transformation of the multiplicative inversion over GF(2^8), S_2 is a combination of x^{247} and the affine transformation. Both S-boxes are defined over GF(2^8) with the following irreducible polynomial:

$$m(x) = x^8 + x^4 + x^3 + x + 1 .$$

This is identical to the polynomial used in AES. Moreover, S_1 is identical to the S-box of AES.

Several papers describe the implementation of the S-box based on a transformation of the original field of GF(2^8) to a composite field of GF(2^4)^2 or GF(((2^2)^2)^2 by isomorphic mapping for the S-box of AES [16], [18]-[21], [23]-[25]. In [18] and [21] the S-box was implemented using composite field arithmetic over GF(2^4)^2. Such use of composite field arithmetic was recommended by Rijmen [22] and is well documented in [18]. Additionally, Satoh and others [20] used the composite field of GF(((2^2)^2)^2, and Zhang and Parhi [23] designed a pipelined S-box using the composite field of GF(((2^2)^2)^2 to achieve high throughput. Both composite field conversions can be efficiently implemented to reduce the required hardware resources and to improve the performance. More precisely, the conversion to the composite field over GF(2^4)^2 proposed by Wolkerstorfer and others has a smaller critical path delay compared to that proposed by Satoh and others [26].

In the case of the ARIA algorithm, one study [11] describes a method of using the composite field of GF(((2^2)^2)^2 for the substitution layer. However, that study did not present the internal matrices for the field conversion and the affine transformation. Such details are crucial if further improvements are to be made.

In this section, the implementation of the substitution layer of ARIA using the composite field of GF(2^4)^2 is presented. Because the method of using the composite field for S_1 is well described in [18], we explain the case of S_2 in detail.

The S-boxes and their inverses in ARIA can be presented as follows:

$$\begin{aligned} S_1(x) &= A \cdot x^{-1} \oplus a, \\ S_1^{-1}(x) &= ((A^{-1} \cdot x) \oplus (A^{-1} \cdot a))^{-1}, \end{aligned} \tag{1}$$

$$\begin{aligned} S_2(x) &= B \cdot x^{247} \oplus b = B \cdot x^{-8} \oplus b \\ &= B \cdot C \cdot x^{-1} \oplus b = D \cdot x^{-1} \oplus b, \\ S_2^{-1}(x) &= ((D^{-1} \cdot x) \oplus (D^{-1} \cdot b))^{-1}, \end{aligned} \tag{2}$$

where x^{-1} refers to the inversion of x over GF(2^8); $\oplus$ denotes 128-bit XOR; A, B, C, and D are 8×8 binary matrices; and a and b are 8×1 binary vectors. For S_2, since x^{254} is x^{-1} in GF(2^8) according to Fermat's little theorem, the term x^{247} can be represented as x^{-8}, and x^{-8} is equivalent to $C{\cdot}x^{-1}$ with a new matrix C. At this point, it is possible to represent $B{\cdot}x^{-8}$ as $B{\cdot}C{\cdot}x^{-1}$. Assuming that $D = B{\cdot}C$, the equation for S_2 becomes $D{\cdot}x^{-1} \oplus b$. Additionally, the inverse of the matrix D must be known to calculate S_2^{-1}. The matrices are the following:

$$B=\begin{bmatrix}0&1&0&1&1&1&1&0\\0&0&1&1&1&1&0&1\\1&1&0&1&0&1&1&1\\1&0&0&1&1&1&0&1\\0&0&1&0&1&1&0&0\\1&0&0&0&0&0&0&1\\0&1&0&1&1&1&0&1\\1&1&0&1&0&0&1&1\end{bmatrix} \quad C=\begin{bmatrix}1&1&0&1&0&0&0&0\\0&1&1&1&0&0&0&0\\0&0&1&0&1&1&0&0\\0&1&1&0&0&0&1&0\\0&1&1&1&0&1&0&0\\0&0&0&1&1&0&1&1\\0&0&1&0&1&0&1&0\\0&0&0&1&1&1&1&0\end{bmatrix}$$

$$D=\begin{bmatrix}0&1&0&1&0&1&1&1\\0&0&1&1&1&1&1&1\\1&1&1&0&1&1&0&1\\1&1&0&0&0&0&1&1\\0&1&0&0&0&0&1&1\\1&1&0&0&1&1&1&0\\0&1&1&0&0&0&1&1\\1&1&1&1&0&1&1&0\end{bmatrix} \quad D^{-1}=\begin{bmatrix}0&0&0&1&1&0&0&0\\0&0&1&0&0&1&1&0\\0&0&0&0&1&0&1&0\\1&1&1&0&0&0&1&1\\1&1&1&0&1&1&0&0\\0&1&1&0&1&0&1&1\\1&0&1&1&1&1&0&1\\1&0&0&1&0&0&1&1\end{bmatrix}$$

Here, the least significant bits are in the upper left corners.

Using these matrices, it is possible to define the affine transformation for S_2 and the inverse affine transformation for S_2^{-1} as (3) and (4), respectively.

$$\begin{bmatrix}y_0\\y_1\\y_2\\y_3\\y_4\\y_5\\y_6\\y_7\end{bmatrix}=\begin{bmatrix}0&1&0&1&0&1&1&1\\0&0&1&1&1&1&1&1\\1&1&1&0&1&1&0&1\\1&1&0&0&0&0&1&1\\0&1&0&0&0&0&1&1\\1&1&0&0&1&1&1&0\\0&1&1&0&0&0&1&1\\1&1&1&1&0&1&1&0\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\end{bmatrix}\oplus\begin{bmatrix}0\\1\\0\\0\\0\\1\\1\\1\end{bmatrix}, \quad (3)$$

$$\begin{bmatrix}y_0\\y_1\\y_2\\y_3\\y_4\\y_5\\y_6\\y_7\end{bmatrix}=\begin{bmatrix}0&0&0&1&1&0&0&0\\0&0&1&0&0&1&1&0\\0&0&0&0&1&0&1&0\\1&1&1&0&0&0&1&1\\1&1&1&0&1&1&0&0\\0&1&1&0&1&0&1&1\\1&0&1&1&1&1&0&1\\1&0&0&1&0&0&1&1\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\end{bmatrix}\oplus\begin{bmatrix}0\\0\\1\\1\\0\\1\\0\\0\end{bmatrix}. \quad (4)$$

Implementation of the S-boxes using combinational logic without tables involves inversion in GF(2^8), which may have high hardware complexity. To reduce the complexity of the required logic for inversion in GF(2^8), the composite field of $G(2^4)^2$ is utilized. An element a in the field GF(2^8) is

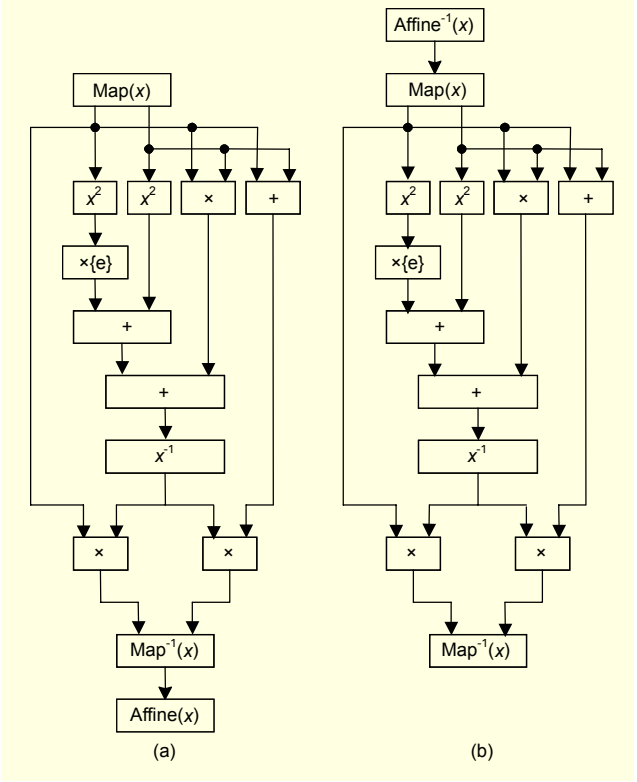


Fig. 1. Architecture of the substitution layers: (a) an S-box and (b) the inverse of the S-box.

represented as a linear polynomial with coefficients in GF(2^4):

$$a = a_h x + a_l, \qquad \text{for} \quad a \in GF(2^8),\ a_h, a_l \in GF(2^4). \quad (5)$$

Multiplication of a two-term polynomial with its inverse yields the 1-element of the field:

$$\begin{aligned}(a_h x + a_l)\times(a_h' x + a_l') &= \{0\}x + \{1\},\\ \text{for} \quad a_h, a_l, a_h', a_l' &\in GF(2^4),\end{aligned} \quad (6)$$

with an irreducible polynomial $n(x) = x^2 + \{1\}x + \{e\}$. Using the extended Euclid algorithm, the inversion from (6) can be derived as

$$\begin{aligned}(a_h x + a_l)^{-1} &= a_h' x + a_l' = (a_h \times d)x + (a_h + a_l)\times d,\\ d &= ((a_h^2 \times \{e\}) + (a_h \times a_l) + a_l^2)^{-1}.\end{aligned} \quad (7)$$

Figure 1 shows how the S-box can be designed using GF(2^4) arithmetic, as described in (7). Figure 1(a) shows the S-box function, which is composed of the inversion over GF(2^8) and the affine transformation, here denoted by Affine(x). Figure 1(b) shows the inverse of the S-box, which consists of the inverse affine transformation and the inversion over GF(2^8). The circuit for the inversion of elements in GF(2^8) occupies most of the S-box functionality. In this approach, the inversion is calculated with combinational logic that operates in GF(2^4).

The components of the inversion are the following:

- Map(x) and Map^{-1}(x): an isomorphic mapping function between GF(2^8) and GF(2^4)^2 and its inverse, respectively
- x^2: square in GF(2^4)
- ×: multiplication in GF(2^4)
- +: addition in GF(2^4) which is equivalent to bitwise XOR
- ×{e}: multiplication of an element in GF(2^4) by the constant {e}
- x^{-1}: inversion in GF(2^4)

The input element of GF(2^8) is mapped to two elements of GF(2^4). The multiplicative inverse is calculated using GF(2^4) operations, and two GF(2^4) elements are then mapped inversely to one element in GF(2^8). Finally, the affine transformation is calculated for the S-box. To compute the inverse of the S-box, inverse affine transformation calculated first, and then the multiplicative inversion over GF(2^8) is performed. The isomorphism may be considered as the matrix multiplication of the 8×8 binary matrix, M, with the 8-bit value in GF(2)^8. The matrix M of Map(x) and its inverse M^{-1} are given by

$$M=\begin{bmatrix}1&0&0&0&1&1&1&0\\0&1&1&0&0&0&0&0\\0&1&0&0&0&0&0&1\\0&0&1&0&1&0&0&0\\0&0&0&0&1&1&1&0\\0&1&0&0&1&0&1&1\\0&0&1&1&0&1&0&1\\0&0&0&0&0&1&0&1\end{bmatrix},\quad M^{-1}=\begin{bmatrix}1&0&0&0&1&0&0&0\\0&0&0&0&1&1&0&1\\0&1&0&0&1&1&0&1\\0&1&0&0&1&1&1&0\\0&1&0&1&1&1&0&1\\0&0&1&0&1&1&0&0\\0&1&1&1&1&0&0&1\\0&0&1&0&1&1&0&1\end{bmatrix}.$$

Further optimization can be achieved to reduce the hardware area. Map^{-1}(x) and Affine(x) can be merged for the S-box, and Affine^{-1}(x) and Map(x) can be combined for the inverse of the S-box. The combination of the isomorphic mapping with the affine transformation for S_1 and S_2 are defined as δ_1 and δ_2, respectively. These combinations and their inverses are the following:

$$\delta_1(x)=\begin{bmatrix}1&0&1&0&1&1&0&1\\1&1&1&1&1&1&0&1\\1&0&0&1&1&1&0&0\\1&0&1&0&1&0&1&1\\1&1&0&1&1&0&1&1\\0&1&1&1&1&1&1&1\\0&0&0&0&1&0&1&1\\0&1&1&0&1&0&1&1\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\end{bmatrix}\oplus\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix},$$

$$\delta_1^{-1}(x)=\begin{bmatrix}1&1&0&0&1&0&1&0\\1&1&0&1&1&0&1&1\\1&1&0&1&1&0&0&0\\0&0&0&1&1&0&1&1\\1&1&1&0&1&1&1&1\\0&0&0&1&1&1&1&0\\1&0&0&0&1&1&1&0\\0&1&1&0&0&0&1&1\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\end{bmatrix}\oplus\begin{bmatrix}1\\1\\0\\1\\0\\0\\1\\0\end{bmatrix},$$

$$\delta_2(x)=\begin{bmatrix}0&0&1&1&1&0&1&1\\0&0&1&0&0&1&1&0\\1&0&0&1&0&1&0&0\\1&1&0&1&0&0&0&1\\0&1&0&1&1&0&0&1\\1&0&0&0&1&1&0&1\\0&0&0&1&0&1&0&0\\1&1&0&1&0&0&1&1\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\end{bmatrix}\oplus\begin{bmatrix}0\\1\\0\\0\\0\\1\\1\\1\end{bmatrix},$$

$$\delta_2^{-1}(x)=\begin{bmatrix}0&0&1&0&0&0&1&0\\0&0&1&0&1&1&0&0\\1&0&1&1&0&1&0&1\\1&1&1&0&0&1&1&0\\0&0&1&1&1&0&1&0\\1&1&1&0&0&1&0&0\\0&0&0&1&0&0&0&1\\1&1&1&1&1&0&0&0\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\end{bmatrix}\oplus\begin{bmatrix}1\\1\\0\\1\\1\\0\\1\\1\end{bmatrix}.$$

Information on these combinations is important if further optimizations are to be made.

2. Diffusion Layer

The diffusion layer consists of a byte-oriented 16×16 binary matrix. There are six XOR operations for each row of the matrix; hence, there are 768 two-input XOR gates in the diffusion layer. However, the proposed diffusion layer has only 480 two-input XOR gates because temporary values of the matrix are used. These are given by

$$\begin{aligned}
&T_0=x_3\oplus x_4\oplus x_9\oplus x_{14}, && T_1=x_2\oplus x_5\oplus x_8\oplus x_{15},\\
&y_0=x_6\oplus x_8\oplus x_{13}\oplus T_0, && y_1=x_7\oplus x_9\oplus x_{12}\oplus T_1,\\
&y_5=x_1\oplus x_{10}\oplus x_{15}\oplus T_0, && y_4=x_0\oplus x_{11}\oplus x_{14}\oplus T_1,\\
&y_{11}=x_2\oplus x_7\oplus x_{12}\oplus T_0, && y_{10}=x_3\oplus x_6\oplus x_{13}\oplus T_1,\\
&y_{14}=x_0\oplus x_5\oplus x_{11}\oplus T_0, && y_{15}=x_1\oplus x_4\oplus x_{10}\oplus T_1,\\
&T_2=x_1\oplus x_6\oplus x_{11}\oplus x_{12}, && T_3=x_0\oplus x_7\oplus x_{10}\oplus x_{13},\\
&y_2=x_4\oplus x_{10}\oplus x_{15}\oplus T_2, && y_3=x_5\oplus x_{11}\oplus x_{14}\oplus T_3,\\
&y_7=x_3\oplus x_8\oplus x_{13}\oplus T_2, && y_6=x_2\oplus x_9\oplus x_{12}\oplus T_3,\\
&y_9=x_0\oplus x_5\oplus x_{14}\oplus T_2, && y_8=x_1\oplus x_4\oplus x_{15}\oplus T_3,\\
&y_{12}=x_2\oplus x_7\oplus x_9\oplus T_2, && y_{13}=x_3\oplus x_6\oplus x_8\oplus T_3.
\end{aligned}$$

3. Round Key Generation

The key scheduling is typically implemented using one of two methods: computing the round keys in advance and storing them in memory or computing them on-the-fly for every block of encrypted data. The former method is suitable for applications in which keys do not change frequently. However, the pre-computing key-generation scheme consumes a considerable amount of memory to store the entire set of round keys. On the other hand, in applications that need to change keys frequently, generating keys on-the-fly is preferred

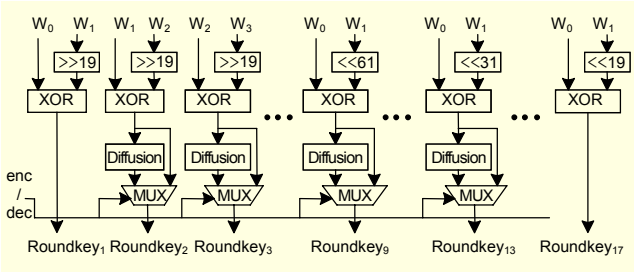


Fig. 2. Block diagram of fully parallel round key generation.

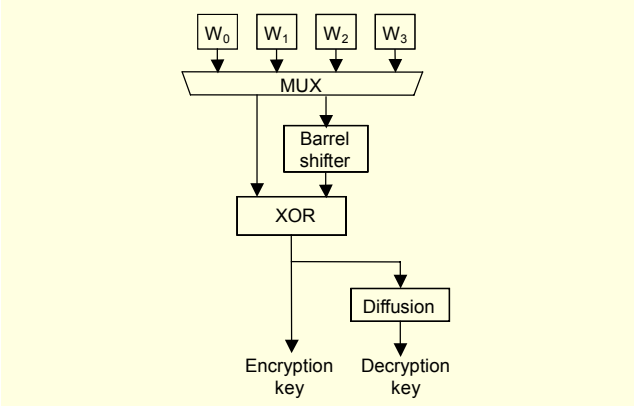


Fig. 3. Block diagram of the on-the-fly key generator.

as it is possible to change keys rapidly without delay.

The key scheduling of ARIA consists of two procedures: key initialization and round key generation. The key initialization process can be implemented by sharing a round operation block with additional XOR arrays. In this section, two types of round key generation blocks are described.

Figure 2 describes a part of the fully parallel round key generation block. In the figure, $<< x$ and $>> x$ denote cyclic shifts by x bits to the left and right, respectively. With the proposed architecture, all the round keys, (for both encryption and decryption), can be generated simultaneously in one clock cycle. This feature is appropriate for applications that support numerous independent streams of data with different keys and that therefore require very high throughput. Resources such as the XORs and the diffusion logic can be shared to reduce the hardware size, but additional clock cycles are required to generate all of the round keys. As the cyclic shifts can be implemented by wiring inputs, no logic gates are needed. Thus, the delay in generating the round keys for encryption is determined by only one XOR gate and the multiplexer. However, there is extra delay for decryption. Here, the critical path of the key generation is composed of one XOR gate, the diffusion layer, and the multiplexer as shown in Fig. 2. To reduce the critical path of the entire system, seventeen round key registers are required. Therefore, the fully parallel architecture of the round key generation is suitable for a pre-computing method that originally requires memory to store the round keys. In addition, this generator is proper for the loop unrolled and pipelined architecture, which requires round key registers to store the entire set of round keys in order to process multiple blocks simultaneously.

Table 1. Gate counts of the proposed key generators.

	Gate count
On-the-fly generator	4,990
Fully parallel generator	46,563

In the other type of architecture, the on-the-fly key scheduler, the registers to store all of the round keys can be removed. In the case of the AES algorithm, it is easy to perform key scheduling procedures in the forward direction for encryption. However, round keys are applied in the reverse order for decryption. Therefore, the on-the-fly key scheduler for AES causes extra clock cycles for decryption since the decryption process can only begin after the last round key is computed. In contrast, the round keys for both encryption and decryption in ARIA can be generated directly from the main key. Thus, the overhead for decryption is eliminated with ARIA. The on-the-fly round key generator is shown in Fig. 3.

Variable amounts of cyclic shift can be implemented by a combinational barrel shifter. In this key-scheduling process, only one 128-bit register (instead of seventeen round key registers) is required in order to store the next round key. Therefore, on-the-fly key scheduling is appropriate for the loop architecture, as it computes one round per clock cycle. Table 1 presents the gate counts of the two proposed round key generators. The fully parallel round key generator has more gates, as it has seventeen 128-bit registers and fifteen diffusion circuits compared to the on-the-fly key generator, which has one key register and one diffusion circuit.

IV. Architectural Optimization and Trade-Offs

In this section, two types of hardware architectures of ARIA are described in relation to the mode of operation. First, the loop architecture is explained, in which one round operation unit is implemented in hardware. This block is reused several times to complete the entire encryption or decryption process depending on the key length. The loop architecture has a small circuit area but has low throughput. Thus, the throughput is improved at the expense of an increase in the area by using a combination of loop unrolling and pipelining. Unrolled architectures have a large number of rounds that are independently implemented in hardware. Pipelining increases the encryption speed by processing multiple blocks of data simultaneously. This is

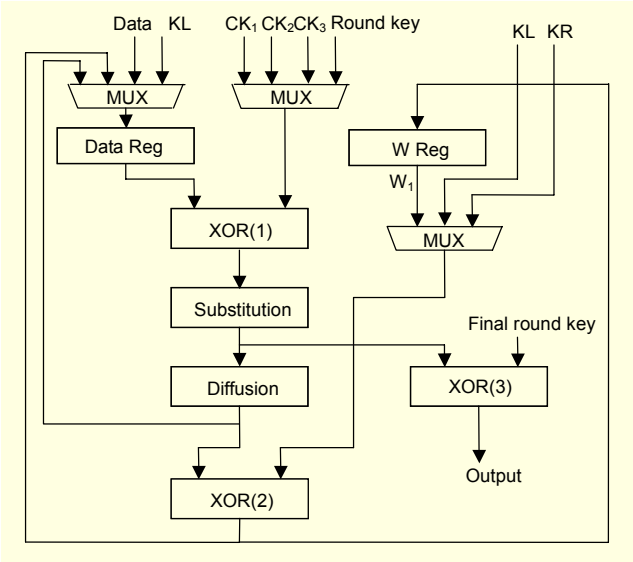


Fig. 4. Block diagram of the loop architecture.

achieved by inserting registers in among the round units. The sub-pipelined architectures with inserted registers in both the inner and outer rounds are then presented.

1. Loop Architecture

The loop architecture unrolls only one full cipher round and iteratively loops data through this round unit until the entire encryption or decryption transformation is completed. Only one block of data is processed at a time. Therefore, the loop architecture is suitable for the feedback modes of ARIA.

Figure 4 shows the loop architecture of ARIA, in which one round operation per clock cycle is performed. It has a 128-bit XOR array, a substitution block, and a diffusion block. The substitution block can be implemented by LUT or the combinational logic using composite field arithmetic. The substitution logic is controlled to execute a different substitution procedure for even or odd rounds, as mentioned in the previous section. If a multiplexer is used to control the inputs of the S-boxes and their inverses, then it is possible to share the S-boxes and their inverses with the even and odd rounds. Thus, the round unit in the loop architecture has one substitution layer.

In addition, the key initialization procedure can be executed using the proposed round function block. To share the resources, the round function block consists of an additional 128-bit XOR array, as denoted by XOR(2), W registers, and an additional multiplexer to select the constant values of CK_1, CK_2, and CK_3 for key initialization. Three clock cycles are required to compute the key initialization values of W_0, W_1, W_2, and W_3, and 12, 14, and 16 clock cycles are necessary for the round

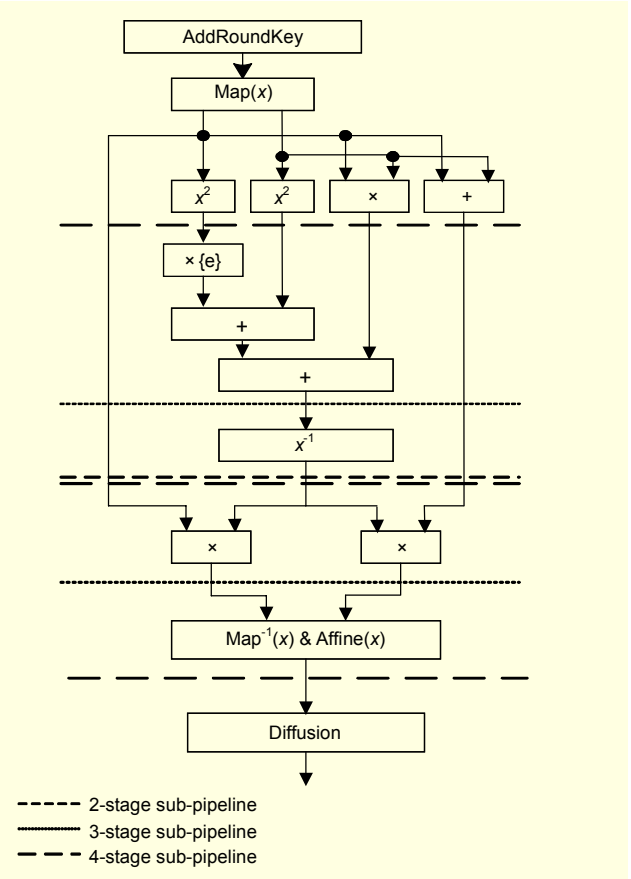


Fig. 5. Sub-pipelined round unit.

operations of 128-bit, 192-bit, and 256-bit key lengths, respectively. As the diffusion step is replaced by the AddRoundKey step in the final round, extra 128-bit XOR gates, denoted by XOR(3), are required in order to perform the final round in one clock cycle. In Fig. 4, KL denotes the left 128 bits of the master key and KR denotes the right 128 bits of the 256-bit key.

2. Sub-pipelined Architecture

In the non-feedback modes, pipelining can be used to achieve high throughput. There are two layers that can be pipelined in ARIA: the inner and outer rounds.

The throughput is calculated according to the clock frequency and the number of clock cycles for encryption or decryption. If one 128-bit data word is encrypted every clock cycle, the throughput is defined by 128 times the clock frequency for the 128-bit data. Therefore, the loop unrolling of all the rounds and pipelining architecture can achieve a throughput that is twelve times higher than the loop architecture for a 128-bit key. This is due to the removal of all the loops to form a loop-unrolled design in which the data is

processed in multiple round units and the insertion of pipeline registers in each round unit. As the loop unrolled and pipelined architecture consumes a considerable amount of hardware resources, implementation of the S-boxes with combinational logic is more suitable than implementation of the LUT scheme. Essentially, the 12th, 14th, and 16th round units differ from the others because they involve an additional AddRoundKey step instead of a diffusion layer.

Although implementation of the S-boxes using composite field arithmetic is highly area-efficient, it involves a long delay compared to other layers. This would greatly degrade the performance of ARIA. To overcome this obstacle, further pipelining can be implemented in the inner round unit. However, dividing each round unit into an arbitrary number of sub-stages does not always speed up the process. Because the minimum clock period is set according to the longest path in the design, inserting additional pipelining registers in the remainder of the round unit with a shorter delay does not further shorten the clock period. Therefore, the sub-pipelined architecture can provide the highest throughput when each round unit is split into multiple sub-stages with equal delays.

Figure 5 describes the critical path of the sub-pipelined round unit. The sub-pipeline cuts are determined through full round operation as AddRoundKey, Substitution, and Diffusion. With careful investigation of the delay in each layer, the round unit can be divided into two, three, or four sub-stages. In the two-stage sub-pipelined architecture, pipeline registers are inserted after the inversion of GF(2^4) because the inversion unit is the most time-consuming operation. In the three-stage sub-pipelining round unit, however, the first pipeline cut is located before the inversion of GF(2^4), and the second pipeline cut is placed before the merged isomorphic map with an affine transformation in order to ensure the same amount of delay. In the four-stage sub-pipelined architecture, two additional pipeline registers are inserted to break down the delay of the previous two-stage pipelining round unit.

V. Implementation Results and Analysis

The proposed hardware architectures for the ARIA algorithm were implemented in Verilog HDL, and the Synopsys Design Compiler was used with a 0.25 μm CMOS standard cell technology library. Table 2 shows the gate counts and the throughputs of the two loop architectures using LUT-based S-boxes or composite-field-based S-boxes with the on-the-fly key scheduler. In Table 2, LUT denotes the LUT-based S-box implementation, while Comp. denotes the composite-field-based implementation. The encryption and decryption of a 128-bit data block with a 128-bit key require 15 clock cycles including the key initialization step which requires three clock cycles. The LUT-based architecture shows a 52% higher throughput than the composite-field-based architecture because the critical path of delay is shorter. However, the area cost shows an increase of 16% compared to the composite-field-based architecture. Table 2 also shows some other loop architectures for comparison. It can be seen that our work offers better performance than those previously reported in the literature. It appears that Koo and others [12] did not include the area of the key scheduler, some of the registers, or selection logics for the inputs. In addition, they only implemented ARIA with a 128-bit key. Park and others [10] and Yang and others [11] proposed an area-efficient architecture based on a 32-bit data path and 16-bit data path, respectively. Their results have a small area but with a very low throughput. Park and others [10] also reported a 128-bit architecture that has a throughput of 757 Mbps. However, their area cost is 43,760 gates because they use LUT-based S-boxes for both the even and odd rounds.

Table 2. Performance of the loop architectures.

Design	Gate count	Freq. (MHz)	Key len. (bit)	Lat.	Thrt. (Mbps)	Thrt./area (kbps/gate)	Tech. (μm)
LUT	25,427	147	128	15	1,254	49.31	0.25
			192	17	1,106	43.49	
			256	19	990	38.93	
Comp.	21,757	97	128	15	827	38.01	
			192	17	730	33.52	
			256	19	653	30.01	
[10] 32-bit	13,893	71	128	356	25	1.79	0.35
[10] 128-bit	43,760	71	128	12	757	17.29	0.35
[11] 16-bit	6,840	15	128	488	3.93	0.57	0.25
[12] 128-bit	15,496	102	128	16	816	52.65	0.25

Table 3. Performance of the pipelined architectures.

Design	Gate count	Freq. (MHz)	Throughput (Mbps)	Throughput / area (kbps/gate)
LUT – pipeline	217,579	203	25,984	124
Comp. – pipeline	181,093	126	16,128	96
2 stage sub-pipeline	203,114	233	29,184	143
3 stage sub-pipeline	216,221	306	39,263	181
4 stage sub-pipeline	260,354	338	43,338	166

Table 4. Performance comparison of this work with the existing AES implementations.

Design	Gate count	Freq. (MHz)	Thrt. (Mbps)	Thrt. / area (kbps/gate)	Tech. (μm)
This work: Loop	25,427	147	1,254	49.31	0.25
[28]	31,957	100	610	19.08	0.25
[29]	119,000	166	2,120	17.81	0.25
[27]	34,300	330	3,840	111.95	0.18
[13]	173,000	154	1,600	9.24	0.18
This work: 4 stage sub-pipeline	260,354	338	43,338	166	0.25
[24]	225,000	445	57,000	253	0.18

Having said that, our proposed loop architecture achieves the highest throughput of 1,254 Mbps with 25,427 equivalent gates for ARIA with a 128-bit key.

Table 3 describes the implementation results of the pipelined architectures with the fully parallel round key generator. To evaluate the effect of the S-box implementation scheme in the pipelined architectures, the result of the LUT-based S-box implementation is also described. In the case of the loop unrolled and pipelined architectures, which only use the outer round pipelining, the LUT-based approach still gives higher throughput compared to that of the composite-field-based scheme. However, the LUT-based S-box cannot be pipelined due to its undivided delay; therefore, we implemented the sub-pipelined architecture using composite field arithmetic to achieve higher performance. Area-throughput trade-offs are determined according to the number of sub-pipeline stages, as this increases the number of pipelining registers in the overall design. A throughput of 43 Gbps was achieved with the four-stage sub-pipelined architecture at the expense of 260,354 equivalent gates. This is the highest throughput in ARIA hardware implementation to date.

Table 4 compares the performance of the two proposed ARIA processors with the existing AES implementations. It is difficult to compare them directly because AES and ARIA are different algorithms in spite of their similarity, and they were implemented with different technologies. For loop architectures, our fast loop architecture is presented in Table 4. The implementation of Hodjat and others [27] has the highest throughput of the previous iterative AES implementations. The performance of our loop architecture of ARIA is slower than that of [27]. However, we should consider that [27] used a more advanced technology library. In particular, the encryption and decryption procedures of ARIA have two more rounds than AES for the same key sizes. For example, ARIA has 12 rounds, but AES has 10 rounds for a 128-bit key. Therefore, the throughput of ARIA in the iterative architectures should be smaller than that of AES with the same clock frequency and key size. However, in the decryption procedure with the on-the-fly key scheduler, ARIA has faster throughput than AES because ARIA does not need extra clock cycles to generate round keys for decryption.

In addition, the pipelined implementations are compared. In reference [24], the throughputs of 30 Gbps to 70 Gbps were achievable in 0.18 μm CMOS technology according to architectural optimization and variable synthesis options. We described the performance of the architecture proposed in [24], which was the same four-stage sub-pipelined architecture as this work in Table 4. Upon comparison, we conclude that our effort to design high-speed hardware architectures for ARIA has been reasonably successful.

VI. Conclusion and Future Work

This article presented efficient high-speed hardware architectures for block cipher ARIA. The two methods of LUT-based or a composite-field-based implementation of S-boxes were evaluated in terms of area-throughput trade-offs. We gave a detailed explanation of the composite-field-based S-box implementation over $GF(2^4)^2$, specifically pertaining to S_2.

Loop architectures for feedback modes and pipelined architectures for non-feedback modes were proposed. The proposed loop architectures showed a 65% increase in throughput over previous designs and achieved a throughput of 1,254 Mbps with 25,427 equivalent gates with a 128-bit key. In addition, fully pipelined high-speed ARIA processors that can provide a throughput of 43 Gbps were presented. This sub-pipelined architecture is implemented by loop unrolling along with inner and outer round pipelining in order to reduce the critical path delay and to increase the maximum throughput. A substitution layer was implemented using combinational logic to avoid the undivided delay of LUTs. Upon comparison, the implementation presented in this work was shown to achieve better performance than previously reported implementations.

Secure implementation of cryptographic algorithms is another important issue in addition to area-throughput trade-offs [9], [30]-[34]. The architectures in this paper would be vulnerable to power analysis attacks. While we have focused on performance and efficiency in this paper, the countermeasures of power analysis attacks would be an important research area. In [9], a masking countermeasure was presented and its second-order side channel resistance was analyzed by using various suitable preprocessing on FPGA implementations of ARIA. Investigation of their countermeasures on our ARIA hardware architectures would

be an interesting work.

In addition, we believe that a power-efficient design in restricted environments such as PDAs and smart cards would be another important topic in relation to ARIA hardware implementations.

Architectural optimization approaches for implementations supporting the modes of operation that have been recently published, such as OCB and GCM, require further study [35], [36]. In particular, the CTR mode is used for encryption in the GCM mode. Therefore, the proposed sub-pipelined architectures could be a good building block in the GCM mode if ARIA is used as an encryption algorithm.

References

[1] NSRI: *Specification of ARIA*, available at: http://www.nsri.re.kr/ARIA/doc/ARIA-specification-e.pdf

[2] D. Kwon et al., "New Block Cipher: ARIA," *Proc. of ICISC-LNCS 2971*, Nov. 2003, pp. 432-445.

[3] FIPS pub. 197: *Specification for the Advanced Encryption Standard (AES)*, Nov. 2001, available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[4] PKCS #11 v2.20 Amendment 3 Rev. 1, *Additional PKCS#11 Mechanisms*, available at: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20a3.pdf.

[5] NIST Special Publication 800-38A: *Recommendation Block Cipher Modes of Operation Methods and Techniques*, 2001, available at http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[6] NSRI: *Security and Performance Analysis of ARIA*, available at: http://www.nsri.re.kr/ARIA/doc/ARIA-COSICreport.pdf.

[7] H. Yoo et al., "Investigations of Power Analysis Attacks and Countermeasures for ARIA," *Proc. of WISA 2006 - LNCS 4298*, 2007, pp. 160-172.

[8] J. Ha et al., "Differential Power Analysis on Block Cipher ARIA," *Proc. of HPCC - LNCS 3726*, 2005, pp. 541-548.

[9] C. Kim, M. Schläffer, and S. Moon, "Differential Side Channel Analysis Attacks on FPGA Implementations of ARIA," *ETRI Journal*, vol. 30, no. 1, Apr. 2008, pp. 315-325.

[10] J. Park et al., "Low Power Compact Design of ARIA Block Cipher," *Proc. of ISCAS*, May 2006, pp. 313-316.

[11] S. Yang, J. Park, and Y. You, "The Smallest ARIA Module with 16-Bit Architecture," *Proc. ICISC-LNCS 4296*, 2006, pp. 107-117.

[12] B. Koo et al., "Design and Implementation of Unified Hardware for 128-Bit Block Ciphers ARIA and AES," *ETRI Journal*, vol. 29, no. 6, Dec. 2007, pp. 820-822.

[13] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and Performance Testing of a 2.29-GB/s Rijndael Processor," *IEEE J. Solid-State Circuits*, vol. 38, Mar. 2003, pp. 569-572.

[14] H. Kuo and I. Verbauwhede, "Architectural Optimization for 1.82Gbits/sec VLSI Implementation of the AES Rijndael Algorithm," *Proc. of CHES, LNCS 2162*, 2001, pp. 51-64.

[15] D. K. Kim et al., "Design and Performance Analysis of Electronic Seal Protection Systems Based on AES," *ETRI Journal*, vol. 29, no. 6, Dec. 2007, pp. 755-768.

[16] F. Standaert et al., "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," *Proc. CHES-LNCS 2779*, 2003, pp. 334-350.

[17] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," *Proc. CHES-LNCS 2779*, 2003, pp. 319-333.

[18] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC Implementation of the AES SBoxes," *Proc. of CT-RSA - LNCS 2271*, 2002, pp. 67-78.

[19] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES Algorithm," *Proc. of CHES – LCNS 3156*, 2004, pp. 357-370.

[20] A. Satoh et al., "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Proc. of ASIACRYPT - LNCS 2248*, 2001, pp. 239-254.

[21] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES Implementation on a Grain of Sand," *IEE Proc. Information Security*, vol. 152, no. 1, 2005, pp. 13-20.

[22] V. Rijmen, "Efficient Implementation of the Rijndael S-Box," available at www.iaik.tugr_az.at/RESE ARCH/krypto/AES/old/~rijmen/rijndael/sbox.pdf

[23] X. Zhang and K.K. Parhi, "High-Speed VLSI Architectures for the AES Algorithm," *IEEE Trans. VLSI System*, vol. 12, no. 9, Sept. 2004, pp. 957-967.

[24] A. Hodjat and I. Verbauwhede, "Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors," *IEEE Trans. Computers*, vol. 55, no. 4, Apr. 2006, pp. 366-372.

[25] T. Good and M. Benaissa, "Pipelined AES on FPGA with Support for Feedback Modes (in a Multi-channel Environment)," *IET Information Security*, vol. 1, no. 1, Mar. 2007, pp. 1-10.

[26] X. Zhang and K.K. Parhi, "On the Optimum Construction of Composite Field for the AES Algorithm," *IEEE Trans. Circuits and Systems*, vol. 53, no. 10, Oct. 2006, pp. 1153-1157.

[27] A. Hodjat et al., "A 3.84 Gbits/s AES Crypto Coprocessor with Modes of Operation in a 0.18-um CMOS Technology," *Proc. 15th ACM Great Lakes Symp. VLSI*, Apr. 17-19, 2005, pp. 60-63.

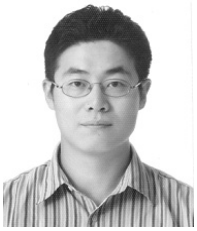
[28] C.C. Lu and S.Y. Tseng, "Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter," *Proc. Application-Specific Systems, Architectures and Processors*, 2002, pp. 277-285.

[29] F.K. Guürkaynak et al., "A 2 Gb/s Balanced AES Crypto-chip Implementation," *Proc. 14th ACM Great Lakes Symp. VLSI*, 2004, pp. 39-44.

[30] http://www.iaik.tu-raz.ac.at/research/krypto/AES/#hardware.

[31] M. Asim and V. Jeoti, "Efficient and Simple Method for Designing Chaotic S-Boxes," *ETRI Journal*, vol. 30, no. 1, Feb.

2008, pp. 170-172.
[32] T. Kim et al., "Power Analysis Attacks and Countermeasures on η_T Pairing over Binary Fields," *ETRI Journal*, vol. 30, no. 1, Feb. 2008, pp. 68-80.
[33] T. Kim et al., "Differential Power Analysis on Countermeasures Using Binary Signed Digit Representations," *ETRI Journal*, vol. 29, no. 5, Oct. 2007, pp. 619-632.
[34] H. Kim et al., "Hyperelliptic Curve Crypto-Coprocessor over Affine and Projective Coordinates," *ETRI Journal*, vol. 30, no. 3, June 2008, pp. 365-376.
[35] P. Rogaway, M. Bellare, and J. Black, "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption," *ACM Trans. Information and System Security (TISSEC)*, vol. 6, no. 3, Aug. 2003, pp. 365-403.
[36] NIST Special Publication 800-38D: *Recommendation Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, Nov. 2007, available at http://csrc.nist.gov/publications/nistpubs/800-38D/SP800-38D.pdf



Sang-Woo Lee received his BS and MS degrees in electronics from Kyungpook National University, Daegu, Rep. of Korea, in 1999 and 2001, respectively. Currently, he is a PhD student in electronics at Kyungpook National University. Since 2001, He has been a senior member of engineering staff with Electronics and Telecommunications Research Institute (ETRI). His research interests include information security based on cryptography, hardware architectures for cryptographic algorithms, and secure embedded systems.

Sang-Jae Moon received his BS and MS degrees in electronics from Seoul National University, Rep. of Korea, in 1972 and 1974, respectively. He received his PhD in communication engineering from the University of California, Los Angeles, USA, in 1984. Currently, he is a professor with the School of Electrical Engineering and Computer Science, Kyungpook National University, Rep. of Korea. He is also the director of the Mobile Network Security Technology Research Center (MSRC). He is an honorary president of the Korea Institute of Information Security and Cryptology. His current research interests are information security in mobile, ubiquitous, and RFID networks including the physical security of smart IC cards. He took part in the Korea Certificate-Based Digital Signature Algorithm (KCDSA) Standard project. He has a number of issued patents and more than one hundred technical publications in international journals and conferences in the area of information security.

Jeong-Nyeo Kim received her BS degree in computer science from Chungnam National University, Rep. of Korea, in 1987. She received her MS and PhD degrees in computer engineering from Chungnam National University, Rep. of Korea, in 2000 and 2004, respectively. She studied computer science at the University of California, Irvine, USA in 2005. Since 1988, she has been a principal member of engineering staff with the Electronics and Telecommunications Research Institute (ETRI), where she is currently working as a team leader of the Knowledge-Based Information Security Research Team. Her research interests include network security and secure operating system.