Hindawi Publishing Corporation
Journal of Engineering
Volume 2013, Article ID 785126, 26 pages
http://dx.doi.org/10.1155/2013/785126

Research Article

A Very Compact AES-SPIHT Selective Encryption Computer Architecture Design with Improved S-Box

Jia Hao Kong,[1] Li-Minn Ang,[2] and Kah Phooi Seng[2]

[1] *Department of Electrical and Electronic Engineering, University of Nottingham, Malaysia Campus, 43500 Semenyih, Malaysia*
[2] *School of Engineering, Edith Cowan University, Joondalup, WA 6027, Australia*

Correspondence should be addressed to Jia Hao Kong; keyx9kjh@nottingham.edu.my

Received 31 August 2012; Revised 5 June 2013; Accepted 5 June 2013

Academic Editor: Alfio D. Grasso

The "S-box" algorithm is a key component in the Advanced Encryption Standard (AES) due to its nonlinear property. Various implementation approaches have been researched and discussed meeting stringent application goals (such as low power, high throughput, low area), but the ultimate goal for many researchers is to find a compact and small hardware footprint for the S-box circuit. In this paper, we present our version of minimized S-box with two separate proposals and improvements in the overall gate count. The compact S-box is adopted with a compact and optimum processor architecture specifically tailored for the AES, namely, the compact instruction set architecture (CISA). To further justify and strengthen the purpose of the compact crypto-processor's application, we have also presented a selective encryption architecture (SEA) which incorporates the CISA as a part of the encryption core, accompanied by the set partitioning in hierarchical trees (SPIHT) algorithm as a complete selective encryption system.

1. Introduction

In the year 1972, the National Institute of Standards and Technology (NIST) has identified and further concluded the study of the US government's computer security needs its own standard for encrypting government-class sensitive information. After various proposal submissions which did not meet their vigorous design requirements, a cipher candidate developed in IBM was deemed suitable and the NSA worked closely with IBM to strengthen that algorithm. Eventually, the Data Encryption Standard (DES) was approved as a federal standard in November 1976. From there onwards, the pillar and model of the encryption for data are formed and established as DES having influenced the advancements of the modern cryptography for many years on.

Since cryptographic solutions are often used to offer integrity and security over the transmission of sensitive data in our communication mediums, it is important for them to have consistent and nondecaying cryptographic strength over time. However, the strength of the encryption is weighted on the key itself, resulting in the strength being exploitable given massive computation strength to search for the key within a finite key space. Over time, the advances of computing technology have dramatically improved the computer processing power and have rendered the earlier DES with the small-sized 56-bit key as no longer safe. This is because of the far more superior computing power we have today, compared to those computers in the earlier days when the DES is proposed. This was quickly rectified later by replacing DES with the triple-DES, which is eventually being out-run by the relentless modern computing advancement. And now, the Advanced Encryption Standard (AES) has eventually replaced the triple-DES for the same reason.

The AES was first specified in 2001 by the NIST to come up with a standard encryption algorithm. It has been fully documented and made available in [1]. Before proceeding into the details of the AES algorithmic structures and descriptions, we would like to discuss more the progresses made by other researchers regarding improvements and implementations of AES. When facing design and development issues for applications, the design outcome is often driven and shaped by the application environment requirements. There are

three common application requirements when designing a security system: minimal hardware circuitry, maximum/high throughput, and minimal power consumption. For example, the work in [2–5] focuses on improving the AES's throughput. High throughput is the most important requirement in a high-speed communication or optical link environment. On the other hand, some work in [6–9] presents the least power consumption. In some resource-constrained environments such as the wireless sensor network (WSN), the lifetime of a node is very limited and power is scarce, making the topic about power consumption vital. This encourages researchers to find better circuitry and power harvesting techniques for extending sensor node's lifespan. Most designs have setbacks and tradeoffs; as such, high-throughput circuit sacrifices design area or low-power techniques result to low-throughput or even a high-throughput, low-power consumption circuit that costs an extremely large circuit size and area. All these are highly dependent on the requirements for their intended applications.

In AES, the most resource consuming and the bottleneck section is the S-box. This is because the S-box is essentially a combination of affine, multiplicative inversion and inversion in the finite field GF (2^8) which requires complex computations. The inversion in the finite field GF (2^8) is practically complex, and therefore it is identified as a design bottleneck. The paper written by [10] has mentioned that the non-LUT-based approaches are fairly attractive since they have breakable delay. The author has elaborated that there are two types of S-box designs. Type 1 is a direct circuit generation using truth table, making use of the sum of products (SOP) or product of sums (POS) and usually features higher throughput at expense of extremely large circuit area. On the other hand, type 2 designs feature higher design area efficiency. Type 2 designs are slowly gaining popularity since the design trend has shifted towards searching efficient logic minimization techniques and circuit depth reduction techniques.

The construction of good combinational circuits is important as it affects almost any metric in a digital circuit design we know. The gate count, critical-path delay, clocking and timing, circuitry jitters, and power consumption are discussed when a circuit is designed. In this work, we focus on our development in the area of low gate count and low-resource environment hardware designs, specifically for resource-constrained environments such as the wireless sensor network (WSN), radio frequency identification (RFID) and even the newly developed wireless identification and sensing platform (WISP). In this paper, we discuss the development of our proposed solution in three areas: the review of current S-box design trend towards the low-gate-count approach, the small and compact footprint for AES designs, and lastly the development of a complete system with AES block for a selective encryption architecture (SEA). The structure of the paper is as follows. Section 1 is the introduction of the paper, which introduces some of the key algorithms adapted in our work and other related work for benchmark and comparison. Section 2 is the review of different design approaches for low-gate-count S-boxes. Section 3 presents our version of a small S-box design with 2 approaches: (1) with an additional instruction set; (2) circuit minimization. Section 4 introduces our proposed instruction set computer architecture, namely, the compact instruction set architecture (CISA) with the adaptation of our proposed small S-box design. Section 5 discusses a higher-level of implementation which incorporates the set partitioning in hierarchical trees (SPIHT) compression algorithm as a source to the CISA running AES for a complete selective encryption architecture. Section 6 is the results and discussions section, and lastly, Section 7 is the conclusion.

1.1. Review of the Advanced Encryption Algorithm (AES). The AES, also known as the Rijndael [1, 11], is a block cipher developed by two Belgian cryptographers, Daemen and Rijmen [11]. It is a symmetric block cipher that consists of 128-bit block length and supports 128, 192, and 256 bits of key length with 10, 12, or 14 iterations of AES transformation, respectively. The encryption and decryption operation is a repetition of the substitute permute network (SPN) operation on the input data. The cipher is applied onto a 2-dimensional 4 by 4 state array. It consists of four rows of bytes containing N_b bytes, where N_b is the block length (128) divided by 32.

There are several modes of operations in which the AES can be configured to. Some of these serve different purposes as their functions vary. One of the most common modes of operation is the "Electronic Code Book" (ECB) mode which does not require any feedback loops, and this is fundamentally a complete round of AES encryption without additional tweaks or changes. The other mode is called the "Cipher Block Chaining" which requires the results of the previously encrypted block. The "Output Feedback" mode is effectively a synchronous stream cipher. It generates key stream blocks using an initialization vector and XORed with the respective plain texts to get the complete cipher. The biggest advantage of this mode is that both encryption and decryption rounds only require the "forward" encryption codes.

Fundamentally, the AES has four basic steps in each round of encryption. The four steps are called *SubBytes* (also known as the byte substitution), *ShiftRows*, *MixColumns*, and *AddRoundKey*. The description of the four basic steps in AES rounds is as follows.

(i) *AddRoundKey*: a simple transformation performs XOR with the *subkey* to the round state.

(ii) *ShiftRow*: shifts the byte location with the offset from zero to three depending on the row location.

(iii) *MixColumns*: column vector is multiplied with a fixed matrix where bytes are treated as polynomials.

(iv) *SubBytes*: nonlinear byte substitution which is composed of multiplicative inverse, affine transformation, and inverse affine transformation.

In brief, the first round is the *AddRoundKey*, the subsequent nine rounds include all the four transformations, and the tenth round omits the *MixColumns*. Note that this only applies to the forward encryption, and as for the decryption rounds, the *AddRoundKey* remains unchanged and the rest of transformation sequences are their mathematical reverses,

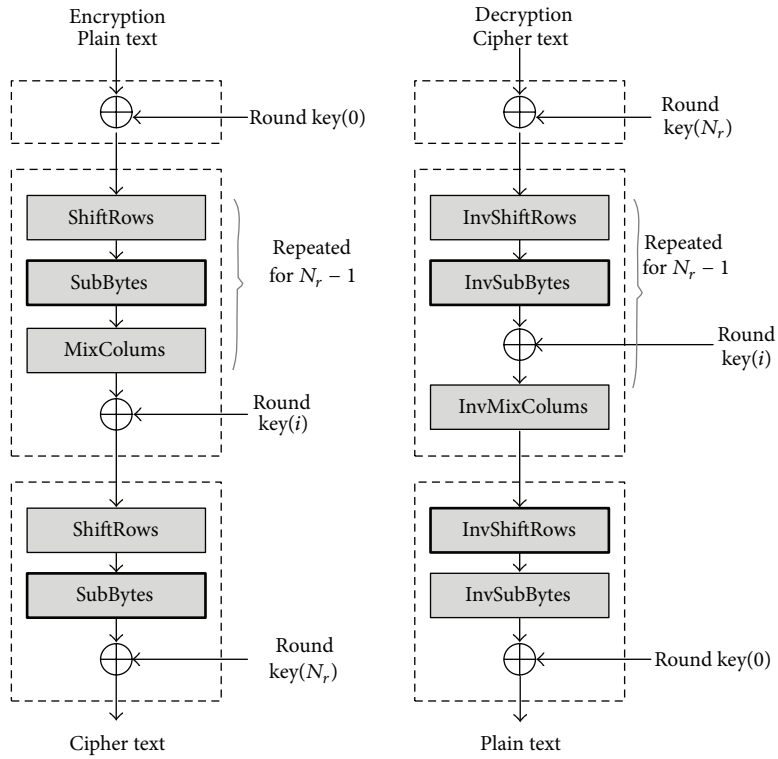

Figure 1: The encryption and decryption processes in AES.

namely, the *InvSubBytes*, *InvShiftRows*, and *InvMixColumns*. Figure 1 shows a block diagram on how the AES works.

2. An Introduction to the AES S-Box (Substitution Box)

The *SubBytes* is unique compared to the other three AES transformations because it is the only nonlinear component in the AES encryption. The *SubBytes* step functions as replacing or substituting an input with another byte, through the "S-box" function. Other than *SubBytes*, the other three transformations are considered modulo 2 bitwise calculations, which can be easily implemented. Conventionally, implementation approach is preferred to storing the values of the S-Box into a ROM and uses it as a look-up table (LUT). Early versions of the S-box circuit are essentially an 8 by 8 look-up table and can be found in the following proposals: [12, 13]. An illustration of the LUT is shown in Table 1.

But for hardware implementations of AES, there is one drawback for the look-up table approach. Each copy of the table requires 256 bytes of storage, along with the circuitry for addressing the table and to fetch the results. The most straightforward way is to store all these values within a memory block. The problem arises when a fully unrolled AES would require 10 rounds of *SubBytes,* and in effect, each byte of data would require an independent S-box. In the end, 160 S-boxes would eventually drain all the available memory. Note that this is assumed to be the worst case of implementation approach and did not consider the pipelining method. Even with the pipelined architecture, the read and write cycle would slow down the architecture. Even though the multiplicative inversion and affine mathematical complexity are hidden by predefining the LUT value and the accesses is merely read and write, the LUT approach has irreducible read-write delays and, therefore, is not suitable for high-speed applications.

On the other hand, some authors suggest that a combinational circuit can be derived using subfield arithmetic. Daemen and Rijmen [11, 15] suggested that using subfield arithmetic in the crucial step of computing an inverse in the Galois field of 256 elements, by reducing an 8-bit input to subcalculations of 4-bit variables, may yield a very small S-box circuit. In [16], the S-box used is derived from the multiplicative inverse over Galois field (2^8). To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation (a matching inverse affine is included in the decryption). Not only the S-box is used in the main AES iterations, it is also shared with the key expansion operation [17]. The key expanding algorithm reuses the forward S-boxes in encryption and decryption. And note that during the AES decryption, the same key expander uses the same forward S-box to generate the round keys. Later on, Satoh et al. [16] further expanded this idea, using the tower-field approach of Paar [18] by suggesting that breaking up the 4-bit calculations into 2-bit variable will result in even smaller circuit blocks.

From the hardware implementation point of view, the search for the multiplicative inversion of GF (2^8) is too complex and resource exhaustive. Being derived from the

Table 1: The look-up table of the 256 substitution values for S-box.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

multiplicative inverse over Galois field (2^8), it is understood that it projects good nonlinearity and may have high hardware complexities. Other than this, in a resource-constrained design environment, this gives a higher impact since implementation is small enough to allow unrolling or parallel designs for higher throughput. Recently, the design trend had shifted to further minimizing and optimizing the S-box circuit [19].

In this section, we will only review the standard hardware proposal and implementation of AES S-box without taking account of the various proposals on variants or tweaks on the AES S-box. We will only focus on the original version of the S-box and its respective minimization techniques; implementation methodologies and design approaches are surveyed and taken into account.

2.1. The Minimized S-Box by Boyar and Peralta. In practice, we build circuit designs using numerous heuristics which potentially led to exponential time complexity which can only be applied onto small-sized circuits. The heuristic approach works naturally fine on circuit function that can be broken down into subfunctions, that is, matrix multiplication, which decomposes into smaller submatrix multiplications. The initial work from Boyar and Peralta [20] proposes a new logic minimization technique, which can be applied to any arbitrary combinational logic problems and even circuits that have been optimized by standard methodologies. The authors described their techniques as a two-step process: nonlinear gate reduction and linear gate reduction. It is by far the smallest S-box combinational circuit that they have come up with. In this section, we are going to review Boyar's first approach in logic minimization which can be found in [20] and his improved work for an even smaller and complete S-box circuit in [21].

In Boyar's paper, the author has carefully explained that the circuit produced for the inverse in GF (2^m) suggested in

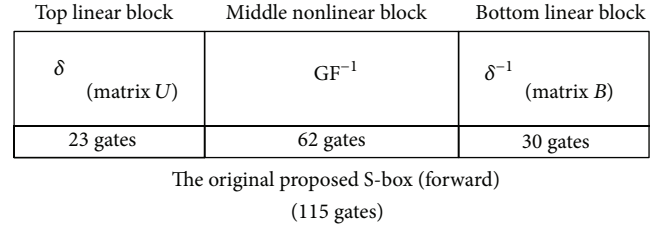

Figure 2: The illustration of Boyar's minimized S-box.

[22] has a tower fields architecture. Since there are multiple representations of Galois fields, there would be multiple versions of efficient circuits. Boyar's approach is to focus on the technique for GF (2^4) inversion computation and then further perform linear circuit's reduction with the inversion circuit placed at a suitable position within the S-box. The first step consists in identifying the nonlinear components and reducing the AND gates. The author chooses to focus on reducing only the GF (2^4) circuit since it would be significantly beneficial. At the end, an inversion in GF (2^4) with only five AND gates poses a higher plausible improvement than Paar's [18].

The second part would be focusing on minimizing linear components with their newly proposed heuristics. Hence, the author has presented two matrices U and B for linear-minimization. The AES's S-box is $S(x) = B * F(U * x) + [11000110]T$, where $*$ is matrix multiplication and x is the 8-bit S-box input. Note that the initial linear expansion and the linear contraction (matrices U and B) were defined to contain as much of the circuit as possible while still maintaining linearity. Thus, the author explains that the portion of the circuit, defined by U, overlaps with the GF (2^8) inversion. So, the true purpose of the second step is to minimize the circuits for computing U and B. The matrices U and B are shown in

(1) and (2). The illustration matrix U (Figure from [20]) is

$$U = \begin{bmatrix} 0&0&0&0&0&0&0&1 \\ 0&1&1&0&0&0&0&1 \\ 1&1&1&0&0&0&0&1 \\ 1&1&1&0&0&1&1&1 \\ 0&1&1&1&0&0&0&1 \\ 0&1&1&0&0&0&1&1 \\ 1&0&0&1&1&0&1&1 \\ 0&1&0&0&1&1&1&1 \\ 1&0&0&0&0&1&0&0 \\ 1&0&0&1&0&0&0&0 \\ 1&1&1&1&1&0&1&0 \\ 0&1&0&0&1&1&1&0 \\ 1&0&0&1&0&1&1&0 \\ 1&0&0&0&0&0&1&0 \\ 0&0&0&1&0&1&0&0 \\ 1&0&0&1&1&0&1&0 \\ 0&0&1&0&1&1&1&0 \\ 1&0&1&1&0&1&0&0 \\ 1&0&1&0&1&1&1&0 \\ 0&1&1&1&1&1&1&0 \\ 1&1&0&1&1&1&1&0 \\ 1&0&1&0&1&1&0&0 \end{bmatrix}. \quad (1)$$

Equation (2) shows the illustration matrix B (Figure from [20]):

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

The Boyar technique has yielded a circuit for the AES S-box composed of three primary parts: the top-linear transformation, the middle nonlinear block, and the bottom-linear transformation [20]. The top-linear transformation is a result of the minimized matrix U, a total of 23 XOR gates used and at depth 7, consisting of 8 inputs and 22 outputs. The middle nonlinear block is a block with 22 inputs and 18 outputs, having a total of 30 XOR and 32 AND gates. And lastly the bottomlinear block converts the 18 inputs from the middle non-linear block to become 8-bit output, having 26 XOR and 4 XNOR gates. All these 3 blocks together form the final circuit of the S-box. Note from the work in [20], the author has only presented the forward version of the S-box, with a total gate count of 115 gates. Figure 2 shows the illustration of the proposed block diagram explaining the S-box in [20].

To further improve the work, the authors have presented their improved work in [21]. Boyar's work has proposed a more complete S-box example, by incorporating the reversed version of the S-box. This time, Boyar attempts to apply a greedy heuristic approach for linear minimization and several depth reduction techniques.

The largest circuit component is the top- and bottom-linear circuits. As explained previously, the top and linear components contain more than just the linear operations in the definition of the complete AES S-box. The reason is that the matrices include some of the field inversion operations. This shows that there would be some amount of AND gates within the U and B matrices, and as mentioned by the author, circuits with fewer AND gates will have larger linear components. This part of the work is optimized on top of the previously minimized circuit (115 gates).

The author's technique is to modify a greedy heuristic approach by Paar [18]. Paar's technique keeps a list of XOR computed variables. Then the steps are repeated to search for the XOR pair of the input which results in the most occurrences in the output. This result is added as a new set of variable to the next stage and repeated until all the most occurred pairs are found. Hence, the "Low_Depth_Greedy" algorithm only allows Paar's greediness as long as the circuit's depth is not increased unnecessarily. Basically, the author has performed the three types of depth-reduction optimizations: (1) applying a greedy heuristics to resynthesize linear components into lower-depth construction of circuits, (2) using techniques from automatic theorem provied to resynthesize nonlinear components, and (3) doing simple depth-reduction along critical paths.

The optimization results have yielded a forward S-box with 128 gates and an inverse S-box with 127 gates. This is considered a significant improvement since the total gate count for a complete bidirectional S-box is amounted to 192 gates, which is less than the total gate count of the two circuits combined. From our understandings, the only tradeoff is to combine both circuits; a multiplexer would be required to switch between encryption and decryption since there is a middle-shared component. Figure 3 shows the illustration of the bidirectional S-box in block diagram form [21].

2.2. The Optimized S-Box by Satoh et al. and the Implementation Results by Edwin. The Rijndael architecture presented by Satoh et al. [16] has been a benchmark for compact AES design for quite some time. The author proposes further optimization of the S-box by introducing a new composite field. The authors adopted the three-stage methodology: extension field, composite field, and extension field. The author has suggested that the composite field can be constructed without applying a single degree-of-8 extension to GF (2), but by applying multiple extensions of smaller degrees. The author built the composite field by repeating the degree-of-2 extensions under the polynomial basis with the irreducible polynomials shown in (3). Hence, Satoh et al. proposed a compact architecture with the introduction of a new composite field of GF $(((2^2)^2)^2)$ and have shown improvement over proposals using the GF $((2^4)^2)$ field approach. Equation (3) shows the irreducible polynomials used in [16]:

$$\begin{aligned} &\mathrm{GF}\left(2^2\right): x^2 + x + 1, \\ &\mathrm{GF}\left(\left(2^2\right)^2\right): x^2 + x + \emptyset, \\ &\mathrm{GF}\left(\left(\left(2^2\right)^2\right)^2\right): x^2 + x + \lambda. \end{aligned} \quad (3)$$

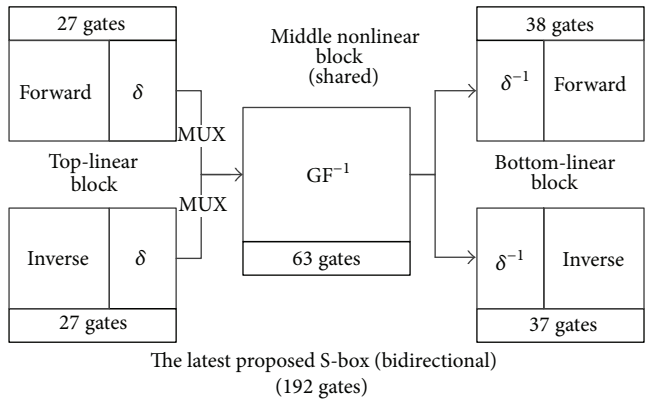

Figure 3: The illustration of Boyar's updated minimized S-box (both forward and inverse S-boxes).

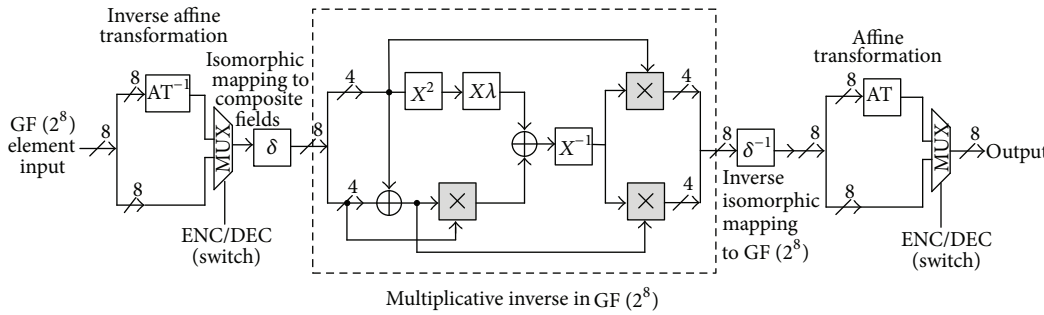

Figure 4: The illustration of the composite field S-box transformation.

Figure 4 shows the overview of the composite field S-box. According to the Satoh et al., the isomorphism functions are located at both ends of the S-box function (both encryption and decryption). The authors have shown the 8 by 8 matrix for the isomorphic mapping into the composite field in Figure 5 and the inverse isomorphic mapping in Figure 6.

For practical implementation, Mui [23] has presented a breakdown of the S-box and the multiplicative inverse GF (2^8) in his paper. The individual blocks within the composite field S-box are shown in Figure 7. With reference to all the subcircuits that Edwin had presented, we have mapped out a circuit excluding isomorphic transformations, showing specifically the circuit layout of the multiplicative inverse in the GF (2^8) in Figure 8. In Figures 8 and 9, we can observe that it utilizes five GF (2^4) multiplier, and each of the blocks uses three GF (2) multipliers. From the schematics that we have drawn, the total gate counts amount to 238. Note that Figure 9 includes only the forward S-box circuit. The total gate count for the bidirectional circuit (excluding the MUX and including the inverse isomorphism circuit) is a total of 261 gates, with inverse isomorphism of 23 gates (referring to Figure 6)

2.3. The Very Compact S-Box by Canright. The work presented by the author Canright [14] aims to find a solution to compute the S-box function by comparing and investigating the normal basis and the polynomial basis inverter. In this section, the authors are not going in depth to explain the design details of Canright's proposed S-box. For design comparison details, please refer to [14]. Table 2 shows the comparison of implementations.

2.4. Other Small S-Boxes. The work presented in [2] is using the same composite field arithmetic approach. The author's contribution is clearly on the division and breakdown of the S-box for subpipelining. The author has also applied the subpipelining architecture on the top-level AES design. This dramatically improves the throughput with a trade-off of larger design size. In Rouvroy et al.'s design [17], the *SubBytes* were combined with *MixColumns* to form a 32-bit "T-box" LUT (18 kbit). This has produced superior throughput however still occupying a relatively large area when the size of the LUT was taken into account. For many applications, throughputs in hundreds of megabits per second would be considered excessive and, therefore, not suitable for resource-constrained environment. And another S-box worth mentioning is the work proposed by Liu and Parhi [10]. Liu and Parhi discussed and broke down various critical path delays within the composite field S-box and attempts to minimize the latency. The authors had presented their findings with improved critical path at the expense of a fairly larger design.

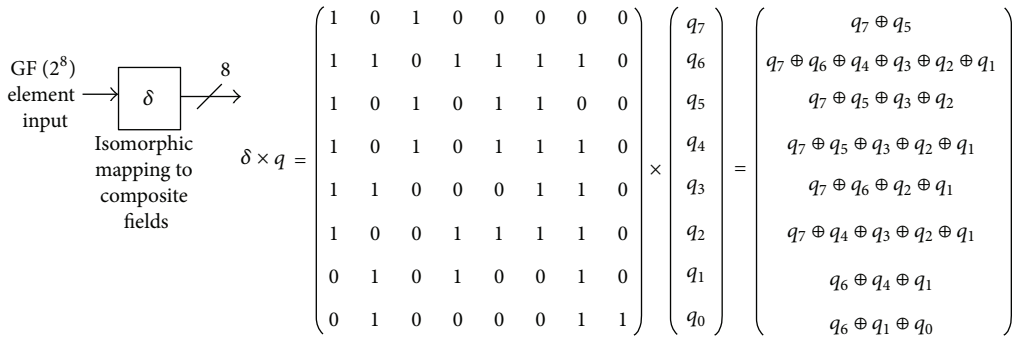

Figure 5: Illustration of isomorphic mapping.

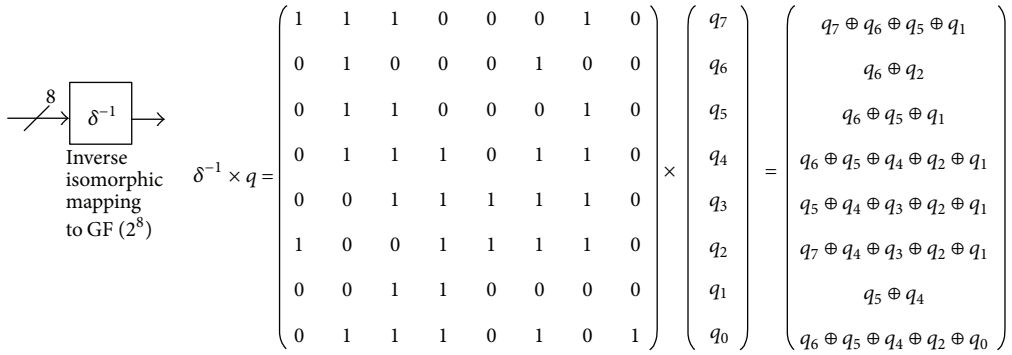

Figure 6: Illustration of inverse isomorphic mapping.

3. The Proposed Methodology for Optimization and Improvement on the Current S-Box

From all the proposals and implementations that we have discussed in the sections above, we have identified a few patterns. The S-box can be presented in the form of tower-field architecture or the GF (2^8) representation. In Boyar's work, the author has broken down the S-box into two linear blocks and one nonlinear block. On the other hand, Satoh's and Canright's works suggest that the GF (2^8) representation can lead to a smaller combinational circuit with the "extension-composite-extension"—three stages. Both represent the S-box but in a different way, but both are logically the same. By comparing both S-boxes, we can see that the function of the extended field in the composite field representation is similar to the linear block of matrices U and B. Since this trait is identified, we choose the smallest S-box design known and apply our method onto the said S-box. To further reduce the size of the S-box, we have proposed two separate methodologies: (1) to use an independent inverse affine circuit and attempt to perform optimization of the circuit; (2) to put the inverse affine as an independent ALU with additional instruction set, without making changes to the current S-box configurations. Note that our approach is to view the S-box as a dedicated component within an ALU of a processor, treating the S-box as a whole and independent hardware.

The Boyar compact S-box offers a total gate count of 115 gates and poses as the smallest forward S-box known. The only drawback is that for a complete bidirectional S-box, the author suggested that a set of "inverse" top- and bottom-linear blocks (Figure 3) has to be added. This increases the gate count by 77 gates (192 gates in total), which is a great amount. And on the other hand, from the composite field point of view, the inverse S-box requires an affine and inverse affine at both ends of the S-box to complete the substitution. This suggests that after a set of bytes underwent the affine transform would only be recovered by applying the inverse affine transform. Both Boyar's and Satoh's S-boxes are completed with the affine transform at the very end of the forward S-box. What we suggest is not to tweak, tamper, or redesign the S-box, but rather adding a small circuit to reverse the effect of the forward S-box, making it a reversible or a bidirectional S-box.

The forward S-box in the composite field has the affine transform in the process while the Boyar triple-stage S-box represents the affine transform "embedded" as a part of the circuit derived from matrix B. From what we see, an inverse affine is the only crucial circuit that determines the inverse S-box. By adding an inverse affine transform at the end of the composite field S-box, we have effectively "cancelled out" the transformation done by the affine transform in the forward S-box. To complete the circuit, another inverse affine transform has to be present at the front to act as the prime component for the inverse S-box. Since we have mentioned

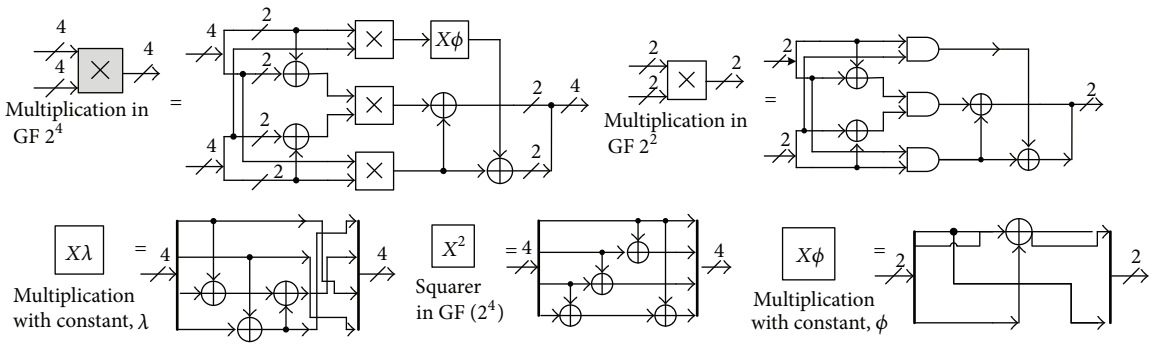

Figure 7: Individual blocks within the composite field S-box.

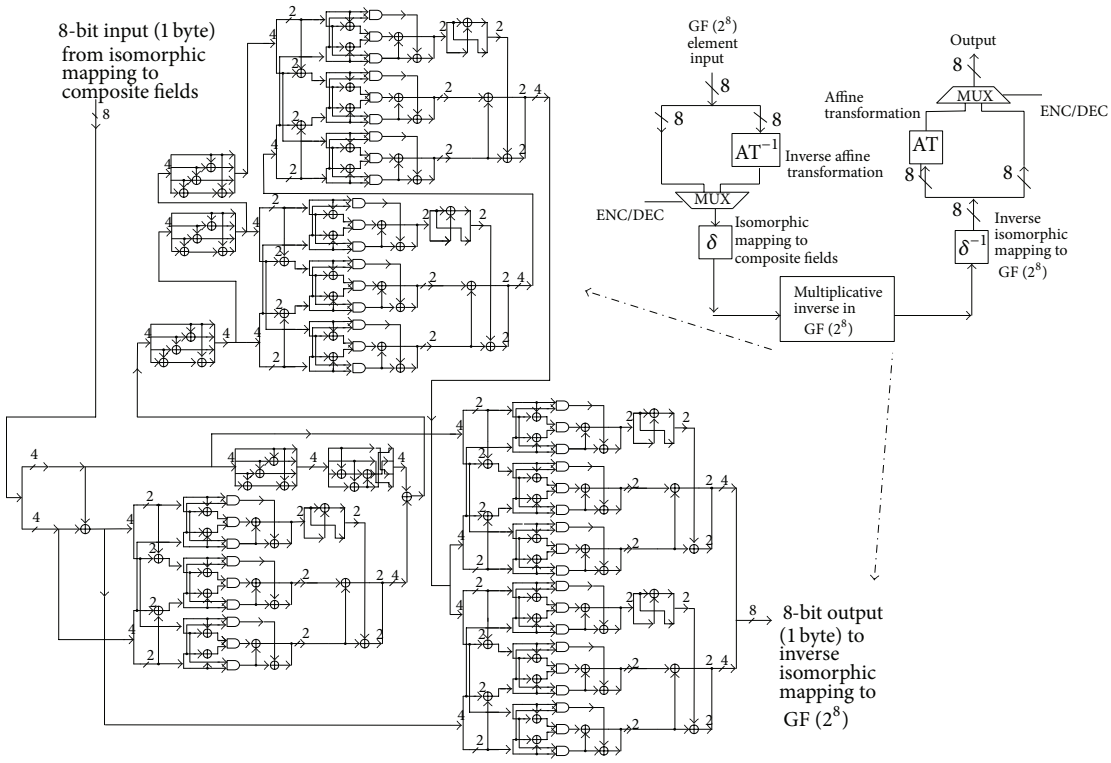

Figure 8: The schematic circuit for the multiplicative inverse of GF (2^8) of the *SubBytes*.

that the smallest S-box still have room for additional circuits (since it is smaller than other S-boxes by a relatively large margin), we suggest that this "inverse affine transform" can be placed at both ends of the Boyar S-box. This results in a complete bidirectional S-box. Figure 11 illustrates the said configuration. Note that this proposal would still require the presence of MUXs to choose the path of the data during encryption and decryption mode selection. In the next section, we will provide more details on the improvements and the proposed optimization techniques. The complete gate layout of our proposed S-box configuration for bidirectional setting is shown in Figure 10.

3.1. The Optimized Inverse Affine Circuit. As we know, in the composite field forward S-box, we have affine and inverse affine transformation at both ends of the circuit. As for the inverse affine transform, the matrix is shown in Figure 12.

In search of an optimization method, we have found out that the work proposed by Bernstein [24] is a good method to optimizing linear matrix mapping. The author has also provided a .cpp file (C++ file) in his website, which is a direct implementation of his algorithm. For more information about the linear maps optimizations, please refer to the paper [24]. We have adopted his algorithm and attempt to feed the matrix into the optimization algorithm for optimization results. The results obtained are shown in a straight-line layout, in other words, the shortest path of calculation. Equation (4) shows the straight-line XOR calculations obtained from the optimization algorithm. The current gate count is 18 XOR

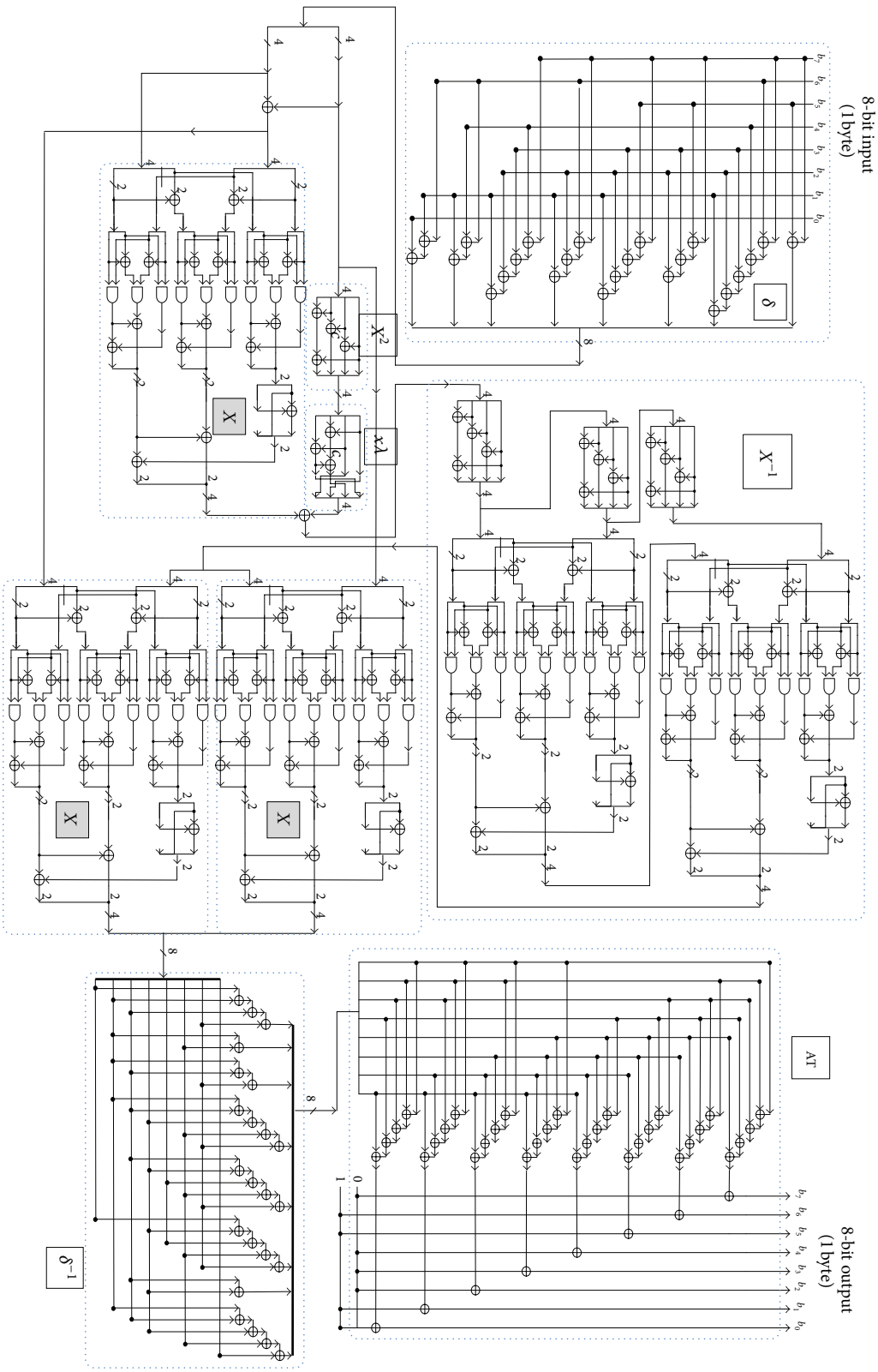

Figure 9: The complete schematic circuit for the forward *SubBytes* with a total gate count of 238.

Table 2: The comparison of S-boxes (table from [14]).

Basis	Type	XOR	NAND	NOT	MUX	Total gates
Canright	Merged	**107**	**36**	**2**	**16**	**253**
	S-box	91	36	0	0	195
	Inv S-box	91	36	0	0	195
Mentens	Merged	**118**	**36**	**0**	**16**	**271**
	S-box	96	36	0	0	204
	Inv S-box	97	36	0	0	206
Satoh	Merged	**119**	**36**	**3**	**16**	**275**
	S-box	100	36	0	0	211
	Inv S-box	99	36	0	0	209
Worst	Merged	**131**	**36**	**0**	**16**	**293**
	S-box	107	36	0	0	223
	Inv S-box	106	36	0	0	222

gates. Equation (4) shows the straight-line circuits for the inverse affine transform:

$$\begin{aligned}
a_0 &= x_0 + x_1, & a_6 &= x_1 + x_4, & a_{12} &= x_6 + a_3,\\
a_1 &= x_0 + x_2, & a_7 &= a_4 + a_6, & a_{13} &= a_0 + a_{12},\\
a_2 &= a_0 + a_1, & a_8 &= x_0 + a_7, & a_{14} &= a_7 + a_{13},\\
a_3 &= x_0 + x_3, & a_9 &= x_5 + a_1, & a_{15} &= x_7 + a_6,\\
a_4 &= a_0 + a_3, & a_{10} &= a_5 + a_9, & a_{16} &= a_2 + a_{15},\\
a_5 &= a_2 + a_4, & a_{11} &= a_8 + a_{10}, & a_{17} &= a_{11} + a_{16}.
\end{aligned} \tag{4}$$

This initial form of circuit uses a total of 18 XOR gates. By sorting out the variables, we have realized that they can be minimized by expanding the equations shown in (4). The mapped-out equations show that there are only 8 outputs at the end, and the equations are shown in (5). The current gate count is 16 XOR gates. Note that from this point onwards, the optimization is done by "hand optimization" since it is a small circuit. Equation (5) shows the minimized equations for the inverse affine transform:

$$\begin{aligned}
b_0 &= x_1 + x_3 + x_6, & b_4 &= x_2 + x_5 + x_7,\\
b_1 &= x_2 + x_4 + x_7, & b_5 &= x_0 + x_3 + x_6,\\
b_2 &= x_0 + x_3 + x_5, & b_6 &= x_1 + x_4 + x_7,\\
b_3 &= x_1 + x_4 + x_6, & b_7 &= x_0 + x_2 + x_5.
\end{aligned} \tag{5}$$

Equation (6) shows the minimized equations for the inverse affine transform:

$$\begin{aligned}
u_0 &= x_1 + x_4, & u_2 &= x_0 + x_5,\\
u_1 &= x_3 + x_6, & u_3 &= x_2 + x_7.
\end{aligned} \tag{6}$$

From here, we further optimize it via variable grouping. Under careful observation, we realized that the circuits have common bases. Meaning, there are XOR operations that can be omitted without contributing to more XOR gates. In (6), we have the common bases that are first acquired and expanded into their respective outputs. Hence, the new circuit is mapped and shown in Figure 13. Note that there is a constant addition at the end of the inverse affine transform, and this requires 2 extra XOR gates. The final circuit is shown in Figure 13, and we have optimized the gate count to a total of 14 gates. The whole circuit can be understood by referring to (4), (5), and (6). Substituting the alternate bases would yield the final circuit.

3.2. Methodology 1: Inverse Affine Transform with Multiplexer. In this section, we propose a methodology by adapting the existing S-box model by Boyar and Peralta [20, 21]. If this is implemented, it will amount to a total of 143 gates (excluding MUX 16 gates). The final circuit is straight-line circuit (with one inverse affine block at both ends) and it is depicted in Figures 10 and 11. Another alternative can be realized considering only one minimized inverse affine transform used. A temporary register can be used to store the results from the S-box, and the same data is fed back to the inverse affine circuit. This method is clearly sharing the inverse affine circuit with the help of using registers (effective total gate count = 129), but the tradeoff is that the circuit now is two times slower. Figure 14 illustrates the alternative method register buffering.

3.3. Methodology 2: S-Box Breakdown (Affine Transform as an Independent ALU). In this section, we present a second approach for improving the gate count of the S-box. In a processor architecture, the ALU is the block that defines the behavior of the machine. In the section later, we will discuss more about the ALUs' role in a computer organization. This configuration yields a final gate count of 129. Figure 15 illustrates the ALU proposal for an independent ALU set for the minimized inverse affine transform. The only tradeoff is that another instruction set has to be defined in the architecture, and it does not comply with the aim of having the least and minimal number of instruction sets for a simple architecture. Note that the methodologies 1 and 2 are aided by the processor architecture. This can be understood by the placement of the minimized inverse affine transform, revolving around the ALU block.

4. The Compact Instruction Set Architecture (CISA) for AES

The ideology of a computing machine appears to be the relationship between the basics of the logical and arithmetic operations. Thus, a computer can be described as the abstraction of the data processing mechanism. In this section, we are presenting a minimalist design of the compact instruction set architecture (CISA) built for AES, explaining details from the hardware architecture to the instruction set synthesis.

4.1. A Brief Review of the Idea behind the CISA: The Ultimate Reduced Instruction Set Computer (URISC). When we use the term "instruction set," it describes a set of commands that elaborates the computer's functionality, behavior, and operations. Most instruction sets make references to memory

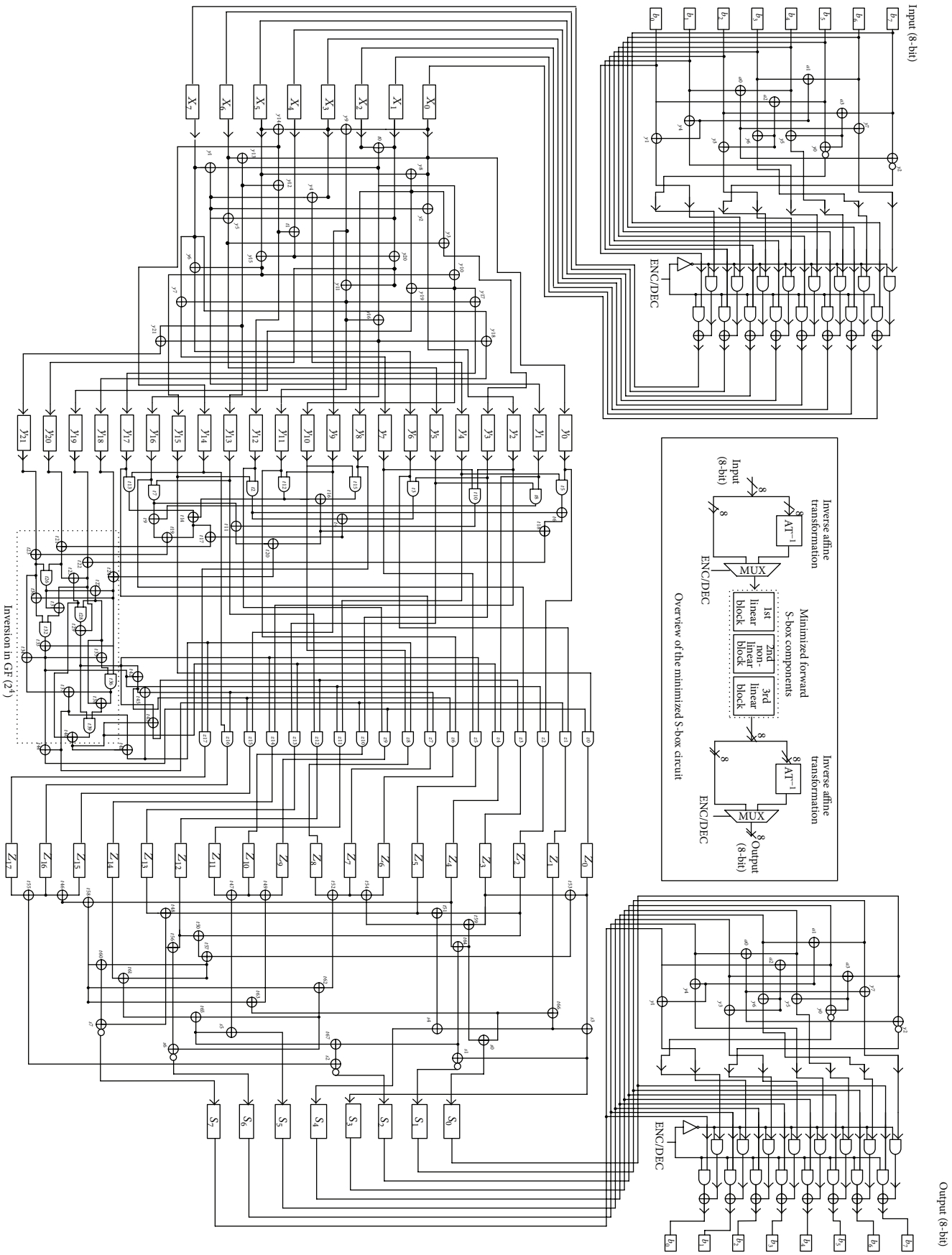

Figure 10: The complete gate layout of the proposed S-box configuration for bidirectional setting.

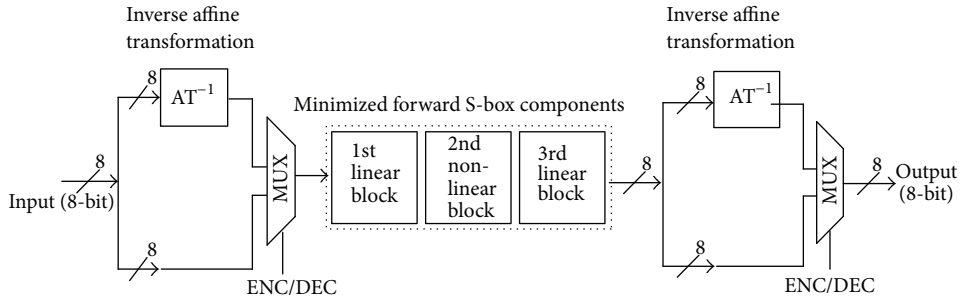

Figure 11: The illustration of the placement of the additional inverse-affine blocks in the Boyar's S-box.

$$\text{Inverse affine transformation: } \mathrm{AT}^{-1}(a) = \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

Figure 12: The matrix for inverse affine transform.

locations, registers, or pointers to a memory location. The referenced memory locations will eventually contain the processed data, which will be used again to produce new data if the next instruction set is defined to do so. In short, computer processors can be viewed as a machine, taking in data and performing transformations and operations onto existing data to become new data, via executing and interpreting instruction codes.

To understand computer architecture, one version of computer organization had caught our attention and it is named the ultimate reduced instruction set computer (URISC). The URISC is known to be an educational model [25] because it is made to be easily understood by novice students. Hence, the design and architecture are simple and yet the idea is grand. The URISC is neither a RISC (reduced instruction set computer) nor a CISC (complex instruction set computer), but it does project similar traits like RISC and CISC in some areas. The idea of URISC is the opposite of a CISC, which incorporates many complex instructions as microprograms within the processor. But at a higher level, a URISC with many heavily synthesized low-level instructions is also a CISC. While for RISCs, the most common features are the single instruction size, small number of addressing modes, and without indirect addressing. These changes made it possible to develop successfully a new set of architectures with simpler instructions. The RISC-based machines focused the attention of designers on two critical performance techniques: the exploitation of instruction-level parallelism (initially through pipelining and later through multiple instruction issue) and the use of caches (in simple forms initially and using more sophisticated organizations and optimizations later).

This simplified model of computer architecture being flexible with only a single instruction incorporated can be further expanded and implemented on hardware easily. The URISC uses only one instruction called the SBN instruction (Subtract and Branch If Negative). By using only the SBN instruction, the URISC is able to perform data addition and subtraction. Logical operations can be performed to execute data movement from one location to another. The URISC consists of an Adder circuit as its sole ALU. Detailed operation of the URISC SBN can be found in [26]. Figure 16 shows the schematic illustration of the URISC SBN architecture.

The "Subtract and Branch if Negative" (SBN) processor was first proposed by Gilreath and Laplante [26]. With this primitive SBN instruction, the URISC is built around this fundamental instruction. The basic operations of URISC are moving operands to and from the memory, with addresses corresponding to the registers. The arithmetic computation can be performed and the results are stored in the 2nd operand's memory location. Similarly, to execute URISC instructions, the core subtracts the 1st operand from the 2nd

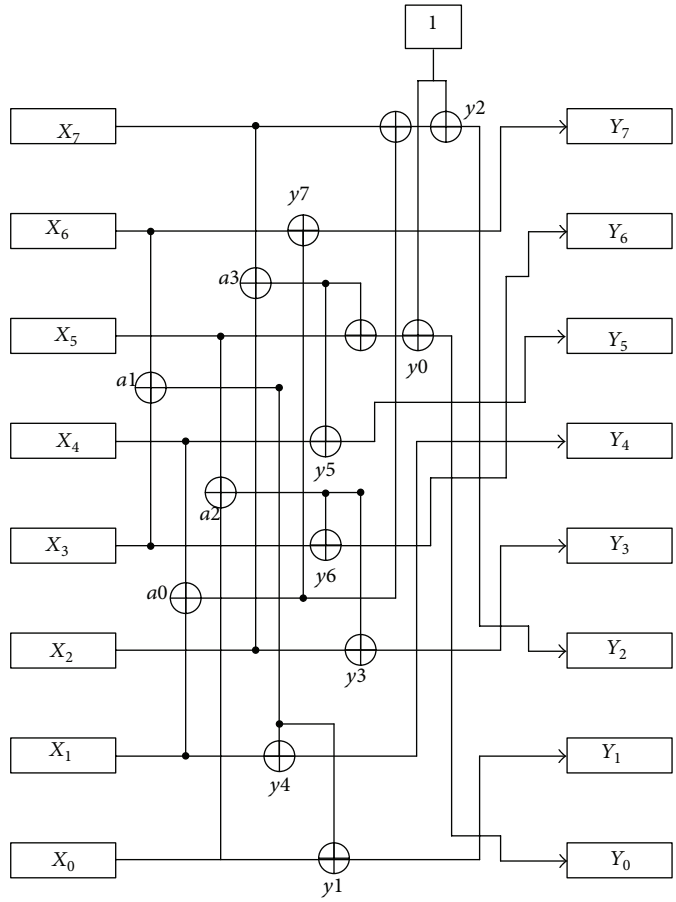

Figure 13: The optimized inverse affine circuit (14 gates).

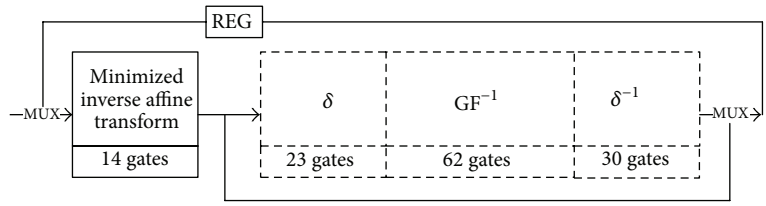

Figure 14: An alternative setup which utilizes registers to store temporary data.

operand, storing the results in the 2nd operand's memory location. If the subtraction results a negative value, it will "jump" to the target address, else it proceeds to execute the next instruction in the following sequence. The advantage of the SBN architecture is that the SBN instruction itself is capable of four different logical operations: ADD/SUBTRACT, MOVE/COPY, JUMP, and CLEAR. In computer programming, such a universal instruction set is deemed powerful as there is no additional hardware or additional instructions set required. With the URISC SBN architecture, a fundamental processing engine is realized. A more detailed description of the extended SBN instructions can be found in Table 3.

4.2. The Compact Instruction Set Architecture for AES (CISA AES). In our work, we use URISC as a platform and further expand it into a customized architecture called the CISA (Von-Neumann due to the single memory configuration). The reason it is called a CISA is due to the minimized and compact instruction sets that the architecture accommodates. There is no need for any additional instruction set in order to complete all the AES transformations, and this is the reason why the computer architecture is "compact." The latter part of this section further explains and dissects the CISA AES architecture into the following subsections: architecture, function codes and instruction sets, memory, FSM control signals, and cipher algorithm program code.

4.2.1. Data-Path Architecture and ALU (Arithmetic-Logic Unit). In the development process, we have extensively

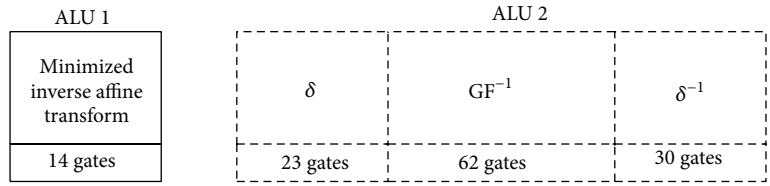

Figure 15: Illustration of the independent ALU set for the minimized inverse affine transform.

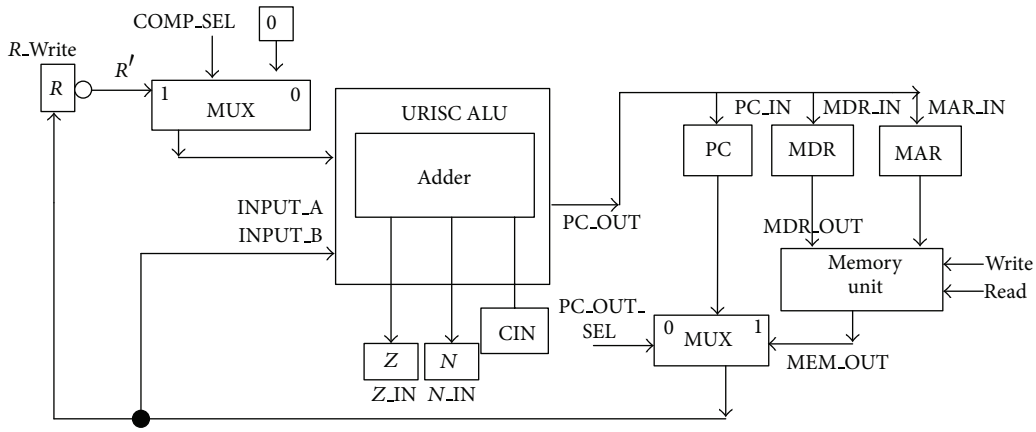

Figure 16: The URISC SBN architecture with Adder block.

studied the AES algorithm together with the basic URISC architecture. By understanding the AES transformations and the other three steps (*ShiftRows*, *MixColumns*, and *AddRoundKey*), this will give insights into how an application-specific integrated Processor (ASIP) can be designed. We have identified that in the AES transformations, there are two specific circuits required. As for the *ShiftRow* and *AddRoundKey*, a simple XOR and memory readdressing would suffice. As for the *SubBytes*, a combinational circuit has to be present. In this work, we are using the S-box that we have proposed in Section 3. As for *MixColumns*, we refer to [15] for the *xTime*-dedicated 4 XOR hardware. Note that to standardize the width of the register and data-path for optimum design, a unified and shared bitwise XOR block will be used to perform MOVE operations instead of SBN MOVE due to memory optimization issues (XOR is more efficient and 1 instruction less compared to SBN MOVE). On the other hand, the *xTime* is known to be using 4 independent XORs. The *xTime* would have a discreet ALU block, and no XORs will be shared with other ALUs due to register-memory width standardization. Unlike URISC which uses only one instruction, the CISA AES that we propose uses four minimized instructions (including SBN) to perform the complete encryption process. This extended version of URISC, together with 4 custom-developed ALU for the AES encryption and decryption, has the ability to perform any transformation in the AES algorithm. At the end, the CISA ALU has the following blocks: *Adder*, *XOR*, *xTime,* and *S-box*. With an external 1-bit input switch, the CISA is able to switch its operational mode between encrypt mode and decrypt mode.

The CISA data-path is shown in Figure 17. It has a single memory unit to store both program and data for the AES algorithm. With the SBN as the core instruction, the CISA is able to branch to any PC values within the memory unit and execute any instructions in any location of the memory unit. With 7 registers, 5 multiplexers, 1 memory unit, and 4 ALU blocks, the CISA is complete and functional. Similar to the structure of URISC, the CISA data-path loads in the first memory address and subsequently loads in the first data item. This operation is repeated for the second data item. Once both data are loaded into the CISA, they are sent into the ALU for computation and the outputs will be chosen with regard to the function code embedded into the first address loaded. The function code is a 2-bit value, concatenated to the first data address in the memory unit. With the 2-bit MSB value, the architecture is able to determine which instruction is used for the current processor cycle and what data are stored back to the memory.

The architecture has 2 input parameters into the CISA AS-ALU: Input_A and Input_B. Since the architecture is run by an FSM, the data movement and processing are fixed within 9 clock cycles. The Adder and XOR block takes in two data items and perform bitwise addition and XOR onto their respective inputs. The *xTime* block is a part of the *MixColumns* transformation. In [2], by using the substructure computation of a byte and between the computations of four bytes in an array of bytes, the derivation of the *MixColumns* transformation can be defined. In [17], the implementation of an "*xTime*" function is used to complete the multiplication of with "02," modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. It is known that the *MixColumns* transformation

Table 3: The SBN instruction sets.

Instruction sets	Description	3-Tuple instruction format
SBN ADD/SUBTRACT	2 SBN instructions would result in an addition operation. Hence, B − (−A) = A + B, and the value of (A + B) is stored in the B address	(1) SBN $A, $C, 0; (C = −A), where the address C contains the value of zero and A is the input A (2) SBN $C, $B, 0; B − (−A) = B + A, where the address B contains the value of input B
SBN MOVE/COPY	2 SBN instructions would result in a MOVE operation. Hence, D = A, and the value of A is stored in the D address	(1) SBN $A, $C, 0; (C = −A), where the address C contains the value of zero and A is the input A (2) SBN $C, $D, 0; (D = +A), where the address D contains the value of zero
SBN JUMP	1 SBN instruction would result in a JUMP operation. The condition would be the resultant output of $X - Y = a$ negative value	SBN $X, $Y, $J; (J = jump address), where $X - Y =$ −ve (value)
SBN CLEAR	1 SBN instruction would result in a CLR memory operation	SBN $A, $A, 0; (A − A = 0), where the address A contains the value of input A

is a process involving several XOR processes and xTime processes. The xTime is a bitwise XOR operation that yields the constant multiplication by (02). By concatenating two *xTime* blocks in serial, constant multiplication by (04) can be achieved. The *MixColumns* circuit in [2] can be used for both *MixColumns* and *Inverse MixColumns*. In Figure 18, part 1 of the circuit is the *MixColumns* transformation. Part 1 together with part 2 of the circuit yields the Inverse *MixColumns* Transformation. The xTime discreet circuit is shown in Figure 19.

4.2.2. Function Codes and Instruction Sets. In order to perform AES computations onto the plain text, byte-oriented methods are adapted from the AES encryption method. To perform tasks, such as *SubBytes* and *MixColumns*, a new series of instructions have to be developed. In this work, the instruction sets are specifically tailored for the ALUs defined for the CISA AES. The CISA instruction sets shown in Table 4 are differentiated using the two MSB of each of the instructions. Based on the operation required for each byte-oriented transformation in the AES algorithm, the four instruction sets used to perform separate operations are shown in Algorithm 1. From Table 4, each of the instruction formats uses 3 bytes in the program memory. The first byte holds the OP Code and the address of Mem_A, the second byte holds the address of Mem_B, and the last byte holds the target address. With four different OP codes embedded in the first byte of the instruction, the CISA selects the appropriate output from the corresponding processor block.

4.2.3. Memory Mapping and Program Structure. The CISA AES von-Neumann architecture includes a 1024 × 10-bit memory unit. The total available memory is 1024 × 8-bit (512 bytes), which accommodates both the data and program codes. The data section is located at the address location of 0 to 127, whereas the program section takes the location of 128 to 1024. In the program section, instructions are sorted in a sequence as the CISA executes in accordance. In the data section, the breakdown of the memory allocation the plain text, master key, and other temporary variables is shown in Figure 20.

For the program design of the CISA AES, functions and modules of a set of the written instructions can be reused for code optimization. During the decryption round, the Key Expansion algorithm has to be executed and the subkeys are stored inside the memory unit. During encryption mode, the program sequence has to start on producing all the subkeys and then proceed to the *AddRoundKey* function. Loop1 and Loop2 are used to branch to any designated memory locations in the memory unit if the resultant value is less than zero of negative. In loop1 and loop2, the addressed memory stores a number that enables the SBN instruction to be executed, and hence, the results will be checked by the CISA FSM controller in order to decide whether a branch instruction has to occur depending on the output of the Adder and the function code of the instruction. The function code tells the data-path that the current instruction performed is an SBN instruction. With the two SBN loops for branching, the AES encrypt mode can be completed.

For decrypt mode, similar to the AES encrypt mode, the decrypt process involves an initial prewhitening transformation of *AddRoundKey*. The subkeys are stored in the memory unit after encryption being done previously. A one-time loop is implemented in order for the CISA to execute the "*AddRoundKey*" once at the start of the decrypt sequence. This is due to the reason that the initial prewhitening step does not have a flow pattern to the programming sequence. In decrypt mode, the data transformation after AddRoundKey is the *Inverse MixColumns*. The initial Add Round Key is a one-time process, so the one-time loop is applied. With another SBN loop applied, the decrypt mode is able to execute the 4 basic inverse transformations with 10 iterations. Figure 21 is the illustration of encryption and decryption program flow in the CISA AES.

4.2.4. FSM Controller. The CISA controller generates a total of 14 control signals. Within 9 clock cycles, a complete instruction is being processed and carried out throughout

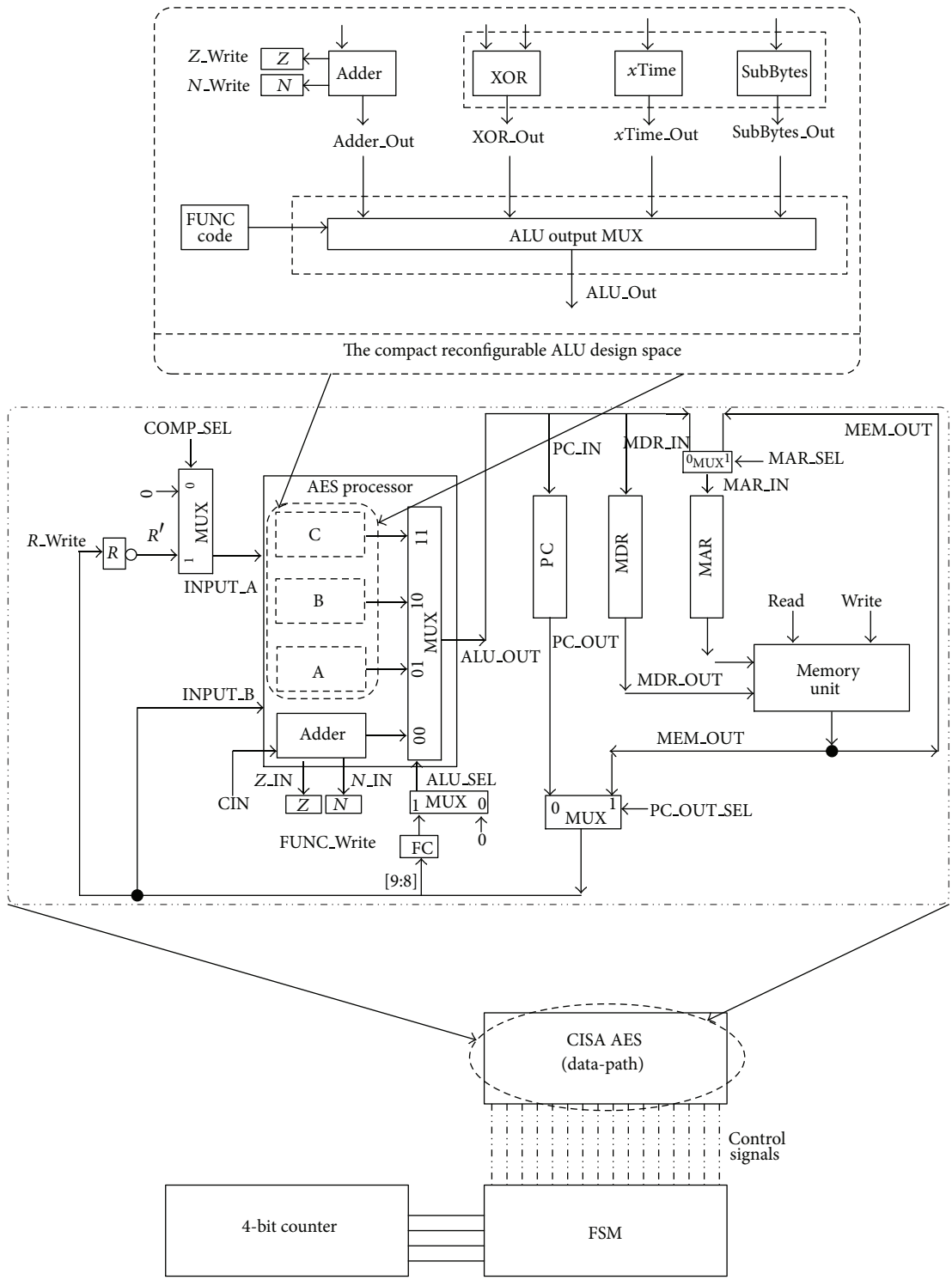

Figure 17: The overview of CISA von-Neumann architecture data-path with a programmable ALU space.

the registers and multiplexers in the architecture. During each clock cycle, operations from program counter (PC) increment, loading data item, writing the MDR register, and storing the results from the ALU, are being carried. Each register is controlled, and the characteristics of the data movement within CISA are predefined and fixed for all 4 basic instructions. The N register is used as an indicator to trigger the controller's outputs for branching. If a branch instruction occurs, the current values in the program counter register will be overwritten with a new target address. This target address was embedded in the last byte of the instructions, and the PC will start at this target address over the next clock cycle.

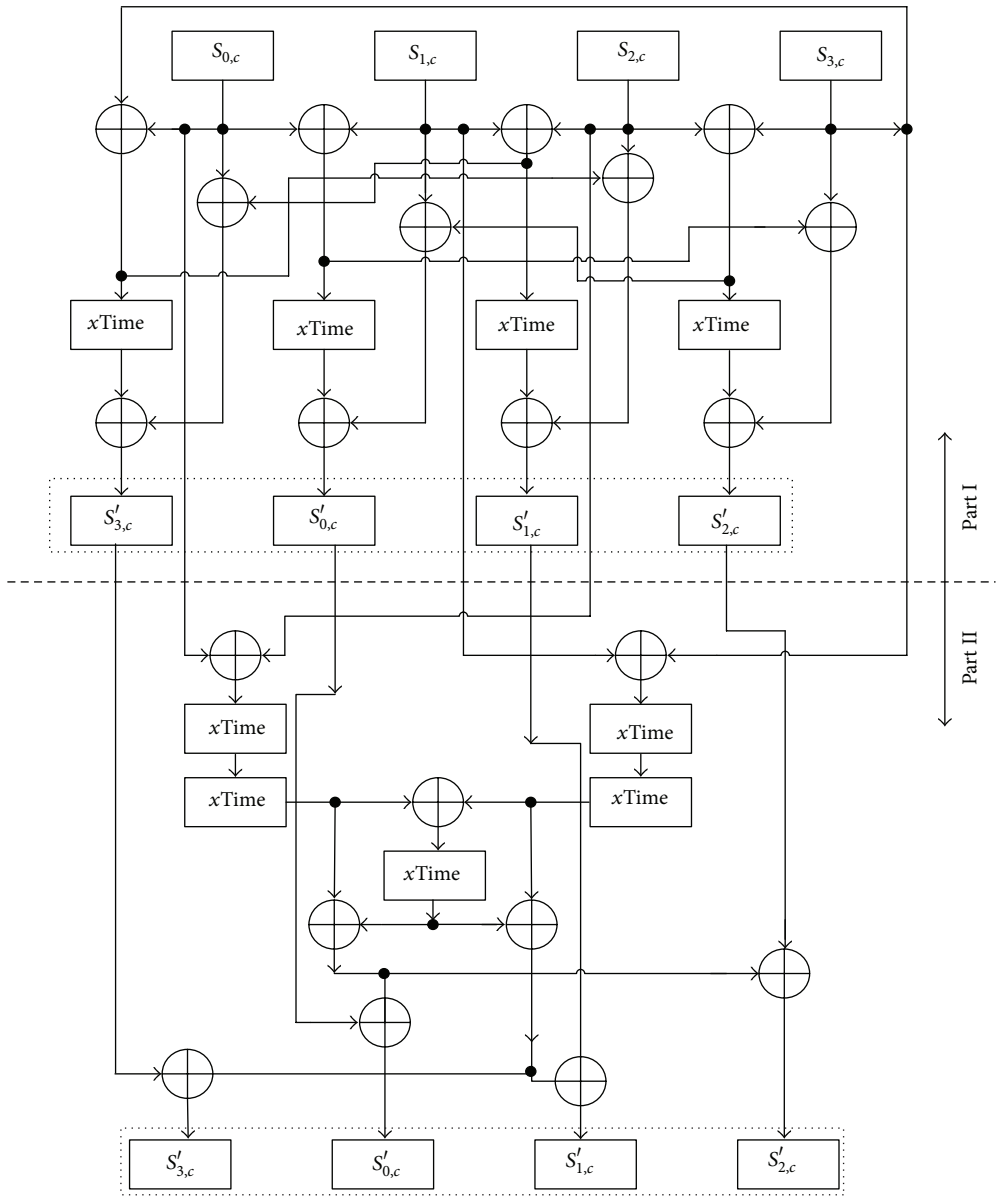

Figure 18: The *MixColumns* transformation process using the *xTime* circuit.

Table 4: The CISA instruction sets break down.

Operation	Function code (2-bit MSB)	Instruction format
SBN (Subtract and Branch if Negative)	00	(0 at address A), address B, Target
XOR	01	(1 at address A), address B, Target
xTime	10	10 at 0[n : 0], address B, Target
SubBytes	11	11 at 0[n : 0], address B, Target

The control circuit is driven by a 4-bit counter ($C_3C_2C_1C_0$). At each clock cycle, the control signals for particular control inputs are different. They are required to control the registers and store memory at any particular clock cycle. During clock cycle 0, the value of the program counter is loaded into the MAR, and at the same clock cycle, the Z register will be set accordingly by the ALU output to determine whether the PC has restarted at 0×00. In the following clock cycles, the data is read from the memory location, addressed by the MAR which stored the value of PC. Subsequently, the read data is written back to MAR, storing the address of the data to be used for computation. These processes are repeated for a second data. The PC value will be increased by 1 after each data loading operation is done.

At clock cycle 1, the address for Mem_A is stored in the MAR; at the same time, the OP Code for the instruction

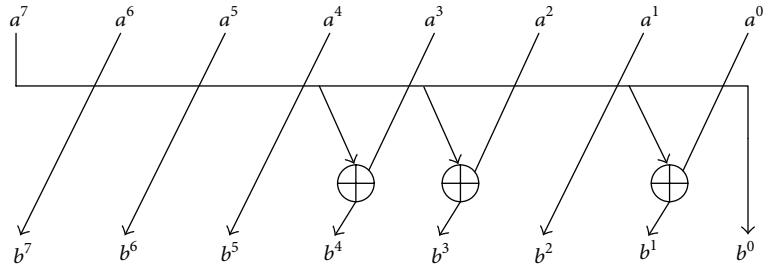

Figure 19: The *xTime* circuit.

```
SBN (Subtract and Branch If Negative)
Mem_B = Mem_B − Mem_A
If Mem_B < 0 Goto (PC + C)
Else Goto (PC + 1)
XOR
Mem_B = Mem_B XOR Mem_A
xTime
Mem_B = xTime(Mem_B)
Sub Bytes
Mem_B = Sub_Bytes(Mem_B)
```

Algorithm 1: The CISA instruction sets in pseudocode form.

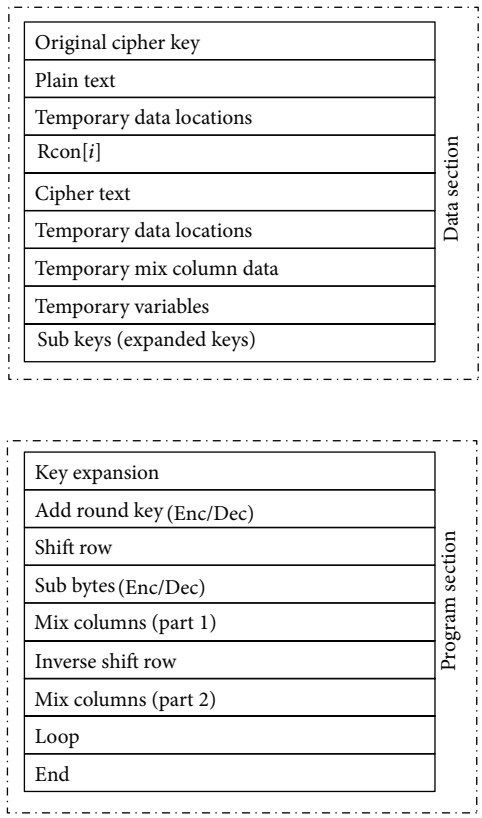

Figure 20: The memory mapping for CISA AES.

is written to the OP Register. At clock cycle 2, the value of Mem_A is read and stored into the R register. At clock cycle 3, the incremented value of PC is stored in the MAR. At clock cycle 4, the address for Mem_B is stored in MAR. At the clock cycle 5, the value of Mem_B is read and sent to the ALU for computation. The Adder and the other hardware blocks will perform their individual operations from the two given inputs (Mem_A and Mem_B). At that particular clock cycle, depending on the value of the OP Register, the desired output will be chosen via an ALU MUX. At clock cycle 6, the output from the ALU is sent to the MDR Register for storage. With the arithmetic operations performed, clock cycle 7 will load the jump address from memory. Then the jump address will be added to the PC value at the same clock cycle. The jump address value will only be added to the PC value, provided that the OP value is corresponding to the SBN instruction and a negative result is found at the output. At clock cycle 8, the value of the PC is incremented.

If a branch occurs, the N register would have a value of 0. So, the PC register would just would take in the value of the jump address and increase by 1. Then, the following instruction in the written program code will be performed. A total of 9 clock cycles are required to perform one instruction written in the program code section. The control signals are produced by a combinational logic circuit during each clock cycle. The whole 9 clock cycles will repeat themselves until the end of the program reached. The 4-bit counter will restart once it reaches the value C = 8. Algorithm 2 shows the Boolean expression of the control signals.

5. The Selective Encryption Architecture (SEA) for Implementation in Radio Frequency Identification (RFID) Environment

5.1. A Brief Review on Selective Encryption Systems for Image Processing and Multimedia Applications. The security of multimedia data in digital distribution networks is commonly provided by encryption. Nevertheless, most of the classical and modern ciphers known were initially developed for the simplest form of data type—"text" and are not made for large quantity of data in real-time environment such as images and video with very large sizes and redundancies. Selective encryption (some sources refer to it as partial encryption) is a highly effective approach to reduce the

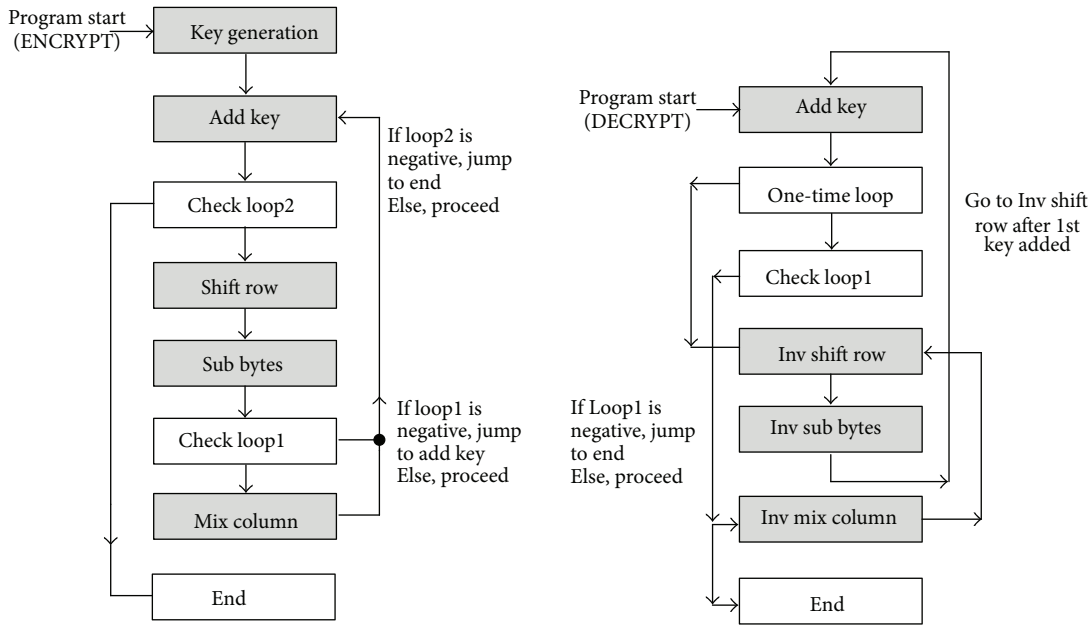

Figure 21: The CISA AES encryption and decryption program flowchart and structure.

$$
\begin{aligned}
&\text{ALU\_B0} = \overline{C_2}\,\overline{C_0} + \overline{C_1}C_0\\
&\text{ALU\_B1} = C_2\overline{C_0}\\
&\text{ALU\_A0} = C_1C_0\\
&\text{ALU\_A1} = \overline{C_2}\,\overline{C_0} + \overline{C_1}C_0\\
&\text{CIN} = C_1 + C_0\\
&\text{PMAR\_Write} = C_1\overline{C_0} + \overline{C_2}\,\overline{C_1}\\
&\text{DMAR\_Write} = C_1\overline{C_0} + \overline{C_2}\,\overline{C_1}C_0\\
&\text{PC\_Write} = \overline{C_2}\,\overline{C_1}C_0 + \overline{C_2}C_1\overline{C_0} + C_2\overline{C_1}\,\overline{C_0}N\\
&\text{PMem\_Read} = C_2\overline{C_0} + C_1\overline{C_0} + \overline{C_2}\,\overline{C_1}C_0\\
&\text{DMem\_Read} = C_1\\
&\text{DMem\_Write} = C_2\overline{C_0}\\
&\text{R\_Write} = C_1\overline{C_0}\\
&\text{Z\_Write} = \overline{C_2}\,\overline{C_1}\,\overline{C_0}\\
&\text{N\_Write} = C_1C_0\\
&\text{MDR\_Write} = C_1C_0\\
&\text{Op\_Write} = \overline{C_2}\,\overline{C_1}C_0\\
&\text{Op\_SEL} = C_1C_0
\end{aligned}
$$

Algorithm 2: The Boolean expression of the CISA FSM controller.

computational requirements for huge volumes of multimedia data in distribution networks. Selective encryption is an approach to "selectively" encrypt the most important portion of the data in order to provide an adequate security and to reduce computational requirements. Inspired by the way compression can strengthen encryption, system designers have found ways to secure compressed data by enciphering only a portion of the compressed bit-stream. Selective encryption is a technique to reduce the computational complexity and enables interesting system functionality by only encrypting a portion of a compressed bit-stream while still achieving a certain degree of security. Figure 22 shows the pictorial explanation of a selective encryption system.

For selective encryption to work, we need to rely not only on the beneficial effects of redundancy reduction described by Shannon [29], but also on a characteristic of many compression algorithms to concentrate important data about reconstruction in a relatively small fraction of the compressed bit-stream. Shannon first pointed out the strong relationship between data compression and encryption [29]. In particular, the author showed how the redundancy in a source (such as the redundancy in the English language associated with the different relative frequencies of letters and letter groupings) makes encryption weak. For example, if an attacker can use the different frequencies in symbols in the source material to guess the encryption mapping from source material to encrypted material. Surprisingly, for the English language and a simple substitution cipher in which each letter is mapped to some other letter, the mapping is close to uniquely determined and the cipher broken after observing about 30 letters of the cipher text. The author then suggested that data compression (by means of removing the redundancy in the source) could improve and strengthen the encryption. Perfect compression, in fact, would eliminate any statistical redundancy in the source to the encryption, and an attacker could do no better than successively guessing the possible encryption mappings (the mappings being indexed by the encryption key shared through some other means by the encryption operations and the decryption operations). There are three requirements to prevent unauthorized access to multimedia content over the communication channel:

(1) the encryption of the digital content;

(2) protection of cryptographic keys;

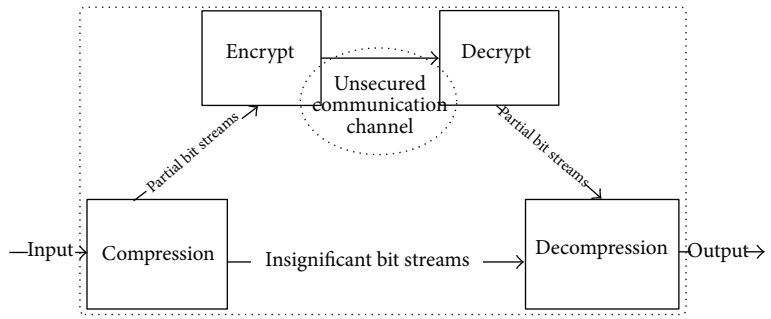

Figure 22: The overview of a selective encryption system (combination of an encryption and compression core).

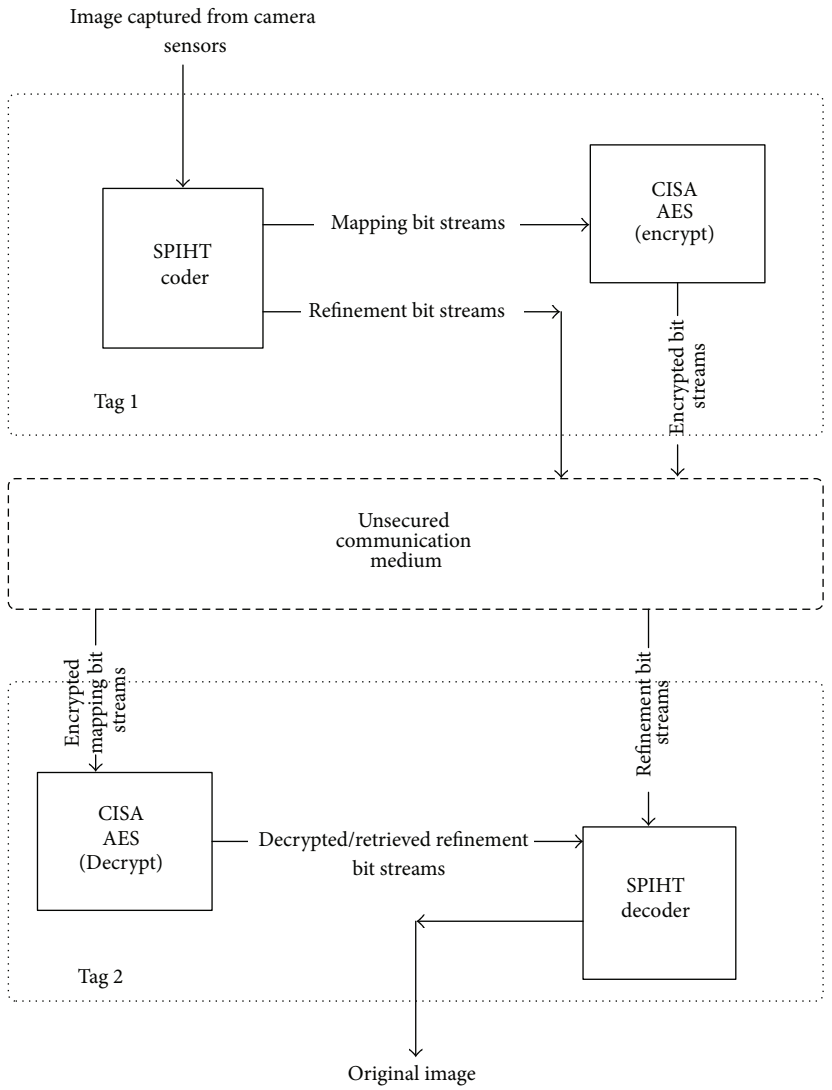

Figure 23: The illustration of the combined system of a SPIHT coder and an AES-based CISA to achieve the selective encryption.

(3) integrity of the critical data (copyright or usage rights) associated with the content.

5.2. A Brief Review of Set Partitioning in Hierarchical Trees (SPIHT) Algorithm. The SPIHT encoder [30] is a highly refined version of the EZW algorithm and is a powerful image compression algorithm that produces an embedded bit stream from which the best reconstructed images in the mean square error sense can be extracted at various bit rates. Some of the best results (highest PSNR values for given

Table 5: The comparison of different S-boxes.

Basis	Type	XOR	XNOR	NAND/AND	NOT	MUX	Total gates
Our work (straight-line)	Merged	**107**	**4**	**32**	—	**16**	**159**
	—	—	—	—	—	—	—
	—	—	—	—	—	—	—
Our work (alternative block sharing)	Merged	**93**	**4**	**32**	—	**16**	**145**
	—	—	—	—	—	—	—
	—	—	—	—	—	—	—
Our work (independent ALU)	Merged	**93**	**4**	**32**	—	**—**	**129**
	—	—	—	—	—	—	—
	—	—	—	—	—	—	—
Boyar (single S-box)	—	—	—	—	—	—	—
	S-box	79	4	32	—	—	115
	—	—	—	—	—	—	—
Boyar (complete) (newer)	Merged	**144**	**14**	**34**	—	**16**	**208**
	S-box	90	4	34	—	—	128
	Inv S-box	83	10	34	—	—	127
Edwin	—	—	—	—	—	—	—
	S-box	193	—	45	—	—	238
	—	—	—	—	—	—	—
Canright	Merged	**107**	**0**	**36**	**2**	**16**	**253**
	S-box	91	0	36	0	0	195
	Inv S-box	91	0	36	0	0	195
Mentens	Merged	**118**	**0**	**36**	**0**	**16**	**271**
	S-box	96	0	36	0	0	204
	Inv S-box	97	0	36	0	0	206
Satoh	Merged	**119**	**0**	**36**	**3**	**16**	**275**
	S-box	100	0	36	0	0	211
	Inv S-box	99	0	36	0	0	209
Worst	Merged	**131**	**0**	**36**	**0**	**16**	**293**
	S-box	107	0	36	0	0	223
	Inv S-box	106	0	36	0	0	222

compression ratios) for a wide variety of images have been obtained with SPIHT. Hence, it has become the benchmark state of the art algorithm for image compression.

The SPIHT method represents an important advance in the field. The method is characterized by the following:

(i) good image quality, high PSNR, especially for color images;

(ii) being optimized for progressive image transmission;

(iii) produces a fully embedded coded file;

(iv) simple quantization algorithm;

(v) fast coding/decoding (nearly symmetric);

(vi) has wide applications, completely adaptive;

(vii) can be used for lossless compression;

(viii) can code to exact bit rate or distortion;

5.3. A Brief Review of RFID/WISP Resource-Constrained Environments. In [31], the authors introduced RFID sensor networks (RSNs), which consist of small, RFID-based sensing and computing devices—WISPs (wireless identification sensing platforms), and RFID readers that are part of the infrastructure and provide operating power. They claim that the RSNs are capable of bringing the advantages of RFID technology to wireless sensor networks but they do not expect them to replace WSNs for all applications. As the WISP became more mature, it is assumed that the WISP is capable of replacing RFIDs. Note that the sensor nodes and the WISP both have sensing abilities but the WISP has an advantage as its energy source comes from energy harvesting. Many sensor network applications such as home sensing and factory automation can be solved where the readers can be installed and carried easily. Like both RFID and WISP platform will have a MCU (microcontroller unit). Hence, this notion suggests that there will be implementation proposal in the future for such a network as the research advances.

5.4. Selective Encryption Architecture (SEA) Based on SPIHT Coder and CISA AES Core. In this section, we are presenting the combined system of a SPIHT coder with a CISA processor. For an image processing system, image compression module is one of the core functionality blocks. In our design and implementation process, our focus was to design and configure a CISA to work with the SPIHT core. A camera is set to capture live images and transfer the coded image into the SEA. The SEA SPIHT coder decomposes input images and creates two separate bit streams: the refinement bits and the mapping bits. As explained in Figure 23, mapping bits are sent to the CISA AES core for encryption, and refinement bits are left and remain as they are without additional processing. Note that for illustration purposes, we choose to depict a system consisting two separate tags which in fact is a setting and network behavior of a RFID/WISP network (tag-to-tag communication or tag-to-reader communication). Both streams are then sent to the receiving party via unsecured communication medium. Both streams pose no security threats as the attacker will not benefit by acquiring the unencrypted refinement bits. Having the refinements bits has no meaning without the tree structures within the encrypted mapping bits. Therefore, the SEA with CISA AES and SPIHT core is deemed a secure system. Note that in this paper, we are not going to discuss the compression quality of the system.

6. Results and Discussion

6.1. Comparison of Various Compact S-Boxes. We have made comparisons with different S-boxes, and the comparison of gate counts is shown in Table 5.

Assuming a multiplexer costs 8 gates (by referring to [14]), our proposed configuration uses 2 MUXes, in which it costs 16 gates in total. Our proposed methodologies had shown an improvement by offering lower gate count of the bidirectional S-box configuration. Since our proposal and methodologies are built on top of Boyar's S-box, the results we offer are only the merged version of the S-box. Our justification is that the merged S-box is more popular when designing an independent system that performs both encryption and decryption on the same platform, without involving a secondary or a host server for decryption.

6.2. Comparison of Various Small AES Processors. Currently, the best work that we are able to find is for the smallest architecture from Good and Benaissa [27]. The authors have presented superb result of 122 slices on a Spartan-II device. The authors' work is ASIP based (application-specific instruction processor) and they believe that the smallest work has to be from Rouvroy et al. [17] and Gaj [28]. The said works opted to iteratively use a reduced fixed-width 32-bit data-path, sacrificing throughput but yielding a much smaller circuit.

The CISA AES von-Neumann architecture is first designed and tested using the DK Design Suite software environment. A Celoxica RC10 board which houses the Spartan-3 XCS1500L-4 FPGA is used, and on-board LEDs are used to observe the data memory items. The implementation results are shown in Table 6. The gate count

Table 6: Implementation results.

Components	Quantity	Total	Usage
No. of slice flip flops	110	26,624	1%
No. of occupied slices	236	13,312	1%
Total no. of 4-input LUTs	428	26,624	1%
No. of LUTs used as logic	405	428	95%
No. of LUTs used as a route-thru	22	428	5%
No. of LUTs used as shift registers	1	428	~0%
No. of bonded IOBs	28	221	12%
No. of BRAMs	3	32	9%
No. of GCLKs	4	8	50%
No. of DCMs	1	4	25%

Table 7: Gate counts on ALU components.

ALU Block	AND	XOR/XNOR	OR
Adder (10-bit)	20	20	10
XOR (8-bit)	—	8	—
$x$time	—	8	—
*SubBytes (dual-inverse affine, straight line circuit)	32	111	—
**SubBytes	32	97	—

*: The proposed S-box with dual-inverse affine, a straight line circuit (refer to Figures 10 and 11).
**: The proposed S-box using methodology 1 and 2 (Sections 3.2 and 3.3).

numbers shown in Table 7 are the schematic gate counts. The numbers will vary after design synthesis and optimization done by Xilinx. Table 8 shows the comparison between CISA and other small FPGA AES implementations.

For comparisons, we would like to compare with the work by Good and Benaissa [27]. The authors' work has an astounding result of 122 slices for the ASIP design. Our goal is to minimize the overall slices' occupancy. The initial step is to focus on reducing the S-box's size. Our aim is to design a much simpler design that uses less complicated instruction sets for the ease of programming. On this platform, we are at advantage since the total numbers of instructions programmed are just 4 while Tim et al.'s design (excluding 2 unused instructions) has 14 instruction sets. Hence, CISA AES is the least complex in terms of programming. Table 9 shows the comparison between our work and Tim's work on instruction count.

6.3. Code Execution and Timing. Our approach of microprogramming is based on the AES ECB mode. In [32, 33], the author has described the method to calculate the throughput for FPGA AES design, where

$$\text{Throughput} = \left[\frac{(\text{bits per block})}{(\text{clock cycle})}\right] * \text{frequency}. \quad (7)$$

The total clock cycles for the CISA AES have to be calculated according to the number of instructions executed for the complete AES operation. Each MISC instruction takes

Table 8: Comparisons with other small AES processors.

Design and FPGA (device)	CISA AES ENC/DEC (von-Neumann) Spartan-III (XC3S50-4)	Good and Benaissa [27] AES ASIP Spartan-II (XC2S15-6)	Good and Benaissa [27] PicoBlaze Spartan-II (XC2S15-6)	Gaj [28] Spartan-II (XC2S30-6)	Rouvroy et al. [17] Spartan-III (XC3S50-4)	Zhang and Parhi [2] Virtex-E (XCV1000e−8)
Encryption algorithm	AES	AES	AES	AES	AES	AES
Max. clock freq. (MHz)	20	72.3	90	60	71	168.4
Data-path bits	12	8	8	32	32	128
No. of slices of flip-flop utilized	110	122	119	222	163	11022
No. of block RAMs used	4	2	2	3	3	0
Block RAM size (kbits)	4	4	4	4	18	—
Bits of block RAM used	49152	4480	10666	9600	34176	0
Equiv. slices for memory	126	140	333	300	1068	0
Total equiv. slices (Est.)	236	262	452	522	1231	11022
Max throughput (Mbps)	—	—	—	166	208	21556
Avg. throughput (Mbps) Average encryption-decryption including key expansion	17.78	2.18	0.71	69	87	21556
Performance, typical throughput per slice (kbps/slice)	75.33	8.3	1.6	132	70	1956
Summary	Smallest	Very small	Software based	Best performance (throughput/slice)	Fastest	Loop unrolled

Table 9: Comparisons with other small AES processors.

Designs	Ours	Good and Benaissa [27]
Instruction set count	4	14

9 cycles to be completed, and the total instructions executed (including the key expansion for AES) are

Key expansion: (189 bytes/3) ∗ 10 = 630,

Shift Rows: (48 bytes/3) ∗ 10 = 160,

Sub Bytes: (96 bytes/3) ∗ 10 = 320,

Add Key: (99 bytes/3) ∗ 10 = 330,

Mix Column: (600 bytes/3) ∗ 10 = 2110.

This is amounted to the total of: 630 + 160 + 320 + 330 + 2110 = 3550 instructions. The total period for an AES encryption using MISC is 3550 × 9 cycles = 31950 cycles. The throughput for [27] is 2.18 Mbps while our MISC AES has an average throughput of 80 kbps [(128-bit/31950 clocks) × 20 MHz], where 20 MHz is the maximum clock frequency and 9 clocks per instruction and giving a 0.738 kbps per slice.

In [17], the author has presented a set of comparisons to related works. By referring to Rouvroy et al., the comparison to our work is shown in Table 10. Our version of CISA AES has shown improvement as compared to the author's work. The rational for comparison to this paper is due to the Rouvroy's aim to propose compact solutions for small embedded applications and the work covers AES and the weaker cipher, DES and 3DES. From the comparison made in Table 10, the proposed CISA AES has an advantage of having significantly reduced area occupancy with a tradeoff of lower throughput. Another comparison can be made to [34] as it has the lowest clock cycles execution for a 128-bit AES encryption. The comparison platform is not very similar but the author has presented a very lightweight design for RFID systems. Our comparison point is to their S-box execution. Technically, our S-box instructions are only 16 sets. 16 instructions with 9 cycles each are equivalent to 144 cycles. Our S-box execution cycle is half of the clock cycles in [34].

6.4. Selective Encryption Architecture (SEA). In our experiment, we have implemented a complete system of selective encryption by coupling the CISA AES processor and a SPIHT coder, which is based on a MIPS processor. The experimental setup and coding are tested using the Agility Design Suite 5.0 software environment, and a Celoxica RC203 board which houses the Vertex XC2V3000 FPGA is used. A still-portrait image is used and displayed on a HP 17-inch LCD as an input source to the camera. The camera that we have used in this experiment is a 330 Line CCD camera. Figure 24 shows our experimental setup.

We have set our program to capture four images simultaneously, and it can be considered a low-frame-rate video. All four images are encrypted onboard and sent to another computer for decryption. Note that due to limited resources, we possess only a single unit of RC203 and are unable to program another RC203 for on-board decryption purpose. The CISA AES is programmed to encrypt only the mapping bits, and both mapping and refinement bits are sent out to the host computer once the encryption has completed. The received bits are then processed in MATLAB environment, and we have chosen the last 2 frames for decryption, solely

Table 10: Comparison with other Rouvroy et al.'s designs in terms of code execution and speed.

	Rouvroy's AES	Rouvroy's AES	Rouvroy's DES	Rouvroy's 3DES	CISA AES
Device	XC3S50-4	XC2S40-6	XC2S40-6	XC2S40-6	XCS1500L-4
Slices	163	146	189	227	110
Throughput (Mbps)	208	358	974	326	0.08
Block RAMs	3	3	0	0	3
Throughput/area (Mbps/slices)	1.26	2.45	5.15	1.44	0.728 kbps

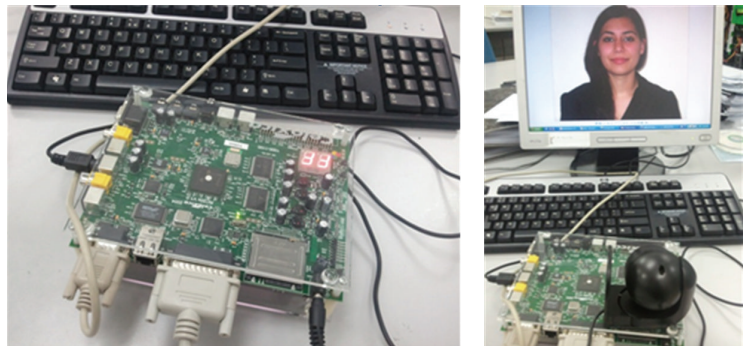

Figure 24: The experimental setup for the development of selective encryption architecture.

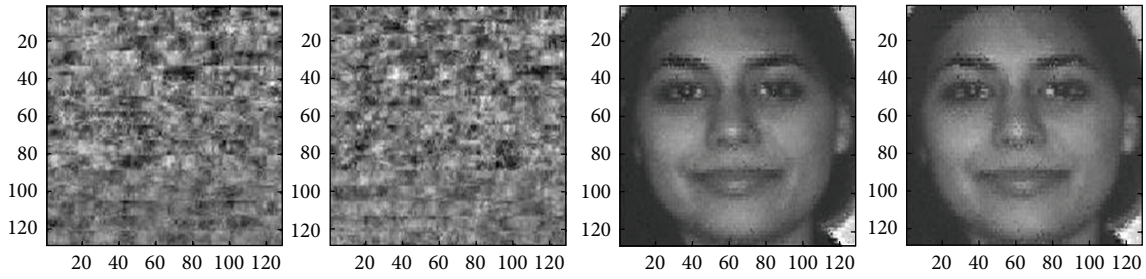

Figure 25: The four selectively encrypted frames with the last two frames decrypted.

Table 11: Logic Utilization of SEA.

Logic utilization	Quantity	Total	Usage
No. of slice flip flops	3692	28672	12%
No. of 4-input LUTs	8793	28672	30%

Table 12: Logic distribution of SEA.

Logic distribution	Quantity	Total	Usage
No. of occupied slices	6251	14336	43%
No. of slices containing only related logic	6251	6251	100%
No. of slices containing unrelated logic	0	6251	0%

Table 13: LUT utilization of SEA.

Components	Quantity	Total	Usage
Total no. of 4-input LUTs	10176	28672	35%
No. of LUTs used as logic	8793	8793	86%
No. of LUTs used as a route-thru	1257	1257	12%
No. of LUTs used for dual-port RAMs	64	64	~1%
No. of LUTs used as 16 × 1 ROMs	30	30	~0.5%
No. of LUTs used as shift registers	32	32	~0.5%

for comparing and verifying the correct encryption and decryption having occurred. Figure 25 shows our results on the SEA system. Note that the cipher mode that we used is ECB mode, and both encryption and decryption parties only target the mapping bits. Figure 26 shows an example of the selective encryption on the Lena image, captured via camera. Tables 11, 12, 13, and 14 show our implementation results.

7. Conclusion

In this paper, we have presented 3 methodologies of complete bidirectional S-box configurations for lower gate count. To incorporate the S-box design into a practical example, we have designed an FPGA implementation of the AES using the CISA architecture. The justification of the CISA's practicality

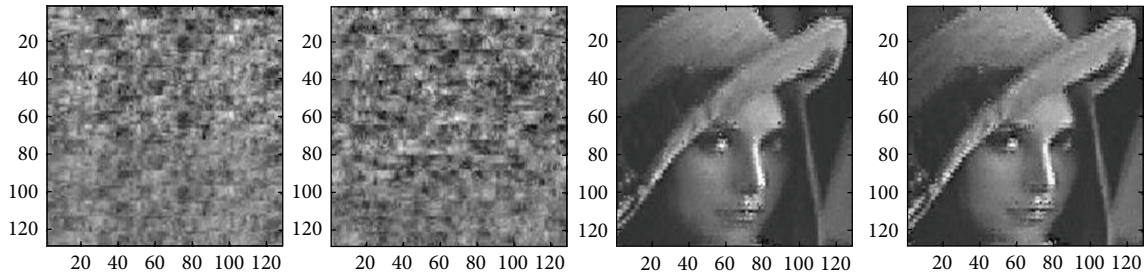

Figure 26: Selective encryption on Lena image.

Table 14: Other components utilized by SEA.

Components	Quantity	Total	Usage
No. of BUFGMUXs	4	16	25%
No. of DCMs	1	12	8%
No. of external IOBs	199	484	41%
No. of LOCed IOBs	199	199	100%
No. of RAMB16s	66	96	68%
No. of slices	6251	14336	1%

is backed by the implementation of a real-time system of selective encryption. The demonstration of a complete system of selective encryption by coupling the CISA AES and the SPIHT coder has further enhanced the confidence of the realization of a real-time image processing and encryption system. We have proposed the smallest S-box configurations with the least gate counts, and we have also presented our smallest version of AES based processor architecture. This work serves as an example for real-time embedded design and also a practical implementation example of image processing systems.

References

[1] N. I. o. S. a. Technology, "Advanced encryption standard," in *NIST FIPS PUB 197*, U. S. D. o. Commerce, 2001.

[2] X. Zhang and K. K. Parhi, "High-speed VLSI architectures for the AES algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 9, pp. 957–967, 2004.

[3] S. Lemsitzer, J. Wolkerstorfer, N. Felber, and M. Braendli, "Multi-gigabit GCM-AES Architecture Optimized for FPGAs," in *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, Vienna, Austria, 2007.

[4] S. Qu, G. Shou, Y. Hu, Z. Guo, and Z. Qian, "High throughput, pipelined implementation of AES on FPGA," in *Proceedings of International Symposium on Information Engineering and Electronic Commerce (IEEC '09)*, pp. 542–545, May 2009.

[5] S. Morioka and A. Satoh, "A 10 Gbps full-AES crypto design with a twisted-BDD S-box architecture," in *Proceedings of International Conference on Computer Design (ICCD '02) VLSI in Copmuters and Processors*, pp. 98–103, September 2002.

[6] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen, "Design and implementation of low-area and low-power AES encryption hardware core," in *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD '06)*, pp. 577–583, September 2006.

[7] J. K. MooSeop Kim and Y. Choi, "Low power circuit architecture of AES crypto module for wireless sensor network," *World Academy of Science, Engineering and Technology*, vol. 8, 2005.

[8] G. N. Selimis, A. P. Kakarountas, A. P. Fournaris, A. Milidonis, and O. Koufopavlou, "A low power design for sbox cryptographic primitive of advanced encryption standard for mobile end-users," *Journal of Low Power Electronics*, vol. 3, pp. 327–336, 2007.

[9] Y.-H. Zeng, X.-C. Zou, Z.-L. Liu, and J.-M. Lei, "Low-power clock-less hardware implementation of the rijndael S-box for wireless sensor networks," *The Journal of China Universities of Posts and Telecommunications*, vol. 14, no. 4, pp. 104–109, 2007.

[10] R. Liu and K. K. Parhi, "Fast composite field S-box architectures for advanced encryption standard," in *Proceedings of the 18th ACM Great Lakes Symposium on VLSI (GLSVLSI '08)*, pp. 65–70, March 2008.

[11] J. Daemen and V. Rijmen, *The Design of Rijndael*, Springer, New York, NY, USA, 2002.

[12] M. McLoone and J. V. McCanny, "Rijndael FPGA implementation utilizing look-up tables," in *Proceedings of IEEE Workshop on Signal Processing Systems (SIPS '01)*, pp. 349–360, October 2001.

[13] V. Fischer and M. Drutarovsk, "Two methods of Rijndael implementation in reconfigurable hardware," in *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems*, 2001.

[14] D. Canright, "A very compact s-box for AES," in *Proceedings of the 7th International Conference on Cryptographic Hardware and Embedded Systems*, Edinburgh, UK, 2005.

[15] J. Daemen and V. Rijmen, "The block cipher Rijndael," in *Proceedings of the the International Conference on Smart Card Research and Applications*, 2000.

[16] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, vol. 2248, pp. 239–254, 2001.

[17] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," in *Proceedings of International Conference on Information Technology: Coding Computing (ITCC '04)*, pp. 583–587, April 2004.

[18] C. Paar, "Some remarks on efficient inversion in finite fields," in *Proceedings of the IEEE International Symposium on Information Theory*, p. 58, September 1995.

[19] S. Morioka and A. Satoh, "An optimized S-box circuit architecture for low power AES design," in *Proceedings of Cryptographic Hardware and Embedded Systems (CHES '02)*, B. Kaliski, ç. Koç, and C. Paar, Eds., vol. 2523, pp. 172–186, Springer, 2003.

[20] J. Boyar and R. Peralta, "A new combinational logic minimization technique with applications to cryptology," in *Experimental Algorithms*, P. Festa, Ed., vol. 6049, pp. 178–189, Springer, Berlin, Germany, 2010.

[21] J. Boyar and R. Peralta, "A small depth-16 circuit for the AES S-box," in *Information Security and Privacy Research*, D. Gritzalis, S. Furnell, and M. Theoharidou, Eds., vol. 376, pp. 287–298, Springer, Berlin, Germany, 2012.

[22] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2m) using normal bases," *Information and Computation*, vol. 78, no. 3, pp. 171–177, 1988.

[23] E. N. Mui, "Practical Implementation of Rijndael S-Box Using Combinational Logic," 2007, http://www.xess.com/projects/Rijndael_SBox.pdf.

[24] D. J. Bernstein, "Optimizing linear maps modulo 2," 2009, http://binary.cr.yp.to/linearmod2-20091005.pdf.

[25] F. Mavaddat, B. Parhami, and URISC:, *Ultimate Reduced Instruction Set Computer*, Faculty of Mathematics, 1987.

[26] W. F. Gilreath and P. A. Laplante, *Computer Architecture: A Minimalist Perspective*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[27] T. Good and M. Benaissa, "Very small FPGA application-specific instruction processor for AES," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 7, pp. 1477–1486, 2006.

[28] K. Gaj, "Very compact FPGA implementation of the AES algorithm," in *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES '03)*, vol. 2779 of *Lecture Notes in Computer Science*, pp. 319–333, Springer, 2003.

[29] C. E. Shannon, *Communication Theory and Secrecy Systems*, Bell Telephone Laboratories, New York, NY, USA, 1949.

[30] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, 1996.

[31] M. Buettner, B. Greenstein, A. Sample, J. Smith, and D. Wetherall, "Revisiting smart dust with RFID sensor networks," in *Proceedings of the 7th ACM Workshop on Hot Topics in Networks (Hotnets-VII '08)*, 2008.

[32] J. M. Granado-Criado, M. A. Vega-Rodríguez, J. M. Sánchez-Pérez, and J. A. Gómez-Pulido, "A new methodology to implement the AES algorithm using partial and dynamic reconfiguration," *Integration, the VLSI Journal*, vol. 43, pp. 72–80, 2010.

[33] C. Chiţu and M. Glesner, "An FPGA implementation of the AES-Rijndael in OCB/ECB modes of operation," *Microelectronics Journal*, vol. 36, no. 2, pp. 139–146, 2005.

[34] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Cryptographic Hardware and Embedded Systems—CHES 2004*, M. Joye and J. J. Quisquater, Eds., vol. 3156, pp. 85–140, Springer, Berlin, Germany, 2004.