

Received January 29, 2021, accepted February 11, 2021, date of publication February 16, 2021, date of current version February 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3059623

# Efficient Implementation of AES and CTR\_DRBG on 8-Bit AVR-Based Sensor Nodes

YOUNGBEOM KIM<sup>ID</sup> AND SEOG CHUNG SEO<sup>ID</sup>, (Member, IEEE)

Department of Financial Information Security, Kookmin University, Seoul 02707, South Korea

Corresponding author: Seog Chung Seo (scseo@kookmin.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] under Grant 2019R1F1A1058494.

**ABSTRACT** Since Rijndael algorithm was selected as the Advanced Encryption Standard (AES) by NIST, optimization research for the AES has been actively conducted on various IoT-based processors. In an 8-bit AVR environment, LIGHT version of Fast AES CTR-mode Encryption (FACE-LIGHT) was proposed at ICISC'2019 conference. However, in a Wireless Sensor Network environment, where sessions are frequently changed, FACE-LIGHT seems not efficient in terms of available memory and generating a pre-computation table. In this article, we present a new column-wise fashion implementation. Unlike previous best AES implementations, our proposed implementation in an 8-bit AVR microcontroller combines SubBytes, ShiftRows, and MixColumns operations and optimizes the operation speed through efficient register scheduling. Our constant-time implementation uses a significantly less table than FACE-LIGHT in an 8-bit AVR microcontroller, achieving 2,251, 2,706, and 3,160 clock cycles when encrypting 128-bit data for each of three security levels. In particular, our 256-bit security level AES implementation is the fastest AES implementation as far as we know in 8-bit AVR microcontroller. Finally, we apply our implementation in CounTeR-mode\_Deterministic Random Bit Generator (CTR\_DRBG), one of the upper algorithms of a symmetric-key algorithm, to prove the generality of our optimization technology in various operating modes of AES.

**INDEX TERMS** AES, software optimization, ATmega, embedded MCUs, wireless sensor networks, sensor device, counter mode of operation, CTR mode, CTR\_DRBG.

## I. INTRODUCTION

Due to the development of Internet of Things (IoT) technology, Wireless Sensor Network (WSN) is a core technology used for communication in various sensor nodes in the edge network [1]. In the WSN environment, various sensor nodes must encrypt data for confidentiality when communicating between sensors or communicating with servers.

However, since 8-bit AVR very resource-constrained only has 32 general-purpose registers of 8-bit size, it is challenging to optimize cryptographic algorithms with more than 128-bit internal state. Until now, optimization studies have been conducted on various cryptographic algorithms in an 8-bit AVR MCUs [2]–[7]. Various cryptographic algorithms, ranging from symmetric key cryptographic algorithms to public key cryptographic algorithms, have been optimized and studied,

suggesting the importance of maximizing the use of registers and efficiently using instruction sets.

Advanced Encryption Standard (AES) is currently the most popular encryption algorithm used by Low and High-end Processors to provide confidentiality. Various AES implementation methods have been proposed to date in 8-bit AVR environments [8]–[11]. At the recent ICISC'2019 conference, FACE-LIGHT, an efficient optimization method for the AES-CounTeR (CTR) mode, was proposed [11]. The main idea of FACE-LIGHT is to generate a LookUp table for duplicate parts during encryption by using the nonce of Initialization Vector (IV), which is the input data of CTR mode, to be reused. In the case of FACE-LIGHT, it achieved 2,218 clock cycles in AES implementation of 128-bit security level, and it is the fastest implementation of the existing in the 8-bit AVR environment.

However, in the case of FACE-LIGHT, there is basically an additional overhead that occurs when generating a LookUp table. In WSN environment using an 8-bit AVR sensor node,

The associate editor coordinating the review of this manuscript and approving it for publication was Yongqiang Cheng<sup>ID</sup>.

sessions are frequently changed. As the session changes, FACE-LIGHT needs to generate LookUp tables for all counters, and the incurred generation cost is about 61,440 clock cycles; therefore, FACE-LIGHT's fixed key scenario reduces the efficiency in terms of speed.

In addition, MixColumns implementation of FACE-LIGHT and AES implementations proposed so far in an 8-bit AVR environment occur a non-constant-time issue. Until now, the presence or absence of carry in the multiplication of MixColumns is implemented through branch instruction. Implementation of cryptography should basically be guaranteed to be efficient and secure. The 8-bit AVR microcontroller, unlike the CPU environment or high-performance embedded device such as ARM-Cortex, does not have cache memory; therefore, it can be considered secure from timing attacks. However, In MixColumns operation of 1 Round, additional information of the master-key may be leaked due to the existence of difference of clock cycles that change by a branch instruction. Therefore, constant-time implementation of MixColumns operation should be guaranteed by default.

In this article, we propose a new column-wise fashion implementation of AES on 8-bit AVR microcontroller. Our main strategy is to combine SubBytes, ShiftRows, and MixColumns operations. This allows a constant-time MixColumns implementation to be achieved. Also, we optimize the operation speed through efficient register scheduling. Unlike FACE-LIGHT requiring LookUp tables of 4 KB which need to be updated when IV value changes, our software utilizes a constant LookUp table of just 256-byte which is sufficient to be loaded in SRAM of 8-bit AVR MCUs. In addition, We apply our implementation in CTR\_DRBG, one of the upper algorithms of a symmetric-key algorithm, to prove the generality of our optimization technology in various operating modes of AES such as Cipher-Block-Chaining (CBC), CTR mode.

## A. CONTRIBUTIONS

The following list details the main contributions of our work:

- **Presenting optimized AES software on 8-bit AVR microcontroller**

We present an efficient AES software on 8-bit AVR MCUs, with the proposed column-wise operation technique. The proposed column-wise operation technique optimizes memory accesses for each state of AES by decreasing the redundant memory accesses. In addition, our implementation integrates a series of SubBytes, ShiftRows, and MixColumns operations into a single process. Thus, our AES software customized with assembly codes can encrypt a 128-bit data block in 2,251 clock cycles (resp. 2,706 and 3,160) with 128-bit (resp. 192-bit and 256-bit) security level. Our software outperforms all the existing AES softwares even FACE-LIGHT with respect to overall performance, including both LookUp table construction and encryption. Specifically, our software gives enhanced

performance compared with FACE-LIGHT until 1,850 blocks encryption (resp. 15,000 blocks) for 128-bit (resp. 192-bit) security. Note that our software always outperforms FACE-LIGHT without including LookUp table construction time for 256-bit security. Actually, as far as we know, for AES-256, our software is the fastest implementation in the world. Furthermore, our software operates in constant manner, which provides the security against timing-based side channel attack.

- **Proving the generality of the proposed AES software through CTR\_DRBG in 8-bit AVR MCUs**

Using the proposed AES implementation, we present CTR\_DRBG software on 8-bit AVR microcontroller. Unlike FACE-LIGHT, our optimized implementation can be applied to Derivation Function; also, our optimized AES-CTR mode can be applied to Update Function. The proposed implementation can expect improved performance regardless of the length of the extracted random bits. With our optimization technique, our Derivation Function implementation outperforms Derivation Function using Otte *et al.*'s implementations by 3.73%, 5.45%, and 5.64% for AES-128, AES-192, and AES-256, respectively. In addition, our Update Function implementation outperforms Update Function using Otte *et al.*'s implementations by 5.71%, 6.64%, and 6.56% for AES-128, AES-192, and AES-256, respectively. Finally, in AES-128, our CTR\_DRBG implementation, which uses the proposed CTR optimization technique, shows up to 10.04% performance improvement when extracting one thousand blocks of the random number, compared to Otte *et al.*'s CTR\_DRBG implementation.

- **Optimized AES implementation for a variety of applications**

we present column-wise AES-ECB mode; and, demonstrate that the proposed implementation is effectively applicable to CBC and CTR mode of the various function in CTR\_DRBG. In addition, we provide accurate performance figures for AES-128/192/256 and CTR\_DRBG (AES-128/192/256) using the proposed technique. The proposed column-wise AES implementation can be used not only in ECB, CBC, and CTR mode but also in all operation modes, e.g., CFB, OFB, CCM, and GCM mode. Unlike FACE-LIGHT, which requires 4 KB of memory space, our implementation, which requires only 512 bytes, is particularly effective on 8-bit AVR MCUs with limited memory and computing speed. Also, in CTR\_DRBG implementation, we demonstrate that using the proposed implementation provides performance improvement compared to the naive CTR\_DRBG implementation based on Otte *et al.*'s AES implementation. Furthermore, our proposed AES implementation is not only applicable to an 8-bit AVR MCUs, but also to other low-end-processors such as 16-bit MSP430, 32-bit ARM Cortex, and 32-bit RISC-V.

TABLE 1. Notations [12].

| Notation     | Meaning                                                                                              |
|--------------|------------------------------------------------------------------------------------------------------|
| $state$      | Internal state consisting of 16 bytes                                                                |
| $Nb$         | Number of columns (32-bit words) comprising the $state$                                              |
| $Nk$         | Number of 32-bit words comprising the $state$                                                        |
| $Nr$         | Number of AES Round                                                                                  |
| SubBytes     | Transformation using a non-linear byte substitution table (SBOX-Table)                               |
| ShiftRows    | Transformation cyclically shifting the last three rows of the $state$ by different offsets           |
| MixColumns   | Transformation taking all of the columns of the $state$ and mixing their data to produce new columns |
| AddRoundKey  | Transformation adding a round Key to the $state$ with XOR operation                                  |
| KeyExpansion | Routine generating a series of round Keys from a master key                                          |

This paper is organized as follows: Section II-A describes overview for the target cipher and target platform. Section III analyzes the existing two efficient AES implementations (Otte *et al.*'s implementation and FACE-LIGHT presented in ICISC'2019) in detail. Section IV propose an efficient AES implementation technique on AVR MCUs and Section VI evaluate the performance of our implementation. Finally, we conclude this paper in Section VII with some future works.

## II. OVERVIEW OF AES BLOCK CIPHER AND 8-BIT AVR MCUs

### A. NOTATIONS

Here, we describe the overview of the AES algorithm and then explain the features (memory size, instruction sets, and register sets) of the target 8-bit AVR-MCU. Table 1 shows the notations of AES block cipher.

### B. OVERVIEW OF AES BLOCK CIPHER

In 2001, three types of Rijndael algorithms were selected as AES standards, with encryption block of 128 bits in size and encryption keys of 128, 192, and 256 bits in length [12]. Algorithm 1 shows the overall structure of the AES. AddRoundKey updates the  $state$  by XORing the  $state$  with a round key generated by the KeyExpansion. AES is a Substitution Permutation Network (SPN) structure that consists of 10, 12, and 14 rounds depending on the key length. The last round does not perform MixColumns due to implementation efficiency and encryption/decryption-symmetry [14].

SubBytes is a nonlinear substitution operation of AES. It is an operation that performs affine-transformation after the inverse operation in the Rijndael field ( $GF(2^8)$ ), and secures stability for differential/linear analysis. For the efficiency of implementation, it is implemented in the form of SBOX-Table with byte input/output. SBOX-Table served this process as a LookUp table and the size of SBOX-Table is 256 bytes. ShiftRows operation is a

### Algorithm 1 Standard AES Encryption Algorithm [12]

**Require:** byte  $state[4 * Nb]$ , word  $rk[Nb * (Nr + 1)]$

**Ensure:** byte  $state[4 * Nb]$

- 1: KeyExpansion( $rk[Nb * (Nr + 1)]$ )
- 2: byte  $state[4 * Nb]$
- 3: AddRoundKey( $state, rk[0, Nb - 1]$ )
- 4: **for**  $rd(round) = 0$  to  $Nr - 1$  **do**
- 5:   SubBytes( $state$ )
- 6:   ShiftRows( $state$ )
- 7:   MixColumns( $state$ )
- 8:   AddRoundKey( $state, rk[rd * Nb, (rd + 1) * Nb - 1]$ )
- 9: **end for**
- 10: SubBytes( $state$ )
- 11: ShiftRows( $state$ )
- 12: AddRoundKey( $state, rk[rd * Nb, (rd + 1) * Nb - 1]$ )
- 13: **return** ( $state$ )

function to move position by row. Each row of the  $state$  is rotated to the left by 0, 1, 2, 3 from the top. Equation 1 is the MixColumns process mixing bytes in each columns to provide high diffusion.

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad (1)$$

In the Equation 1, each byte in a column is updated with following Equation 2.

$$\begin{aligned} S'_{0,c} &= (02 \cdot S_{0,c}) \oplus (03 \cdot S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S'_{1,c} &= S_{0,c} \oplus (02 \cdot S_{1,c}) \oplus (03 \cdot S_{2,c}) \oplus S_{3,c} \\ S'_{2,c} &= S_{0,c} \oplus S_{1,c} \oplus (02 \cdot S_{2,c}) \oplus (03 \cdot S_{3,c}) \\ S'_{3,c} &= (03 \cdot S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (02 \cdot S_{3,c}) \end{aligned} \quad (2)$$

**TABLE 2.** AVR assembly instructions [13].

| Instruction | Operands | Operation                                                             | Clocks |
|-------------|----------|-----------------------------------------------------------------------|--------|
| MOV         | $Rd, Rr$ | Copy register from $R_r$ to $R_d$ , $Rd \leftarrow Rr$                | 1      |
| LD          | $Rd, X$  | Load memory data from $X$ to $R_d$ , $Rd \leftarrow (X)$              | 2      |
| LDI         | $Rd, k$  | Load immediate constant from $k$ to $R_d$ , $Rd \leftarrow k$         | 1      |
| LPM         | $Rd, Z$  | Load program memory from $Z$ to $R_d$ , $Rd \leftarrow (Z)$           | 3      |
| ST          | $Rr, X$  | Store memory from $R_r$ to $X$ , $(X) \leftarrow Rr$                  | 2      |
| EOR         | $Rd, Rr$ | Exclusive-OR two register, $Rd \leftarrow Rd \oplus Rr$               | 1      |
| LSL         | $Rd$     | Logical shift left, $Rd(n) \leftarrow Rd(n)$ , $Rd(0) \leftarrow 0$   | 1      |
| SET         | None     | Set T in SREG, $T \leftarrow 1$                                       | 1      |
| RJMP        | $k$      | Relative jump, $PC \leftarrow PC + k + 1$                             | 2      |
| BRCC        | $k$      | Branch if Carry cleared, if $(C = 0)$ then $PC \leftarrow PC + k + 1$ | 1/2    |

In the above system,  $(02 \cdot S_{i,c})$  can be computed as a multiplication in  $GF(2^8)$  with an irreducible polynomial  $x^8 + x^4 + x^3 + x + 1$ .  $(03 \cdot S_{i,c})$  can be executed with a multiplication in  $GF(2^8)$  and extra addition in  $GF(2^8)$ . Since all bytes in the state need to be updated following Equation 2, MixColumns causes the most computational load in AES.

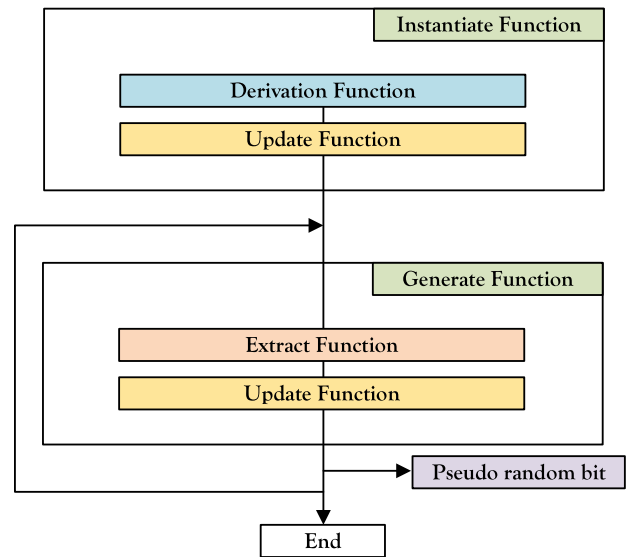
### C. OVERVIEW OF 8-BIT AVR MCUs

The 8-bit AVR microcontroller is an embedded device made into a single integrated circuit by adding memory and I/P to the microprocessor. And this microcontroller is currently the most used worldwide in the WSNs environment [16], [17]. The 8-bit AVR microcontroller's commands comprise operation codes and an operand. Table 2 shows the operands and clock cycles of the commands used in this paper [13]. The 8-bit AVR-MCU consists of flash memory, SRAM, and EEPROM with a Harvard architecture. The AVR-MCU has 32 8-bit general-purpose registers, which are used for various purposes, e.g., basic private operations and bit operations. The Status REGISTER (SREG) shows the status and result after Arithmetic Logic Unit (ALU) calculations [5].

### D. OVERVIEW OF CTR\_DRBG

The secure key is a fundamental prerequisite for operating secure block cipher algorithms to protect secret information. To securely generate secret keys, a random bit generator needs to be utilized [18]. The ideal random bit generator should satisfy the following three conditions: unpredictability, unbiasedness, and inter-bit independence [15]. The random bit that satisfies these three conditions is called the true random bit.

Theoretically, however, the creation of a true random bit is almost impossible [18]. Therefore, it generates and uses pseudo-random bit that are difficult to distinguish from the true random bit. As pseudo-random bit, Deterministic Random Bit Generators (DRBGs) are used to securely generate random bit information

**FIGURE 1.** Overview of CTR\_DRBG [15].

including secret keys, Initial Vectors (IV), nonces, and so on. CTR\_DRBG use block cipher algorithm, HASH\_DRBG use hash function, and HMAC\_DRBG use HMAC, among other types of pseudo-random number generators. We note the cryptographic algorithm (AES) used in CTR\_DRBG and the CTR modes used in Update Function and Extract Function. Various symmetric key algorithms (AES, LEA, HIGHT and so on) used in CTR\_DRBG, unlike the HASH functions (SHA-2 Family) used in HASH\_DRBG and HMAC\_DRBG, can store the internal state in the general purpose register and reduce memory access efficiently in 8-bit AVR-MCUs.

Table 3 defines the notations used in this paper. Table 4 defines the constant parameters used in this paper. Figure 1 shows an overview of CTR\_DRBG, which has no option for Reseed Function. To extract a random number, first, initialize the Internal State by executing the Instantiate Function; and, it uses the Generate Function to extract random

**TABLE 3.** Notations for CTR\_DRBG [15].

| Notation               | Descriptions                                                                                                                                                             |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Internal State         | Representing the internal state of CTR_DRBG, and includes values of variables and variable information                                                                   |
| Operational Status     | Information directly used for random number output. Consisting of C and V, where C is the key used for a block cipher, and V is the plaintext used for the block cipher. |
| Instantiate Function   | Function to create and initialize CTR_DRBG instances as needed.                                                                                                          |
| Derivation Function    | Function to receives input variables from the user and generates a seed value                                                                                            |
| Update Function        | Function to update the internal state.                                                                                                                                   |
| Generate Function      | Function to update internal state according to options and generate random-bit                                                                                           |
| Extract Function       | Function to extract random-bit using internal state and CTR mode                                                                                                         |
| Uninstantiate Function | Function to make Internal State zero and delete instances after CTR_DRBG use.                                                                                            |

**TABLE 4.** Constant parameters of CTR\_DRBG using AES [15].

| Parameters | AES-128 | AES-192 | AES-256 |
|------------|---------|---------|---------|
| Key Bit    | 128     | 192     | 256     |
| Block Bit  | 128     | 128     | 128     |
| Seed Bit   | 256     | 320     | 384     |
| Len_seed   | 2       | 3       | 3       |

numbers. Derivation Function makes use of CBC-MAC in order to produce an output, which has the length of Seed Bit, by inputting variable-length S. The Derivation Function is called in the Instantiate Function. Update Function updates C and V of Operational Status in Internal State via CTR mode using input data. Update Function is called by some functions, such as the Instantiate Function and Generate Function. Using C as the Key and V as the CTR, the Update Function performs encryption as much Len\_seed with CTR mode. The Generate Function consists of Extract Function, and Update Function. The Generate Function generates random numbers using Operational Status C and V, and updates the Operational Status using Update Function. Extract Function is a function that uses the same CTR mode as Update Function. Therefore, using Operational Status C as the key and V as the CTR, Extract Function generates random numbers using CTR mode by the length of random numbers set by the user. If there is no additional random number generation, then the use of CTR\_DRBG is terminated, otherwise the process for additional random number generation is returned.

### III. ANALYSIS OF EXISTING AES IMPLEMENTATIONS ON 8-BIT AVR MCUS

In this section, we analyze Otte et al.'s AES implementation in AVR-Crypto Library [9], which is widely used, in detail from a basic operation point of view; and, discuss the limitations of FACE-LIGHT recently proposed in ICISC'19 [11]. We will

present a new implementation technique that can handle the discussed weakness in Section IV.

#### A. ANALYSIS OF OTTE ET AL.'S IMPLEMENTATION

Due to the characteristic of AVR with 32 general-purpose registers of 8-bit size, it seems that the internal state of 16 bytes of AES can be maintained in the general-purpose register and can be easily implemented. However, in the case of an AVR environment, there are only two operands of the instruction, and it is quite complicated to arrange the registers efficiently and design the operation structure because multiple storage values must be used when performing complex operations such as MixColumns.

Algorithm 2 shows the assembly code for the MixColumns by Otte *et al.* [9]. It performs MixColumns corresponding to Equation 1,2 by 4 iterations. In order to perform iteration efficiently, Otte *et al.* operate MixColumns using the macro **.irp** and **.endr** in lines 1 and 9. In the case of Line 2-10 of Algorithm 2, MixColumns performs an operation of the first row of MixColumns matrix. At this time, the multiplication operation in  $GF(2^8)$  is performed by LSL instruction. when carry occurs in multiplication,  $0 \times 1B$  is XORed for operating reduction. If this process is repeated 4 times, MixColumns for one column in the internal state of AES can be completed, and the internal state is finally updated through a 4 times iteration. Since only 3 Temp registers are used for MixColumns except for 16 registers for internal state storage, Otte *et al.*'s implementation is an efficient MixColumns implementation.

However, Otte *et al.*'s MixColumns implementation using BRCC instruction is a non constant-time implementation. There are some differences depending on the operating mode, but master-key is used continuously in most cryptographic protocols in the WSN environment. In AddRoundKey of Round 0, the master-key information remains in the internal state until the master-key is XORed and spreads in the 1 Round of MixColumns. In addition, the result of SubBytes and ShiftRows is information that an attacker can

**Algorithm 2** MixColumns of Otte *et al.* [9]

---

|                                                                          |                                                   |                                                |
|--------------------------------------------------------------------------|---------------------------------------------------|------------------------------------------------|
| <b>Require:</b> byte $ST_{ij}$ , ( $i, j \in 0.3$ )                      | 10: EOR $ST\backslash row\backslash()0$ , $1B$    | 21: BRCC $3f$                                  |
| <b>Ensure:</b> byte $ST_{ij}$ , ( $i, j \in 0.3$ )                       | 11: 3: EOR $ST\backslash row\backslash()0$ , $r0$ | 22: EOR $T2$ , $1B$                            |
| 1: .irp row, in 0,1,2,3                                                  | 12: EOR $ST\backslash row\backslash()0$ , $T0$    | 23: 3: EOR $T2$ , $r0$                         |
| 2: MOV $r0$ , $ST\backslash row\backslash()2$                            | 13: MOV $T1$ , $ST\backslash row\backslash()1$    | 24: EOR $ST\backslash row\backslash()2$ , $T2$ |
| 3: EOR $r0$ , $ST\backslash row\backslash()3$                            | 14: EOR $T1$ , $ST\backslash row\backslash()2$    | 25: EOR $T0$ , $ST\backslash row\backslash()3$ |
| 4: MOV $T2$ , $r0$                                                       | 15: LSL $T1$                                      | 26: LSL $T0$                                   |
|                                                                          | 16: BRCC $3f$                                     | 27: BRCC $3f$                                  |
| 5: MOV $T0$ , $ST\backslash row\backslash()0$                            | 17: EOR $T1$ , $1B$                               | 28: EOR $T0$ , $1B$                            |
| 6: EOR $ST\backslash row\backslash()0$ , $ST\backslash row\backslash()1$ | 18: 3: EOR $T1$ , $r0$                            | 29: 3: EOR $T0$ , $r0$                         |
| 7: EOR $r0$ , $ST\backslash row\backslash()0$                            | 19: EOR $ST\backslash row\backslash()1$ , $T1$    | 30: EOR $ST\backslash row\backslash()3$ , $T0$ |
| 8: LSL $ST\backslash row\backslash()0$                                   | 20: LSL $T2$                                      | 31: .endr                                      |
| 9: BRCC $3f$                                                             |                                                   |                                                |

---

easily predict; therefore, some information of master-key is leaked due to the change of clock cycles by carry during MixColumns. Since master-key information leakage can be fatal from a side-channel-attack perspective, countermeasure of non constant-time issue are essential.

In Otte *et al.*'s AES implementation, SubBytes and ShiftRows are implemented as a combined operation. 8-bit unit rotate operation of ShiftRows can be performed implicitly by allocating registers directly. In addition, since the AVR environment is not a high-performance embedded device such as ARM and the register is 8-bit, it is efficient to implement it using SBOX-Table rather than the Bit-Slicing technique. In Otte *et al.*'s implementation, 256 bytes of SBOX-Table are stored in Flash memory. When data is loaded into registers, there is a difference of 1 clock cycle depending on the space stored. Data stored in Flash memory costs 3 clock cycles when loaded, vice versa data stored in SRAM costs 2 clock cycles when loaded. Therefore, in 8-bit AVR microcontroller other than an ATiny processor, which is a very constrained device, it may be useful to store SBOX-Table in SRAM.

## B. ANALYSIS OF FACE-LIGHT

In CHES' 2018, Fast AES CTR mode Encryption (FACE) was proposed [19]. FACE is an algorithm that increases the encryption speed by replacing the Repetitive part with LookUp table using the characteristics of CTR mode. However, since FACE requires a 5KB lookup table and needs to be updated at regular intervals, there are many limitations in applying it to low-end devices such as 8-bit AVR or 16-bit MSP. To solve this issue, FACE-LIGHT was proposed at ICISC' 2019 [11]. Unlike FACE, FACE-LIGHT proposed in the 8-bit AVR environment has the advantage that it generates a 4KB LookUp table and does not need to update its table.

Figure 2 shows an overview of FACE-LIGHT. Note that the input data for encryption in CTR mode is the IV. FACE-LIGHT uses a CTR mode consists of a 32-bit counter and a 96-bit nonce. The main idea of FACE-LIGHT is to generate a LookUp table until each 8-bit

of the counter affects each other. Since only  $S[0] - S[3]$  is the part that changes according to the counter in the 0 Round, the value corresponding to  $S[4] - S[15]$  can be pre-computed. SubBytes and ShiftRows in 1 Round can be pre-computed because only 4 bytes are changed from the previous encryption block. In the case of the 1Round MixColumns, the value of each counter is spread in the direction of the column. Here, the counters do not affect each other; therefore, pre-computation is possible. The area affected by the 8-bit counter is 4 bytes. In SubBytes and ShiftRows of 2 Round, the values do not spread, so they can be saved as a LookUp table. However, in MixColumns of 2 Round, all value affected by counters is mixed. Therefore, In here, it is difficult to predict the value. Therefore, in the case of FACE-LIGHT, all encryption processes up to the 2 Round MixColumns can be omitted using the LookUp table. Since 8-bit counters of IV have 256 cases and 8-bit counters are 4, a total of  $4KB = 4096$  bytes ( $4 \times 4 \times 256$ ) of LookUp table is required.

FACE-LIGHT, which omits about 2 rounds of encryption using a 4KB LookUp table, can be expected to show high performance in communication environments such as TLS/SSL using CTR mode. However, there are some issues in applying FACE-LIGHT in the actual WSNs environment. First, there are limitations in generating the LookUp table due to session changes. In the case of the FACE-LIGHT proposed in the fixed key scenario, a new LookUp table should be generated when the session changes. Algorithm 3 shows the code which generates the LookUp table. The code of FACE-LIGHT is open source, but the part about generating the LookUp table is omitted [20]. Therefore, we estimate the cost of generating the LookUp table by referring to the FACE-LIGHT paper and open source code. FACE-LIGHT follows the implementation of Otte *et al.*, except for some code referencing the LookUp table. We conservatively measured FACE-LIGHT's LookUp table generation cost estimate. In other words, additional overhead may occur during the actual generation of a LookUp table. From this, we estimated that it takes about 15,360 clock cycles to generate a LookUp table for an 8-bit counter. Since FACE-LIGHT uses a total of 32-bit counters,

Initialization Vector (128bits Counter Value: 32 Counter + 96 Nonce)

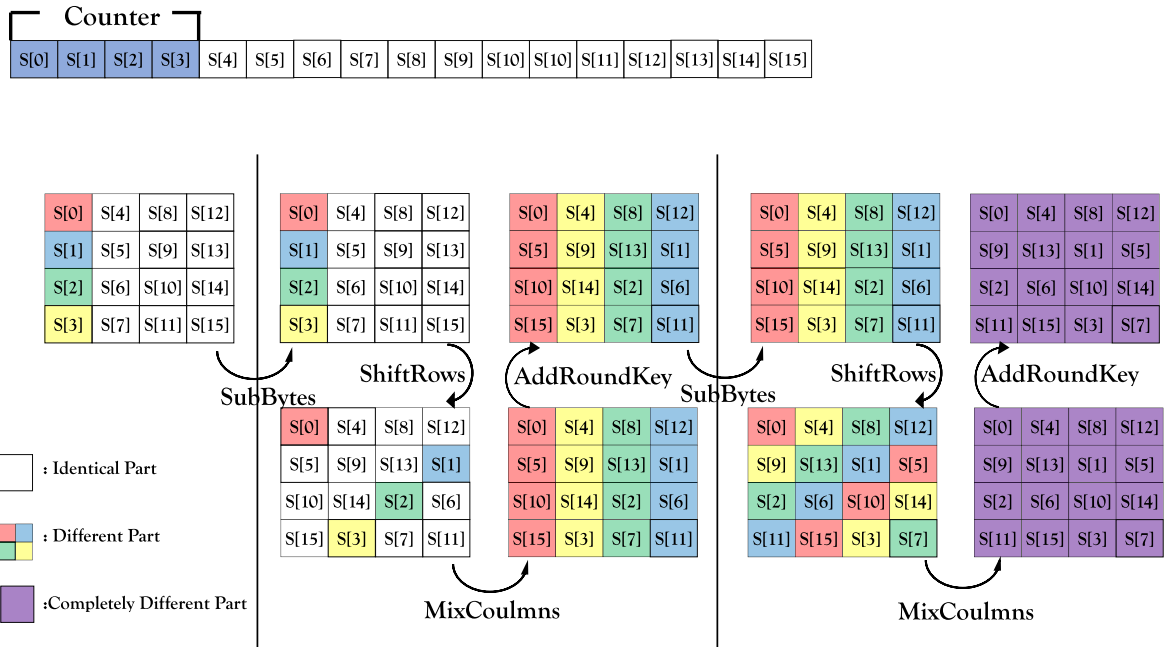


FIGURE 2. Overview of FACE-LIGHT [11].

**Algorithm 3** Generation of Pre-Computation Table for S[3] of IV in FACE-LIGHT

**Require:** byte *Counter*[4], byte *key*[4]

**Ensure:** byte *LookUpTable*[4][256]

```

1: byte state[4]
2: for i = 0 to 255 do
3:   state = Counter
4:   MixColumns(state)
5:   AddRoundKey(Counter, key)
6:   Subbyte(state)
7:   LookUpTable[4][i] = state
8:   Counter = Counter + 1
9:   i = i + 1
10: end for

```

it can be inferred that approximately 61,440 clock cycles are required to generate a 4KB LookUp table. This is approximately equal to the cost of 24 AES encryption. In other words, if the session changes in cryptographic protocol, users have to generate a new master-key; therefore, it is difficult to apply FACE-LIGHT technique in WSNs environment that session changes are frequent.

Also, additional issues may arise when generating a 4KB LookUp table of FACE-LIGHT. In the case of ATmega 328p, the target device of FACE-LIGHT, SRAM is 2KB, so the 4KB LookUp table must be saved in Flash memory. Since the 8-bit AVR microcontroller cannot store data in Flash memory during communication and function operation, rebooting

using a boot loader is essential to exchange session keys with the server and generate the LookUp tables. Therefore, the FACE-LIGHT is a difficult algorithm to actually apply in the WSNs environment. In addition, FACE-LIGHT is the same as Otte *et al.*'s implementation except for the second round optimization; therefore, at the 192-bit and 256-bit security levels where the number of encryption rounds increases, the optimization performance is degraded, and there is a non constant-time issue of Otte *et al.*'s MixColumns implementation. Therefore, in the next section, we propose an efficient AES implementation for the WSNs environment.

#### IV. PROPOSED AES IMPLEMENTATION ON 8-BIT AVR MCUs

##### A. MAIN IDEA

As mentioned in Section III, the existing AES optimization implementations had some limitations. In this section, we introduce a new implementation that fixes the shortcomings of the existing implementation.

Unlike the existing implementation, we propose a 256 bytes MIX-Table to achieve a constant-time implementation. In the case of the existing implementation methodology in the 8-bit AVR environment, the fastest MixColumns implementation was achieved; however, there is a disadvantage that the use of branch instructions is inevitable such as BRCC instruction. In our implementation,  $02 \cdot S_{i,c}$ , ( $i \in [0, 3]$ ) of Equation 2 is stored in the form of MIX-Table like SBOX-Table. MIX-Table is a constant table that has a size of 256 bytes and does not need to be updated. However, even if a constant-time implementation is achieved using MIX-Table, performance may be degraded due to the addi-

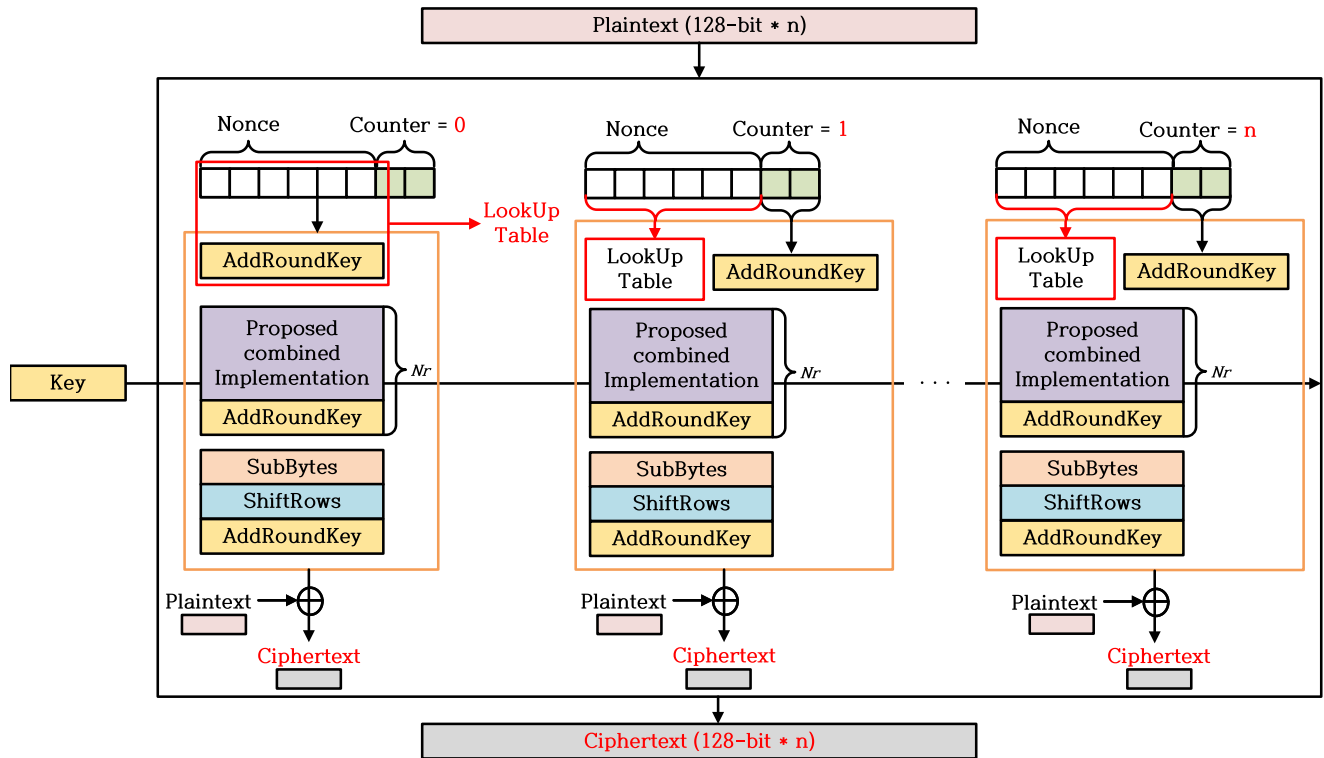


FIGURE 3. Overall structure of the proposed AES CTR software.

tional computational load. For using this technique, we need to compare the cost of clock cycles incurred when using a branch instruction and using a MIX-Table. According to Algorithm 2, for MixColumns of Otte *et al.*, a total of 16 branch instructions are used. In here, the expected value is 24 ( $24 = 16 \times 1.5$ ) clock cycles. In the other case of using MIX-Table, the 8-bit value of the internal state should move to the Z register and then refer to the memory. In here, the cost needed to MIX-Table is 48 ( $48 = 16 \times (1 + 2)$ ) clock cycles. From this, using a MIX-Table can seem inefficient.

Therefore, to solving this problem, we propose a new implementation technique using column-wise fashion in an 8-bit AVR environment. Otte *et al.*'s implementation of MixColumns is a row-wise fashion implementation, and the internal state of AES is calculated for each row of MixColumns. However, our implementation operates the columns of the MixColumns matrix as the starting point. Through this, the three main operations of AES (SubBytes, ShiftRows, and MixColumns) can be calculated at once. In addition, we adjust SBOX and MIX-Table to be placed in the SRAM. Data stored in flash memory requires 3 clock cycles to load into registers, but 2 clock cycles to load data stored in SRAM. Through this, our implementation can benefit from 1 clock cycles per memory access in SubBytes operation.

Figure 3 shows the overview of the proposed AES encryption software using CTR mode of operation. Proposed

combined implementation is used repeatedly for the number of rounds of AES. Therefore, unlike FACE-LIGHT, which optimized up to the initial 3 Rounds of AES, the proposed implementation can be applied to all rounds of AES. In addition, we present an optimized software for AES-CTR mode. Instead of calculating all values according to the counter, we struct a LookUp table that omits 0 round AddRoundKey operation, corresponding to a fixed nonce value of Initial Vector (IV).

## B. PROPOSED REGISTER SCHEDULING

Since the size of the general-purpose register of AVR is 8-bit, it is effective to update AES internal state by making the most of the register. In our column-wise implementation, since MIX-Table is used differently from the existing implementation, temporary registers for the variables to be actually handled are required. Figure 4 represent the register scheduling used in our software. In the case of Otte *et al.*'s implementation, the internal state of AES is arranged and maintained in order from R0-R15. However, in our software, we place some registers of the state in reverse order. By using this arrangement, we can use MOVW instructions and reduce some clock cycles. Eight temporary registers are used to maintain the middle value of the operation.  $M_i$  and  $T_i$  ( $i \in [0, 3]$ ) temporary registers store  $S'_{i,c}$ , ( $i \in [0, 3]$ ) of MixColumns operation in Equation 2. The X, Y, and Z register store the address of key used in AddRoundKey, MIX-Table, and SBOX-Table, respectively.

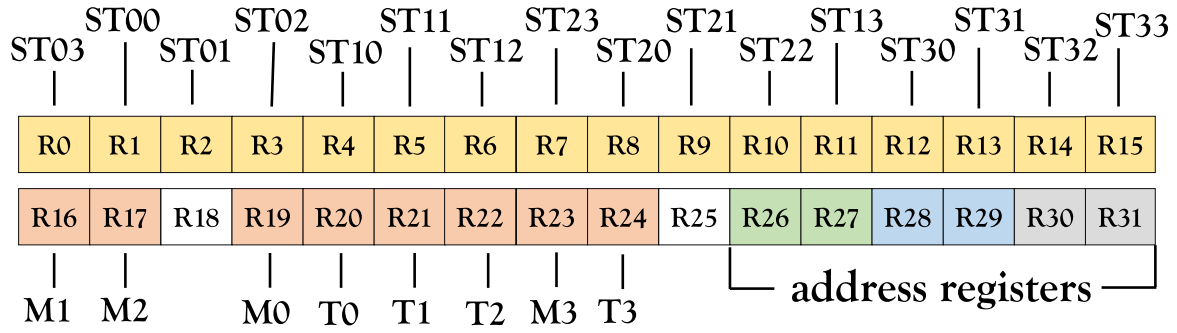


FIGURE 4. Register scheduling for proposed implementation in the target AVR MCUs.

$$\bar{S}_{i,c} = SBOX(S_{i,c \ll i}) \text{ and } \tilde{S}_{i,c} = MIX(S_{i,c \ll i}), \text{ for } i \in [0, 3]$$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \underbrace{\begin{bmatrix} \tilde{S}_{0,c} \\ \bar{S}_{0,c} \\ \bar{S}_{0,c} \\ \tilde{S}_{0,c} \oplus \bar{S}_{0,c} \end{bmatrix}}_A \oplus \underbrace{\begin{bmatrix} \tilde{S}_{1,c} \oplus \bar{S}_{1,c} \\ \tilde{S}_{1,c} \\ \bar{S}_{1,c} \\ \bar{S}_{1,c} \end{bmatrix}}_B \oplus \underbrace{\begin{bmatrix} \bar{S}_{2,c} \\ \tilde{S}_{2,c} \oplus \bar{S}_{2,c} \\ \tilde{S}_{2,c} \\ \bar{S}_{2,c} \end{bmatrix}}_C \oplus \underbrace{\begin{bmatrix} \bar{S}_{3,c} \\ \bar{S}_{3,c} \\ \tilde{S}_{3,c} \oplus \bar{S}_{3,c} \\ \tilde{S}_{3,c} \end{bmatrix}}_D$$

FIGURE 5. Equation for proposed implementation which combine SubBytes, ShiftRows, and MixColumns into a single process.

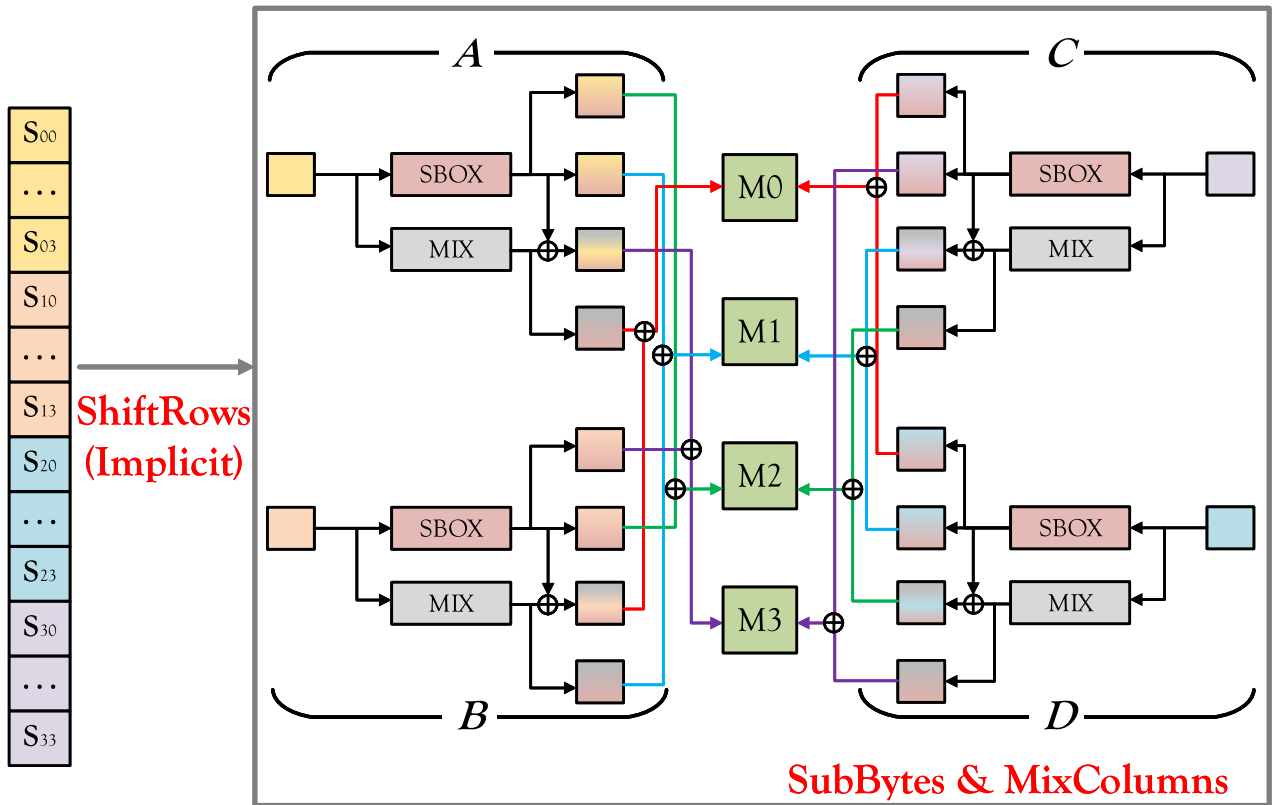


FIGURE 6. Work flow for the proposed equation in Figure 5.

**Algorithm 4** Codes for Figure 5,6 Proposed Combined Subbytes, ShiftRows and MixColumns Operations With Equation 5 When  $c = 0$

**Require:** byte  $STij$ , ( $i, j \in 0.3$ ), in  $R0-R15$

**Ensure:** byte  $STij$ , ( $i, j \in 0.3$ ), in  $R0-R15$

**Part A Computation**

```
1: MOV r30, ST00
2: LD M1, Z // M1 ←  $\tilde{S}_{0,0}$ 
3: MOV M2, M1 // M2 ←  $\tilde{S}_{0,0}$ 
4: MOV M3, M2
5: MOV r28, ST00
6: LD M0, Y // M0 ←  $\tilde{S}_{0,0}$ 
7: EOR M3, M0 // M3 ←  $\tilde{S}_{0,0} \oplus \tilde{S}_{0,0}$ 
```

**Part B Computation**

```
8: MOV r30, ST11
9: LD T0, Z
10: EOR M0, T0
11: EOR M2, T0 // M2 ←  $M2 \oplus \tilde{S}_{1,0}$ 
12: EOR M3, T0 // M3 ←  $M3 \oplus \tilde{S}_{1,0}$ 
13: MOV r28, ST11
14: LD T0, Y
15: EOR M1, T0 // M1 ←  $M1 \oplus \tilde{S}_{1,0}$ 
16: EOR M0, T0 // M0 ←  $M0 \oplus \tilde{S}_{1,0} \oplus \tilde{S}_{1,0}$ 
```

**Part C Computation**

```
17: MOV r30, ST22
18: LD T0, Z
19: EOR M0, T0 // M0 ←  $M0 \oplus \tilde{S}_{2,0}$ 
20: EOR M1, T0
21: EOR M3, T0 // M3 ←  $M3 \oplus \tilde{S}_{2,0}$ 
22: MOV r28, ST22
23: LD T0, Y
24: EOR M2, T0 // M2 ←  $M2 \oplus \tilde{S}_{2,0}$ 
25: EOR M1, T0 // M1 ←  $M1 \oplus \tilde{S}_{2,0} \oplus \tilde{S}_{2,0}$ 
```

**Part D Computation**

```
26: MOV r30, ST33
27: LD T0, Z
28: EOR M0, T0 // M0 ←  $M0 \oplus \tilde{S}_{3,0}$ 
29: EOR M1, T0 // M1 ←  $M1 \oplus \tilde{S}_{3,0}$ 
30: EOR M2, T0
31: MOV r28, ST33
32: LD T0, Y
33: EOR M3, T0 // M3 ←  $M3 \oplus \tilde{S}_{3,0}$ 
34: EOR M2, T0 // M2 ←  $M2 \oplus \tilde{S}_{3,0} \oplus \tilde{S}_{3,0}$ 
```

### C. PROPOSED IMPLEMENTATION ON 8-BIT AVR MCUS

Our column-wise implementation updates  $[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}]$  simultaneously over 4 processes. In the case of the existing MixColumns implementation, as shown in Otte *et al.*'s MixColumns based on Equation 2,  $S'_{1,c}$  is calculated after the  $S'_{0,c}$  calculation is completed. On the other hand, in column-wise implementation, The value of  $[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}]$  is updated simultaneously. Figure 5 shows the detailed Equation of our constant-time software. The proposed Implementation updates the four  $S'_{i,c} (i \in [1, 3])$  by four stages: A, B, C, and D. The advantage of the proposed software is that SubBytes, ShiftRows, and MixColumns can be combined into one process, using MIX-Table mentioned in section IV-A. Using SBOX and MIX-Table in step A of Figure 5, we calculate  $\tilde{S}_{0,c}$  and  $\tilde{S}_{0,c}$ ; Here,  $\tilde{S}_{0,c}$  is the value where SubBytes is completed, and  $\tilde{S}_{0,c}$  is the value where  $02 \cdot S_{0,c}$  calculation of MixColumns is completed. After that, the process for Part A is completed through XOR-operation. In the case of ShiftRows, since the general-register size is 8-bit in AVR environments, it can be executed implicitly with no additional overhead. If this process is repeated 4 times, performing  $A \rightarrow B \rightarrow C \rightarrow D$  parts, the calculation for  $[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}]$  is completed. Figure 6 shows the workflow for implementing the proposal. When the calculation for  $[S'_{0,1}, S'_{1,1}, S'_{2,1}, S'_{3,1}]$  is completed, the value except  $M0$  is kept without updating to the state. After that, the calculation for  $[S'_{0,2}, S'_{1,2}, S'_{2,2}, S'_{3,2}]$  uses  $T0, T1, T2$ , and  $T3$  registers. The reason Temp Register is actively used is because of ShiftRows operation. If the calculated value from  $M1, M2$ , and  $M3$  is immediately updated to the state, it affects  $[S'_{0,2}, S'_{1,2},$

$S'_{2,2}, S'_{3,2}]$ . Therefore, the state is continuously updated while using  $Mi$  and  $Ti$   $i \in [0, 3]$  registers alternately. Since using temp-register has no additional overhead, our constant-time implementation is highly efficient.

Algorithm 4 is an 8-bit AVR assembly code for the workflow in figure 6. The presented Algorithm is the Assembly code that calculates  $[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}]$  when  $c = 0$  in Figure 5. In other words, if Algorithm 4 is repeated 4 times, SubBytes, ShiftRows, and MixColumns for one round of AES are completed. As mentioned earlier, the temp register to store each calculated value is used alternately. Lines 1-7 are the assembly code for part A in Figure 6. Here,  $S_{0,0}$  is stored in  $ST00$ . Via line 1-2, our software references SBOX-Table value and load the value. To store the value of Part A in the  $Mi$  ( $i \in [0, 3]$ ) register, our implementation copies the value through Line 3-4 and refers to the MIX-Table value through Line 5-6. Finally, in line 7,  $(02 \cdot S_{0,0}) \oplus (03 \cdot S_{0,0})$  is calculated through XOR-operation. Using the same logic, Part B, C, and D can be calculated; therefore, if Algorithm 4 repeat 4 times, SubBytes, ShiftRows, and MixColumns are completed. Since we do not use the branch statement, our software is a constant-time implementation. Unlike the existing Otte *et al.* and FACE-LIGHT's implementation, we benefit 16 clock cycles per round because of SBOX-Table stored in SRAM. FACE-LIGHT can only be applied in CTR mode; however, our implementation is applicable to all operation modes such as CBC mode, OFB mode, and CFB mode. In addition, since it can be applied compactly to all rounds of AES, the higher the security level, the higher the optimization efficiency.

#### D. CTR OPTIMIZATION

Among several encryption operation modes, CTR mode is mainly used in security protocols such as TLS/SSL [21]. In particular, in an edge network environment using sensor nodes based on an embedded device, since session changes are irregular, it is recommended to use CTR mode which facilitates parallelism of encryption/decryption and error recovery [1]. FACE-LIGHT suggested an optimization plan for the initial 3 Rounds, as mentioned in section III, there is a disadvantage of generating a LookUp table every time a session starts. Since the 4KB pre-computation table of FACE-LIGHT must be calculated in advance, it is not free to use in the actual sensor network.

The proposed column-wise fashion implementation is applicable to all operation modes. In addition, we present optimization software for the CTR mode based on the proposed constant-time implementation. The most important thing when considering practicality and applicability as a focus is to minimize the extra overhead of applying the optimization method even if the session changes. FACE and FACE-LIGHT generate a pre-computation table for all cases where the counter value changes; however, we focused on unchanging nonces of IV, not counter values of IV, and focused on generating the LookUp table during encryption, not pre-computation. As a result of configuring various scenarios, it was confirmed that it is effective when generating a LookUp table in Round 0. The operation of 0 Round is only AddRoundKey; therefore, 12 bytes of nonce excluding counter of IV can be predicted. Most importantly, we can generate a LookUp Table during the encryption process. Therefore, AddRoundKey in Round 0 after the first encryption process is completed, it can be saved and used in the stack through LD instruction. The cost of generating the LookUp table is only 24 clock cycles. When using the LookUp table, our software can omit the operation of loading IV and master-key in 0 Round.

In summary, the CTR mode based on proposed AES column-wise software is effective in the WSN environment where sessions are frequently changed compared to FACE-LIGHT. In the case of FACE-LIGHT, there is a high cost of generating a pre-computation table, so in order to gain the actual performance, FACE-LIGHT needs to proceed with the encryption process more than 24 times. However, in our AES-CTR mode implementation, if only 2 encryption are performed, we can get a greater benefit than the cost of generating the LookUp table.

#### V. PROPOSED AES-BASED CTR\_DRBG IMPLEMENTATION ON 8-BIT AVR MCUs

Our proposed implementation of AES-CTR mode is effectively available for all functions of CTR\_DRBG. As explained in Section II-D, Derivation Function makes use of CBC-MAC in order to produce an output, which has the length of Seed Bit, by inputting variable-length S. Since Derivation function uses CBC mode, FACE-LIGHT

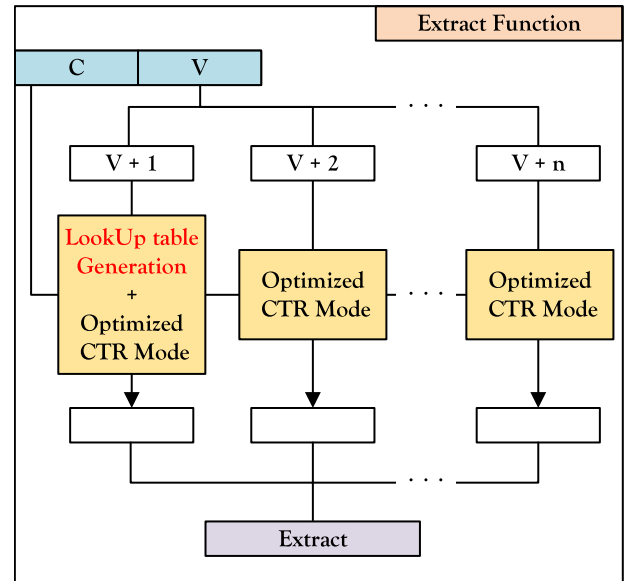


FIGURE 7. Overview of optimized Extract Function.

based on CTR mode cannot be used; however, our proposed implementation can be effectively applied to the AES-CBC mode of Derivation Function.

In Derivation Function, encryption is performed according to the length of Personalization String and Nonce (input data of Derivation Function). These non-deterministic parts require at least 112-bit data [15]. In general, in an 8-bit AVR environment, the size of input data is mainly used as 64 bytes due to resource constraints [4]. In this case, if the 128-bit key is used, the implementation proposed in this paper can be applied to 14 AES-CBC encryption, and if a key of 192-bit or higher is used, our implementation can be applied to 21 AES-CBC encryption.

Update Function performs CTR mode encryption as much as Len seed. For CTR\_DRBG using AES encryption, Len seed is up to 3. The optimization method of FACE-LIGHT is to generate a LookUp table for all CTR values; however, it is difficult to apply the optimization method of FACE-LIGHT to Update Function in reality, because AES-CTR mode is called up to 3 times in Update Function and the cost of creating LookUp table of FACE-LIGHT is 61,440 clock cycle. Unlike FACE-LIGHT, the proposed AES-CTR mode implementation has the advantage of not only requiring 24 clock cycles to generate LookUp table but also being able to generate the LookUp table during the first AES-CTR mode encryption process. By using this, it is possible to optimize the CTR mode encryption process up to 3 times in Update Function. Update Function is called at least twice in CTR\_DRBG, therefore up to six CTR mode encryption processes can be optimized.

Extract Function executes CTR mode and extracts random numbers using C in the Internal State as Key and V as Counter. After that, Generate Function calls Update Function to update the Internal State. The Internal State is updated continuously unless the UnInstance function is called.

**TABLE 5.** Comparison of AES implementations on AVR in terms of clock cycles.

| Security       | Dinu et al. [8] | Otte et al. [9] | FACE-Light [11] | This work | This work(CTR) |
|----------------|-----------------|-----------------|-----------------|-----------|----------------|
| <b>AES-128</b> | 2,835           | 2,507           | 2,218           | 2,289     | 2,251          |
| <b>AES-192</b> | N/A             | 2,991           | 2,702           | 2,746     | 2,706          |
| <b>AES-256</b> | N/A             | 3,473           | 3,184           | 3,209     | 3,160          |

**TABLE 6.** Running time comparison of AES implementations using 128, 192, and 256-bit key (Timings are measured by clock cycles. *NB* means the number of blocks).

| AES-CTR mode using 128-bit key |                     |                      |                      |                      |                        |                        |
|--------------------------------|---------------------|----------------------|----------------------|----------------------|------------------------|------------------------|
| <i>NB</i>                      | 1                   | 50                   | 100                  | 200                  | 800                    | 1,850                  |
| <b>Otte et al. [9]</b>         | 2,507<br>( - )      | 125,350<br>( - )     | 250,700<br>( - )     | 501,400<br>( - )     | 2,005,600<br>( - )     | 4,665,527<br>( - )     |
| <b>FACE-LIGHT. [11]</b>        | 63,658<br>(-2,439%) | 172,340<br>(-37.49%) | 283,240<br>(-12.98%) | 505,040<br>(-0.73%)  | 1,835,840<br>(+8.46%)  | 4,189,138<br>(+10.20%) |
| <b>This Work<br/>(CTR)</b>     | 2,275<br>(+9.3%)    | 112,574<br>(+10.2%)  | 225,124<br>(+10.20%) | 450,224<br>(+10.20%) | 1,800,824<br>(+10.20%) | 4,189,135<br>(+10.20%) |

| AES-CTR mode using 192-bit key |                     |                      |                       |                       |                        |                        |
|--------------------------------|---------------------|----------------------|-----------------------|-----------------------|------------------------|------------------------|
| <i>NB</i>                      | 1                   | 100                  | 500                   | 1,000                 | 4,000                  | 15,000                 |
| <b>Otte et al. [9]</b>         | 2,991<br>( - )      | 299,100<br>( - )     | 1,495,500<br>( - )    | 2,991,000<br>( - )    | 11,964,000<br>( - )    | 44,865,000<br>( - )    |
| <b>FACE-LIGHT. [11]</b>        | 64,142<br>(-2,044%) | 331,640<br>(-10.88%) | 1,412,440<br>(+5.55%) | 2,763,440<br>(+7.61%) | 10,869,440<br>(+9.15%) | 40,591,440<br>(+9.53%) |
| <b>This Work<br/>(CTR)</b>     | 2,730<br>(+8.73%)   | 270,624<br>(+9.52%)  | 1,353,024<br>(+9.53%) | 2,706,024<br>(+9.53%) | 10,824,024<br>(+9.53%) | 40,590,024<br>(+9.53%) |

| AES-CTR mode using 256-bit key |                     |                     |                       |                        |                        |                        |
|--------------------------------|---------------------|---------------------|-----------------------|------------------------|------------------------|------------------------|
| <i>NB</i>                      | 1                   | 100                 | 500                   | 5,000                  | 15,000                 | 25,000                 |
| <b>Otte et al. [9]</b>         | 3,473<br>( - )      | 347,300<br>( - )    | 1,736,500<br>( - )    | 17,365,000<br>( - )    | 52,095,000<br>( - )    | 86,825,000<br>( - )    |
| <b>FACE-LIGHT. [11]</b>        | 64,624<br>(-1,760%) | 379,840<br>(-9.37%) | 1,653,440<br>(+4.78%) | 15,981,440<br>(+7.97%) | 47,821,440<br>(+8.20%) | 79,661,440<br>(+8.25%) |
| <b>This Work<br/>(CTR)</b>     | 3,184<br>(+8.32%)   | 316,024<br>(+9.01%) | 1,580,024<br>(+9.01%) | 15,800,024<br>(+9.01%) | 47,400,024<br>(+9.01%) | 79,000,024<br>(+9.01%) |

Figure 7 shows Extract Function optimized in Generate Function. Optimized CTR mode is FACE-LIGHT or This work(CTR). For CTR\_DRBG with FACE-LIGHT, a LookUp table should be generated whenever Extract Function is executed. Therefore, if FACE-LIGHT is used in Extract Function, it is effective to extract random numbers of lengths that offset the cost of generating a Look-Up table. In the WSNs environment, especially using 8-bit AVR MCUs, the session is short, but it is generated repeatedly. Therefore, it is recommended to make the extract random number as long as possible when booting 8-bit AVR MCUs. However, it is almost impossible in a limited environment. In particular, FACE-LIGHT also needs memory space to store a 4 KB Look-Up table. Therefore, there is not enough memory space to store long random numbers in 8-bit AVR-MCUs. On the other hand, CTR\_DRBG, which uses our optimized implementation, can optimize Extract Function by additionally

storing only 268 bytes of constant data (MIX-table + LookUp table) to SRAM without generating Look-Up table which requires 4 KB. In addition, our implementation can expect improved performance regardless of the length of the extract random number.

## VI. PERFORMANCE ANALYSIS

### A. PERFORMANCE ANALYSIS OF PROPOSED AES IMPLEMENTATION

In an 8-bit AVR environment, we evaluate our implementation on an Arduino-Uno equipped with the ATmega 328p, an experimental instrument from FACE-LIGHT. The implementation environment is Atmel Studio 7, and the compilation option is -OS. Table 5 shows the clock cycles for each security level. For ECB mode, our column-wise software achieves higher performance at all security levels than Dinu *et al.* and Otte *et al.*'s implementation [8], [9].

**TABLE 7.** Running time comparison of Derivation Function implementations on CTR\_DRBG according to security level of AES. Input Data of Derivation Function is 32 bytes.

| Security                | AES-128            | AES-192            | AES-256            |
|-------------------------|--------------------|--------------------|--------------------|
| <b>Otte et al. [9]</b>  | 46,779<br>( - )    | 62,615<br>( - )    | 68,399<br>( - )    |
| <b>FACE-LIGHT. [11]</b> | 46,779<br>(+0%)    | 62,615<br>(+0%)    | 68,399<br>(+0%)    |
| <b>This Work.</b>       | 45,035<br>(+3.73%) | 59,195<br>(+5.45%) | 64,543<br>(+5.64%) |

In AES-CTR mode, our implementation is about 40 clock cycles performance difference at 128 and 192-bit security level compared to FACE-LIGHT and at the 256-bit security level, our implementation is about 50 clock cycles faster than FACE-LIGHT. We construct a scenario based on actual communication in a WSN environment to justify the implementation of the proposal. The session is generated and FACE-LIGHT and the proposed CTR mode software should generate a LookUp table. Table 6 is a performance comparison including the cost of generating a LookUp table by security level. First of all, our software outperforms Otte *et al.*'s implementation at all security levels and all encryption blocks. In the case of FACE-LIGHT, a large overhead is incurred at the beginning due to the cost of generating the LookUp table. In this case, at the 128-bit security level, our implementation outperforms FACE-LIGHT by up to 1,850 blocks. However, since 1,850 encryption variables cannot be stored realistically in ATmega328p, our implementation in a WSN environment can outperform FACE-LIGHT in all cases.

### B. PERFORMANCE ANALYSIS OF PROPOSED CTR\_DRBG

Table 7 is a performance table of Derivation Function, called in Instantiate Function, according to the security level of AES. In this paper, when measuring the ratio of performance improvement, Input Data of Derivation Function was fixed at 32 bytes (Which is reasonable because on AVR platforms, the noise data is typically collected from hardware noise sources, which contain large entropy than software noise sources). As mentioned in Section II-D, since the operation mode used in Derivation Function for CBC-MAC is CBC mode, It is impossible to apply FACE-LIGHT's LookUp table technique to Derivation Function. However, our implementation can be applied to Derivation Function. With the proposed optimization technique, our Derivation Function implementations outperform the Otte *et al.*'s implementations by 3.73%, 5.45%, and 5.64% for AES-128, AES-192, and AES-256, respectively.

Table 8 is a performance table of Update Function, called in Instantiate Function and Generate Function, according to the security level of AES. As mentioned in Section II-D, it is almost impossible to apply FACE-LIGHT's LookUp table technique to Update Function. However, our implementation can be applied to the Update Function. With the proposed

**TABLE 8.** Running time comparison of Update Function implementations on CTR\_DRBG according to security level of AES.

| Security                | AES-128           | AES-192            | AES-256            |
|-------------------------|-------------------|--------------------|--------------------|
| Len seed                | 2                 | 3                  | 3                  |
| <b>Otte et al. [9]</b>  | 8545<br>( - )     | 12,504<br>( - )    | 13,950<br>( - )    |
| <b>FACE-LIGHT. [11]</b> | 8545<br>(+0%)     | 12,504<br>(+0%)    | 13,950<br>(+0%)    |
| <b>This Work (CTR)</b>  | 8,057<br>(+5.71%) | 11,673<br>(+6.64%) | 13,035<br>(+6.56%) |

optimization technique, proposed optimized Update Function implementations outperform the Otte *et al.*'s implementations by 5.71%, 6.64%, and 6.56% for AES-128, AES-192, and AES-256, respectively. In 192 and 256-bit security using the same Len seeds in Update Function, the performance improvement of 256-bit implementation is slightly lower than that of 192-bit. The reason is that our Round 0 optimization implementation reduces slightly as rounds increase; however, this is a very minor issue, so it does not affect much on actual performance.

Table 9 is a performance table of CTR\_DRBG according to the extracted random number blocks. Our implementation of CTR\_DRBG, which uses the proposed AES-CTR (128-bit) optimization technique, shows up to 10.04% performance improvement when extracting one thousand blocks of the random number. When Extract Function, in Generate Function, is executed, the values of C (Key) and V (Counter) for Internal State differ from those for Internal State when the previous Extract Function was executed. Therefore, FACE-LIGHT should generate a Look-Up table when executing the initial CTR mode encryption of Extract Function. Since the cost of generating a Look-Up table is high, extracting small lengths of random numbers requires higher clock cycles than Otte *et al.*'s implementation. Our column-wise implementation is optimized without generating a Look-Up table. Thus, our implementation shows a high optimization performance. The ratio of performance improvement of CTR\_DRBG decreases slightly as the security bit increases; because, as like Update Function, the effect of Round 0 optimization decreases as the number of rounds increases.

When using AES-128 in CTR\_DRBG, FACE-LIGHT and our implementation show the same optimization performance when those generate a random number as much as 1300 blocks. 1300 blocks is a random number of 20,800 bytes. However, the optimization performance of FACE-LIGHT is demonstrated in a fixed-key scenario. Realistically, in an 8-bit AVR-MCUs used as a sensor node for WSNs, it is impossible to extract 20,800 bytes of random numbers at once. Note that 20,800 bytes = 20 KB. In WSN environment using 8-bit AVR-MCUs, it is advisable to extract a random number every time a session is created. Therefore, our optimized implementation not only shows performance improvement better than FACE-LIGHT in the

**TABLE 9.** Running time comparison of CTR\_DRBG implementations using 128, 192, 256-bit key (Timings are measured by clock cycles). *RB* means the extracted random number of blocks (size of one block is 16 bytes), Input Data of Derivation Function is 32 bytes.

| CTR_DRBG using AES-128 |                      |                      |                     |                       |                        |                        |
|------------------------|----------------------|----------------------|---------------------|-----------------------|------------------------|------------------------|
| <i>RB</i>              | 50                   | 100                  | 200                 | 400                   | 800                    | 1000                   |
| Otte et al. [9]        | 186,223<br>( - )     | 311,573<br>( - )     | 562,273<br>( - )    | 1,063,673<br>( - )    | 2,066,473<br>( - )     | 2,567,873<br>( - )     |
| FACE-LIGHT. [11]       | 233,213<br>(-25,23%) | 344,113<br>(-10,44%) | 565,913<br>(-0.65%) | 1,009,513<br>(+5.09%) | 1,896,713<br>(+8.21%)  | 2,340,313<br>(+8.86%)  |
| This Work<br>(CTR)     | 171,679<br>(+7.81%)  | 284,229<br>(+8.78%)  | 509,329<br>(+9.42%) | 959,529<br>(+9.79%)   | 1,859,929<br>(+10.00%) | 2,310,129<br>(+10.04%) |

| CTR_DRBG using AES-192 |                      |                     |                     |                       |                       |                       |
|------------------------|----------------------|---------------------|---------------------|-----------------------|-----------------------|-----------------------|
| <i>RB</i>              | 50                   | 100                 | 200                 | 400                   | 1000                  | 1500                  |
| Otte et al. [9]        | 226,259<br>( - )     | 375,809<br>( - )    | 674,909<br>( - )    | 1,273,109<br>( - )    | 3,067,709<br>( - )    | 4,563,209<br>( - )    |
| FACE-LIGHT. [11]       | 273,249<br>(-20,77%) | 408,349<br>(-8,66%) | 678,549<br>(-0.54%) | 1,218,949<br>(+4.25%) | 2,840,149<br>(+7.42%) | 4,191,149<br>(+8.15%) |
| This Work<br>(CTR)     | 208,589<br>(+7.81%)  | 343,889<br>(+8.49%) | 614,489<br>(+8.95%) | 1,155,689<br>(+9.22%) | 2,779,289<br>(+9.40%) | 4,132,289<br>(+9.44%) |

| CTR_DRBG using AES-256 |                      |                     |                     |                       |                       |                       |
|------------------------|----------------------|---------------------|---------------------|-----------------------|-----------------------|-----------------------|
| <i>RB</i>              | 50                   | 100                 | 200                 | 400                   | 1500                  | 3000                  |
| Otte et al. [9]        | 256,143<br>( - )     | 429,793<br>( - )    | 777,093<br>( - )    | 1,471,693<br>( - )    | 3,555,493<br>( - )    | 10,501,493<br>( - )   |
| FACE-LIGHT. [11]       | 299,665<br>(-16,99%) | 458,865<br>(-6,76%) | 777,265<br>(-0.02%) | 1,414,065<br>(+3.92%) | 3,324,465<br>(+6.50%) | 9,692,465<br>(+7.70%) |
| This Work<br>(CTR)     | 236,737<br>(+7.58%)  | 394,737<br>(+8.16%) | 710,737<br>(+8.54%) | 1,342,737<br>(+8.76%) | 3,238,737<br>(+8.91%) | 9,558,737<br>(+8.98%) |

WSN environment that has a short length session but also in CTR\_DRBG. In the same logic, in CTR\_DRBG using AES-192 and AES-256, our work is more effective than FACE-LIGHT.

## VII. CONCLUDING REMARKS

In this paper, we proposed a new column-wise implementation of AES in an 8-bit AVR environment. We solved the non-constant time issue of Otte et al and FACE-LIGHT proposed in the existing AVR environment, and efficiently arranged registers to compactly combine key operations of AES. In addition, unlike the existing implementation methodology, the optimization factor was maximized by utilizing the available SRAM. The proposed software in ECB mode achieves up to 10.2% performance improvement over the previous best implementation at all security levels. Moreover, our optimized technique can be used for various modes of operations, e.g., CBC, CTR, CFB, OFB, CCM, and GCM. We present an optimized CTR\_DRBG implementation using the proposed AES-CTR mode. In CTR\_DRBG implementation, which uses the proposed technique and AES-128, we achieved a performance improvement of up to 10.04% for CTR\_DRBG over the existing implementation. In addition, With the development of quantum computing systems in the

future, according to the Groover algorithm, the key length must be 256-bit, which is twice the current 128-bit, in order to ensure the safety of the symmetric key cryptosystem [22]. AES-256 proposed in this paper is the fastest among all AES implementations that exist in an 8-bit AVR environment. Based on this research, we will not only study our proposed column-wise fashion implementation in other low-power IoT devices but also conduct research on various Light-Weight Cryptography for edge networks.

## REFERENCES

- [1] L. Tawalbeh, S. Hashish, and H. Tawalbeh, "Quality of service requirements and challenges in generic WSN infrastructures," in *Proc. 8th Int. Conf. Ambient Syst., Netw. Technol.* (Procedia Computer Science), vol. 109, E. M. Shakshuki, Ed. Amsterdam, The Netherlands: Elsevier, 2017, pp. 1116–1121.
- [2] H. Kwon, S. An, Y. Kim, H. Kim, S. J. Choi, K. Jang, J. Park, H. Kim, S. C. Seo, and H. Seo, "Designing a CHAM block cipher on low-end microcontrollers for Internet of Things," *Electronics*, vol. 9, no. 9, p. 1548, Sep. 2020, doi: [10.3390/electronics9091548](https://doi.org/10.3390/electronics9091548).
- [3] H. Seo, I. Jeong, J. Lee, and W.-H. Kim, "Compact implementations of ARX-based block ciphers on IoT processors," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 3, pp. 1–16, Jun. 2018.
- [4] Y. Kim, H. Kwon, S. An, H. Seo, and S. C. Seo, "Efficient implementation of ARX-based block ciphers on 8-Bit AVR microcontrollers," *Mathematics*, vol. 8, no. 10, p. 1837, Oct. 2020, doi: [10.3390/math8101837](https://doi.org/10.3390/math8101837).

- [5] S. C. Seo and H. Kim, "SCA-resistant GCM implementation on 8-Bit AVR microcontrollers," *IEEE Access*, vol. 7, pp. 103961–103978, 2019.
- [6] Y. Kim, H. Choi, and S. C. Seo, "Efficient implementation of SHA-3 hash function on 8-bit AVR-based sensor nodes," in *Proc. 23rd Annu. Int. Conf. Inf. Secur. Cryptol.*, 2020, pp. 140–154.
- [7] H. Cheng, D. Dinu, and J. Großschädl, "Efficient implementation of the SHA-512 hash function for 8-bit AVR microcontrollers," in *Innovative Security Solutions for Information Technology and Communications*, J.-L. Lanet and C. Toma, Eds. Cham, Switzerland: Springer, 2019.
- [8] D. Dinu, A. Biryukov, J. Großschädl, D. Khovratovich, Y. L. Corre, and L. Perrin. (2015). *Felics-Fair Evaluation of Lightweight Cryptographic Systems*. [Online]. Available: <https://www.cryptolux.org/images/8/80/Session7-dinu-paper.pdf>
- [9] (2015). *Vr-Crypto-Lib*. [Online]. Available: <https://wiki.das-labor.org/w/Crypto-avr-lib/en>
- [10] J. Balasch, B. Ege, T. Eisenbarth, B. Gérard, Z. Gong, T. Güneysu, S. Heyse, S. Kerckhof, F. Koeune, T. Plos, T. Pöppelmann, F. Regazzoni, F. Standaert, G. V. Assche, R. V. Keer, L. van Oldeneel tot Oldenzeel, and I. von Maurich, "Compact implementation and performance evaluation of hash functions in attiny devices," in *Proc. 11th Int. Conf.*, vol. 7771, Graz, Austria: Springer, Nov. 2012, pp. 158–172.
- [11] K. Kim, S. Choi, H. Kwon, Z. Liu, and H. Seo, "FACE-LIGHT: Fast AES-CTR mode encryption for low-end microcontrollers," in *Proc. 22nd Int. Conf. (Lecture Notes in Computer Science)*, vol. 11975, J. H. Seo, Ed. Seoul, South Korea: Springer, 2019, pp. 102–114.
- [12] NIST. (2021). *Announcing the Advanced Encryption Standard (AES)*. [Online]. Available: <https://www.cisco.com/c/dam/en/us/products/collateral/security/anyconnect-secure-mobility-client/fips.pdf>
- [13] Atmel. (2012). *AVR Instruction Set Manual*. [Online]. Available: <http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf>
- [14] A. Adomnican and T. Peyrin, "Fixslicing AES-like ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V," *IACR Trans. Cryptograph. Hardware Embedded Syst.*, vol. 2021, no. 1, pp. 402–425, 2020, doi: 10.46586/tches.v2021.i1.402-425.
- [15] S. T. Meltem, B. Elaine, K. John, M. Kerry, B. Mary, and B. Michael, "Recommendation for the entropy sources used for random bit generation," in *Proc. CSRC*, 2018, pp. 4–28.
- [16] S.-E. Park, C.-G. Hwang, and D.-C. Park, "Internet of Things(IoT) ON system implementation with minimal arduino based appliances standby power using a smartphone alarm in the environment," *J. Korea Inst. Electron. Commun. Sci.*, vol. 10, no. 10, pp. 1175–1182, Oct. 2015.
- [17] Z. Liu, H. Seo, J. Grossschädl, and H. Kim, "Efficient implementation of NIST-compliant elliptic curve cryptography for 8-bit AVR-based sensor nodes," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 7, pp. 1385–1397, Jul. 2016.
- [18] KISA. (2011). *Research for Random Number Generator Using a Domestic Cipher Algorithm*. [Online]. Available: <https://kisa.or.kr/jsp/common/libraryDown.jsp?folder=018905>
- [19] J. H. Park and D. H. Lee, "FACE: Fast AES CTR mode encryption techniques based on the reuse of repetitive data," in *Proc. IACR*, Aug. 2018, pp. 469–499.
- [20] K. Kim. (2019). *FACE-LIGHT 8-bit Assembly Codes*. [Online]. Available: <https://github.com/solowal/DEVELOP/tree/master/Source%20Code>
- [21] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Advances in Cryptology (Lecture Notes in Computer Science)*, vol. 3621. Berlin, Germany: Springer-Verlag, 2005, pp. 258–275.
- [22] V. Mavroeidis, K. Vishi, M. D., and A. Jøsang, "The impact of quantum computing on present cryptography," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 3, pp. 1–10, 2018.

...