applied sciences

MDPI

Article

PAGE—Practical AES-GCM Encryption for Low-End Microcontrollers

Kyungho Kim [1], Seungju Choi [1], Hyeokdong Kwon [1], Hyunjun Kim [1] and Zhe Liu [2] and Hwajeong Seo [1,*]

[1] Division of IT Convergence Engineering, Hansung University, Seoul 02876, Korea; pgm.kkh@gmail.com (K.K); bookingstore3@gmail.com (S.C.); korlethean@gmail.com (H.K.); khj930704@gmail.com (H.K.)

[2] Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China; sduliuzhe@gmail.com

* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-2-760-8033

Received: 25 March 2020; Accepted: 28 April 2020; Published: 30 April 2020

Abstract: An optimized AES (Advanced Encryption Standard) implementation of Galois Counter Mode of operation (GCM) on low-end microcontrollers is presented in this paper. Two optimization methods are applied to proposed implementations. First, the AES counter (CTR) mode of operation is speed-optimized and ensures constant timing. The main idea is replacing expensive AES operations, including AddRound Key, SubBytes, ShiftRows, and MixColumns, into simple look-up table access. Unlike previous works, the look-up table does not require look-up table updates during the entire encryption life-cycle. Second, the core operation of Galois Counter Mode (GCM) is optimized further by using Karatsuba algorithm, compact register utilization, and pre-computed operands. With above optimization techniques, proposed AES-GCM on 8-bit AVR (Alf and Vegard's RISC processor) architecture from short-term, middle-term to long-term security levels achieved 415, 466, and 477 clock cycles per byte, respectively.

Keywords: AES; fast software encryption; Galois Counter Mode of operation; low-end microcontrollers; side channel attack countermeasure

1. Introduction

Resource constrained devices for Internet of Things (IoT) applications only equip limited RAM, ROM, computation capability and battery power. Under these hard conditions, the secure and robust network connection is a fundamental building block for IoT services. General cryptography solutions, such as encryption and authentication, for high-end desktop can be straightforwardly adopted to low-end microcontrollers. However, these approaches require heavy computation overheads since it is targeting for high-end processors. For this reason, the lightweight but secure cryptography solution is an important research area for practical and real world applications. The lightweight cryptography ensures low-cost and simple computations. Many lightweight block ciphers have been investigated. Block ciphers are largely based on ARX (Addition, Rotation, and XOR) architecture, such as SIMON, SPECK, CHAM, HIGHT, and LEA block ciphers [1–5]. However, ARX based block cipher requires many computations to prevent the side channel attack with a masking method [6]. On the other hand, AES block cipher (i.e., SPN architecture) is efficient for secure block cipher with the masking method [7]. The AES block cipher is not lightweight block cipher. However, this is used in almost all security solutions. For this reason, the optimized implementation of AES block cipher on the low-end microcontroller is important for security and generality.

Another concern is mode of operation. Among a number of mode of operations, the counter (CTR) mode of operation and the Galois Counter Mode (GCM) of operation are widely used for encryption

and authenticated encryption, respectively [8]. In CHES'18, the fast implementation of AES-CTR is introduced [9]. The method utilized the repeated data to pre-compute the part of AES operations. The FACE method is targeting for high-end processors, where they require high memory demands and frequent table updates.

For the authenticated encryption, AES-GCM is widely used in practice. The core operation of GCM is a binary field multiplication. In Reference [10], fast and secure binary field multiplication for GCM was introduced. They utilized Karatsuba algorithm and Instruction Level Atomicity (ILA). However, there is a large room to enhance the performance by using Karatsuba algorithm and engineering techniques.

In this paper, we optimized FACE technique for low-end microcontrollers (i.e., FACE–LIGHT). Afterward, the implementation of AES-GCM is efficiently constructed with the optimized GHASH function.

1.1. Extended Version of ICISC'19

In this paper, we extended our previous work published in ICISC'19 [11]. In ICISC'19, only AES-CTR implementations for 128-bit, 192-bit, and 256-bit security levels on low-end microcontrollers were investigated (i.e., FACE-LIGHT). This work further optimized the well-known mode of operation for authenticated encryption (i.e., GCM) on low-end microcontrollers. In particular, the binary field multiplication is optimized with Karatsuba algorithm and compact register assignments. The proposed implementation (i.e., PAGE) is also secure against Simple Power Analysis (SPA), Timing Attack (TA), and Correlation Power Analysis (CPA).

1.2. Research Contributions

- Compact Implementation of AES-CTR on Low-end Embedded Processors Previous AES-CTR implementation methods (i.e., FACE) mainly focused on high-end processors, such as Intel processors. Furthermore, the FACE implementation requires recalculation of the cache table after 256 encryption operations. The proposed method optimized the implementation of AES-CTR on low-end microcontrollers. The previous method is re-designed for the target low-end microcontroller by considering supported instruction sets and the number of general purpose registers. The proposed implementation requires the cache table update in the initialization step once. Proposed implementations achieved 138, 168, and 199 clock cycles per byte for 128-bit AES-CTR, 192-bit AES-CTR, and 256-bit AES-CTR on 8-bit AVR microcontrollers, respectively. Furthermore, the masked AES implementation is also accelerated with the proposed method for high performance.
- FACE Extension for Round 3 The original FACE method pre-computes the AES operation by ShiftRows of Round 2. In this paper, we further extended the FACE method by caching one Sub-Bytes and one Shift-Rows operations more (i.e., Round 3). Proposed implementations achieved 122, 153, and 183 clock cycles for 128-bit AES-CTR, 192-bit AES-CTR, and 256-bit AES-CTR, respectively.
- Optimized Implementation of GHASH Function for AES-GCM Encryption Binary field multiplication is a performance-critical building block for the GHASH function. The target 128-bit binary field multiplication is optimized with Karatsuba Block-Comb method and optimal register utilization. The countermeasure is also applied to the implementation, which ensures high security against SPA, TA, and CPA. The previous GHASH implementation requires 7172 clock cycles on 8-bit AVR microcontrollers, while the proposed method requires only 4230 clock cycles. With the optimized implementation of GHASH, AES-GCM for 128-bit, 192-bit, and 256-bit require 415, 466, and 477 clock cycles per byte, respectively.

The organization of the paper is as follows. Section 2 presents Fast implementation of AES–CTR Encryption (i.e., FACE), Galois Counter Mode (GCM) of operation, and secure implementation of binary field multiplication. Sections 3 and 4 introduce the fast implementation of AES-CTR (i.e., FACE–LIGHT)

and AES-GCM (i.e., PAGE) on microcontrollers, respectively. In Section 5, we review the performance of the suggested implementation. Section 6 concludes the paper.

2. Related Work

2.1. Galois Counter Mode (GCM) of Operation

GCM mode is the most commonly used encryption mode that protects message confidentiality, authentication, and integrity. GCM mode performs encryption using CTR mode and generates Message Authentication Code (MAC) using GHASH function using 128-bit binary multiplication. Security is ensured because the receiver can know the integrity of the message by comparing the MAC value before decrypting the ciphertext.

AES-GCM is a well known Authenticated Encryption with Associated Data (AEAD) encryption algorithm that sets a specific part of the message as associated data and transmits it without encryption. Even though the associated data is not encrypted, it is still often used in network packet encryption since it can be still used to verify the authenticity and integrity of the message. Currently, AES-GCM is adopted by NIST (National Institute of Standards and Technology) authentication encryption standard (SP 800-38D) and Transport Layer Security (TLS) as authentication encryption.

2.2. FACE: Fast AES-CTR Mode Encryption

The compact implementation of AES-CTR (i.e., FACE) for Intel processor was presented in CHES'18 [9]. The FACE method took advantage of the fact that small portion of Initial Vector (IV) value is only affected by the change of counter values. By utilizing this case, the FACE stores repeated values in a form of cache table and reused, which reduced the encryption timing. The FACE method largely consists of four steps.

The first step is called FACE_{rd0}, which makes a pre-computed table of results of Round 0's Add-RoundKey. In particular, it stores 12 bytes out of 16 bytes in the IV block. In the CTR, only 1 counter is different between the first and the second IV block. The Add-RoundKey operation can be optimized away by pre-computing 12 bytes out of 16 bytes in the IV block. The pre-computed table can be utilized by 2^{32-1} times of encryption operations.

The second step of FACE is FACE_{rd1} for Round 1. After the Mix-Columns operation of Round 1, only the first column of the state is updated. This is impact of the last byte, which is the only distinct byte from previous block of AES-CTR. The remaining columns maintain previous values since they are not affected by the last byte. This pre-computation can be used up-to 256 times, which may update other counter values.

The third step of FACE is FACE_{rd1+}. The only difference between first and second blocks is the first column value, which may be updated in the FACE_{rd1} step. The others are not changed and can be computed with the cached table. The first column is updated with 1-byte in the Mix-Columns operation.

The fourth step of FACE is FACE_{rd2}. The counter value updates the first column of Round 2. In Round 2, counter values are updated. These values update other columns in the Sub-Bytes operation. Afterward, all 16 bytes are updated in the Mix-Columns operation. However, many values can be reused during the Mix-Columns operation.

However, the original FACE method is efficient for 8-bit counter mode. After the 256-th block, the look-up table is changed. This frequent updates can be used by the attacker as an attack point. In order to avoid this incident, the FACE method should be implemented in a regular pattern.

2.3. Binary Field Multiplication

The binary field multiplication involves multiplying two polynomials and modular reduction. Various methods have been studied to enhance the performance of binary field multiplication.

First method is look-up table based polynomial multiplication by Lopez and Dahab [12]. The method involves of making the look-up table and accessing the table. It first construct the look-up

table with all of the possible result values of multiplication of one operand. After the construction, the binary field multiplication is carried out while referencing the table with operand offsets.

Another approach is the Block-Comb method [13] which carries out the polynomial multiplication with bit-wise XOR operation over multiple blocks. Unless the operand is set to 1, no other operation is performed. When the operand is set to 1, the operand is then bit-wise XORed with intermediate results. By utilizing this method, the number of redundant memory accesses can be reduced. Seo et al. proposed the Karatsuba Block-Comb technique by merging the Karatsuba with Block-Comb [14].

This leads to reduction of the number of partial products further at the cost of several inexpensive field additions when computing a polynomial multiplication. However, these Block-Comb methods are vulnerable to timing attacks since the method uses `if-else` statements which does not ensure constant computation patterns.

Therefore, the research has been conducted in order to have a countermeasure against side channel attacks using masking or dummy XOR operations. Liu et al. proposed a method that always works in a constant-time with the same pattern regardless of the multiplier using the method called Masked Block-Comb [15]. However, the power consumption pattern can be leaked if the MASK value is set to 0 which leads to using zero register. In addition, Liu et al. did not take consideration of the CPA security during the GHASH function.

To solve the limitation of Liu et al's method, Seo and Kim proposed an implementation using Dummy XOR with garbage register [10]. A real accumulator register for the result of multiplication operation and a garbage register for the meaningless result were utilized. This ensures that each real and fake power trace remains identical which makes it resistant against SPA and TA.

3. FACE-LIGHT: Fast AES-CTR Mode Encryption for Low-End Microcontrollers

In this section, we present an advanced implementation method for AES-CTR mode on low-end embedded processors. Compared to previous work (i.e., FACE), the proposed technique focused on the low-end microcontroller together with the masking operation. Nonce and counter values construct the packet of the CTR mode. For the universal setting, 96-bit nonce values and 32-bit counter values are assigned. Throughout the whole process, the nonce remains same while the counter value gets transformed in each processing. The previous work is based on the 8-bit unit, which requires the table update in every 256 counts during encryption procedures. The proposed implementation is targeting for the 32-bit counter, which only needs to set the pre-calculated table in the initialization stage, where the cache table is not updated throughout the whole process. The explanation are presented in Figure 1. The top and bottom of the Figure 1 indicate first block and n-th block, respectively. Each square holds 8-bit data. White and colored blocks portray identical and different parts, respectively.

3.1. Round 0

During Round 0, Add-RoundKey operation is performed. In the operation, plaintext and round key are XORed. Differences between adjacent blocks are only 4 bytes.

3.2. Round 1

In Round 1, Sub-Bytes, Shift-Rows, Mix-Columns, and Add-RoundKey operations are performed in order. The Sub-Bytes operation merely changes the 8-bit input value into the 8-bit output value. This operation does not affect other blocks. The Shift-Rows literally shifts rows of the block by certain offsets each. In the Mix-Columns, the 8-bit data gets mixed in 32-bit column wise. Lastly, the Add-RoundKey adds round keys to each square of the block. Throughout the first and n-th blocks, there are no identical values. However, both results of the first block and the n-th blocks are originated from the same IV.

In particular, the 32-bit of counter values from the IV (i.e., $X[0]$, $X[1]$, $X[2]$, and $X[3]$) are different between blocks. In the Mix-Columns, from the bottom of the Figure 1, the $X[0]$ affects the first column (i.e., $X[5]$, $X[10]$, and $X[15]$). Other columns get affected by the change of each square (i.e., $X[1]$, $X[2]$, and $X[3]$). Likewise, the value of each column (i.e., 32-bit) depends on the 8-bit square (i.e., $X[0]$, $X[1]$, $X[2]$, and $X[3]$). Each 32-bit column has 256 conditions. The following Add-RoundKey operation does

not mix up values of each column but rather only has affects on each square, independently. There is no impact on the complexity for pre-computing the result value of the column.

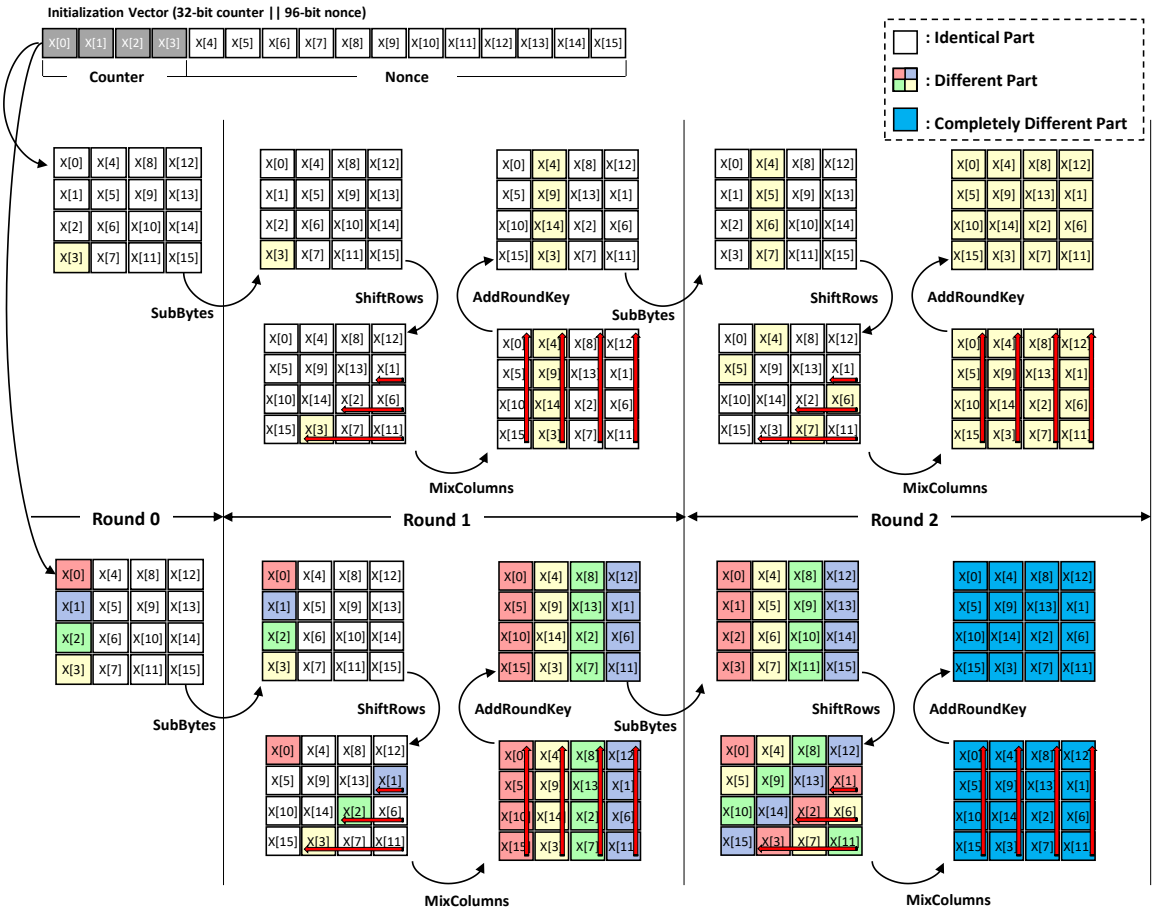

Figure 1. Overview of FACE–LIGHT.

3.3. Round 2

In the previous stage, each value of column gets decided based on the 8-bit value (i.e., $X[0]$, $X[1]$, $X[2]$, and $X[3]$). Since this does not perform any mixing operations across columns, the condition is still valid in the Sub-Bytes operation. The Sub-Byte operation only replaces the 8-bit input value to the 8-bit output value. For the pre-computed table, four look-up tables are required. Input values are $X[0]$, $X[1]$, $X[2]$, and $X[3]$ in 8-bit wise and the length of output value is set to 32-bit wise. The whole size of the look-up table is 4 KB in total. Intermediate results of specific 8-bit input values are assigned to skip the process of 2 Round.

3.4. Extended Round for FACE

Proposed method accelerates implementations of AES-CTR by ShiftRows of Round 2. This approach can be applied even for the original FACE method by ShiftRows of Round 3. Operations from Round 0 to SubBytes of Round 1 are computed with FACE method. Afterward, FACE–LIGHT method is used to cache computations from Round 2 to ShiftRows of Round 3.

3.5. Optimized Implementation of FACE–LIGHT

The pre-computed table is generated off-line and stored in memory before the encryption operation. Since 8-bit AVR embedded processor has small `SRAM` size, `PROGRAM`, which is larger than `SRAM`, is utilized. Whenever the look-up table gets accessed, pre-computed 32-bit results get obtained.

The 8-bit input offset is re-scaled to the 32-bit input offset by quadrupling the 8-bit offset. The input offset gets added to the base address of each look-up table.

During the substitute operation, 256 bytes of pre-computed result are reserved as a form of look-up table. When the value in the look-up table is loaded with the Z pointer (R30 and R31), the lower part of memory address and the upper part of memory address of the look-up table are loaded in registers (R30 and R31). Afterward, the offset is added to the pointer. This may generate carry bits. In order to avoid the carry propagation, memory address is 8-bit aligned. With the aligned memory address, the lower address is always set to 0. By assigning the 8-bit offset to the lower address, the offset calculation is simply completed. In order to load the round key, the LD instruction is used. In the target microcontroller, the post incremental address mode is used with X, Y, and Z pointers. Using the address with indexed mode, the LDD instruction is used.

3.6. Optimized Implementation of Masking Operation

The masking method prevents the potential attacks with additional computations to reduce information leakages. However, the countermeasure needs longer execution timing than basic implementations. For the high performance, two implementation techniques are applied.

Before performing the encryption operation, round keys are pre-computed. This is XOR with the value M0 from Round 0 to Round 9. By performing XOR operation in advance, the last round does not include M6, M7, M8, and M9 values. However, the M1 value is applied to all keys. The aforementioned operation is performed in advance to optimize the repetition of the XOR operation and loading the masking value in each round. M0 is XORed with M6, M7, M8, and M9 from line 4. In label 1, M6, M7, M8, and M9 get XORed in row by row for a total of 10 key sets from Round 0 to 9. By doing the XOR operation with M0 to each mask value beforehand, 160 times of bit-wise exclusive-or operations are reduced. M1 XOR operations of the final round key are given in label2.

After the processing Sub-Bytes operation, M0 is replaced with M1. For optimized implementations, Masked-SBOX table is pre-computed and referenced during the substitute operation.

4. PAGE: Practical AES-GCM Encryption for Microcontrollers

This section describes distinguished features of suggested method by comparing previous works by Seo and Kim [10]. First, the previous work utilized eight garbage registers to calculate the GHASH function. However, the proposed method only utilizes one garbage register without loss of security against CPA, SPA, and TA. In addition, multiplication computations are optimized to save the register. In total, eight more registers are utilized for Karatsuba algorithm than the previous method. Furthermore, the register optimized version is also investigated Second, the repeated value (H_{GHASH}) of GHASH function allows to pre-compute the part of Karatsuba operand. The pre-computed value is generated once and called several times, which reduces the execution timing. Third, the fast reduction by Shay and Kounavis performs faster reduction than the implementation by Seo and Kim [10,16]. The method is optimized for the low-end microcontroller.

4.1. Optimized Implementation of PAGE

The Block Comb method by Seo and Kim uses 8 garbage registers for Dummy XOR operation to prevent SPA, TA, and CPA [10]. Values stored in eight garbage registers are not used during the multiplication. Garbage registers only load meaningless data in order not to be distinguished from the real operation. The attacker cannot distinguish the real operation from the original power trace, which prevents leakage of secret value.

The same effect can be achieved by using only one of eight garbage registers. In Figure 2, the power trace using eight garbage registers in 32-bit multiplication is shown. Two power traces have regular pattern. In the 11-th line of Algorithm 1, the result of XOR operation using eight garbage registers (R0∼R7) can be obtained using only one register (R24).

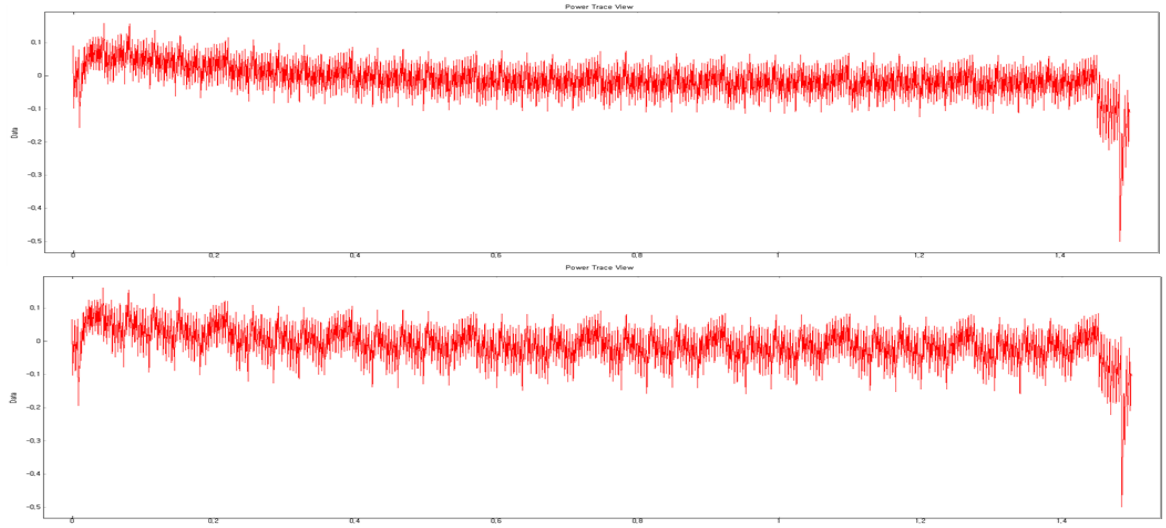

Figure 2. Comparison of 32-bit multiplication in terms of power trace, (**Top**) Seo and Kim [10], (**Bottom**) this work, Figure is generated by us in this paper.

Algorithm 1 Proposed method for 32-bit multiplication.

Input: : 32-bit multiplicand A ($R_{19}, \ldots, R_{16}$), 32-bit multiplier B ($R_{23}, \ldots, R_{20}$), Garbage result (R_{24}), and real result ($R_{15}, \ldots, R_8$).
Output: : Result C (64-bit) = $A \times B$.

...
$R_{25} \leftarrow 0x06$
for l = 7 to 0 **do**
  for m = 3 to 0 **do**
    if the l-th bit of R_{16+m}==1 **then**
      $R_0 \leftarrow R_0 + R_{25}$
      for k = 0 to 3 **do**
        $R_{8+m+k} \leftarrow R_{8+m+k} \bigoplus R_{20+k}$
      end for
    else
      for k = 0 to 3 **do**
        $R_{24} \leftarrow R_{24} \bigoplus R_{20+k}$
      end for
    end if
  end for
  $(R_{15}, \ldots, R_8) \leftarrow (R_{15}, \ldots, R_8) \ll 1$
end for
...

In addition, the performance is improved further by applying Liu et al's multiplication method to the proposed method. Seo and Kim proposed to shift 40-bit multiplicand (A) for a 64-bit multiplier, rather than shifting 64-bit accumulator. This approach reduces 29 shift instructions per 32-bit multiplication operation. However, the multiplication method suggested by Seo and Kim does not show a big difference in performance compared to the method suggested by Liu et al. According to Seo and Kim, the number of shift instructions can be reduced compare to Liu et al's method. However, the XOR instruction goes one more operation per bit, which leads to addition of 32 more XOR instructions when calculation 40-bit multiplicand (A). Since five registers are needed to store the 40-bit multiplicand (A), one more register is required compared to the Liu et al's technique. Liu et al.'s approach is used

for multiplication and one register is saved. These spare registers are used in the Karatsuba algorithm to improve the performance. For the optimal number of register utilization, the version without using spare registers is also investigated. Currently, RISC-V introduces new architecture for future microcontrollers. The optimal register utilization can contribute to the optimal architecture design.

4.2. Karatsuba Algorithm for GHASH

Karatsuba algorithm is well known asymptotically fast multiplication method and the proposed implementation also utilizes the Karatsuba algorithm for high performance.

First, the multiplication is performed with lower 32-bit of 64-bit operands ($A[3 \sim 0], B[3 \sim 0]$) and 64-bit result (L) is obtained. Second, the multiplication is performed with lower 32-bit of 64-bit operands ($A[7 \sim 4], B[7 \sim 4]$) and 64-bit result (H) is obtained. Third, XOR operation is performed between the upper part and the lower part of the A and B, respectively. Values are multiplied and output the final result (M). The L, H, and M are XORed together and stored in M, again. Lastly, shifted H ($H \ll 64$), shifted M ($M \ll 32$) and L are XORed and the final 128-bit result (C) is obtained.

Karatsuba algorithm needs a buffer to store intermediate results of each operation. In the Block Comb method proposed by Seo and Kim, the size of intermediate results is over than given size of general purpose registers. For this reason, the part of intermediate result should be stored in `STACK`. However, accessing to `STACK` memory (i.e., 2 clock cycles) requires high-overheads than register accesses (i.e., 1 clock cycle). The proposed method uses eight more registers to maintain the intermediate result. For the register optimized version, the intermediate result is stored in `STACK` memory.

In Algorithm 2, the proposed implementation does not reserve the intermediate result of the Karatsuba calculation, such as L, H, and M, in the `STACK`. Instead, only values needed for the following operation are stored using the $K0$ to $K7$ registers.

Algorithm 2 Proposed implementation of 1-level Karatsuba Block-Comb in source code level.

```
1: ROUND32
2: STD Z+0, C0
3: STD Z+1, C1
4: STD Z+2, C2
5: STD Z+3, C3
6: EOR C0, C4
7: EOR C1, C5
8: EOR C2, C6
9: EOR C3, C7
10: MOVW K0, C0
11: MOVW K2, C2
12: MOVW K4, C4
13: MOVW K6, C6
14: ROUND32
15: STD Z+12, C4
16: STD Z+13, C5
17: STD Z+14, C6
18: STD Z+15, C7
19: EOR K0, C0
20: EOR K1, C1
21: EOR K2, C2
22: EOR K3, C3
23: EOR C0, C4
24: EOR C1, C5
25: EOR C2, C6
26: EOR C3, C7
27: EOR K4, C0
28: EOR K5, C1
29: EOR K6, C2
30: EOR K7, C3
31: ROUND32
32: EOR C0, K0
33: EOR C1, K1
34: EOR C2, K2
35: EOR C3, K3
36: EOR C4, K4
37: EOR C5, K5
38: EOR C6, K6
39: EOR C7, K7
40: STD Z+4, C0
41: STD Z+5, C1
42: STD Z+6, C2
43: STD Z+7, C3
44: STD Z+8, C4
45: STD Z+9, C5
46: STD Z+10, C6
47: STD Z+11, C7
```

When the 128-bit result (C) and the 64-bit intermediate result (M) are divided into 8-bit units, it can be expressed as $(C[0], C[1], ..., C[15])$ and $(M[0], M[1], ..., M[8])$, respectively. Among them, $C[0 \sim 3]$ and $C[12 \sim 15]$ store the upper 32-bit multiplication result and the lower 32-bit multiplication result.

The result from $C[4]$ to $C[11]$ is XORed with the M. The final result of $C[4]$ is calculated by $C[4] \bigoplus M[0] \bigoplus C[0] \bigoplus C[8]$. Values affected by XOR, such as $C[0] \sim C[3]$ and $C[12] \sim C[15]$, are stored after each 32-bit multiplication operation. In addition, values used to make $C[4] \sim C[11]$ can be XORed and stored in advance. Afterward, 64-bit multiplication is performed without loading and storing procedures from the `STACK`.

In the case of the GHASH function, the associated data and cipher text are XORed and multiplied by the hash key value (H_{GHASH}). In this process, the B value, which is a multiplier of all 128-bit multiplication operations, is fixed to the H_{GHASH} value. This repeated value (i.e., $H_{GHASH}[7 \sim 0] \bigoplus H_{GHASH}[3 \sim 0]$) can be pre-computed and reserved to reduce the number of XOR operations and memory accesses.

4.3. Fast Reduction for GHASH

In order to optimize the modular operation, the fast reduction modulo algorithm by Gueron and Kounavis [16] was applied to the proposed implementation. The algorithm optimizes the reduction operation with an irreducible polynomial ($x^{128} + x^7 + x^2 + x + 1$) in a 64-bit environment. The algorithm is optimized for 8-bit AVR platforms. In Algorithm 3, input values ($K[0] \sim K[31]$) were reflected in advance. This is the reason why all of operations are reversed to the existing original algorithm. The algorithm requires a number of bit-wise shift operation. The 8-bit wise shift operation is optimized away.

Algorithm 3 Proposed method for fast modular reduction.

Input: : 256-bit data $K[0], ..., K[31]$, where $K[0 \sim 31]$ are 8-bit long each.

Output: : $K[16], ..., K[31]$ (reduced result).

1: $A \leftarrow (K[31]\ \&\ 0b1) \ll 6$
2: $B \leftarrow (K[31]\ \&\ 0b10) \ll 5$
3: $C \leftarrow (K[31]\ \&\ 0b1111111)$
4: $K[16] \leftarrow K[16] \oplus ((A \oplus B \oplus C) \ll 1)$
5: $K[8] \leftarrow K[8] \oplus K[24] \oplus ((K[23] \ll 7) \mid (K[24] \gg 1)) \oplus ((K[23] \ll 6) \mid ((K[24] \gg 2)) \oplus ((K[23] \ll 1) \mid (K[24] \gg 7))$
6: $K[0] \leftarrow K[0] \oplus K[16] \oplus (K[16] \gg 2) \oplus (K[16] \gg 7)$
7: **for** l = 1 to 7 **do**
8: $K[i+8] \leftarrow K[i+8] \oplus K[i+24] \oplus ((K[i+23] \ll 7) \mid (K[i+24] \gg 1)) \oplus ((K[i+23] \ll 6) \mid (K[i+24] \gg 2)) \oplus ((K[i+23] \ll 1) \mid (K[i+24] \gg 7))$
9: $K[i] \leftarrow K[i] \oplus K[i+16] \oplus ((K[i+15] \ll 7) \mid (K[i+16] \gg 1)) \oplus ((K[i+15] \ll 6) \mid (K[i+16] \gg 2)) \oplus ((K[i+15] \ll 1) \mid (K[i+16] \gg 7))$
10: **end for**

5. Evaluation

Low-end 8-bit AVR microcontollers were used to run the proposed method. `Arduino UNO` board equipped with the `ATmega328` microcontroller working at 16MHz is used. The size of `FLASH`, `EEPROM`, and internal `SRAM` is 32KB, 1KB, and 2KB, respectively. 32 8-bit general purpose registers and 131 instructions are supported. The code is complied in `-OS` option and the executiong timing is measured in clock cycles. The software is implemented over `Arduino IDE` and `Atmel Studio 7`. Majority of implementation is written in assembly language. To evaluate the side channel attack countermeasure, the power usage during encryption operation is measured by Chipwhisperer-Lite (`CW1173`).

5.1. Performance Evaluation of FACE-LIGHT

Table 1 presents the comparison of execution timing for various AES implementations. Previous AES implementations support the ECB (Electronic CodeBook) mode of operation, while the proposed implementation is targeting for the CTR mode of operation. Both implementations perform same AES encryption as a core operation. Dinu et al. [17] proposed the size optimized implementation, while Otte et al. [18] proposed the speed optimized implementation. FACE–LIGHT and Extended FACE for AES-128 are faster than previous works by 18 and 34 clock cycles per byte, respectively. Implementations of FACE–LIGHT are 11.5%, 9.6%, and 8.3% faster than Otte et al. for 128-bit, 192-bit, and 256-bit security levels, respectively. Implementations of Extended FACE are 21.5%, 18.1%, and 15.6% faster than Otte et al. for 128-bit, 192-bit, and 256-bit security levels, respectively.

Table 1. Comparison of AES on 8-bit AVR (Alf and Vegard's RISC processor)microcontrollers, in terms of clock cycles per byte.

Security Level	Dinu et al. [17]	Otte et al. [18]	FACE–LIGHT (This work)	Extended FACE (This work)
128-bit	177	156	138	122
192-bit	N/A	186	168	153
256-bit	N/A	217	199	183

In Table 2, the comparison of functionality is given. FACE–LIGHT does not update the table throughout the execution. The execution timing of encryption keeps constant timing. Second, target applications are different. FACE–LIGHT targets for the 8-bit microcontrollers or above, while FACE is targeting for 32-bit processors or above. The FACE method only supports by Round 2, while FACE–LIGHT is expandable up-to the Round 3.

Lightweight ciphers can be implemented in masked way to deal with side channel attacks. Most lightweight ciphers are based on the ARX structure, which have significant overheads for Arithmetic-to-Boolean operation during masking operations. On the contrary, AES based on SPN structures has a relatively short operation time with the masked implementation. This is an advantage over other lightweight ciphers when it comes to the masking operation.

Table 2. Comparison of functionality between FACE and FACE–LIGHT.

	FACE [9]	FACE-LIGHT (This Work)
Table update	√	–
Constant timing	–	√
Target application	High-end processor	Low-end microcontroller or high-end processor
Supported round	Round 2	Round 3

Table 3 presents the execution timing of unmasked LEA, masked LEA, and masked AES implementations. For the LEA–128 with masking technique, the execution timing increases when the masking method is adopted, while masked AES ensures relatively short execution timing.

Table 3. Evaluation of LEA (Lightweight Encryption Algorithm) and AES implementations on 8-bit AVR microcontrollers, in terms of clock cycles per byte.

Basic LEA-128 [19]	Masked LEA-128 [20]	Masked AES-128 (This Work)
168	2,286	388

Figure 3 presents a graph of key values and coefficient of correlation estimated via CPA on the non-masked AES and masked AES implementation. In the AES implementation without masking technique, the coefficient of correlation of all of the key values is notably higher than others. The masked AES implementation shows all key values have equal coefficients of correlation.

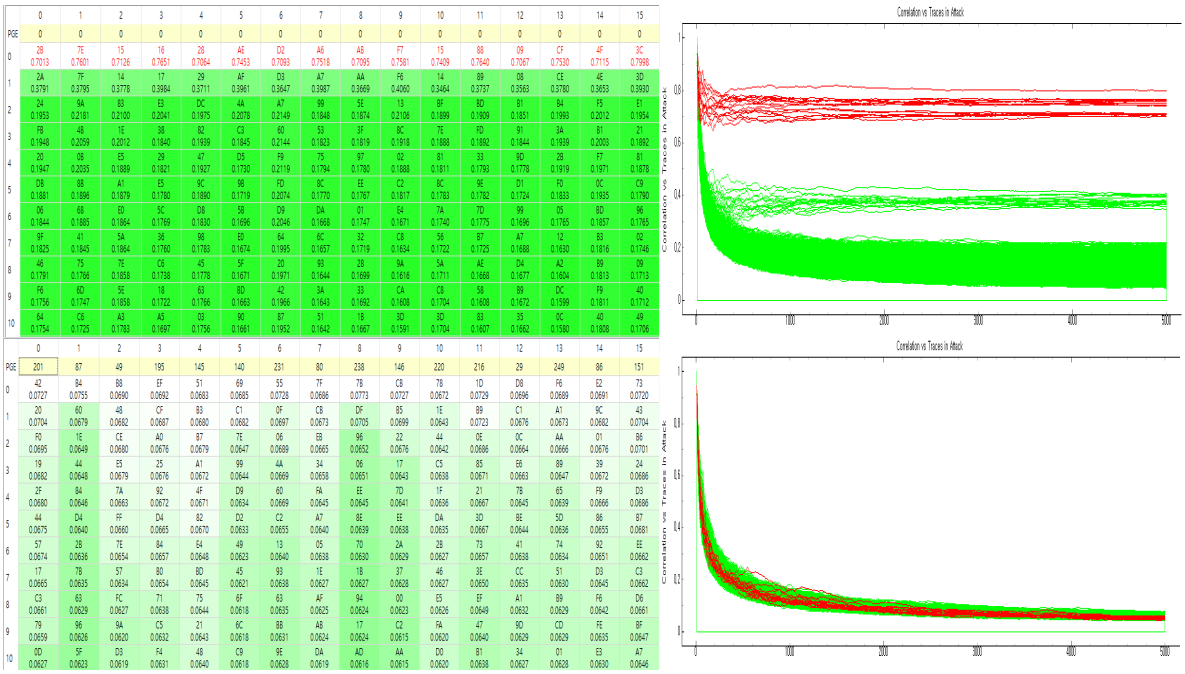

Figure 3. CPA (Correlation Power Analysis) attack result comparison between non-masked AES and masked FACE-LIGHT implementations, (**Top**) analyzed key value table of non-masked AES through CPA attack and graphical result of correlation of key values via CPA attack on non-masking AES, (**Bottom**) analyzed key value table of masked AES through CPA attack and graphical result of the correlation of key values via CPA attack on masked AES.

5.2. Performance Evaluation of PAGE

In Table 4, the performance evaluation of AES-GCM operation is given. In case of AES-GCM-128, the clock cycles per byte of 16-byte, 64-byte and 1024-byte is 807, 510, and 415, respectively. The rest of the AES-GCM-192 and AES-GCM-256 follows the same pattern. The long message encryption is more efficient than short message encryption, because all AES-GCM computations need one encryption on the counter value and one pre-calculation of multiplication value during the initial stage. The register optimized version is also evaluated. The performance is slower than normal version by 2.8%∼3.5%. The result shows that register optimized version is also competitive.

Table 4. Performance of AES–GCM evaluated by byte length for 16-byte, 64-byte, and 1024-byte messages (clock cycles per byte). [1]: register optimized version.

Security Level	16-Byte	64-Byte	1024-Byte
128-bit	807	510	415
192-bit	868	548	446
256-bit	928	586	477
128-bit [1]	843	529	430
192-bit [1]	903	567	461
256-bit [1]	964	604	491

In Table 5, the performance comparison of polynomial multiplication is given. For 128-bit multiplication, the previous work requires 5675 clock cycles, while the proposed implementation requires 3896 clock cycles, which is 31.3% reduction than the previous work.

Table 5. Comparison of execution timing (in terms of clock cycles) for 128-bit wise polynomial multiplication. [1]: register optimized version.

Seo and Kim [10]	This Work	This Work [1]
5,675	3,896	4,175

Table 6 compares clock cycles for 128-bit binary field multiplication. Proposed method requires 4,230 clock cycles. This high performance was achieved by optimizing polynomial multiplication and reduction operations. This is 40.9% faster than work by Seo and Kim.

Table 6. Comparison of binary field multiplication in terms of execution timing (in terms of clock cycles) and countermeasures. [1]: register optimized version.

Contributor	Method	TA/SPA Security	Timing (cc)
Liu et al. [15]	Karatsuba + Masked Block Comb	TA only	14,878
Seo and Kim [10]	Karatsuba + Block-Comb with Dummy XOR and ILA	TA/SPA both	7,162
This Work	Karatsuba + Block Comb with Dummy XOR and ILA	TA/SPA both	4,230
This Work [1]	Karatsuba + Block Comb with Dummy XOR and ILA	TA/SPA both	4,497

Figure 4 shows the result of CPA on the proposed GHASH function. Gray waveform shows the correlations of wrong keys and the black waveform shows the correlation of a correct key. The correlation of the correct key is indistinguishable from the wrong keys. The proposed method successfully prevents the CPA.

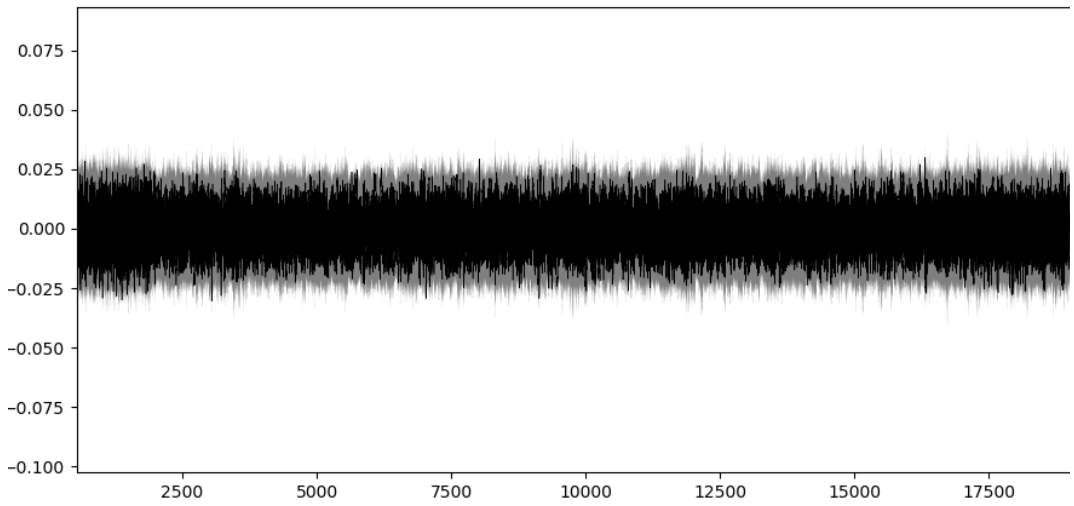

Figure 4. CPA on the proposed implementation of GHASH function.

6. Conclusions

In this paper, we proposed optimized implementations of AES–CTR and AES–GCM on low-end 8-bit AVR microcontrollers. The implementation of AES–CTR is accelerated with the re-designed look-up table. This novel approach skips AES–CTR computations by ShiftRows of Round 2. The proposed method is also applied to the previous work to improve the performance, further.

The implementation of AES–GCM is accelerated with the efficient binary field multiplication. The required number of general purpose registers is optimized and certain part of Karatsuba algorithm is cached. This approach is also secure against SPA, TA, and CPA.

With above efficient implementations, low-end microcontrollers can provide secure and fast encryption operations. Next research topic is efficient implementation of other lightweight block ciphers with other mode of operations on low-end microcontrollers.

Author Contributions: K.K. and S.C. designed the entire architecture. H.K. (Hyunjun Kim), H.K. (Hyeokdong Kwon), and Z.L. performed experiments. H.S. supervised the whole process as a corresponding author. All authors discussed contents of the manuscript and wrote the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported as part of Military Crypto Research Center (UD170109ED) funded by Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD) and this work was partly supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00264, Research on Blockchain Security Technology for IoT Services). This research was financially supported by Hansung University for Hwajeong Seo.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hong, D.; Lee, J.; Kim, D.; Kwon, D.; Ryu, K.H.; Lee, D. LEA: A 128-bit block cipher for fast encryption on common processors. In Proceedings of the International Workshop on Information Security Applications, Jeju Island, Korea, 19–21 August 2013; pp. 3–27.
2. Hong, D.; Sung, J.; Hong, S.; Lim, J.; Lee, S.; Koo, B.; Lee, C.; Chang, D.; Lee, J.; Jeong, K. HIGHT: A new block cipher suitable for low-resource device. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Yokohama, Japan, 10–13 October 2006; pp. 46–59.
3. Beaulieu, R.; Treatman-Clark, S.; Shors, D.; Weeks, B.; Smith, J.; Wingers, L. The SIMON and SPECK lightweight block ciphers. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 8–12 June 2015; pp. 1–6.
4. Koo, B.; Roh, D.; Kim, H.; Jung, Y.; Lee, D.; Kwon, D. CHAM: a family of lightweight block ciphers for resource-constrained devices. In Proceedings of the International Conference on Information Security and Cryptology, Seoul, Korea, 29 November–1 December 2017; pp. 3–25.
5. Roh, D.; Koo, B.; Jung, Y.; Jeong, I.W.; Lee, D.G.; Kwon, D.; Kim, W.H. Revised Version of Block Cipher CHAM. In Proceedings of the International Conference on Information Security and Cryptology, Seoul, Korea, 4–6 December 2019; pp. 1–19.
6. Goubin, L. A sound method for switching between boolean and arithmetic masking. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Paris, France, 14–16 May 2001; pp. 3–15.
7. Standard, N.F. Announcing the advanced encryption standard (AES). *Fed. Inf. Process. Stand. Publ.* **2001**, *197*, 3-3.
8. McGrew, D.; Viega, J. The Galois/counter mode of operation (GCM). *NIST Mod. Oper. Process* **2004**, *20*, in submission.
9. Park, J.H.; Lee, D.H. FACE: Fast AES CTR mode Encryption Techniques based on the Reuse of Repetitive Data. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, pp. 469–499.
10. Seo, S.C.; Kim, H. SCA-resistant GCM implementation on 8-Bit AVR microcontrollers. *IEEE Access* **2019**, *7*, 103961–103978. [CrossRef]
11. Kim, K.; Choi, S.; Kwon, H.; Liu, Z.; Seo, H. FACE–LIGHT: Fast AES–CTR Mode Encryption for Low-End Microcontrollers. In Proceedings of the International Conference on Information Security and Cryptology, Seoul, Korea, 4–6 December 2019; pp. 102–114.
12. López, J.; Dahab, R. High-speed software multiplication in f_{2^m}. In Proceedings of the International Conference on Cryptology, Calcutta, India, 10–13 December 2000; pp. 203–212.
13. Shirase, M.; Miyazaki, Y.; Takagi, T.; Han, D.G.; Choi, D. Efficient implementation of pairing-based cryptography on a sensor node. *IEICE Trans. Inf. Syst.* **2009**, *92*, 909–917. [CrossRef]
14. Seo, H.; Liu, Z.; Choi, J.; Kim, H. Karatsuba–Block-Comb technique for elliptic curve cryptography over binary fields. *Secur. Commun. Netw.* **2015**, *8*, 3121–3130. [CrossRef]
15. Liu, Z.; Seo, H.; Chen, C.N.; Nogami, Y.; Park, T.; Choi, J.; Kim, H. Secure GCM implementation on AVR. *Discrete Appl. Math.* **2018**, *241*, 58–66. [CrossRef]
16. Gueron, S.; Kounavis, M. Efficient implementation of the Galois Counter Mode using a carry-less multiplier and a fast reduction algorithm. *Inf. Process. Lett.* **2010**, *110*, 549–553. [CrossRef]

17. Dinu, D.; Biryukov, A.; Großschädl, J.; Khovratovich, D.; Le Corre, Y.; Perrin, L. FELICS–fair evaluation of lightweight cryptographic systems. In Proceedings of the NIST Workshop on Lightweight Cryptography, Gaithersburg, MD, USA, 4–6 November 2015; Volume 128.
18. Otte, D. AVR-crypto-lib. Available online: https://wiki.das-labor.org/w/AVR-Crypto-Lib/en (accessed on 29 April 2020).
19. Seo, H.; Jeong, I.; Lee, J.; Kim, W. Compact implementations of ARX-based block ciphers on IoT processors. *ACM Trans. Embed. Comput. Syst. (TECS)* **2018**, *17*, 60. [CrossRef]
20. Park, E.; Oh, S.; Ha, J. Masking-Based Block Cipher LEA Resistant to Side Channel Attacks. *J. Korea Inst. Inf. Secur. Cryptol.* **2017**, *27*, 1023–1032.