FACE–LIGHT: Fast AES–CTR Mode Encryption for Low-end Microcontrollers

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

[1]Division of IT Convergence engineering, Hansung University, Seoul, South Korea,
[2]Aeronautics and Astronautics, Nanjing University, Nanjing, Jiangsu, China,
{pgm.kkh, bookingstore3, korlethean, sduliuzhe, hwajeong84}@gmail.com

Abstract. In this paper, we revisited the previous Fast AES–CTR mode Encryption (FACE) method for high-end processors and tailored the method to the microcontrollers, namely FACE–LIGHT. We targeted the 32-bit counter mode of operation for AES in constant timing. This optimized technique pre-computes the 2 Add-RoundKey, 2 Sub-Bytes, 2 Shift-Rows and 1 Mix-Columns operations. The FACE–LIGHT is implemented on the representative low-end microcontrollers (e.g. 8-bit AVR). The execution timing of AES–CTR algorithm for 128-bit and 256-bit security levels achieved the 2,218 and 3,184 clock cycles, respectively. This is faster than previous works by 22 % for 128-bit security level. The FACE–LIGHT can be used to extend the FACE to round 3. The AES is also implemented to be secure against the CPA (Correlation Power Analysis).

Keywords: AES, Software Implementation, Counter Mode, Microcontroller, Correlation Power Analysis

1 Introduction

Low-end IoT (Internet of Things) platforms are resource constrained devices, which have limited memory size and low computing power. In order to apply the cryptography protocols, the research on lightweight cryptography, which is relatively simple and has short computation time, has been actively conducted. Typical lightweight encryption algorithms include LEA, HIGHT, SIMON, SPECK, and CHAM [1–4]. However, most of these lightweight cipher algorithms adopt the ARX (Addition, Rotation, and XOR) structure, which has the disadvantage that it takes much additional time when applying the masking operation to prevent the side channel attack [5].

Even though the AES encryption algorithm is not considered as a lightweight encryption due to its relatively long computation time than other lightweight block ciphers, AES has been the worlds most used encryption algorithm for a long period of time, and is an international standard encryption algorithm with

* Corresponding Author

high security and generality [6]. In addition, AES adopts the SPN (Substitution-Permutation-Network) structure, which requires relatively less time for masking computation than the ARX structure. If the AES can be optimized and implemented on a low-end processor, it can be used as a lightweight cipher with high security and generality. In contrast to most of previous AES-CTR implementations, which focused mostly on the high-end processors, this paper concretes on low-end microcontrollers. We revisited the previous works and tailored the method to fit into the low-end environments.

This paper is organized as follows. Section 2 discusses the previous AES implementations and Fast AES CTR mode Encryption (FACE) technique, which is the fastest AES-CTR implementation method. In Section 3, we introduce FACE–LIGHT method for microcontrollers. In Section 4, we evaluate the performance of the proposed implementation. Section 5 concludes this paper.

2 FACE: Fast AES CTR mode Encryption

In CHES'18, the efficient AES-CTR implementation (i.e. FACE) for high-end processor was suggested [7]. The FACE method utilizes the value of IV depending only on the change of counter values. Since the IV value, except for the counter value, remains the same as the following blocks, an identical pattern is repeated in specific part of the encrypted value until the Round 2 of AES. By utilizing this feature, repeated values can be stored in the cache table and used, which minimizes the encryption operation of subsequent blocks during the encryption operation, thereby effectively reducing the encryption operation time.

The first step is the FACE$_{rd0}$. In this step the FACE utilizes the fact that in Round 0, only the Add-RoundKey operation is performed. In the case of Add-RoundKey operation, a byte calculation is not affected by other bytes since it is a `XOR` operation which only deals with single bytes independently. The only byte difference between the first IV block and the second IV block is the last byte that is used as a counter. The Add-RoundKey operation can be minimized by storing the previous 12 bytes out of 16 bytes in the precomputed table. The table is only replaced after a 2^{32-1} block operation where all unused 4 bytes are `0xFFFFFFFF`. This approach requires only one cache update while processing 65.5GB of plaintext. The description of the step is shown in Figure 1 and the Add-RoundKey result values, the State, can be seen stored in the cache. The cache consists of 4×4 bytes in total. Among the cache, only the values in the 0, 1, and 2 columns are reused. The values in the third column are not reused to minimize the cache update.

The second step is FACE$_{rd1}$. The State value from Round 0 is used as the input value of Round 1, where FACE$_{rd1}$ reuses some of the values stored in the cache. In Figure 2, it is shown that after the Round 1, the value of $S[15]$, which was the only different byte from previous block, affects the whole first column through the Mix-Columns operation. The remaining columns except the first can be reused since they are not affected by $S[15]$. The reusable column values can be used until the value of $S[15]$ exceeds `0xFF` and affects $S[14]$, which can be used up to 256 times.

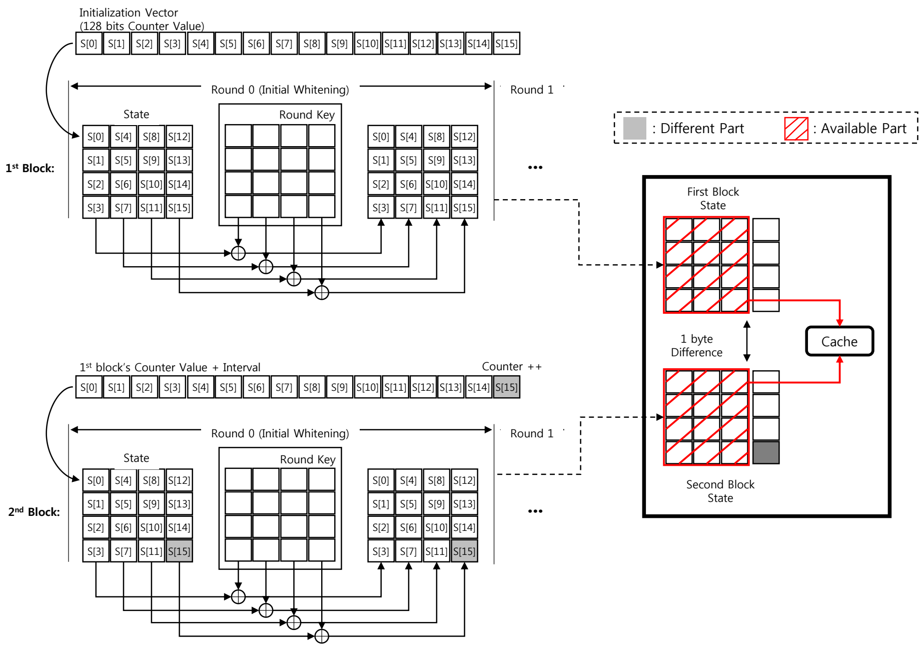


Fig. 1: Initial whitening of the first and the second block in CTR mode

The third step is FACE$_{rd1+}$. In this step, it is suggested that the value of the first column, which gets changed in the step FACE$_{rd1}$, be created as a cache table through a precomputation. The value in the first column consists of $S[0]$, $S[5]$, $S[10]$ and $S[15]$, which get affected by $S[15]$ in the Mix-Columns operation. Therefore, 1KB (256 x 4) of cache can be generated through a pre-calculation based on the $S[15]$ value that changes according to the counter value. The cache table can be created beforehand based on $S[15]$ since the value of $S[10]$, the high byte of $S[15]$, is not affected until a total of `0xFFFFFFFFFF` blocks are initialized from $S[15]$ to $S[11]$, the 1,099,511,627,776-th block(16 TB), is calculated.

The forth step is FACE$_{rd2}$. This step deals with Round 2 which utilizes the output of the Round 1. Through Round 1 operation, the value of the first column of Round 2 is affected by the changes of the counter value. Figure 3 shows the Round 2 process of the first and second blocks. $S[0], S[1], S[2]$ and $S[3]$, which are affected by the counter value, are spread to other columns by the Sub-Bytes operation. The values affect all 16 Bytes through Mix-Columns operation. In conclusion, the whole byte is affected by the counter value. However, during the Mix-Columns operation, some of the values can be reused. Shown in Figure 3, operation values except $S[0]$ can be reused such as $S[5], S[10], S[15]$ and round

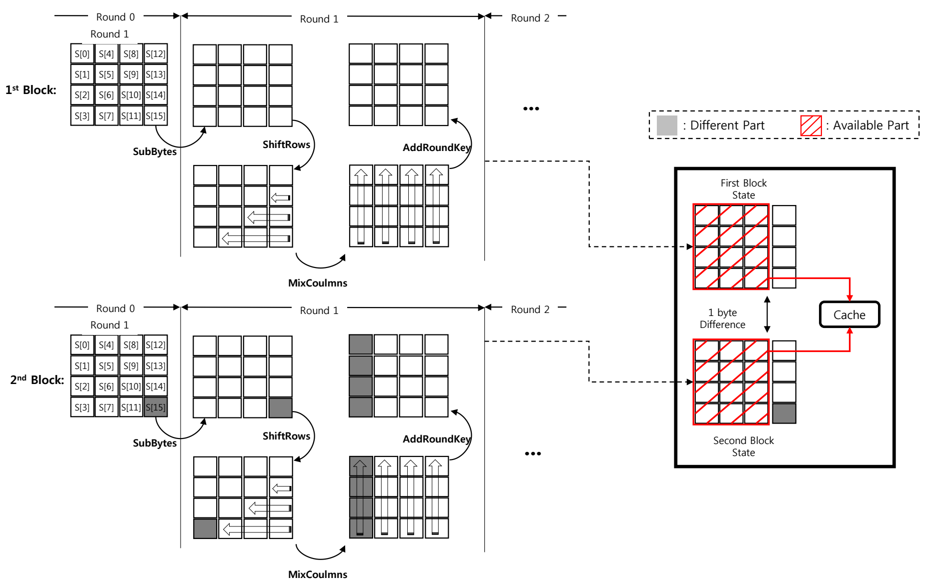


Fig. 2: Round 1 and the difference between the first and the second block

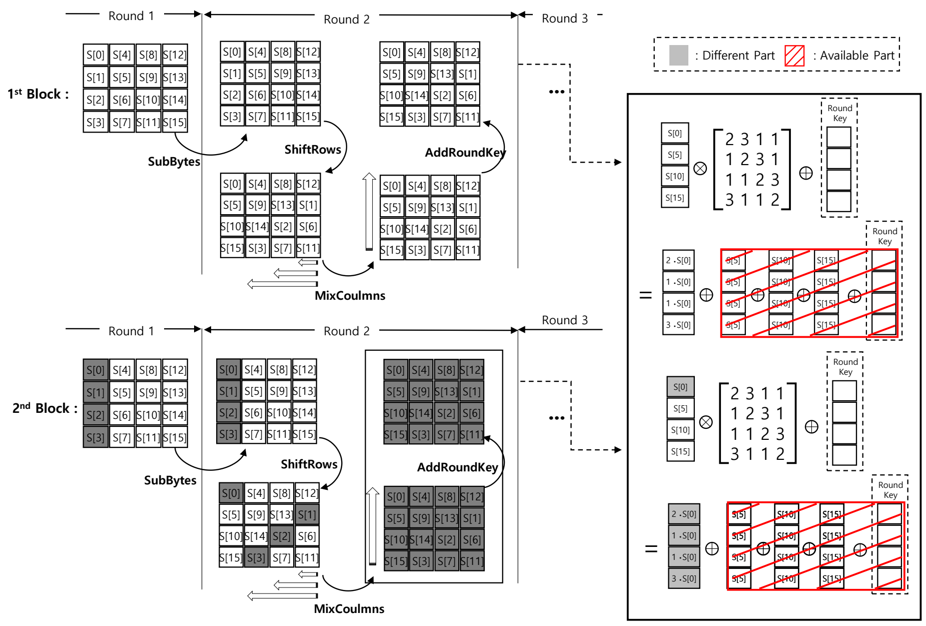


Fig. 3: Round 2 and the difference between the first and the second block

key. These values will not change and repeat until the IV $S[14]$ is not affected

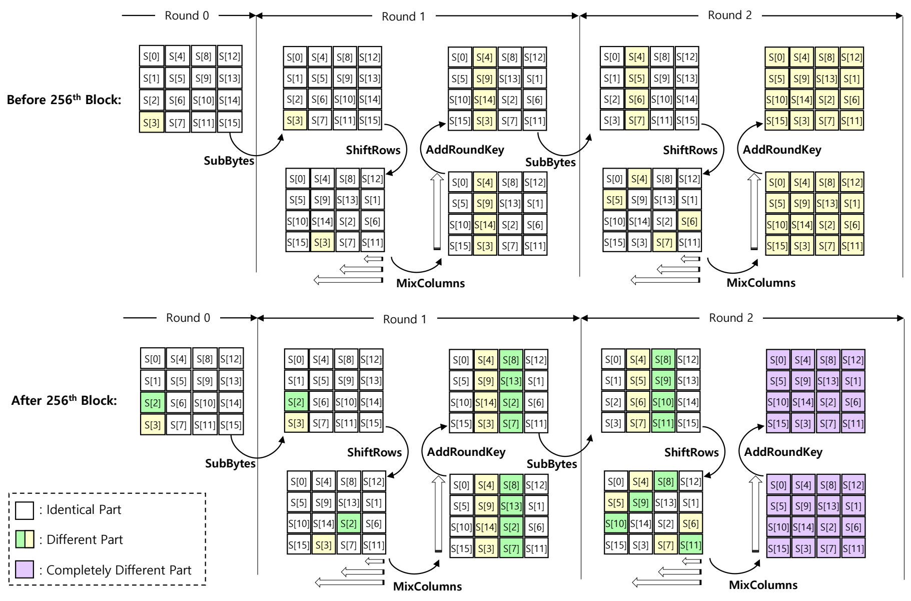


Fig. 4: Overview of FACE

by the counter increment. Therefore, making these 16 Bytes of unaffected values into cache table, can be used during operation up to the 256^{th} block.

In the final step, FACE$_{rd2+}$, it is suggested to create and store 4KBytes (16 x 256) table which contains the changing operation values which were excluded from FACE$_{rd2}$ and involves around $S[0], S[1], S[2]$ and $S[3]$. As in the case of FACE$_{rd1+}$, while calculating the 1,099,511,627,776-th block (16 TB), a previously calculated table can be used. The use of these lookup tables can provide significant advantages in cryptographic computation time, since it minimize Round 0, Round 1 and Round 2 encryption operations in repeated blocks and uses the same pre-computed values. The detailed descriptions of FACE are given in Figure 4.

However, this work is only efficient for 8-bit counter mode. In Figure 4, if the last 8-bit counter value is 0 to `0xFF`, it can be used as a table except for the different parts painted in black. However, after 256 blocks, more values are affected as the value of $S[14]$ is changed as shown below. In this case, since we can not use pre-stored table, we need to update the cache table in each 256 times encryption. Since this frequent updates can be abused by attacker as an attack point (i.e. fault attack), we need to implement the FACE in regular form. In this paper, we revisit the FACE and optimize the method for low-end microcontrollers. The proposed FACE–LIGHT is regular fashion and we don't need to update the cache table throughout the whole AES-CTR life-cycle.

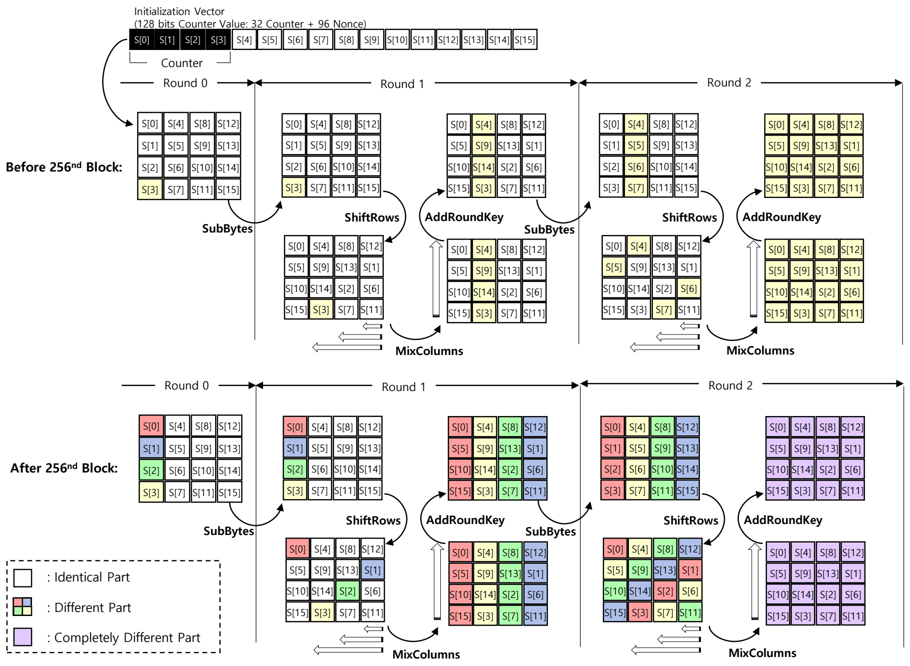


Fig. 5: Overview of FACE–LIGHT

3 FACE-LIGHT

In this section, we introduce the new implementation techniques for AES-CTR mode. Compared with previous work (i.e. FACE), we optimized the method for low-end microcontrollers and evaluated the masked implementation as well. The CTR mode consists of nonce and counter values. For the general setting, nonce and counter are set to 96-bit and 32-bit, respectively. The nonce is not changed throughout the whole sessions but the counter is changed in each transaction. Previous work focused on the low 8-bit and this needs to update the cache table in 256-times of encryption. In proposed work, we consider the 32-bit counter. We only need to set the pre-computed table in the initialization stage and it is not changed during the whole sessions. The detailed descriptions are given in Figure 5. The above figure indicates first block and the below figure indicates n-th block. Each square contains 8-bit data and the white and black colors represent identical and different parts, respectively.

Round 0 In Round 0, the only computation is Add-RoundKey. The plaintext and round key is xored. The only difference between two blocks is 4 bytes.

Round 1 In Round 1, Sub-Bytes, Shift-Rows, Mix-Columns, and Add-RoundKey operations are performed in order. The Sub-Bytes operation only changes the

8-bit input into the 8-bit output. This does not spread the values to the adjacent squares. The Shift-Rows tries to shift the square by certain offsets. In the Mix-Columns, the 8-bit data is mixed with 32-bit column. Finally, the Add-RoundKey adds the round keys to each square. Between first and N-th blocks, there is no common values. However, the both results are originated from IV. In particular, the low 32-bit value, namely counter value (i.e. $S[0], S[1], S[2]$, and $S[3]$), mainly contributes to the differences. In the Mix-Columns, $S[0]$ square influences to the first column (i.e. $S[5], S[10]$, and $S[15]$). For this reason, each column (i.e. 32-bit) depends on the 8-bit square. Similarly, other columns are also based on the squares (i.e. $S[1], S[2]$, and $S[3]$). This means each 32-bit column has only 256 cases. The following Add-RoundKey only applied to the each square so the pre-computation complexity is not changed. For this stage, the FACE performs in round 2 since they concern the 8-bit counter case with table update, while FACE–LIGHT is 32-bit counter case without table update.

Round 2 In the previous stage, each column is based on the 8-bit square value. This is still maintained in Sub-Bytes operation which is only based on the each square value and the value is determined by 8-bit square of previous round. Since the Shift-Rows operation does not perform any mixing and changing on the value, the pre-computation is still working. For pre-computation, we need to keep 4 look-up tables. The input values are $S[0], S[1], S[2]$, and $S[3]$ in 8-bit wise. The length of output value is 32-bit wise. The total size for look-up table is 4 KB. In Figure 6, pre-computed table for FACE–LIGHT is described. In each table, we only receive the 8-bit input value and the value goes through the Mix-Columns, Add-RoundKey, and Sub-Bytes in this order. The Shift-Rows operation is not combined in the pre-computed table. It is optimized away by directly assigning the results to the specific squares.

Extended round for FACE The FACE–LIGHT is applied to the Round 2. This can be utilized for the FACE Round 3 since the different value is same with Round 2 of FACE–LIGHT. By using this technique, the Sub-Bytes and Add-RoundKey of Round 3 can be also cached. The detailed extended round for FACE are given in Figure 7. We used FACE Round 1 method for AES Round 1. From Round 2 to 3, we used the FACE-LIGHT strategy to extend the 1 round more than FACE method.

Optimized Implementation The pre-computed table is stored before encryption operation. The 8-bit AVR microcontroller has very limited `SRAM`. For storing huge pre-computed table, we used the `PROGRAM MEMORY`. In each look-up table access, 32-bit results are extracted. For this reason, 8-bit input offset is extended to the 32-bit input offset by using quadrupling on the offset. Afterward, the input offset is added to the based address of each look-up table.

In AES algorithm, a substitution operation is performed during Sub-Bytes operation. During the operation, value of 256 Bytes to be substituted is stored in

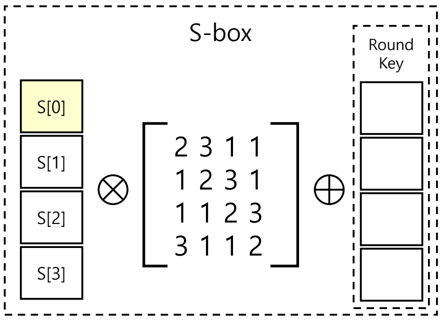


Fig. 6: Pre-computation for FACE–LIGHT

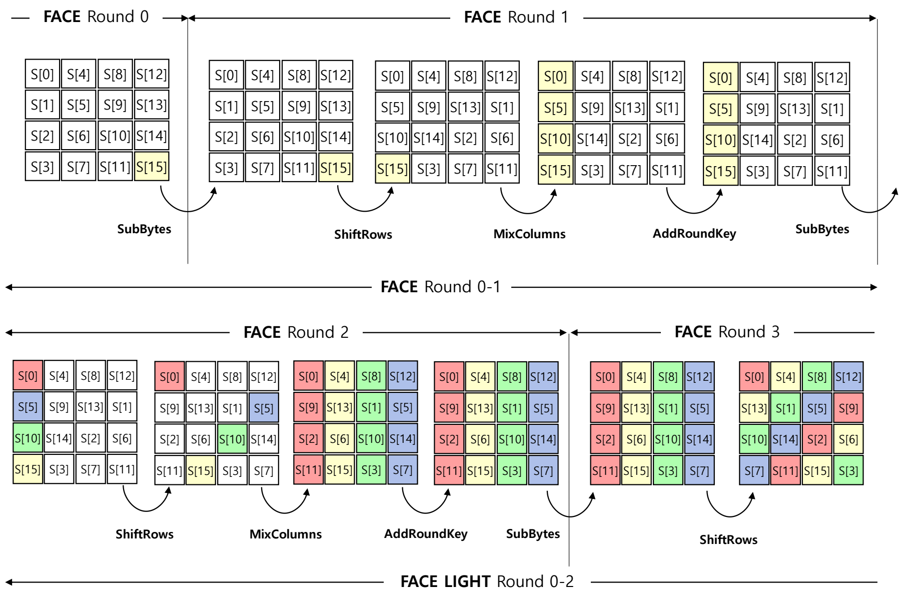


Fig. 7: Extended round for FACE round 3

memory as a table called SBOX in advance to shorten the computation time. In Sub-Bytes operation, the values which to be substituted are loaded based on the input ciphertext. Therefore, when the value stored in the SBOX is loaded using the `Z` pointer (`R30, R31`), the lower memory address and the upper memory address of the SBOX are loaded in `R30` and `R31`, respectively, and the ciphertext value to be used as the index value should be added to `R30` to match the index. In this process, a carry might occur at the lower address of the SBOX memory loaded in `R30`, so that the value of `R31` might needs to be increased by one. Therefore, if the ciphertext value is added to `R30`, the lower address of SBOX memory, to refer to the SBOX memory, the `ADC` operation which performs carry operation, must be performed on R31, which stores the upper address of SBOX memory.

However, if the carry does not occur, the ADC is not only a meaningless operation, but it cause a big overhead since two operations (ADD, ADC) is performed for each Sub-Bytes operation for the index operation. To solve this problem, we utilize the memory size of the SBOX, 256 Bytes. If the lower address of memory is set to 0x00, the carry value does not occur when indexing for memory access with the last index value of 0xFF. Therefore, the ADC operation on R31, including the carry value can be omitted and the ciphertext can be loaded directly without performing the ADD operation on R30. In the same way, the computation time when storing or loading values in the Masked–SBOX table, a table created for masking, can also be reduced using the same address sorting method.

LD and ST instructions are used to load or store the key and the round key values stored in consecutive memory. In the case of the AVR, 16-bit addresses are accessed using the X(R26, R27), Y(R28, R29) and Z(R30, R31) pointers. Therefore, when accessing the repetitive memory or the peripheral memory, the memory address to be accessed should be set by using the ADD and ADC operators in the register that constitutes the pointer. In this case, carry value operation should be included as described above which causes a large overhead. Therefore, the address access without additional operator was performed by using LDD and STD or memory address post incremental instruction provided by AVR.

Optimization for Masking Operation Masking operations aim to prevent the attacker from accurately measuring power usage by adding useless operations without affecting the encryption operations themselves. Therefore, the clock cycle is inevitably longer than the conventional encryption operation. In this paper, in order to minimize this disadvantage, we propose two methods.

The first method is as follows. Before performing encryption operation, 10 sets of round keys, from Round 0 to Round 9, pre-computed by fixed key encryption, should perform XOR operation with the value M0. Then carry out XOR operation on the first row with M6, the second row with M7, the third row with M8 and the last row with M9. In the case of Round 10, the last round, does not include M6, M7, M8 and M9 operations, but XORs the M1 value to all of the key values. If the corresponding operation is performed in advance, the XOR operation repeated in each round is minimized, and the memory in which the masking value is stored does not have to be loaded in each round. It can be observed in Algorithm 1 that M0 is XORed with M6, M7, M8, M9 in line 4. At label 1, M6, M7, M8 and M9 are XORed row by row for a total of 10 key sets from Round 0 to Round 9. By XORing M0 value to each mask value in advance, 160 XOR operations can be reduced. It is also shown in label 2, M1 XORs with the last round key.

The second method is as follows. Sub-Bytes operation is a relatively clock cycle consuming since it needs to access SBOX memory stored as a table. In addition, after performing Sub-Bytes operation, the masked value, M0, needs to be replaced with M1. Therefore, overhead occurs when executing the memory load

Algorithm 1 Masking operation optimization

```
mov HI, ROUND
lsl HI
lsl HI

eor MASK6, MASK0
eor MASK7, MASK0
eor MASK8, MASK0
eor MASK9, MASK0

1:
ld r0, Z
eor r0, MASK6
st Z+, r0

ld r0, Z
eor r0, MASK7
st Z+, r0

ld r0, Z
eor r0, MASK8
st Z+, r0

ld r0, Z
eor r0, MASK9
st Z+, r0

dec HI
brne 1b

ldi HI, 4

2:
ld r0, Z
eor r0, MASK1
st Z+, r0

ld r0, Z
eor r0, MASK1
st Z+, r0

ld r0, Z
eor r0, MASK1
st Z+, r0

ld r0, Z
eor r0, MASK1
st Z+, r0

dec HI
brne 2b
```

Algorithm 2 Generating Masked SBOX

```
ldi r31, hi8(MSBOX)
ldi r29, hi8(SBOX)
ldi xREDUCER, 0x1b

1:
mov r28, r0
ld r26, Y
eor r26, T1

mov r30, r0
eor r30, T0
st Z, r26

inc r0
brne 1b
```

operation and XOR operation every round. In order to minimize the overhead, Masked-SBOX table should have been precomputed and be referenced in the Sub-Bytes operation. The detailed descriptions of the implementation are given in Algorithm 2.

4 Evaluation

In this paper, we utilized the 8-bit AVR microcontoller. In particular, we used the `Arduino UNO` platform, which equips the `ATmega328` processor. The hardware follows the Harvard architecture and the working frequency is 16MHz. It contains 32 8-bit general purpose registers and has a total of 131 instruction sets. Flash memory is 32KB in size, with 1KB of `EEPROM` and 2KB of internal `SRAM`. We compile the code in `-OS` option and the performance is compared in terms of clock cycles. For performance measurement, the software was developed with `Arduino IDE` and `Atmel Studio 7` on an `Arduino UNO` board with `Atmega328p`. All the functions in the program were implemented in Assembly except the loop function. The `Arduino UNO` board's frequency was 16MHz and the

Table 1: Comparison of AES implementations on 8-bit AVR, in terms of clock cycles.

Security level	Dinu et al. [8]	Otte et al. [9]	**FACE–LIGHT (This work)**	**Extended FACE (This work)**
AES-128	2,835	2,507	**2,218**	**1,967**
AES-192	N/A	2,991	**2,702**	**2,449**
AES-256	N/A	3,473	**3,184**	**2,931**

Table 2: Comparison of FACE and FACE–LIGHT.

	FACE [7]	FACE–LIGHT (This work)
Table update	√	–
Constant timing	–	√
Target processor	32-bit or above	8-bit or above
Expandable Round	Round 2	Round 3

results were obtained using the `Arduino IDE` and `Atmel Studio 7` for accurate performance measurements. In addition, in order to confirm that the proposed AES is safe against power analysis attack, the power consumption during encryption operation is measured by Chipwhisperer-Lite (`CW1173`). In addition, CPA was performed by collecting 5,000 waveforms in the third round with the output value of the Sub-Bytes operation as the middle value.

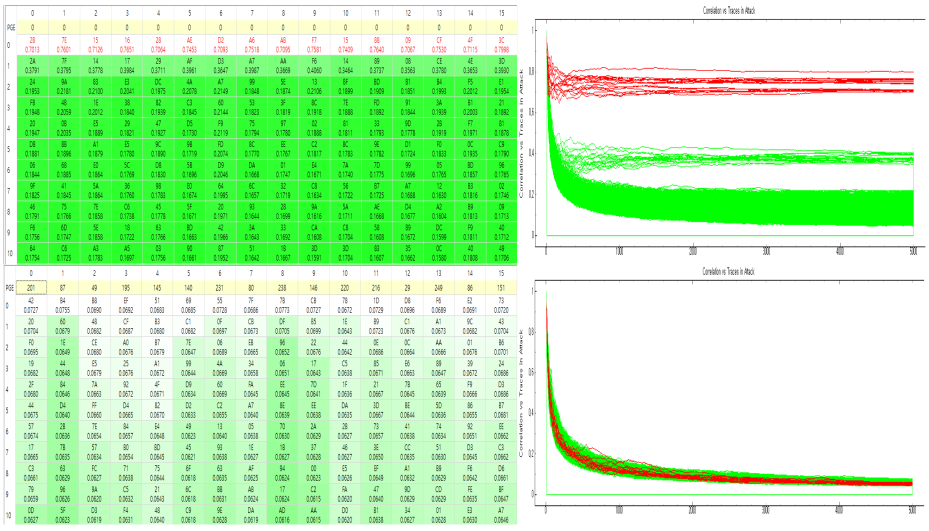
Table 1 shows the comparison of the clock cycles of four AES. Previous AES implementations are under ECB mode. The proposed method is CTR mode by using FACE–LIGHT method and extended FACE which FACE–LIGHT is applied. The main differences are mode of operation and their input values. For more details, Dinu et al. [8] is the result of code size software optimization and Otte et al. [9] is the result of clock cycle software optimization. In our setting, we used some special cases as mentioned in Chapter 3. For this reason, FACE–LIGHT is faster than previous state-of-art by 617 clock cycles and Extended FACE is faster by 868 clock cycles. The proposed method efficiently passes the certain routines with pre-computed tables.

In Table 2, the comparison results between FACE and FACE–LIGHT are given. The FACE–LIGHT does not require table update during computations. This means the encryption timing is always regular fashion. The timing information indicates the order of messages. By measuring the timing, we can get these information and this can be linked to the privacy issue. The target processor varies in each method. The FACE is for 32-bit processor while the FACE–LIGHT is for 8-bit microcontrollers. In addition, the FACE can be expanded till Round 2 while our work, FACE–LIGHT, can be expanded to the Round 3.

Existing lightweight ciphers have also been studied for the implementation of additional masking techniques to cope with side channel attacks. However, most lightweight ciphers with ARX structures have significant overhead since the ciphers must undergo Arithmetic-to-Boolean operation during masking operations. On the other hand, the SPN structure, AES, has a relatively short operation time. Therefore it has an advantage over other lightweight ciphers in the masking operation, which is a side channel countermeasure.

Table 3: Comparison of LEA and optimized masked AES implementations on 8-bit AVR microcontroller, in terms of clock cycles.

LEA-128 [10]	Masked LEA-128 [11]	Masked AES-128 (This work)
2,688	36,589	**6,219**



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2B 0.7013	7E 0.7601	15 0.7126	16 0.7651	28 0.7064	AE 0.7453	D2 0.7093	A6 0.7518	AB 0.7095	F7 0.7581	15 0.7409	88 0.7640	09 0.7067	CF 0.7530	4F 0.7115	3C 0.7998
1	2A 0.3791	7F 0.3795	14 0.3778	17 0.3984	29 0.3711	AF 0.3961	D3 0.3647	A7 0.3987	AA 0.3669	F6 0.4060	14 0.3464	89 0.3737	08 0.3563	CE 0.3780	4E 0.3653	3D 0.3930
2	24 0.1953	9A 0.2181	83 0.2100	E3 0.2041	DC 0.1975	4A 0.2078	A7 0.2149	99 0.1848	5E 0.1874	13 0.2106	BF 0.1899	BD 0.1909	B1 0.1851	B4 0.1993	F5 0.2012	E1 0.1954
3	FB 0.1948	4B 0.2059	1E 0.2012	38 0.1840	82 0.1939	C3 0.1845	60 0.2144	53 0.1823	3F 0.1819	8C 0.1918	7E 0.1888	FD 0.1892	91 0.1844	3A 0.1939	B1 0.2003	21 0.1892
4	20 0.1947	0B 0.2035	E5 0.1889	29 0.1821	47 0.1927	D5 0.1730	F9 0.2119	75 0.1794	97 0.1780	02 0.1888	81 0.1811	33 0.1793	9D 0.1778	2B 0.1919	F7 0.1971	81 0.1878
5	DB 0.1881	8B 0.1896	A1 0.1879	E5 0.1780	9C 0.1890	9B 0.1719	FD 0.2074	8C 0.1770	EE 0.1767	C2 0.1817	8C 0.1783	9E 0.1782	D1 0.1724	F0 0.1833	0C 0.1935	C9 0.1790
6	06 0.1844	68 0.1885	E0 0.1864	5C 0.1769	D8 0.1830	5B 0.1696	D9 0.2046	DA 0.1668	01 0.1747	E4 0.1671	7A 0.1740	7D 0.1775	99 0.1696	05 0.1765	BD 0.1857	96 0.1765
7	9F 0.1825	41 0.1845	5A 0.1864	36 0.1760	98 0.1783	E0 0.1674	64 0.1995	6C 0.1657	32 0.1719	C8 0.1634	56 0.1722	B7 0.1725	A7 0.1688	12 0.1630	B3 0.1816	02 0.1746
8	46 0.1791	75 0.1766	7E 0.1858	C6 0.1738	45 0.1778	5F 0.1671	20 0.1971	93 0.1644	28 0.1699	9A 0.1616	5A 0.1711	AE 0.1668	D4 0.1677	A2 0.1604	B9 0.1813	09 0.1713
9	F6 0.1756	6D 0.1747	5E 0.1858	18 0.1722	63 0.1766	BD 0.1663	42 0.1966	3A 0.1643	33 0.1692	CA 0.1608	C8 0.1704	58 0.1608	B9 0.1672	DC 0.1599	F9 0.1811	40 0.1712
10	64 0.1754	C6 0.1725	A3 0.1783	A5 0.1697	03 0.1756	90 0.1661	87 0.1952	51 0.1642	1B 0.1667	3D 0.1591	3D 0.1704	83 0.1607	35 0.1662	0C 0.1580	40 0.1808	49 0.1706

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	201	87	49	195	145	140	231	80	238	146	220	216	29	249	86	151
0	42 0.0727	B4 0.0755	B8 0.0690	EF 0.0692	51 0.0683	69 0.0685	55 0.0728	7F 0.0686	7B 0.0773	CB 0.0727	78 0.0672	1D 0.0729	D8 0.0696	F6 0.0689	E2 0.0691	73 0.0720
1	20 0.0704	60 0.0679	48 0.0682	CF 0.0687	B3 0.0680	C1 0.0682	0F 0.0697	CB 0.0673	DF 0.0705	B5 0.0699	1E 0.0643	B9 0.0723	C1 0.0676	A1 0.0673	9C 0.0682	43 0.0704
2	F0 0.0695	1E 0.0649	CE 0.0680	A0 0.0676	B7 0.0679	7E 0.0647	06 0.0689	EB 0.0665	96 0.0652	22 0.0676	44 0.0642	0E 0.0686	0C 0.0664	AA 0.0666	01 0.0676	B6 0.0701
3	19 0.0682	44 0.0648	E5 0.0679	25 0.0676	A1 0.0672	99 0.0644	4A 0.0669	34 0.0658	06 0.0651	17 0.0643	C5 0.0638	85 0.0671	E6 0.0663	89 0.0647	39 0.0672	24 0.0686
4	2F 0.0680	84 0.0646	7A 0.0663	92 0.0672	4F 0.0671	D9 0.0634	60 0.0669	FA 0.0645	EE 0.0645	7D 0.0641	1F 0.0636	21 0.0667	7B 0.0645	65 0.0639	F9 0.0666	D3 0.0686
5	44 0.0675	D4 0.0640	FF 0.0660	D4 0.0665	82 0.0670	D2 0.0633	C2 0.0655	A7 0.0640	8E 0.0639	EE 0.0638	DA 0.0635	3D 0.0667	BE 0.0644	5D 0.0636	86 0.0655	B7 0.0681
6	57 0.0674	2B 0.0636	7E 0.0654	84 0.0657	E4 0.0648	49 0.0623	13 0.0640	05 0.0638	70 0.0630	2A 0.0629	2B 0.0627	73 0.0657	41 0.0638	74 0.0634	92 0.0651	EE 0.0662
7	17 0.0665	7B 0.0635	57 0.0634	B0 0.0654	BD 0.0645	45 0.0621	93 0.0638	1E 0.0627	1B 0.0627	37 0.0628	46 0.0627	3E 0.0650	CC 0.0635	51 0.0630	D3 0.0645	C3 0.0662
8	C3 0.0661	63 0.0629	FC 0.0627	71 0.0638	75 0.0644	6F 0.0618	63 0.0635	AF 0.0625	94 0.0624	00 0.0623	E5 0.0626	EF 0.0649	A1 0.0632	B9 0.0629	F6 0.0642	D6 0.0661
9	79 0.0659	96 0.0626	9A 0.0620	C5 0.0632	21 0.0643	6C 0.0618	BB 0.0631	AB 0.0624	17 0.0624	C2 0.0615	FA 0.0620	47 0.0640	9D 0.0629	CD 0.0629	FE 0.0635	BF 0.0647
10	0D 0.0627	5F 0.0623	D3 0.0619	F4 0.0631	48 0.0640	C9 0.0618	9E 0.0628	DA 0.0619	AD 0.0616	AA 0.0615	D0 0.0620	B1 0.0638	34 0.0627	01 0.0628	E3 0.0630	A7 0.0646

Fig. 8: Comparison of non-Masked AES and Masked FACE–LIGHT on CPA attack
(Top Left)Table of Analyzed Key Value via CPA Attack on Non-masking AES
(Top Right)Correlation graph of key values through CPA Attack on Non-Masking AES
(Bottom Left)Table of Analyzed Key Value via CPA Attack on Masking AES
(Bottom Right)Correlation graph of key values through CPA Attack on Masking AES

Table 3 shows the clock cycles of none masked LEA, masked LEA and AES encryption algorithm with masking operation. In the case of masked LEA–128, the encryption operation time increases rapidly when masking is applied to LEA. However, it can be observed that the proposed AES has robustness against power consumption analysis attack while having less computation time.

Figure 8 shows a graph of key value and correlation coefficient estimated by performing CPA on AES without mask and AES presented in this paper. In case of the AES without masking operation, the correlation coefficient of all key values is significantly higher than other values. However, in the case of the AES proposed in this paper, the attacker cannot guess the key value since all of the key values have equal correlation coefficients.

5 Conclusion

In this paper, we demonstrate the implementation of AES–CTR encryption on 8-bit AVR microcontrollers. The proposed FACE–LIGHT efficiently improves the performance. Furthermore, we investigated the extended round method and masked AES implementations. For future works, We will apply this method to the AES–GCM, which consists of CTR and polynomial multiplication and other block ciphers/mode of operations.

6 Acknowledgement

This work was supported as part of Military Crypto Research Center(UD170109ED) funded by Defense Acquisition Program Administration(DAPA) and Agency for Defense Development(ADD).

References

1. D. Hong, J. Lee, D. Kim, D. Kwon, K. H. Ryu, and D. Lee, "LEA: A 128-bit block cipher for fast encryption on common processors," in *International Workshop on Information Security Applications*, pp. 3–27, Springer, 2013.
2. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, *et al.*, "HIGHT: A new block cipher suitable for low-resource device," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 46–59, Springer, 2006.
3. R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The SIMON and SPECK lightweight block ciphers," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2015.
4. B. Koo, D. Roh, H. Kim, Y. Jung, D. Lee, and D. Kwon, "CHAM: a family of lightweight block ciphers for resource-constrained devices," in *International Conference on Information Security and Cryptology*, pp. 3–25, Springer, 2017.
5. L. Goubin, "A sound method for switching between boolean and arithmetic masking," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 3–15, Springer, 2001.
6. N.-F. Standard, "Announcing the advanced encryption standard (AES)," *Federal Information Processing Standards Publication*, vol. 197, no. 1-51, pp. 3–3, 2001.
7. J. H. Park and D. H. Lee, "FACE: Fast AES CTR mode encryption techniques based on the reuse of repetitive data," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 469–499, 2018.
8. D. Dinu, A. Biryukov, J. Großschädl, D. Khovratovich, Y. Le Corre, and L. Perrin, "FELICS–fair evaluation of lightweight cryptographic systems," in *NIST Workshop on Lightweight Cryptography*, vol. 128, 2015.
9. D. Otte *et al.*, "AVR-crypto-lib," *Online: http://www. das–labor. org/wiki/AVR–Crypto–Lib/en*, 2009.
10. H. Seo, I. Jeong, J. Lee, and W. Kim, "Compact implementations of ARX-based block ciphers on IoT processors," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 3, p. 60, 2018.
11. E. Park, S. Oh, and J. Ha, "Masking-based block cipher LEA resistant to side channel attacks," *Journal of the Korea Institute of Information Security and Cryptology*, vol. 27, no. 5, pp. 1023–1032, 2017.