mathematics

*Article*

# Masked Implementation of Format Preserving Encryption on Low-End AVR Microcontrollers and High-End ARM Processors

**Hyunjun Kim, Minjoo Sim, Kyoungbae Jang, Hyeokdong Kwon, Siwoo Uhm and Hwajeong Seo ***

Division of IT Convergence Engineering, Hansung University, Seoul 02876, Korea; amdjd0704@hansung.ac.kr (H.K.); alswntla@hansung.ac.kr (M.S.); starj1234@hansung.ac.kr (K.J.); hyeok@hansung.ac.kr (H.K.); smile267@hansung.ac.kr (S.U.)
* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-2-760-8033

**Abstract:** Format-Preserving Encryption (FPE) for Internet of Things (IoT) enables the data encryption while preserving the format and length of original data. With these advantages, FPE can be utilized in many IoT applications. However, FPE requires complicated computations and these are high overheads on IoT embedded devices. In this paper, we proposed an efficient implementation of Format-preserving Encryption Algorithm (FEA), which is the Korean standard of FPE, and the first-order masked implementation of FEA on both low-end (i.e., AVR microcontroller) and high-end (i.e., ARM processor) IoT devices. Firstly, we show the vulnerability of FEA when it comes to the Correlation Power Analysis (CPA) approach. Afterward, we propose an efficient implementation method and the masking technique for both low-end IoT device and high-end IoT device. The proposed method is secure against power analysis attacks but the performance degradation of masked measure is only 2.53∼3.77% than the naïve FEA implementation.

**Keywords:** format preserving encryption; masked implementation; AVR microcontroller; ARM processor

**Citation:** Kim, H.; Sim, M.; Jang, K.; Kwon, H.; Uhm, S.; Seo, H. Masked Implementation of Format Preserving Encryption on Low-End AVR Microcontrollers and High-End ARM Processors. *Mathematics* **2021**, *9*, 1294. https://doi.org/10.3390/math9111294

Academic Editor: Eckhard Pfluegel

Received: 2 April 2021
Accepted: 2 June 2021
Published: 4 June 2021

## 1. Introduction

Format-preserving encryption is a cryptography algorithm that maintains the format or type of plaintext after the encryption, (i.e., format and length of ciphertext). With these nice properties, the data format conversion (e.g., padding) is not required before the encryption. In traditional block ciphers (e.g., AES [1]), the length of block size should be the multiple of block size (e.g., 128-bit wise). When the block size is not divisible by 128-bit, the padding should be added to the plaintext to ensure the multiple of 128-bit wise data format.

When only four out of sixteen numbers (i.e., bytes) of credit card are encrypted with the AES block cipher, the ciphertext should be extended to 128-bit wise and this is four times longer than the size of original plaintext to be encrypted. On the other hand, the format-preserving encryption can preserve the 4-digit number format, which is the storage efficient approach than the traditional block cipher method. By using the format-preserving encryption, the existing file system is still available without altering the structure of system.

The typical format-preserving encryption includes the NIST standard format-preserving encryption of FF1, FF3-1, Visa Format Preserving Encryption (VFPE), and the Korean standard Format-preserving Encryption Algorithm (FEA) [2–5]. The FEA algorithm, presented by National Security Research (NSR) institute, uses Feistel structures similar to NIST standards, such as FF1 and FF3-1. However, FF1 and FF3-1 algorithms use block ciphers as $F$ functions, while the FEA uses its own dedicated functions. This feature allows the high-speed encryption than other algorithms designed for the format preserving encryption. The FEA can be an appropriate choice when the sensitive personal information is generally short length.

Internet of Things (IoT) is a network of low-power devices designed to interact and communicate with each other by adding sensors and communication functions to various IoT objects. At this point, IoT devices are easily exposed to viruses or malicious attacks. For this reason, the security should be considered along with the development of IoT services. The format-preserving encryption for IoT devices enables the encryption of data values that maintain the format and length of the data [6–8].

When utilizing block ciphers on IoT devices, it is necessary to consider a side-channel attack. The side-channel attack exploits the information when the device is working, (e.g., electromagnetic wave or heat) [9]. There are several proposals for countermeasures [10–14]. All of these countermeasures reduce the direct relationship between the key and power consumption, allowing less information to be retrieved from the power trace. The masking is a popular mitigation technique used against power analysis attacks. Common masking methods consist of S-box masking and higher order masking. In this countermeasure, intermediate values are hidden with random values. The `F` function of FEA consists of two layers including a substitution layer and a diffusion layer. The substitution layer uses an 8-bit wise S-Box, which is known to be vulnerable to side-channel attacks, such as DPA. Therefore, it is necessary to apply side-channel resistance techniques to IoT devices. In this paper, we propose a FEA implementation technique considering the IoT environment using masking techniques.

*Contribution*

- Correlation power analysis on the FEA: We implemented the FEA on low-end AVR microcontrollers. Without secure countermeasures on the FEA implementation, the design is vulnerable to side channel attacks. We evaluated the security strength through the Correlation Power Analysis (CPA) and successfully extracted the secret information.
- Optimization techniques for Galois field multiplication on both low-end and high-end IoT devices: In low-end IoT environments, the *log/anti-log* table based Galois field multiplication is utilized. This approach replaces the complicated multiplication into the simple addition operation. For high-end IoT environments, both S-Box operation and Galois field multiplication are combined and executed at once. This approach eliminates unnecessary multiplication operations by considering bits of the removed block at the last stage of the `F` function.
- Novel masking technique for the FEA: Masking techniques should be carefully designed for different cryptographic structures. The additional computation for masking techniques degrades the execution timing. In order to ensure the high performance, we propose the optimized first-order masking technique for the FEA implementation.

## 2. Related Works

### 2.1. Format-Preserving Encryption Algorithm (FEA)

FEA is the standard FPE algorithm of Korea Telecommunications Technology Association selected in 2015 [5]. It consists of two algorithms, including FEA-1 and FEA-2. As shown in Table 1, both algorithms are supporting various key bit lengths (i.e., 128, 192, and 256). The difference between two algorithms is that FEA-1 has a tweak bit length of $128 - n$ bits, each with a number of 12, 14, and 16 rounds at key bit lengths of 128, 192, and 256, respectively. FEA-2 has a fixed tweak bit length of 128 bits with 18, 21, and 24 rounds at key bit lengths of 128, 192, and 256, respectively.

**Table 1.** Parameters of FEA-1 and FEA-2.

| Algorithm | Key Size (bit) | Block Size | Tweak Size | Round |
|---|---|---|---|---|
| FEA-1 | 128, 192, 256 | $n1 = \lceil \frac{n}{2} \rceil, n2 = \lfloor \frac{n}{2} \rfloor$ | $128 - n$ | 12, 14, 16 |
| FEA-2 | 128, 192, 256 | $n1 = \lceil \frac{n}{2} \rceil, n2 = \lfloor \frac{n}{2} \rfloor$ | 128 | 18, 21, 24 |

- **Encryption and Decryption.** Figure 1 shows that encrypting and decryption functions are Feistel structures using tweak. When implementing the decryption function, it has the advantage that does not require the inverse circuit of the round function. The round function `F` is consist of `Tw-KSP-KSP-Tr` functions, where `Tw` is a round tweak insertion, `K` is a key addition layer, `S` is a substitution layer, `P` is a diffusion layer, and `Tr` is truncation. In the substitution layer, 8 8-bit S-boxes are computed in the parallel way, and the diffusion layer is defined as the product of an $8 \times 8$ Maximum Distance Separable) matrix on $GF(2^8(MDS)$.
- **Key schedule.** The key schedule of the FEA has a secret key and a bit length of $n$ as input, and outputs $r$ (number of round) 128-bit round keys. It uses the Lai-Massey structure. The Lai-Massey [15] structure needs to recover the $i$-th and $(i+1)$-th round keys for every odd number $i$ to recover the secret key. Encryption and decryption of key schedules are used by changing only the order of the generated round keys using the same key schedule. Encryption and decryption are used by changing the order of same round keys generated. The operation process of the key schedule is shown in Figure 2. If the key is 128-bit long, $Ka||Kb$ is equal to the key value, and $Kc$ and $Kd$ are 0. If the key is 192 bits long, $Ka||Kb||Kc$ is equal to the key value and $Kd$ is 0. If $K$ is 256 bits, the key value is the same as $Ka||Kb||Kc||Kd$.
- **Tweak schedule.** FEA Type 1 and Type 2 use different tweak schedules. In Type 1, the tweak $T$ is divided into $Ta = T[0 : 64 - 2n - 1]$ and $Tb = T[64 - 2n : 128 - n - 1]$. $Ta$ in odd rounds and $Tb$ in even rounds are used. In Type 2, the tweak $T$ is divided into $Ta = T[0 : 63]$ and $Tb = T[64 : 127]$, and 0, $Ta$, and $Tb$ are used, repeatedly. The decryption uses the Round Tweak in the reverse order.

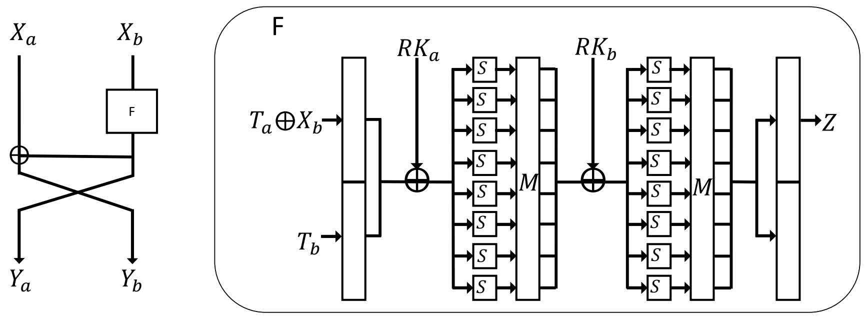

**Figure 1.** Feistel based round structure of FEA.

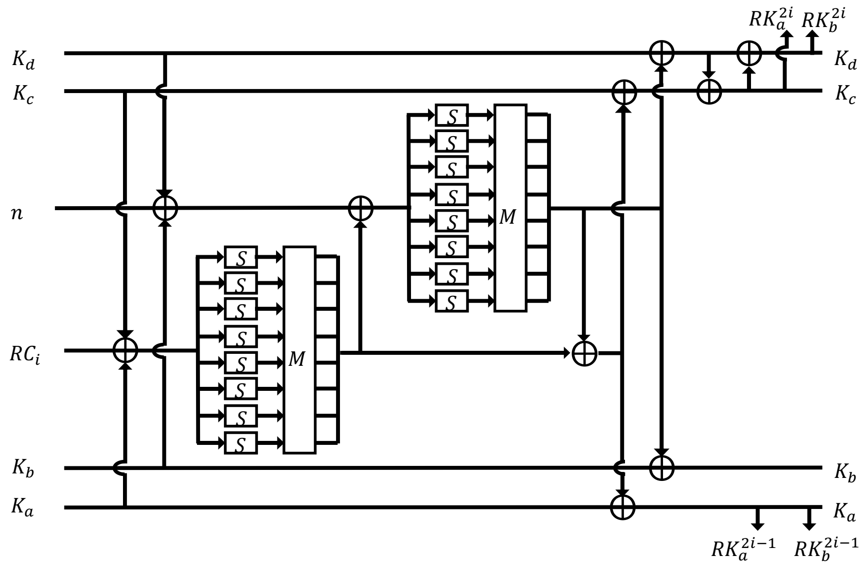

**Figure 2.** Key scheduling of FEA.

### *2.2. Previous Implementations of FEA*

There are several works on optimization of FEA implementations. In [16], they analyzed FEA and measured the performance of FEA by implementing it optimally in various environments. On a high-end device, SIMD was used to perform the operation in a parallel way. In addition, on ARM Cortex-A53, a memory look-up table of 2568 byte was used to design the FEA serial implementation. ARM Cortex-A53 was more efficient than SSE and AVX2 parallelism in PC. Reference [17] implemented FEA in AVR environment for BLE communication between IoT and users. The lookup table technique of [16] is used, and when the size of SRAM is larger than 2KB, faster performance is achieved by storing and reading LUTs (2KB) in the Flash Memory.

### *2.3. Power Analysis Attack on Block Cipher*

IoT devices are more accessible to attackers than general-purpose computing devices. Therefore, an attacker can exploit Side Channel Attack (SCA) by obtaining the information of physical leakages. Modern block ciphers are designed to resist all kinds of attacks. These ciphers have been analyzed over the years and they are widely believed to be theoretically secure. However, most of the implementations are vulnerable to side-channel attacks. The Correlated Power Analysis (CPA) is one of the SCA types and is the most effective method, and it has been shown that several block ciphers (e.g., AES, DES [18], PRESENT [19], SIMON [20], LED [21]) are vulnerable [13,22–24]. The CPA attack collects the power consumption of the part where the operation is performed with information that can be manipulated and the secret key to be guessed in the encryption algorithm. In addition, the power consumption is modeled for all cases of the guessed secret key. Hamming distance models can be used in CPA attacks. The actual power consumption and the modeled power consumption are compared using the Pearson correlation coefficient.

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sqrt{E[(X-\mu_X)^2]E[(Y-\mu_Y)^2]}}$$

The guessed key representing the highest correlation coefficient is determined as the key. S-Box, which is used to perform non-linear replacements in block ciphers, becomes more vulnerable to side-channel attacks, such as Differential Power Analysis (DPA) [25]. Therefore, it can be expected that FEA using S-Box will also be vulnerable to DPA attacks. The masking technique is an efficient SCA countermeasure to randomize sensitive median values. For the intermediate value ($v$), a new random value is generated for each encryption operation, and the intermediate value is maintained ($v_m = vs. \perp m$). The masking technique is an efficient SCA countermeasure to randomize sensitive median values. For the intermediate value ($v$), a new random value is generated for each encryption operation, and the intermediate value is maintained as follows. Different masking techniques are applied depending on which operation is selected for $\perp$. Different masking logic is applied to the linear operation and the nonlinear operation of the encryption algorithm. In the case of a nonlinear operation, since an additional operation is required compared to a linear operation, speed and memory consumption may increase. In order to reduce the masking overhead, an efficient masking method is also being studied by reducing the number of nonlinear operations [26–29]. PIPO [30] is one of these ciphers.

## **3. Proposed Methods**

### *3.1. 8-Bit AVR Microcontroller*

The 8-bit AVR MCU is a RISC-structured 8-bit microcontroller manufactured by Atmel. It is a low power consumption chip suitable for an embedded environment, and 32 general-purpose registers are embedded inside. The number of instructions and addressing methods is large, and the design of 32 general-purpose registers and RISC structure helps to develop products, quickly, and is suitable for small systems for ubiquitous and sensor networks. In the AVR MCU, optimization studies are mainly performed using memory access minimization and pre-computed tables [17].

### *3.2. Implementation on 8-Bit AVR Microcontrollers*

- **log and anti-log LUTs:** To speed-up the computation of multiplication, there are various LUT methods that pre-calculate and store results of the computation in a table. The multiplication with *log* and *anti-log* tables utilized the property of *log* operations. The multiplication $a \times b$ for the two elements $a$ and $b$ of a finite field is calculated as $log^{-1}(log(a) + log(b))$. 8-bit Galois field multiplication of the FEA can be implemented by using 512 bytes of small memory.
- **Removing unnecessary multiplication operations:** The last step of the $F$ function is to remove the bit length of the block using the split function. No multiplication result of the removed bit is required. Therefore, it is possible to speed-up the computation during the encryption and decryption process without performing the multiplication operation of parts being removed. The technique is efficient in complex multiplication operations. When unnecessary multiplication operations were eliminated, clock cycle changes were measured to estimate the efficiency of the input length. To this end, we implement S-Box operations and multiplication operations in C language using LUT techniques and uses Atmel-Studio framework. The target processor is 8-bit ATmega1280 with `-O3` level of optimization option. Figure 3 is the result of the clock cycles, when the length of the plaintext $n$ is encryption in FEA-1/128 from 8 to 128 in 4-bit increments. When $n$ is 128, it takes 18,264 cycles. However, in the case of $n$ is 8, it takes 12,984 cycles, that shows 40% higher performance. Each time $n$ decreases by 16, it decreases by about 800 cycles. The length decreased, resulting in approximately 6% improvements in the computational speed. As a result, we confirm that faster operations are possible if unnecessary multiplication operations are removed during the computation process.

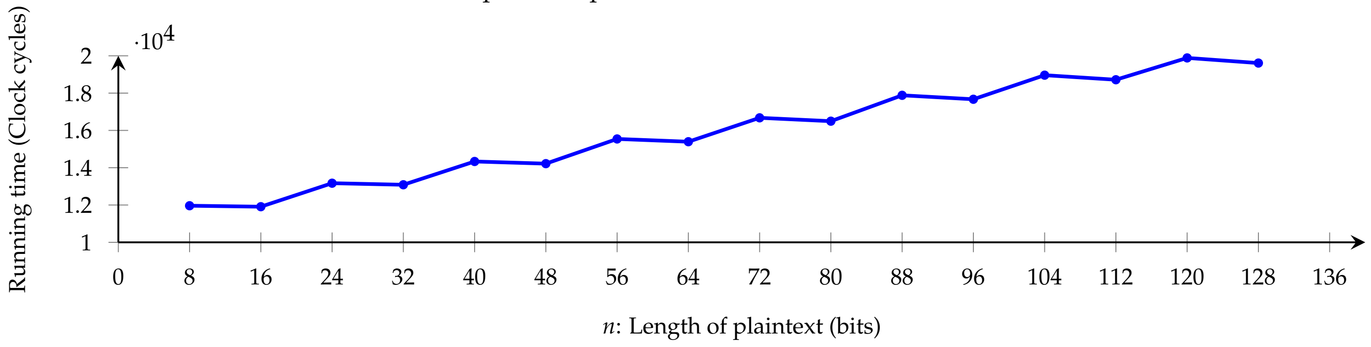

**Figure 3.** Differences of clock cycles depending on input $n$ with AVR processor.

### *3.3. 64-Bit ARMV Processor*

64-bit ARMv8-A processor supports three instruction sets (i.e., A32, T32, and A64). Both A32 and T32 instruction sets are used when executing the AArch32 execution state, and the A64 instruction set is used when executing the AArch64 execution state. In this paper, we cover the A64. There are 31 general-purpose 64-bit registers, and most commands support both 32-bit or 64-bit operands. Registers `W0` to `W30` are 32 bits and registers `X0` to `X30` are 64 bits and are mapped one-to-one to the lower bits of the large register. It provides a single instruction multiple data (SIMD) instruction set that can perform mathematical operations on multiple data streams in a parallel way. All 128 bits of NEON registers can be used to simultaneously operate $2 \times$ 64-bit, $4 \times$ 32-bit, $8 \times$ 16-bit, or $16 \times$ 8-bit integer data elements. In this paper, we perform the fast multiplication operation using `TBL`/`TBX` instructions.

### *3.4. Implementation on 64-Bit ARM Processor*

- **LUT based multiplication:** The LUT pre-computes the result of an operation for every pair of elements in a field, and stores the result in a table (See Table generations in Algorithms 1 and 2.). If the result of the multiplication operation is used as a precomputed table, the storage complexity of $GF(2n)$ is $(n/8) \times 2^{2n+1}$. Since it is the multiplication on $GF(2^8)$, $2^{17}$ bytes of storage space is required.
  However, the diffusion layer DL of FEA is performed by the product of a predefined matrix $M$. $M$ has elements `0x28, 0x1a, 0x7b, 0x78, 0xc3, 0xd0, 0x42,` and `0x40`. Therefore, the product of 8 elements of `M` is written as a dictionary table. When we create a table, including not only multiplication, but also the S-Box operation, which is the previous step of the multiplication operation in the `F` function. We can perform two operations by querying a table once by writing. S-box and multiplication operations can be performed simultaneously using 2 Kbytes of memory. This table lookup method can be efficiently implemented in high-performance embedded devices. We implemented it as shown in Listing 1 a using the NEON instruction set from ARMv8, which provides scalar/vector instructions and registers. Detailed instructions are given in Table 2. At the same time, it performs 8-byte unit table inquiry effectively by using `TBL` and `TBX` commands that allow table inquiry as described in Algorithm 3.

**Table 2.** Summary of instruction set for polynomial multiplication.

| asm | Operands | Description | Operation |
|---|---|---|---|
| `adr` | `Xd, Label` | Form PC-relative address | `Xd` ← `PC-relative address` |
| `sub` | `Vd.T, Vn.T, Vm.T` | Subtract | `Vd` ← `Vn-Tm` |
| `movi` | `Vd.T, #imm8` | Move Immediate 8-bit value | `Vd.T` ← `imm` |
| `ld1` | `Vt.T-Vt4.T, [Xn], imm` | Load multiple single-element structures 4 registers with immediate offset | `Vt-Vt4` ← `(Xd)` |
| `tbl` | `Vd.T, Vn.16b-Vn+3.16b, Vm.T` | Table vector Lookup | `Vd` ← `(Vn-Vn+3)[Vm]` |
| `tbx` | `Vd.T, Vn.16b-Vn+3.16b, Vm.T` | Table vector Lookup extension | `Vd` ← `(Vn-Vn+3)[Vm]` |

**Algorithm 1** $2^8(GF)$ anti-log table generation.

```
Require: The finite field $2^8(GF)$, generator polynomial P.
Output: The anti-log table.
1: anti-log$[0] = 0$
2: for i in range $(1, 256)$ do
3:     anti-log$[i]$ = anti-log$[i-1] << 1$
4:     if anti-log$[i-1]$ = 128 then
5:         anti-log$[i]$ = anti-log$[i] \oplus P$
```

**Algorithm 2** $2^8(GF)$ log table generation.

```
Require: antilog table.
Output: log table.
1: log$[0] = -1$
2: log$[1] = 0$
3: for i in range $(1, 256)$ do
4:     log[antilog[i]] = i
```

**Algorithm 3** tbl/tbx instructions based polynomial multiplication.

**Input:** intermediate result (`v13.8b`), s-box address (`.sboxNmul`)

**Output:** substituted result (`v15.8b`)

```
adr x28, .sboxNmul
movi v13.8b, #0x40
movi v14.8b, #0x0
sub v10.8b, v0.8b, v13.8b
sub v11.8b, v10.8b, v13.8b
sub v12.8b, v11.8b, v13.8b
ld1 v16.16b-v19.16b, [x28], #64
ld1 v20.16b-v23.16b, [x28], #64
ld1 v24.16b-v27.16b, [x28], #64
ld1 v28.16b-v31.16b, [x28], #64
tbl v15.8b, v16.16b-v19.16b, v0.8b
tbx v15.8b, v20.16b-v23.16b, v10.8b
tbx v15.8b, v24.16b-v27.16b, v11.8b
tbx v15.8b, v28.16b-v31.16b, v12.8b
return v15.8b
```

### 3.5. Side Channel Analysis on the FEA

In this chapter, CPA attack is actually performed to check the side channel vulnerability of the FEA. the goal of the attack is to obtain the master key of the FEA, and the attack point is the point after the S-Box operation of the `F` function, the part marked in red dots in Figure 4. The CPA attack on these attack points can obtain the round key calculated by XOR operation in the process prior to the S-Box operation. The first process of an attack uses known plaintext and round tweak values to attack the point immediately after the S-Box operation in the first round to obtain a 64-bit key. In the SBL operation, a full investigation of $2^8 \times 8$ is performed, because 8-bit S-Box is performed 8 times. The acquired round key values are used to calculate intermediate values for next round key values. Furthermore, the value of the 64-bit round key of the next round can be found by performing a CPA attack to the point after the subsequent S-Box operation. To obtain the master key value, we need to obtain four 64-bit round keys used in the first and second rounds. Thus, the attack collects two rounds of consumption power and performs four CPA attacks to obtain four 64-bit round keys. Algorithm 4 is the process of finding the master key value through 4 round key values.

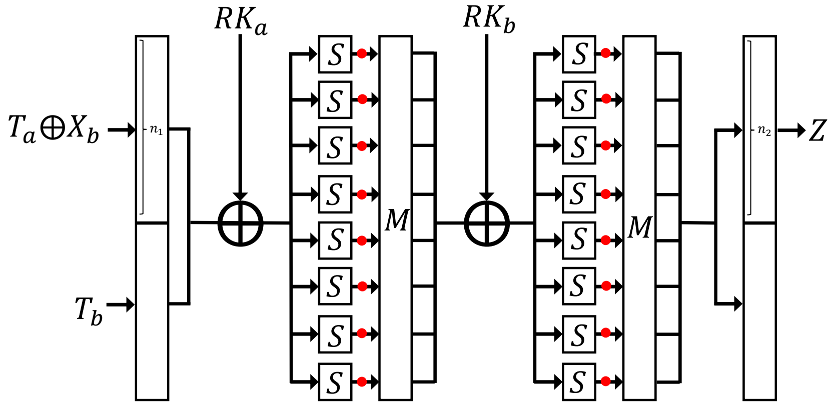

**Figure 4.** Attack points of CPA on FEA.

**Algorithm 4** Process of finding the master key value through 4 round key values.

```
Input: Round Key ka′, kb′, kb′, kd′
Output: Maskter Key mk = ka||kb||kc||kd
1: $x = DL(SBL(ka' \bigoplus kd' \bigoplus RC1))$
2: $y = DL(SBL(kb' \bigoplus kd'))$
3: $ka = ka' \bigoplus x \bigoplus y$
4: $kb = kb' \bigoplus y$
5: $kc = kd' \bigoplus x \bigoplus y$
6: $kd = kc' \bigoplus kd' \bigoplus y$
7: return mk = ka||kb||kc||kd
```

## 4. Proposed Masking Method

Masking techniques must be applied differently depending on the cryptographic structure, and the additional computation of masking technique affects the computational speed. Considering this point, we propose an appropriate and fast first-order masking technique in this paper. In the masking technique, a random mask value is generated as shown in Figure 5. Afterward, it added to the plaintext and round key. After the encryption, the masking technique is applied, the mask value is removed, and the cipher text is output.

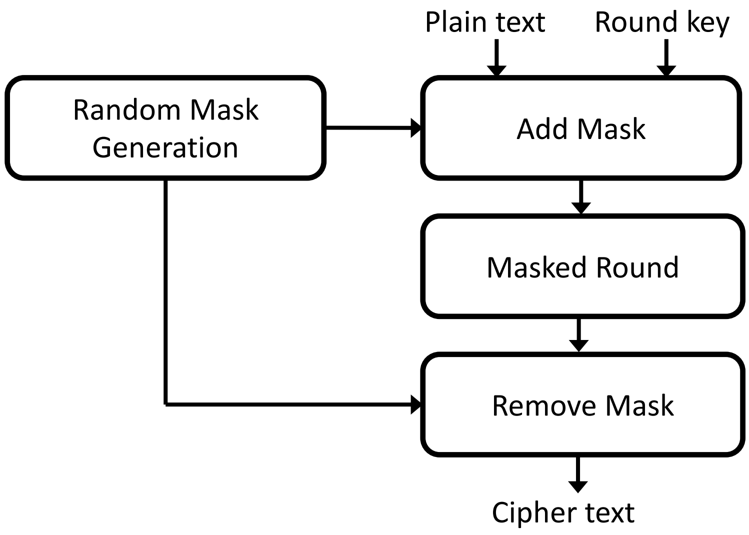

**Figure 5.** Masking step on FEA.

Figure 6 shows an encryption process with the proposed masking technique. In the round function, We designed to increase the efficiency of the implementation by masking the split plaintext $X_a$ and $X_b$. Masks of $M$ and $M'_i$ are maintained. Mask correction values are calculated in advance to maintain the mask value. First, 8-bit masks $M$, $M'$ and 8 masks $M_i$ ($M_1$, $M_2$, $M_3$, $M_4$, $M_5$, $M_6$, $M_7$, $M_8$) are created. In addition, several mask values are calculated to apply the mask. The mask value is not removed during the DL operation. One performs an ML operation on $M_i$ to generate $M'_i$ ($M'_1$, $M'_2$, $M'_3$, $M'_4$, $M'_5$, $M'_6$, $M'_7$, $M'_8$). The other is XOR operation the mask $M$ and the mask $M_i$ values to produce the resulting value $M''_i$ ($M''_1$, $M''_2$, $M''_3$, $M''_4$, $M''_5$, $M''_6$, $M''_7$, $M''_8$).

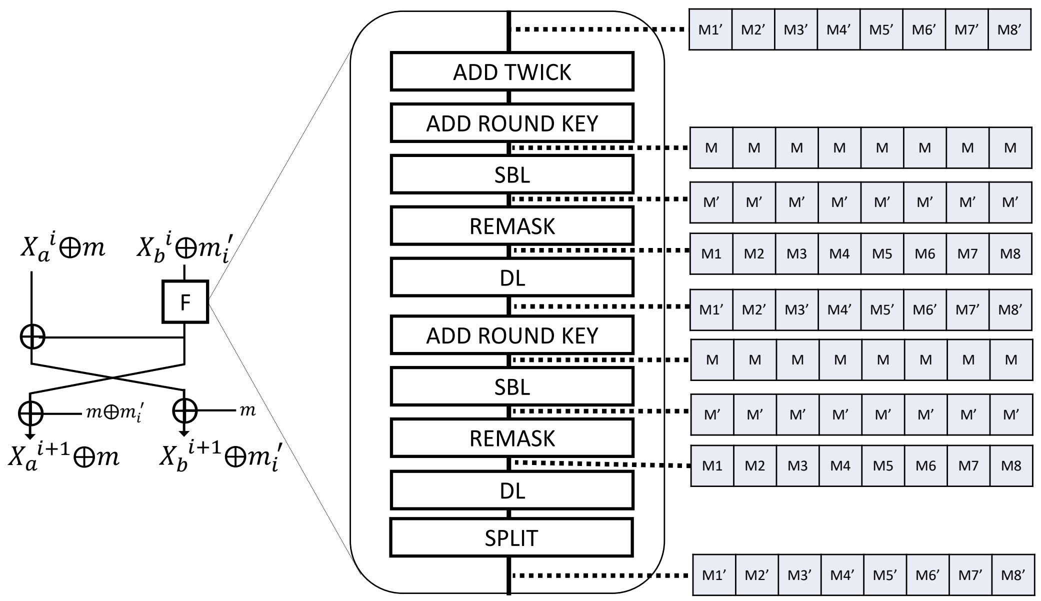

**Figure 6.** Proposed masking method.

The mask $M_i'$ is used to maintain the mask value after the ML operation, and the mask $M_i''$ is used to change the mask state to $M_i$ in the REMASK operation. Next, all round keys and $M'[0:7]$ perform XOR operations, and Mask $M$ performs XOR operations for each 8-bit block. The masked round key changes the mask state to $M$ when it is operated with the input value. We then generate an MS-Box table with masking values applied to the S-Box table. It is created as MS-Box[$i \oplus M$] = S-Box[$i$] $\oplus$ $M'$ and creates a new masking table, whenever the mask value changes.

The change of the mask state when performing the encryption is as follows. First, we perform an XOR operation of the input plaintext $X_a$, $X_b$ and each mask $M$, $M_i{}'$. This mask value is maintained during the round and is removed after the last round. The mask state of the block inputted from the `F` function is first calculated, resulting in the mask $M$ state. Next, in the SBL step, the mask state becomes $M'$ as an inquiry is performed in the MS-Box table. Thereafter, in the REMASK step, the XOR operation is performed on $M_i''$ to change the mask value from $M'$ to $M_i$. In addition, in the DL operation, the mask value becomes the same as the initial $M_i'$, and the mask state becomes the same as the first time. From the next step, the same operation is performed and the mask state is repeated. In the last step of the `F` function, the last splitting process, the mask value of the truncated bit is removed. Only, the bit is masked back to $M_i$. Thereafter, in order to keep the plaintext $X$ in the mask $M$ state and $X_b$ in the $M_i'$ state, the value of $M_i''$ is XORed in the block $X_a$, and $M$ is XORed in $X_b$.

The application of masking to the implementation of the SBL-multiplication table is shown in Figure 7. When the SBL operation and the multiplication operation are implemented as a table, a difference occurs in the masking scheme because the SBL and the multiplication process are combined. The masking application for SBL-multiplication table implementation is as follows. First, mask values used in the mask generation process are 8-bit mask $M$, $M'$ 64-bit mask $M_i$, and 8-bit $M''$ by XORing each byte of $M_i$. Finally, $M'''$ is created by XORing the masks $M$ and $M''$. Furthermore, the SBL multiplication table is masked like an MS-box using masks $M'$ and $M_i$. When the matrix multiplication is performed in the DL operation, these mask values are XORed with each other to become $M''$. Next, the round key $RK_a$ generated by the key schedule is XORed with the masks $M$ and $M'$. Then, the round key $RK_b$ generates a masked round key by XORing the masks $M$ and $M'''$. The change in mask state when performing encryption is shown in Figure 6. First, the input plaintext $X_a$ and $X_b$ are XORed with $M$. This mask value is retained as the round progresses and is removed after the last round. The mask state of the block input from the

F function becomes the mask $M$ state as the masked round key $RK_a$ is calculated first. Next, in the SBL-DL process, the multiplied value is XORed and the mask state becomes $M''$. Then, as the masked round key $RK_b$ is XORed, the mask state becomes $M$ again, and the same process is carried out. In the final split process, which is the last step of the F function, the mask value of the truncated bit is removed. Only the corresponding bit is masked again $M''$. Thereafter, in order to keep the plaintext $X_a$ and $X_b$ in the mask $M$ state, the value of $M'''$ is XORed in the block $X_a$, and $M''$ is XORed in $X_b$.

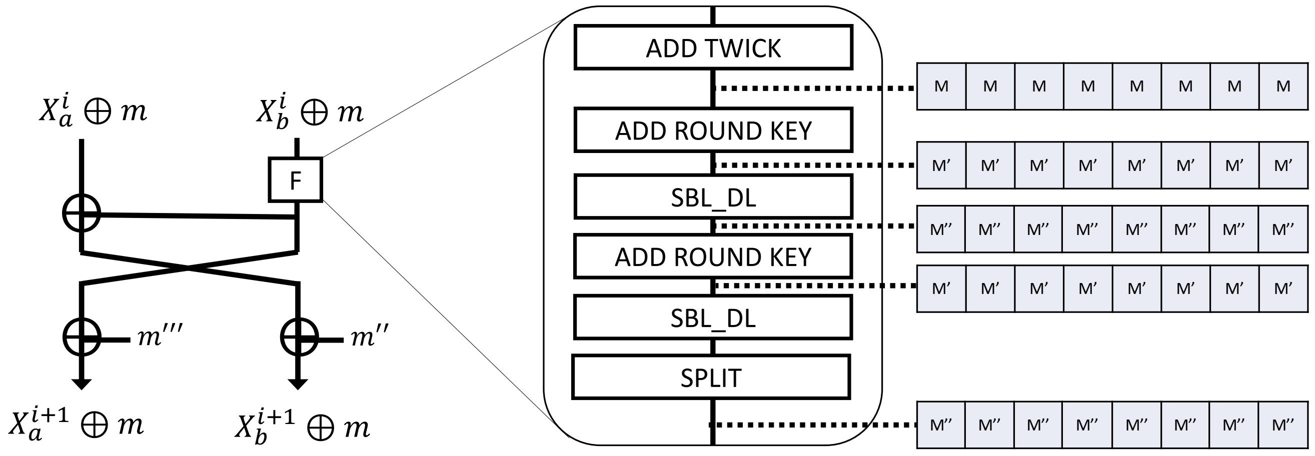

**Figure 7.** Masking implementation using SBL-multiplication table.

## 5. Evaluation

### *5.1. Result of CPA on the FEA*

The target is FEA-2 with a 128-bit key, and the input value is 128-bit. We collected the power-consumption of the microcontroller through ChipWhisperer-Lite i.e., 8-bit AVR processor (XMEGA). The sampling rate is 7.38 million samples per second (MS/s) and this is working frequency of target processor. The source code of microcontroller is written in C language and compiled with the AVR-GCC. We collected 1000 waveforms from the S-Box computation point in the first round of the FEA.

The attack performs each 8 bits 8 times, where the S-Box operation is performed. S-Box calculation results for all cases of the guessed secret keys 0 to 255 were converted into a Hamming Weight model, and the Pearson correlation coefficient between the modeled power consumption and the collected actual power consumption was calculated. After the measurement, $D$ traces($t$) are generated and each trace has $T$ data points. In $t_{d,j}$, $d$ is the trace index, $j$ is the data point of the trace ($d(1 \leq d \leq D, 0 \leq j < T)$). When there are different subkeys $i(1 \leq d \leq D, 0 \leq i < I)$., $h_{d,i}$ means the amount of power consumed when subkey $i$ in trace $d$. Based on these data, the correlation between t and h on the D trace can be found as follows.

$$r_{i,j} = \frac{\sum_{d=1}^{D}\left[\left(h_{d,i} - \overline{h_i}\right)\left(t_{d,j} - \overline{t_j}\right)\right]}{\sqrt{\sum_{d=1}^{D}\left(h_{d,i} - \overline{h_i}\right)^2 \sum_{d=1}^{D}\left(t_{d,j} - \overline{t_j}\right)^2}}$$

The top of Figure 8 shows CPA attack results. As a result of the attack, the highest correlation coefficient was shown at `0x8E, 0x37, 0x64, 0x48, 0xBC, 0x18, 0xC8,` and `0x91`. Round key values of round 1 used in the encryption were the same as `0x8E, 0x37, 0x64, 0x48, 0xBC, 0x18, 0xC8,` and `0x91`. This shows that the CPA was successfully performed. The bottom of Figure 8 is the result of the correlation test. We compute the correlation between model values and real-world waveforms for known round key values. The eight repetitive waveforms at the top are the S-Box calculation process, and it can be seen that the correlation coefficient is high for each waveform and the leakage occurs at the corresponding point where the S-Box calculation is performed. As a result, the FEA is vulnerable to power analysis attacks. It takes $2^{(128)}$ times to try all possible keys, but it takes $16 \times 2^{(8)} = 2^{(12)}$ times because it finds a key of 16 bytes by performing a CPA attack one byte at a time.

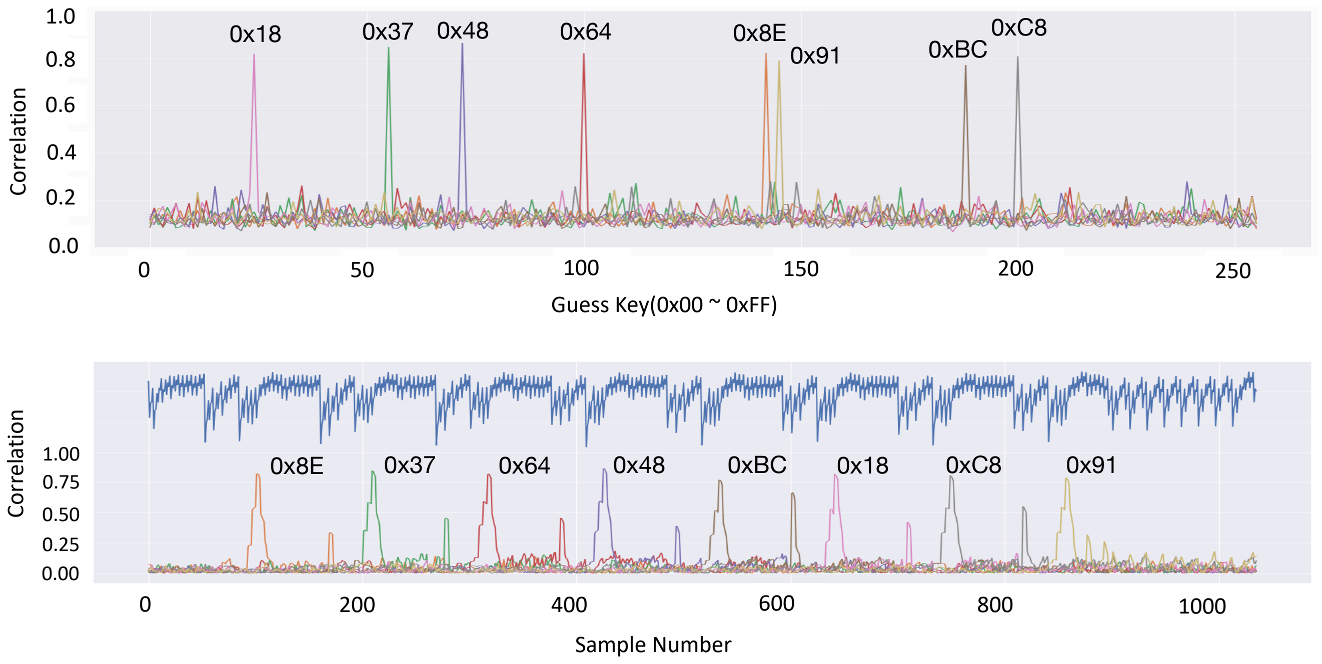

**Figure 8. Top**: correlation attack results, **bottom**: correlation test results.

## 5.2. Security and Efficiency of the Proposed Masking Technique

In this chapter, we verify the security and efficiency of the proposed masking technique. First, we check whether the proposed masking technique can safely defend against power analysis attacks. The CPA attack was successfully performed on the FEA and it is vulnerable to power analysis attacks. Based on this point, the security is confirmed by performing the same CPA attack on the proposed technique. The experimental environment of the CPA attack collected the power consumption of the microcontroller through the ChipWhisperer-Lite 8-bit processor (XMEGA). The sampling rate is 7.38 million samples per second (MS/s), and the source code of the microcontroller is written in C language and compiled with AVR-GCC. Attacking FEA-1/128 with plaintext length $n = 128$, collecting 1000 traces in the first round of the encryption process.

Second, we investigate the amount of computation and memory added by the proposed masking technique. Compare the difference between the clock cycle and the memory added to the masking table when the masking technique was applied to the FEA implementation and before. AtmegaStudio's simulator was used to investigate the computational amount and memory of AVR microcontrollers. The device is an 8-bit ATmega1280, implemented in C language, the optimization level is `-O2`.

The top part of Figure 9 shows the correlation coefficient of the set of guess keys by the CPA attack. The attack result in Section 3 showed a high correlation coefficient of 0.8 or more, but a correlation coefficient less than 0.18 in the masked implementation. The guess keys that showed the highest correlation coefficient were `0x67, 0x04, 0x09, 0x56, 0x91, 0x8D, 0xAB,` and `0x92`, which were not related to the actual round key values `0x8E, 0x37, 0x64, 0x48, 0xBC, 0x18, 0xC8,` and `0x91`. The middle of Figure 9 is the result of performing the correlation coefficient test in the first multiplication process with the matrix `M` at the bottom of Figure 9 in the SBL operation process. This is the result of calculating the correlation between the model value and the actual trace for the round key value known at each point. A low correlation coefficient was shown in all collected intervals. As a result, there is no correlation between the power and the median value of the data predicted for the correct key, which means that the proposed masking method is safe for the first-order CPA.

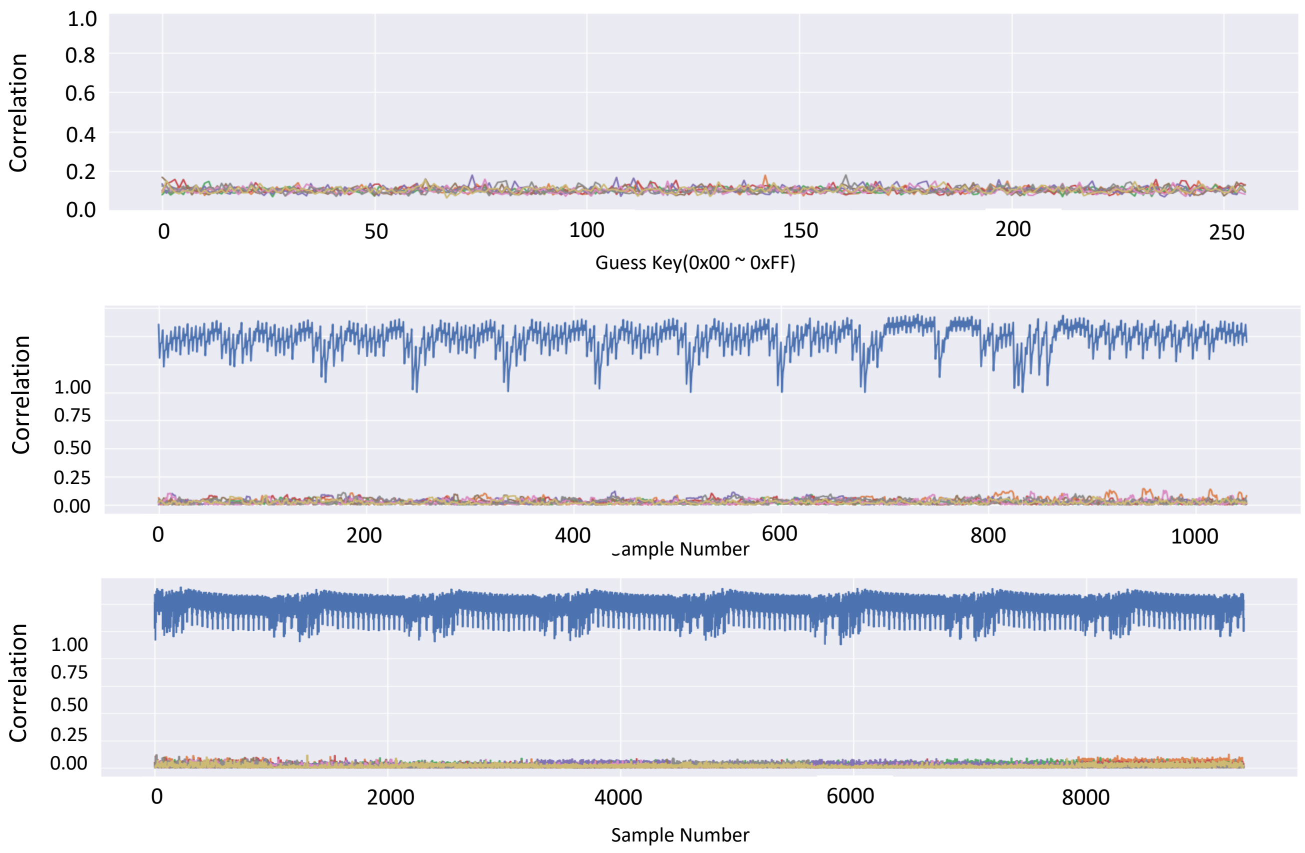

**Figure 9.** Top: correlation attack results, middle: correlation test results in multiplication, bottom: correlation test results in multiplication.

The difference between before and after the application of the proposed masking technique is shown in Table 3. Among the implementation techniques, when the proposed masking technique was applied to the $log/anti\text{-}log$ table, an additional calculation of 2866 cycles occurred, resulting in a small difference in calculation speed of about 3.77%. 512 bytes used for $log/anti\text{-}log$ table and 768 bytes added to 256 bytes of Masked S-BOX are used for the masking technique. In the lookup table where S-BOX and multiplication are combined, when the masking technique is applied, an additional operation of 725 cycles occurred, indicating a small difference in operation speed of about 2.53%. In addition, a total of 4 Kbytes is used due to the masked lookup table from 2 Kbytes used in the lookup table in which S-Box and multiplication are combined. The added operation in the two techniques was not large, and the two masking techniques showed a great difference in the size of the table used. As a result, it can be confirmed that the proposed technique was effectively applied.

Differences before and after the application of the proposed masking method of the FEA encryption implementation in ARMv8 is given in Table 4. It was written in the assembly language using Xcode and the execution timing was measured with Apple A13 Bionic (2.65 GHz). The difference before and after masking was 725 Cycles, resulting in a small calculation speed difference of about 2.36%. In addition, a total of 4 Kbytes is used due to the 2 Kbyte masked lookup table used in the lookup table in which S-Box and multiplication are combined. As a result, it can be confirmed that the proposed method was effectively applied.

**Table 3.** Performance comparison result on the AVR microcontroller.

| Metric | *log*/*anti-log* Table | | S-BOX/MUL Table | |
|---|---|---|---|---|
| | Before Masking | After Masking | Before Masking | After Masking |
| Execution Timing [clock cycles] | 75,889 | 78,755 | 28,621 | 29,346 |
| Table Size [bytes] | 512 | 768 | 2048 | 4096 |
| DPA Resistance | – | √ | – | √ |

**Table 4.** Performance comparison result on the ARM processor.

| Metric | S-BOX/MUL Table | |
|---|---|---|
| | Before Masking | After Masking |
| Execution Timing [clock cycles] | 28,621 | 29,346 |
| Table Size [bytes] | 2048 | 4096 |
| DPA Resistance | – | √ |

## 6. Conclusions

In this paper, we presented secure and efficient masked FEA implementations on low-end AVR processors (i.e., AVR) and high-end ARM processors (i.e., ARM64). In particular, the `F` function is re-designed to ensure the side channel resistance on target devices by utilizing the masked approach. To optimize the multiplication operation, *log* and *anti-log* tables are utilized. For the S-Box computation, the look-up table based approach is performed. The table based implementation uses huge memory. To reduce the memory consumption, the masking is implemented in two ways (one with table and the other one without table). The masked result shows safe because the median value did not appear in the correlation comparison. The proposed method is secure against power analysis attacks but the performance degradation of masked approach is only 2.53∼3.77% than the naïve FEA implementation. In the future work, the design of newly proposed ciphers will be considered as an efficient method for coping with side-channel attacks. Currently, NIST is standardizing a lightweight encryption algorithm that uses fewer resources through a lightweight encryption contest. We can apply our techniques to these standards.

**Author Contributions:** Data curation, H.K. (Hyunjun Kim); Investigation, M.S., K.J., H.K. (Hyeok-dong Kwon), and S.U.; Software, H.K. (Hyunjun Kim); Supervision, H.S.; Writing—original draft, H.K. (Hyunjun Kim); Writing—review and editing, H.K. (Hyunjun Kim) and H.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** 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, 90%) and this work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2020R1F1A1048478, 10%).

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

## References

1. Daemen, J.; Rijmen, V. *AES Proposal: Rijndael*. 1999 . Available online: https://www.cs.miami.edu/home/burt/learning/Csc688.012/rijndael/rijndael_doc_V2.pdf (accessed on 1 April 2021).
2. Dworkin, M. Recommendation for block cipher modes of operation: Methods for format preserving encryption. *NIST Spec. Publ.* **2016**, *800*, 38G.
3. Bellare, M.; Rogaway, P.; Spies, T. The FFX mode of operation for format-preserving encryption. *NIST Submiss.* **2010**, *20*, 19.
4. Brier, E.; Peyrin, T.; Stern, J. BPS: A Format-Preserving Encryption Proposal. 2010 . Available online: https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/bps/bps-spec.pdf (accessed on 1 April 2021).

5. Lee, J.K.; Koo, B.; Roh, D.; Kim, W.H.; Kwon, D. Format-preserving encryption algorithms using families of tweakable blockciphers. In *International Conference on Information Security and Cryptology*; Springer: Cham, Switzerland, 2014; pp. 132–159.
6. Baccarini, A.N.; Hayajneh, T. *Evolution of Format Preserving Encryption on IoT Devices-FF1+*. In Proceedings of the 52nd Hawaii International Conference on System Sciences, Grand Wailea, HI, USA, 8–11 January 2019; pp. 1628–1637.
7. Lenk, A.; Marcus, P.; Povoa, I. Format Preserving Encryption of Geospatial Data for the Internet of Things. In Proceedings of the 2018 IEEE International Congress on Internet of Things (ICIOT), San Francisco, CA, USA, 2–7 July 2018.
8. Jang, W.; Lee, S.Y. Partial image encryption using format-preserving encryption in image processing systems for Internet of things environment. *Int. J. Distrib. Sens. Netw.* **2020**, *16*, 1550147720914779. [CrossRef]
9. Kasper, T.; Oswald, D.; Paar, C. Sweet Dreams and Nightmares: Security in the Internet of Things. In *Information Security Theory and Practice. Securing the Internet of Things (WISTP)*; IEEE: Heraklion, Greece, 2014; pp. 1–9.
10. Chari, S.; Jutla, C.S.; Rao, J.R.; Rohatgi, P. Towards sound approaches to counteract power-analysis attacks. In *Annual International Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 398–412.
11. Chen, Z.; Zhou, Y. Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In *Cryptographic Hardware and Embedded Systems—CHES 2006*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 242–254.
12. Nassar, M.; Souissi, Y.; Guilley, S.; Danger, J.L. RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 12–16 March 2012; pp. 1173–1178.
13. Popp, T.; Mangard, S.; Oswald, E. Power Analysis Attacks and Countermeasures. *IEEE Des. Test Comput.* **2007**, *24*, 535–543. [CrossRef]
14. Rivain, M.; Prouff, E. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010*; Springer: Berlin/Heidelberg, Germany, 2010.
15. Lai, X.; Massey, J.L. A proposal for a new block encryption standard. In *Workshop on the Theory and Application of of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1990; pp. 389–404.
16. Park, C.; Jeong, S.; Hong, D.; Seo, C. Optimal Implementation of Format Preserving Encryption Algorithm FEA in Various Environments. *J. Korea Inst. Inf. Secur. Cryptol.* **2018**, *28*, 41–51.
17. Lim, J.H.; Kwon, H.D.; Woo, J.M.; An, K.H.; Kim, D.Y.; Seo, H.J. Utilization and Optimized Implementation of Format Preserving Encryption Algorithm for IoT and BLE Communications. *J. Korea Inst. Inf. Secur. Cryptol.* **2018**, *28*, 1371–1378.
18. National Institute of Standards and Technology. Data Encryption Standard (DES). FIPS Publication 46-3, October 25, 1999. Available online: https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub46-2.pdf (accessed on 1 April 2021).
19. Bogdanov, A.; Knudsen, L.R.; Leander, G.; Paar, C.; Poschmann, A.; Robshaw, M.J.B.; Seurin, Y.; Vikkelsoe, C. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems—CHES 2007*; Springer: Berlin/Heidelberg, Germany, 2007.
20. Beaulieu, R.; Treatman-Clark, S.; Shors, D.; Bryan, W.; 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.
21. Guo, J.; Peyrin, T.; Poschmann, A.; Matt, R. The LED Block Cipher. In *Cryptographic Hardware and Embedded Systems—CHES 2011*; Springer: Berlin/Heidelberg, Germany, 2011.
22. Goubin, L.; Patarin, J. DES and Differential Power Analysis The "Duplication" Method. In *Cryptographic Hardware and Embedded Systems, CHES 1999*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 158–172.
23. Bellizia, D.; Scotti, G.; Trifiletti, A. Implementation of the PRESENT-80 block cipher and analysis of its vulnerability to Side Channel Attacks Exploiting Static Power. In Proceedings of the 2016 MIXDES—23rd International Conference Mixed Design of Integrated Circuits and Systems, Lodz, Poland, 23–25 June 2016.
24. Shanmugam, D.; Selvam, R.; Annadurai, S. Differential Power Analysis Attack on SIMON and LED Block Ciphers. In Proceedings of the SPACE 2014: Security, Privacy, and Applied Cryptography Engineering, Pune, India, 18–22 October 2014.
25. Prouff, E. DPA attacks and S-boxes. In *International Workshop on Fast Software Encryption*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 424–441.
26. Gérard, B.; Grosso, V.; Naya-Plasencia, M.; Standaert, F.X. Block Ciphers that are Easier to Mask: How Far Can we Go? In *International Conference on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 383–399. [CrossRef]
27. Grosso, V.; Leurent, G.; Standaert, F.X.; Varıcı, K. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In *International Workshop on Fast Software Encryption*; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8540. [CrossRef]
28. Canteaut, A.; Duval, S.; Leurent, G. Construction of Lightweight S-Boxes using Feistel and MISTY structures. In *International Conference on Selected Areas in Cryptography*; Springer: Cham, Switerzerland, 2015; Volume 9566. [CrossRef]
29. Boss, E.; Grosso, V.; Güneysu, T.; Leander, G.; Moradi, A.; Schneider, T. Strong 8-bit Sboxes with Efficient Masking in Hardware. In *International Conference on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9813, pp. 171–193. [CrossRef]
30. Kim, H.; Jeon, Y.; Kim, G.; Kim, J.; Sim, B.Y.; Han, D.G.; Seo, H.; Kim, S.; Hong, S.; Sung, J.; Hong, D. PIPO: A Lightweight Block Cipher with Efficient Higher-Order Masking Software Implementations. In *International Conference on Information Security and Cryptology*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 99–122. [CrossRef]