Conditional Re-encoding Method for Cryptanalysis-Resistant White-Box AES

Seungkwang Lee, Dooho Choi, and Yong-Je Choi

Conventional cryptographic algorithms are not sufficient to protect secret keys and data in white-box environments, where an attacker has full visibility and control over an executing software code. For this reason, cryptographic algorithms have been redesigned to be resistant to white-box attacks. The first white-box AES (WB-AES) implementation was thought to provide reliable security in that all brute force attacks are infeasible even in white-box environments; however, this proved not to be the case. In particular, Billet and others presented a cryptanalysis of WB-AES with 230 time complexity, and Michiels and others generalized it for all substitution-linear transformation ciphers. Recently, a collision-based cryptanalysis was also reported. In this paper, we revisit Chow and others's first WB-AES implementation and present a conditional re-encoding method for cryptanalysis protection. The experimental results show that there is approximately a 57% increase in the memory requirement and a 20% increase in execution speed.

Keywords: White-box cryptography, cryptanalysis, countermeasure, AES.

I. Introduction

Data encryption is, without a doubt, one of the best ways to protect information stored in a digital device. In most cases, cryptographic algorithms are open to all for review and the secrecy aspect is maintained through the use of secret keys. If an attacker needs to perform an impractically large number of mathematical tests to reveal a secret key, then it indicates that the cryptographic algorithms are computationally secure. In the case of the Advanced Encryption Standard (AES), there are 3.4×10^{38} possible key combinations with respect to a 128-bit key size. Cracking the key using a supercomputer would require an average of 1.02×10^{18} years of testing at a rate of 10.51×10^{15} floating point operations per second. For this reason, if the secret key is kept secure, then it is also assumed to be secure.

Most cryptographic algorithms are designed to protect a secret key against what is known as a "black-box attack." In a black-box attack model, an attacker has knowledge of the cryptographic algorithm and is able to examine various inputs and outputs, but is unable to monitor its execution or any intermediate results generated during the computation. Based on these conditions, a black-box attacker can mount various types of attacks including known-plaintext, ciphertext-only, chosen-ciphertext, and adaptive chosen-plaintext attacks.

While these attacks seem to be powerful enough in a black-box attack model, the physical properties of cryptosystems mean software is vulnerable to advanced attacks. Often, if an attacker has sufficient access to a target cipher to mount adaptive attacks, then they may obtain some useful information about the execution of an algorithm and its secret key. For example, side-channel attacks, such as differential power analysis (DPA) [1] or correlation power analysis (CPA) [2],

Manuscript received Jan. 27, 2014; revised June 25, 2015; accepted Aug. 7, 2015.

This work was supported by the K-SCARF project, the ICT R&D program of ETRI (Research on Key Leakage Analysis and Response Technologies).

Seungkwang Lee (skwang@etri.re.kr) and Yong-Je Choi (choiyj@etri.re.kr) are with the SW & Contents Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Dooho Choi (corresponding author, dhchoi@ etri.re.kr) is with the SW & Contents Research Laboratory, ETRI, and with the University of Science & Technology (UST), Daejeon, Rep. of Korea.

enable an attacker to crack smart cards protected with cryptographic algorithms in a matter of minutes rather than the theoretical billion years of testing. This type of attack exploits the fact that the power consumption of a cryptographic device at any given point in time strongly depends on the data it processes and the operation it performs. Provided that enough power traces are given, an attacker can perform a statistical analysis and discover a secret key very quickly [3]. Another type of side-channel attack is realized through maliciously injecting faults into a cryptographic device and observing the corresponding erroneous outputs; this is a so-called fault injection attack. In some extreme cases such as the RSA algorithm using the Chinese Remainder Theorem (CRT-RSA), even a single fault injection enables an attacker to reveal a secret key [3]–[4].

Side-channel attacks, however, are characterized as a "grey-box attack" because an attacker has access to a small part of the execution. From this standpoint, the countermeasures against a grey-box attack do not fully reflect reality. A much more powerful type of attack is a "white-box attack." In this attack model, an attacker has total visibility into an execution platform and the software implementation of an algorithm. This means that an attacker can analyze and tamper with the binary code and corresponding memory page of the application during execution. To this end, an attacker may use an assistant attack tool such as a disassembler or debugger. Unfortunately, most cryptographic algorithms, including those for digital signatures, authentication, smart cards, and the like, are vulnerable to white-box attacks. There is therefore a need for white-box cryptography in any software protection strategy. The protection of AES against a white-box attack (WB-AES) was started in [5] with the addition of cryptanalysis for this technique [6], and continuous research on WB-AES has followed [7]–[10].

Overall, each WB-AES algorithm has a corresponding cryptanalysis of its weakness. There is therefore a need for a secure WB-AES algorithm that is resistant to cryptanalysis.

In this paper, we revisit the vulnerabilities of Chow's WB-AES [5] and improve it in such a way so as to protect against a white-box attack, with low additional costs. Our solution changes a portion of the encoding method in addition to the construction of a specific type of lookup table. This new encoding method is the key idea behind the protection of previous white-box attacks. Compared to Chow's WB-AES, the additional costs are 1.57-times the size of the lookup tables, and 1.2-times the runtime with an Intel Core i7 at 3.4 GHz.

The rest of this paper is organized as follows. Section II briefly reviews the WB-AES algorithm proposed by Chow and others. In Section III, we describe the previous cryptanalysis. We then propose our WB-AES algorithm, which is resistant to cryptanalysis, and analyze its security and performance in Section IV. Finally, we provide some concluding remarks in Section V.

II. White-Box AES Implementation

In this section, we review the WB-AES algorithm proposed by Chow and others [4] with respect to a 128-bit key size, which can be extended to a 192- or 256-bit key size. In WB-AES, AddRoundKey, SubBytes, and part of the MixColumns are combined into a series of table lookups. To this end, the secret key is first integrated into an S-box. The MixColumns step is then combined to a key-customized S-box. For the details, we start with the description of a conventional AES-128 encryption [11] and its variation. AES-128 encryption can generally be described as follows:

state ← *plaintext*
AddRoundKey(state, k^0)
for $r = 1$ to 9
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, k^r)
SubBytes(state)
ShiftRows(state)
AddRoundKey(state, k^{10})
ciphertext ← state

Here, we know the following [5], [12]:

1) It is possible to put AddRoundKey(state, k^0) into the for-loop while removing AddRound(state, k^9).
2) SubBytes followed by ShiftRows produces the same result as ShiftRows followed by SubBytes.
3) AddRoundKey(state, k^{r-1}) followed by ShiftRows(state) produces the same result as ShiftRows(state) followed by AddRoundKey(state, $\hat{k}^{r-1}$), where $\hat{k}^{r-1}$ is the result of applying ShiftRows to k^{r-1}.

These observations give us the following description by which AES is converted into a table-based implementation.

state ← *plaintext*
for $r = 1$ to 9
    ShiftRows(state)
    AddRoundKey(state, $\hat{k}^{r-1}$)
    SubBytes(state)
    MixColumns(state)
ShiftRows(state)
AddRoundKey(state, $\hat{k}^9$)
SubBytes(state)
AddRoundKey(state, k^{10})

ciphertext ← state

1. AddRoundKey and SubBytes

For the key-customized instances of AES-128, AddRoundKey and SubBytes are first combined into the so-called T-boxes — a series of 160 lookup tables that map bytes to bytes. Theses T-boxes are defined as follows:

$T^r_{i,j}(x) = S(x \oplus \hat{k}^{r-1}_{i,j})$ for $i = 0, \ldots, 3, j = 0, \ldots, 3, r = 1, \ldots, 9$,

$T^{10}_{i,j}(x) = S(x \oplus \hat{k}^{9}_{i,j}) \oplus k^{10}_{i,j}$ for $i = 0, \ldots, 3, j = 0, \ldots, 3$.

Note that $S(x)$ is an S-box with an 8-bit input value, x, and T-boxes for round 10 have to absorb two round keys, $\hat{k}^9$ and k^{10}, where $\hat{k}^r_{i,j} = k^r_{i,(j+i) \bmod 4}$.

2. T-Box and MixColumns

After each byte in the state is mapped through a T-box, multiplication of a 32-bit vector by a MixColumns matrix (MC) is performed. To reduce the total size of lookup tables, these multiplications are divided as four separate multiplications of an 8-bit vector by subdividing MC. Let x_0, x_1, x_2, and x_3 be four bytes to be multiplied by MC. The multiplication can be decomposed as follows:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= x_0 \times \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus x_1 \times \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus x_2 \times \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus x_3 \times \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix}.$$

Let y_0, y_1, y_2, and y_3 denote the four terms on the right-hand side. The so-called Ty_i tables that map 8-bits to 32-bits are defined as follows:

$$Ty_0(x) = x \times [02\ 01\ 01\ 03]^T,$$
$$Ty_1(x) = x \times [03\ 02\ 01\ 01]^T,$$
$$Ty_2(x) = x \times [01\ 03\ 02\ 01]^T,$$
$$Ty_3(x) = x \times [01\ 01\ 03\ 02]^T.$$

Then, multiplication of the four bytes x_0, x_1, x_2, and x_3 by MC can be computed by four table lookups and three XORs: $Ty_0(x) \oplus Ty_1(x) \oplus Ty_2(x) \oplus Ty_3(x)$. In rounds 1 to 9, 144 Ty_i tables are needed to accept the 8-bit outputs of T-boxes.

3. Encodings

Since an attacker can access the lookup tables from round 1, they can easily reveal the secret key. There is therefore a need for input and output encodings to obfuscate all lookup tables. There are two types of encodings — non-linear encodings[1] on its input and output, and affine mixing bijections applied to all key-dependent lookup tables. The input and output encodings achieve *confusion*, as defined by Shannon [13], while mixing bijections achieve *diffusion*. In the Type II tables, 8 × 8 mixing bijections are used to diffuse the input to T-boxes, and 32 × 32 mixing bijections, MB, are inserted after MixColumns. A trapezoid that is labeled "32 × 8" in Type II shown in Fig. 1 indicates a result of applying the mixing bijections to the Ty_i tables. The Type III tables combine MB^{-1} with the inverse of the input 8 × 8 mixing bijections for T-boxes of the next round. To reduce the size of lookup tables, MB^{-1} is also divided into four submatrices, MB_i^{-1}. Another trapezoid that is labeled "32 × 8" in Type III depicted in Fig. 1 means a combination of MB_i^{-1} and the inverse of the 8 × 8 mixing bijections [5].

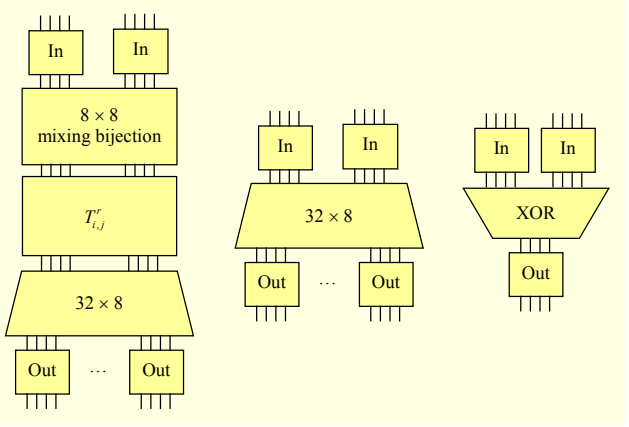

Fig. 1. Type II, III, and IV tables (from left).

4. XOR

The lookup values from Type II and Type III tables should be XOR-ed so that multiplications become complete. The Type IV tables define XOR operations that take in four bits from each of two previous computations and map them to their XOR; $XOR(x, y) = x \oplus y$.

There are two variations of Type IV in WB-AES — one to XOR (and combine) 4-bit chunks for a 32-bit result of Type II and Type III tables, and the other for a 128-bit result of Type IA and Type IB tables. Each 4-bit input has to be decoded and the 4-bit output has to be encoded non-linearly. The XOR operation of 4-bit chunks using Type IV tables is given below for illustration purposes [14]:

- $E1(x_0), \ldots, E1(x_3)$: Encoded 4-bit inputs to Type IV table.
- InvE1: Input decoding table.
- E2: Output encoding table.

1) The input decoding and the output encoding are performed in 4-bit unit with a concatenated form to avoid large tables.

ETRI Journal, Volume 37, Number 5, October 2015
http://dx.doi.org/10.4218/etrij.15.0114.0025

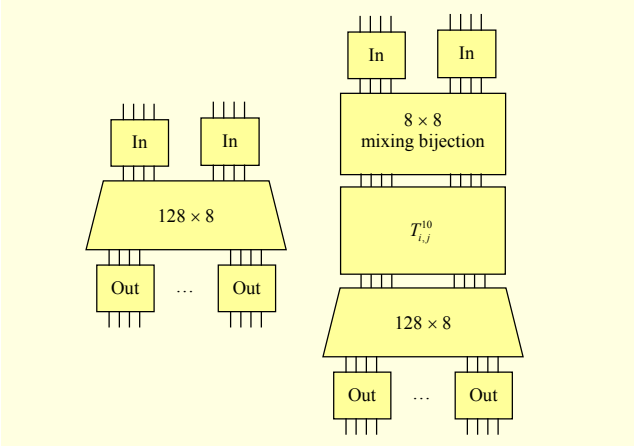

Fig. 2. Type IA and IB tables.

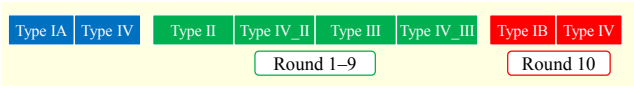

Fig. 3. Sequence of table lookups.

▪ InvE2: Output decoding table, used in the next stage.

The resulting value of E2($x_0 \oplus x_1 \oplus x_2 \oplus x_3$) can be computed in the following way:

$$\text{pair2} = \text{E2}(\text{InvE1}(\text{E1}(x_2)) \oplus \text{InvE1}(\text{E1}(x_3))),$$
$$\text{pair1} = \text{E2}(\text{InvE1}(\text{E1}(x_1)) \oplus \text{InvE2}(\text{pair2})),$$
$$\text{pair0} = \text{E2}(\text{InvE1}(\text{E1}(x_0)) \oplus \text{InvE2}(\text{pair1})).$$

Here, pair0 has a value of E2($x_0 \oplus x_1 \oplus x_2 \oplus x_3$). Type II, III, and IV tables are depicted in Fig. 1.

5. External Encodings

External input and output encodings are composed of two sets of sixteen 8-bit to 128-bit lookup tables. Each external input encoding table (Type IA) contains one 128 × 8 vertical section of a 128 × 128 matrix (that is, the composition of the inverse of the initial encoding prior to round 1 and the concatenation of the input mixing bijections for the inverse of $T^1_{i,j}$) surrounded by 4-bit input and output encodings. Each external output encoding table (Type IB), on the other hand, includes the composition of the mixing bijection combined with $T^{10}_{i,j}$, and a 128 × 8 vertical section of a 128 × 128 matrix for the final encoding, surrounded by 4-bit input and output encodings. These result in the lookup tables depicted in Fig. 2. Consequently, all of the tables described thus far are looked up in the order shown in Fig. 3.

For later use, we distinguish Type IV following Type II and Type III tables as Type IV_II and Type IV_III tables, respectively. For compatibility with an original AES, the input to Type IA should be first encoded by the corresponding linear and non-linear transformations. On the other hand, the output of round 10 should be decoded by the corresponding non-linear and linear transformations.

All encoding tables have to be deleted after generating the lookup tables. All we need is a set of lookup tables for encryption/decryption.

III. Cryptanalysis of WB-AES

In this section, we briefly introduce three methods of cryptanalysis of WB-AES and point out a common starting point between them. As explained in the previous section, Chow and others' WB-AES obfuscates the lookup tables using input/output encodings and mixing bijections. However, linear mixing bijections along with output encodings and the corresponding input decodings along with linear mixing bijections cancel out at the boundary between two consecutive rounds. Billet and others' cryptanalysis [6] starts with this fact. We first review Billet and others' cryptanalysis and then Michiels and others' generalized cryptanalysis of white-box ciphers [15]. In addition, we review a collision-based attack introduced by Lepoint and others [10]. Our purpose is to find a common starting point of cryptanalysis, and we thus propose a countermeasure in which the starting point is not applied.

1. Billet and Others' Cryptanalysis

In [6], the cryptanalysis begins with the conceptualized R^r_j boxes depicted in Fig. 4. Each R^r_j box consists of four 8-bit to 8-bit input permutations $P^r_{i,j}$ (and the respective output permutations $Q^r_{i,j}$), made up of the composition of two concatenated 4-bit to 4-bit input (and the respective output) encodings, and one 8-bit to 8-bit mixing bijection. Each $Q^r_{i,j}$ is the inverse of $P^{r+1}_{i,j}$, where $r \in [0, 9]$. Using this R^r_j representation, the cryptanalysis defines a function y_i of (x_0, x_1, x_2, x_3) as follows:

$$y_i(x_0, x_1, x_2, x_3) = Q^r_i\left(\alpha_{i,0}T^r_0\left(P^r_0(x_0)\right) \oplus \alpha_{i,1}T^r_1\left(P^r_1(x_1)\right) \oplus \alpha_{i,2}T^r_2\left(P^r_2(x_2)\right) \oplus \alpha_{i,3}T^r_3\left(P^r_3(x_3)\right)\right), \quad (1)$$

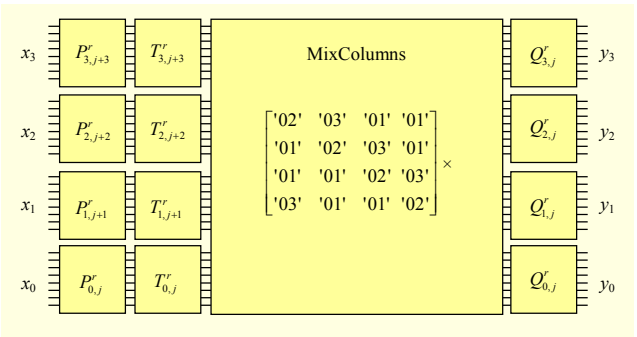

Fig. 4. One of four R^r_j mappings, $j = 0, \ldots, 3$ [6].

where α_{ij} are the coefficients of MixColumns. If x_1, x_2, and x_3 are fixed to certain constants, say c_1, c_2, and c_3, respectively, then we have

$$y_0(x, c_1, c_2, c_3) = Q^r_{0,j}\left(\alpha T^r_{0,j}\left(P^r_{0,j}(x)\right) \oplus \beta_c\right). \quad (2)$$

Since x only takes 256 values and constant β_c is fixed, these mappings are known by the input/output, as well as by their inverses. Changing one of the constants, c_1, c_2, or c_3, for example, c_1, c_2, and c_3', produces the following function as a lookup table:

$$\begin{aligned} & y_0(x, c_1, c_2, c_3) \circ y_0(x, c_1, c_2, c_3')^{-1} \\ &= Q^r_{0,j}\left(\alpha T^r_{0,j}\left(P^r_{0,j}{}^{-1}\left(T^r_{0,j}{}^{-1}\left(\alpha^{-1}(Q^r_{0,j}{}^{-1} \oplus \beta_{c'})\right)\right)\right) \oplus \beta_c\right) \\ &= Q^r_{0,j}\left((Q^r_{0,j}{}^{-1} \oplus \beta_{c'}) \oplus \beta_c\right), \\ &= Q^r_{0,j}(Q^r_{0,j}{}^{-1} \oplus \beta), \end{aligned} \quad (3)$$

where $\beta = \beta_c \oplus \beta_c'$ takes all values in GF(2^8). Theorem 1 (below) then enables an attacker to recover the non-linear part $\tilde{Q}^r_{i,j}$ of $Q^r_{i,j}$ such that $\tilde{Q}^r_{i,j}{}^{-1} \circ Q^r_{i,j}$ is an affine mapping, $A^r_{i,j}{}^{-1}$, for any round $r \in [1, 9]$ with time complexity 2^{24}. Because there is no MixColumns step, the last round is excluded from the cryptanalysis. The open dot " $\circ$ " means a composition of functions.

Theorem 1. Given a set of functions $S = \{Q \circ \oplus_\beta \circ Q^{-1}\}_\beta$, where $\beta \in$ GF(2^8), Q is a permutation of GF(2^8), and $\oplus_\beta$ is the translation by β in GF(2^8), one can construct a particular solution, $\tilde{Q}$, such that there exists an affine mapping, A, so that $\tilde{Q} = Q \circ A$.

The next step is to recover the affine maps $Q^r_0 = A_0 \oplus q_0$, where A_0 is linear and q_0 is a constant. To this end, (1) is used to determine a unique mapping, L, and a constant, c, such that for all $x_0 \in$ GF(2^8) we have $y_i(x_0, 0, 0, 0) = L(y_j(x_0, 0, 0, 0)) \oplus c$.

The time complexity to solve this is much lower than 2^{16} with a highly over-defined linear system of $2^8 \times 8$ equations, involving the 64 entries of L as well as the eight entries of c as unknowns over GF(2), using the knowledge of the functions y_i and y_j based on their values.

It is then possible to recover A_0, the linear part of Q_0, with time complexity 2^{24} by determining the characteristic polynomial of L. At the same time, the constant part q_0 of Q_0 is also computed using the knowledge of α_{ij} by setting the four variables in (1) to zero. It was shown that the linear parts of Q_1, Q_2, and Q_3 can be directly computed from the knowledge of the linear part of Q_0 with time complexity 2^{16}. In a similar way, all Q^r_i of a round can be computed. At the same time, P^{r+1}_i is computed because we know Q^r_i is the inverse of P^{r+1}_i, as stated previously. After the mappings are recovered, the round key bytes embedded in the AES-128 white-box implementation can be retrieved. Even though they are not necessarily in the right order, they can be rearranged owing to the constraint in the key derivation algorithm of AES-128. In conclusion, the total complexity of this cryptanalysis is bounded by 2^{30}. In [16], Tolhuizen improved it to 2^{22} using a preprocessing step.

2. Michiels and Others' Cryptanalysis

Michiels and others generalized Billet and others' strategy against substitution-linear transformation (SLT) ciphers [15]. Their generalization is composed of the following three steps:

1) Removing the non-linear part of the mixing bijection encodings using Theorem 1
2) Removing the linear part of the encodings by solving a linear equivalence problem [17]–[18]
3) Extracting the secret key information by solving a matrix equivalence problem

To be more specific, a generic substitution-affine transformation (SAT) cipher is defined in the second step, where a round consists of T-boxes T^r_i, followed by an invertible affine function ϵ^r. Because MixColumns, the affine function of AES, are known to an attacker, T^r_i and ϵ^r can be computed. Biryukov and others' algorithm [17] is used to solve the equivalence problem $T_i = \gamma_i \circ S_i \circ \delta_i$, where γ and δ are the affine functions that describe the affine relation between T_i and S_i. Since T_i is a part of the SAT cipher, it contains the key k^r. Note that this cryptanalysis also begins with Theorem 1 to remove the non-linear part.

3. Lepoint and Others' Cryptanalysis

A collision-based attack [10] was recently introduced in SAC2013. As shown in Fig. 5, this cryptanalysis finds collisions to recover functions S_0, S_1, S_2, and S_3 and associated key bytes using the equation

$$\begin{aligned} & Q_0(02 \otimes S_0(\alpha) \oplus 03 \otimes S_1(0) \oplus c) \\ &= Q_0(02 \otimes S_0(0) \oplus 03 \otimes S_1(\beta) \oplus c). \end{aligned} \quad (4)$$

As they mentioned, there are 256 pairs (α, β) with a trivial solution (0, 0). The attacker constructs a linear system to recover S_0 and S_1. After S_0 is recovered, S_1 can be recovered through an exhaustive search. The remaining functions, S_2 and S_3, are then recovered similarly by solving the linear systems. Once the S_i functions have been recovered, one can easily recover Q_i in the output of the first round. The total time complexity of the attack is 2^{22}.

For the cryptanalysis above, we can see through Theorem 1 that the first two methods of cryptanalysis used and the collision equation of the last method are both dependent on Q. More precisely, their assertion assumes that for $x, y \in$ GF(2^8), if $x = y$, then $Q(x) = Q(y)$; and if $x \neq y$, then $Q(x) \neq Q(y)$.

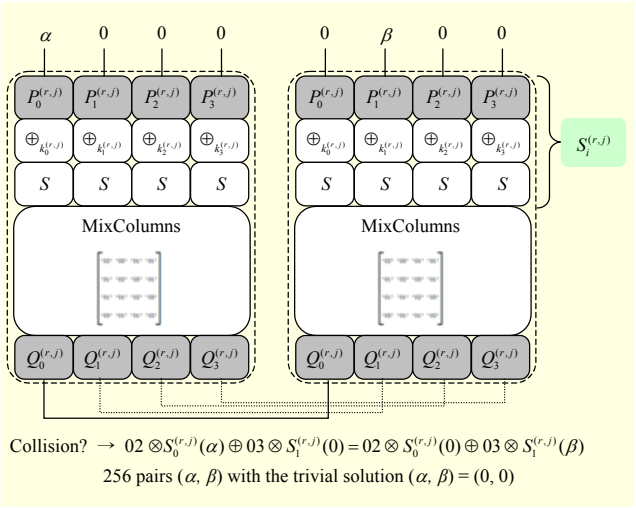

Fig. 5. Lepoint and others' attack [10].

What is important here is that intermediate values are combined by XORs, and the non-linear encoding is then followed in the 4-bit unit. In the following section, based on this fact, we propose a conditional re-encoding method and show that it can protect against three methods of cryptanalysis.

IV. Proposed Encoding Method

The key idea behind the conditional re-encoding we propose is to create many-to-many relationships in the non-linear encoding for the round output. To this end, the non-linear encoding table for the round output is switched depending on the characteristic of the inputs to Type IV_III (XOR tables). We also analyze its security against cryptanalysis in the next section.

1. Basic Idea and Its Extension

A. Round Output Encoding

Let δ denote the non-linear encoding function for the round output. Given $x, y \in \{0, 1\}^4$ and $z = x \oplus y$, δ depends on x, y, and z instead of only z. This is as follows:

1) Encode z using table E2: $z' \leftarrow$ E2(z).
2) Let ϕ be a function examining the characteristics of x and y based on a predefined rule. For example, for two randomly selected i, j, where $1 \le i, j \le 4$, and $i \ne j$, we have

$$\phi(x, y) = \begin{cases} 1 & \text{if } x(i) \ne y(j), \\ 0 & \text{otherwise,} \end{cases}$$

where $x(n)$ indicates the nth bit of x.

3) If $\phi(x, y) = 1$, then re-encode z' using an additional encoding table E3: $z' \leftarrow$ E3(z').
4) Return $\delta(x, y, z) = z'$. Here, the choice of ϕ is not limited to this example, but can be extended to various scenarios including a comparison of multiple bits of x and y. We will discuss the security aspect of this later.

This conditional re-encoding depending on the value of ϕ creates a many-to-many relationship between z and z'. To be more precise, given $x1, x2, y1, y2 \in \{0, 1\}^4$ such that $x1 \oplus y1 = z1$ and $x2 \oplus y2 = z2$, let us assume that $\phi(x1, y1) = 1$ and $\phi(x2, y2) = 0$. We then know that $\delta(x1, y1, z1) \ne \delta(x2, y2, z2)$. For this reason, z' requires the value of $\phi(x, y)$ to be correctly decoded. Because the value of $\phi(x, y)$ determines whether a re-encoding takes place, δ is a function of $\phi(x, y)$ and z.

B. Round Input Decoding

To properly decode z' back to z, we need to keep track of the occurrence of the re-encoding in a satellite value. We denote the satellite variable by γ, and its value is determined by the value of $\phi(x, y)$. If $\gamma = 1$, then it means that z' = E3(E2(z)); otherwise, z' = E2(z). The input decoding δ^{-1} of the next round is then a function of z' and γ as follows:

$$z = \delta^{-1}(z', \gamma) = \begin{cases} \text{InvE2}(\text{InvE3}(z')) & \text{if } \gamma = 1, \\ \text{InvE2}(z') & \text{otherwise.} \end{cases}$$

The last step is to apply the conditional re-encoding and decoding during the generation of the lookup table. The table generation for Type IV_III and Type II tables needs to be modified as follows.

C. Type IV_III Generation and Lookup

A sub-byte of a round output is computed using three XOR operations combining four intermediate values, and we denote each intermediate result by pair2, pair1, and pair0, as in Section II. Without having to apply the conditional re-encoding to all pairs, we apply it only when pair0 is computed for simplicity. Thus, a Type IV_III table should reflect the conditional re-encoding based on the characteristics of the two 4-bit inputs of pair0 when it is generated (see Appendix).

To properly look up a Type IV_III table, we must consider that all encodings and XOR operations are performed in the 4-bit unit, and γ needs to be two bits to track the re-encodings of the upper and lower 4-bit chunks separately; in each round, the total size of γ is then 16 × 2 bits. The satellite variable γ provides no additional secret information to the attacker because the two inputs are encoded values.

D. Type II Generation and Lookup

To be properly cooperated with the modified Type IV_III tables, the Type II tables must also be modified to correctly decode the input. The conditional re-encoding is independently applied to the upper and lower 4-bit output of a Type IV_III

Table 1. Four cases of input decoding in a Type II table, where γ is corresponding value of ϕ for each case.

$Y1$	$Y0$	γ
InvE2($X1$)	InvE2($X0$)	00
InvE2($X1$)	InvE3(InvE2($X0$))	01
InvE3(InvE2($X1$))	InvE2($X0$)	10
InvE3(InvE2($X1$))	InvE3(InvE2($X0$))	11

table. For this reason, a Type II table contains four decoding cases, as shown in Table 1. To look up a correctly decoded input for a Type II table, γ has to be referred. Based on Table 1, γ can be used as an index to select the decoded value.

2. Security Analysis

As stated previously, three methods of cryptanalysis depend on Theorem 1 or the collision equation. In turn, Theorem 1 and the collision equation strongly rely on Q. Specifically, Theorem 1 basically uses $Q^{-1} \circ Q(z) = z$ and the collision equation uses the fact that $z = z'$ if $Q(z) = Q(z')$. Owing to the many-to-many relationship by the conditional re-encoding, Theorem 1 and the collision equation are not acceptable for our proposed scheme. The details are as follows.

A. Against Theorem 1

When we say $Q^{-1} \circ Q(z) = z$, this means that the output of the non-linear encoding is determined by only its 8-bit value. To be more precise, a particular 8-bit z at the same round, row, column, and pair has to be encoded to a particular output z'. In addition, z' has to be decoded to z. Based on the property of the conditional re-encoding, this is not always guaranteed.

Let us recall (3). An attacker assumed that the table for the non-linear encoding at the same round, row, column, and pair is always the same. However, we know that the non-linear encoding table for the round output is

$$\begin{cases} \text{E3} \circ \text{E2} & \text{if } \phi(x, y) = 1, \\ \text{E2} & \text{otherwise,} \end{cases}$$

where x and y are 4-bit inputs to be XORed. Given a particular round, row, column, and pair0, we need to conduct the XOR operations between 8-bit inputs and encode the result for the round output. The following examples illustrate how this disturbs Theorem 1. Table 2 explains the notation to be used.

Example 1. X = 0x11, Y = 0x34, E2_high (0x2) = 0xA, E2_low (0x5) = 0x1, and E3_high (0xA) = 0x7, E3_low (0x1) = 0xC. The decision function ϕ outputs 1 if the LSB of the first input and the second LSB of the second input are different. The following computation is then depicted in Fig. 6, and we know that

$$Z1 = X \oplus Y = \text{0x25 and } \gamma = 01.$$

This gives us the final encoded output of 0xAC because the lower 4-bit of $Z1$ has to be re-encoded.

Table 2. Notations used.

Notation	Usage
E2_high	Non-linear encoding table for the upper 4 bits
E2_low	For the lower 4 bits
E3_high	Re-encoding for the upper 4 bits
E3_low	For the lower 4 bits
x_0	The lower 4-bit of an 8-bit X
x_1	The upper 4-bit of X
δ	Non-linear encoding function
ϕ	Decision function for re-encoding
γ	Satellite variable storing the occurrences of re-encodings for an 8-bit value

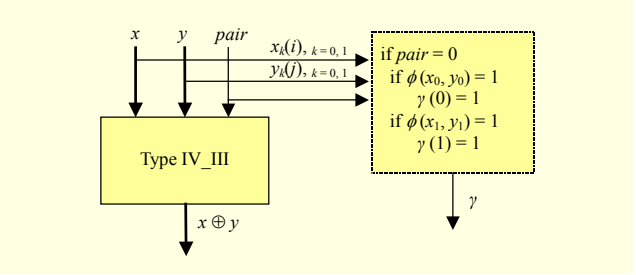

Fig. 6. Type IV_III lookups for XOR operations.

Example 2. X = 0xB7, Y = 0x92, and the other conditions are the same; thus, we have

$$Z2 = X \oplus Y = \text{0x25 and } \gamma = 10.$$

In this case, the final encoded output is 0x71 because only the upper 4-bit of $Z2$ has to be re-encoded.

Example 3. X = 0x6C, Y = 0x41, E2_low (0xD) = 0xC, and the other conditions are the same; thus, we have

$$Z3 = X \oplus Y = \text{0x2D and } \gamma = 00.$$

The final encoded output is then 0xAC because there is no re-encoding. In the first two examples, we showed that the two encoded outputs can be different due to the re-encoding even though $Z1$ and $Z2$ are the same.

In contrast, the last two examples showed that the two encoded outputs can be the same even though $Z2$ and $Z3$ are different. The decoding for the next round's input also has a many-to-many relationship because the input decoding is

ETRI Journal, Volume 37, Number 5, October 2015
http://dx.doi.org/10.4218/etrij.15.0114.0025

determined by γ. For this reason, a successful cryptanalysis using Theorem 1 is possible if there is *always no re-encoding* or *always a re-encoding*. According to (3) and Theorem 1, an attacker needs $S = \{Q \circ \oplus_\beta \circ Q^{-1}\}(x)$, where $x \in \mathrm{GF}(2^8)$. Because we know that the probability that γ = '00' is 1/4, the probability of "*always no re-encoding*" happening is $(1/4)^{256}$. Similarly, the probability of "*always a re-encoding*" is also $(1/4)^{256}$. Thus, there is a negligible probability of constructing a correct S for an attacker who has one of the assumptions, "*always no re-encoding*" or "*always a re-encoding*." Without a correct set of functions S, an attacker is unable to construct a particular solution $\tilde{Q}$ in Theorem 1 and therefore the cryptanalysis cannot work.

Without loss of generality, if we compare n bits of x_k and y_k, where k = 0 or 1, to test the condition for the re-encoding, then there is a $1/2^{2n}$ probability of a re-encoding not happening. Therefore, if we compare four bits of x_k and y_k, a sub-byte of a round output is re-encoded with a 255/256 probability. In this case, the attacker can be convinced that the output was probably encoded by the same table, E3 ∘ E2. The problem then returns to its starting point. For this reason, we recommend comparing a single bit between x_k and y_k. In addition, one might be curious if the many-to-many relationship of the non-linear encoding lowers the security level of WB-AES. However, the encoding becomes complex because the number of encoding cases is significantly increased.

B. Against Collision Attacks

Lepoint and others' cryptanalysis needs to find a collision by Q. We now show that the many-to-many relationship by the conditional re-encoding prevents this cryptanalysis from finding such a collision. As pointed out previously, the attacker first tries to recover functions S_0 and S_1 using (4) as follows:

$$Q_0(02 \otimes S_0(\alpha) \oplus 03 \otimes S_1(0) \oplus c) = Q_0(02 \otimes S_0(0) \oplus 03 \otimes S_1(\beta) \oplus c).$$

In this cryptanalysis, this means that $02 \otimes S_0(\alpha) \oplus 03 \otimes S_1(0) = 02 \otimes S_0(0) \oplus 03 \otimes S_1(\beta)$, because the non-linear encoding for the round output guarantees a one-to-one mapping regardless of how the mixing bijection obfuscates each byte.

However, as demonstrated in the previous analysis, even if both sides of the equation are the same, then their encoded outputs can be different owing to the conditional re-encoding. On the contrary, there can be collisions even when

$$02 \otimes S_0(\alpha) \oplus 03 \otimes S_1(0) \neq 02 \otimes S_0(0) \oplus 03 \otimes S_1(\beta).$$

We expect that each intermediate bit becomes 0 or 1 with a 1/2 probability owing to the confusion and diffusion property of S-box, MixColumns, and mixing bijection. In this assumption, half of the (α, β) pairs satisfying

$$02 \otimes S_0(\alpha) \oplus 03 \otimes S_1(0) = 02 \otimes S_0(0) \oplus 03 \otimes S_1(\beta)$$

may not lead to collisions if the re-encoding occurs by a 1-bit comparison. The attacker's disadvantage here is that it is not allowed to fully recover S_0. In turn, S_1, S_2, and S_3 cannot be recovered without recovering S_0. Since the attacker is unable to find the correct collision sets, the strategy for the cryptanalysis is not complete for the proposed scheme.

3. Size and Performance

We provide additional costs of our proposed algorithm in terms of the memory requirement and execution speed. The only operation left unchanged from the AES implementation of Daemen and Rijmen [11] is ShiftRows while the changed lookup tables from Chow and others' WB-AES are Type II and IV_III tables. The elapsed time for the table generation is not included in the runtime overhead.

A. Memory Requirement

The total size of the lookup tables is 1,212,416 bytes. A reasonable comparison is Chow and others' WB-AES, which requires 770,048 bytes for the lookup tables. Only the Type II table increases its size, with a four-fold increase. This is due to the fact that there are four cases of input decoding in a Type II table, as stated previously. As a consequence, our proposed WB-AES requires 1.57-times the size of the lookup tables in total, compared to Chow and others' WB-AES, shown in Table 3.

B. Execution Time

The number of lookups is left unchanged from Chow and

Table 3. Comparison of table size: dash symbol (-) indicates that proposed algorithm is same size as Chow and others' algorithm, in unit of bytes.

Type	Chow's	Proposed
Type IA	65,536	-
Type IA_IV	61,440	-
Type II	147,456	589,824
Type IV_II	110,592	-
Type III	147,456	-
Type IV_III	110,592	-
Type IB	65,536	-
Type IV_IB	61,440	-
Total	770,048	1,212,416

Table 4. Comparison of runtime. Number of clock ticks per second is 1,000.

Algorithm	CPU ticks
AES-128	1
Chow and others	10
Our WB-AES	12

others' WB-AES, but some additional operations are needed to compute and read y. To compare the runtime of the standard AES implementation, for both Chow and others' WB-AES and our own, we performed experiments using a desktop PC with an Intel Core i7 CPU at 3.4 GHz. We repeated each algorithm 100 times and measured the number of elapsed CPU ticks during the execution. The average CPU tick is shown in Table 4. Because the number of clock ticks per second of our PC is 1,000, one CPU tick is equal to 1 ms.

Compared to AES, Chow and others' WB-AES is approximately ten-times slower, whereas our WB-AES is twelve-times slower. The interesting point here is that we expected a much lower execution time because of a number of lookups compared to AES. This result indicates that other operations, such as an XOR, in AES probably take more time than the lookups.

C. Table Generation

The first step of white-box cryptography is to generate the lookup tables. However, generating the lookup tables in advance is not included in the runtime overhead in our case as previously mentioned. For a comparison, we provide the elapsed time for the table generation of the two WB-AES algorithms. The main reasons for the additional time, as compared to Chow and others' WB-AES, are as follows:

- generating additional encoding tables, E3 and InvE3
- conditional re-encoding of the output of Type IV_III tables
- generating the new Type II tables, which are four-times bigger than the original type

As a result, our algorithm requires approximately 8,100 ticks, whereas that of Chow and others requires 5,200 ticks on average.

V. Conclusion

To provide a secure WB-AES implementation, we proposed a conditional re-encoding method applied to the round outputs, and changed the corresponding input decoding of the next round. In addition, we described how to properly look up the tables and showed that the encoding variance protects against cryptanalysis. Additional costs compared to Chow and others' WB-AES are about 1.57-times the total size of the lookup tables and 1.2-times the execution speed.

Appendix

Algorithm 1. Generating the original Type IV_III.

```
for   (rnd = 0; rnd < 16;   ++ rnd)   {
 for   (row = 0; row < 4;   ++ row)   {
  for   (col = 0; col < 4;    ++ col)   {
   for   (pair = 0; pair < 3;   ++pair)   {
   \* offSet distinguishes
      the upper and the lower 4 bits of a byte *\
    for (offSet = 0;   offSet < 2;   ++offSet)   {
     x_dec_table =
            & (InvE1) [rnd] [row] [col] [pair × 2 + offSet] [0];
     y_dec_table =
            & (InvE2) [rnd] [row] [col] [offSet] [0];

     if   (pair = = 2)   {
      y_dec_table =
        & InvE1 [rnd] [row] [col] [((pair + 1) × 2) + offSet] [0];
     }
     for   (x = 0; x < 16;   ++x)   {
      for   (y = 0; y < 16;   ++y)   {
       x_dec = x_dec_table [x] ;
       y_dec = y_dec_table [y] ;
       xy_index = x || y;
       xy_xored = x_dec ^ y_dec;

            Type IV [rnd] [row] [col] [pair] [offSet] [xy_index]
                = E2 [rnd] [row] [col] [offSet] [xy_xored];
     }
  …
}
```

Algorithm 1 shows how to generate the original Type IV_III table in [5]. It first decides the proper decoding tables for x and y based on a pair. Then, it decodes all possible 4-bit inputs x and y, and computes the XOR value between them. To encode each resulting value, there is only one encoding table, E2. The encoding is simply done by looking up the values of E2 at the index *xy_xored*; the encoded values are stored in Type IV_III. This is simplified in Fig. 7.

In Section IV, we recommended comparing a single bit of x and y to decide whether to conduct the re-encoding. For better understanding, Fig. 8 simplifies the flow of the generation of Type IV_III if the re-encoding is decided by a single bit comparison between x and y. Algorithm 2 explains the generation of the proposed Type IV_III in detail on the condition that the re-encoding depends on the comparison of the LSB of x and the second LSB of y. For all possible pairs of

4-bit x and y, the proper decoding tables are decided first, and the XOR result is encoded by E2. Based on the result of the comparison, the re-encoding using an additional encoding table E3 is performed.

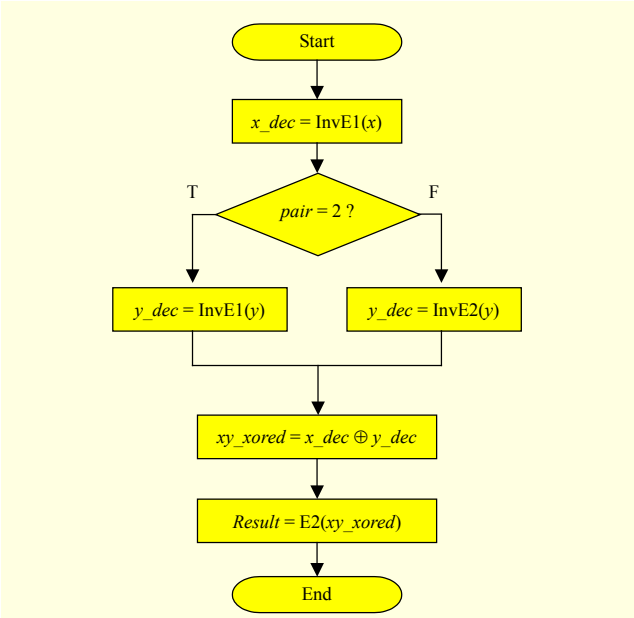

Fig. 7. Simplified flow chart of Algorithm 1. InvE1 and InvE2 are decoding tables, while E2 is an encoding table.

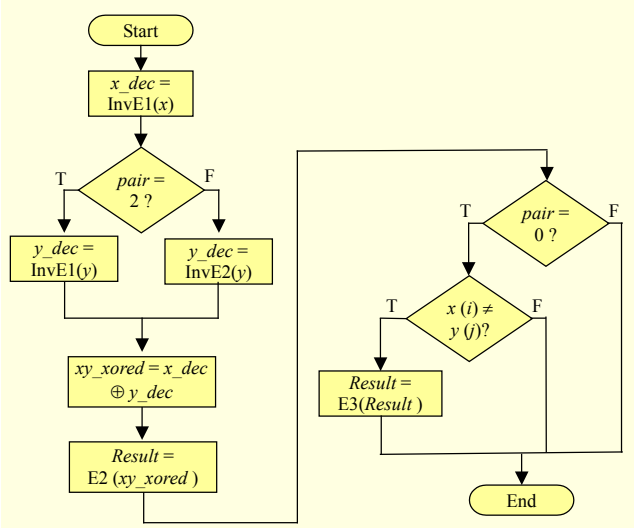

Fig. 8. Simplified example of proposed encoding method; $x(n)$ is the nth bit of x in binary.

Algorithm 2. Generating the proposed Type IV_III.

```
for  (rnd = 0; rnd < 16;  ++ rnd)  {
 for  (row = 0; row < 4;  ++ row)  {
  for  (col = 0; col < 4;  ++ col)  {
   for  (pair = 0; pair < 3;  ++pair)  {
    \* offSet distinguishes
     the upper and the lower 4 bits of a byte *\
    for (offSet = 0;  offSet < 2;  ++offSet)  {
     x_dec_table =
          & (InvE1) [rnd] [row] [col] [pair × 2 + offSet] [0];
     y_dec_table =
          & (InvE2) [rnd] [row] [col] [offSet] [0];

     if  (pair = = 2)  {
      y_dec_table =
        & InvE1 [rnd] [row] [col] [((pair + 1) × 2) + offSet] [0];
     }
     for  (x = 0; x < 16;  ++x)  {
      for  (y = 0; y < 16;  ++y)  {
       x_dec = x_dec_table [x] ;
       y_dec = y_dec_table [y] ;
       xy_index = x | | y;
       xy_xored = x_dec ^ y_dec;
       encoded_output = E2 [rnd] [row] [col] [offSet] [xy_xored];

       \* compare x(1) and y(2) at pair 0 *\
       if ((pair = = 0) && ((x & 0x01) ^ (y & 0x02) = = 1))
            encoded_output =
                 E3 [rnd] [row] [col] [offSet] [encoded_output];

       Type IV [rnd] [row] [col] [pair] [offSet] [xy_index]
                 = encoded_output;
     }
   …
}
```

Algorithm 2 shows how to generate the proposed Type IV_III table; here, E3 is an additional encoding table used for the conditional re-encoding.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Int. Cryptology Conf. Adv. Cryptology*, Santa Barbara, CA, USA, Aug. 15–19, 1999, pp. 388–397.

[2] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," *Cryptographic Hardware Embedded Syst.*, Cambridge, MA, USA, Aug. 11–13, 2004, pp. 16–29.

[3] S. Lee, D. Choi, and Y. Choi, "Improved Shamir's CRT-RSA Algorithm: Revisit with the Modulus Chaining Method," *ETRI J.*, vol. 36, no. 3, June 2014, pp. 469–478.

[4] D. Boneh, R.A. DeMillo, and R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Int. Conf. Theory Appl. Cryptographic Techn.*, Konstanz, Germany, May 11–15, 1997, pp. 37–51.

[5] S. Chow et al., "White-Box Cryptography and an AES Implementation," *Workshop Sel. Areas Cryptography*, Madrid,

Spain, Aug. 15–16, 2002, pp. 250–270.
[6] O. Billet, H. Gilbert, and C. Ech-Chatbi, "Cryptanalysis of a White Box AES implementation," *Int. Conf. Sel. Areas Cryptography*, Waterloo, Canada, Aug. 9–10, 2004, pp. 227–240.
[7] J. Bringer, H. Chabanne, and E. Dottax, "White Box Cryptography: Another Attempt," *IACR Cryptology ePrint Archive*, vol. 2006, Dec. 2006, p. 468.
[8] Y.D. Mulder, P. Roelse, and B. Preneel, "Cryptanalysis of the Xiao-Lai White-Box AES Implementation," *Int. Conf. Sel. Areas Cryptography*, Windsor, Canada, Aug. 15–16, 2012, pp. 34–49.
[9] Y.D. Mulder, B. Wyseur, and B. Preneel, "Cryptanalysis of a Perturbated White-Box AES Implementation," *Int. Conf. Cryptology India*, Hyderabad, India, Dec. 12–15, 2010, pp. 292–310.
[10] T. Lepoint et al., "Two Attacks on a White-Box AES Implementation," *Int. Workshop Sel. Areas Cryptography*, Burnaby, Canada, Aug. 14–16, 2013, pp. 265–285.
[11] J. Daemen and V. Rijmen, *AES Proposal: Rijndael*, 1998. Accessed Aug. 30, 2014. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf
[12] J.A. Muir, "A Tutorial on White-Box AES," in *IACR Cryptology ePrint Archive*, 2013, p. 104.
[13] Shannon, "Communication Theory of Secrecy Systems," *Bell Syst. Techn. J.*, vol. 28, no. 4, Oct. 1949, pp. 656–715.
[14] Jeff Saremi, *White-Box AES Project for Educational Purposes.* Accessed Apr. 1, 2014. https://github.com/wbaes
[15] W. Michiels, P. Gorissen, and H.D. Hollmann, "Cryptanalysis of a Generic Class of White-Box Implementations," *Int. Conf. Sel. Areas Cryptography*, Sackville, Canada, Aug. 14–15, 2008, pp. 414–428.
[16] L. Tolhuizen, "Improved Cryptanalysis of an AES Implementation," *WIC Symp. Inf. Theory Benelux*, Boekelo, Netherlands, May 24–25, 2012.
[17] A. Biryukov et al., "A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms," *Int. Conf. Theory Appl. Cryptographic Techn.*, Warsaw, Poland, May 4–8, 2003, pp. 33–50.
[18] J. Fuller and W. Millan, "Linear Redundancy in S-Boxes," *Int. Workshop Fast Softw. Encryption*, Lund, Sweden, Feb. 24–26, 2003, pp. 74–86.

Seungkwang Lee received his BS degree in computer science and electronic engineering from Handong University, Pohang, Rep. of Korea, in 2009 and his MS degree in computer science from Pohang University of Science and Technology, Rep. of Korea, in 2011. He is currently working as a researcher at ETRI. His research interests include cryptography, side-channel analysis, and fault injection attacks.

Dooho Choi received his BS degree in mathematics from Sungkyunkwan University, Suwon, Rep. of Korea, in 1994 and his MS and PhD degrees in mathematics from the Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea, in 1996 and 2002, respectively. He has been a principal researcher at ETRI since January 2002, and he is also an assistant professor at the University of Science & Technology, Daejeon, Rep. of Korea. His current research interests include cryptographic engineering, including side-channel analysis and its countermeasure design; security technologies of RFID and IoT; and lightweight and secure cryptographic module HW/SW design. He was the editor of the ITU-T Rec. X.1171.

Yong-Je Choi received his BS and MS degrees from Chonnam National University, Gwangju, Rep. of Korea, in 1996 and 1999, respectively. He is currently a senior member of technical staff at ETRI. His research interests include VLSI design, crypto processor design, side-channel analysis, and information security.