Adaptive Side-Channel Analysis Model and Its Applications to White-Box Block Cipher Implementations

Yufeng Tang[1], Zheng Gong[1](✉), Tao Sun[2], Jinhai Chen[1], and Fan Zhang[3,4]

[1] School of Computer Science, South China Normal University, Guangzhou, China
cis.gong@gmail.com
[2] China Information Technology Security Evaluation Center (Guangdong Office), Guangzhou, China
[3] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[4] Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province, Hangzhou, China

Abstract. White-box block cipher (WBC) aims at protecting the secret key of a block cipher even if an adversary has full control over the implementations. At CHES 2016, Bos *et al.* proved that WBC are also threatened by *side-channel analysis* (SCA), e.g., *differential fault analysis* (DFA) and *differential computation analysis* (DCA). Therefore, advanced countermeasures have been proposed by Lee *et al.* for resisting DFA and DCA, such as table redundancy and improved masking methods, respectively. In this paper, we introduce a new *adaptive side-channel analysis* model which assumes that an adversary adaptively collects the intermediate values of a specific function and can mount the DFA/DCA attack with chosen inputs. In the adaptive SCA model, both theoretical analysis and experimental results show that Lee *et al.*'s proposed methods are vulnerable to DFA and DCA attacks. Moreover, a negative proposition is also demonstrated on the corresponding *high-order* countermeasures under our new model.

Keywords: White-box block cipher · Adaptive side-channel analysis · Differential fault analysis · Differential computation analysis

1 Introduction

The concept of *white-box attack context* was introduced in 2002 by Chow, Eisen, Johnson, and van Oorschot (CEJO) [15, 16]. It assumes that an adversary can analyze the details of a cryptosystem with full control over its execution. White-box block cipher (WBC) aims at preventing the secret key of a block cipher algorithm from being extracted in a white-box environment. The first two attempts of WBC were white-box AES [16] (CEJO-WBAES) and DES [15] proposed by CEJO. The fundamental idea of them is to convert the operations of a cryptographic algorithm with a secret key into look-up tables (LUTs) and

apply linear and non-linear encodings to protect the intermediate values. In 2004, Billet *et al.* [7] presented an algebraic attack which was named BGE attack against CEJO-WBAES. From then on, although several improvements on white-box AES [14, 24, 39] were mentioned to resist cryptanalysis, all of them were subsequently broken [17–19].

In addition to algebraic attacks, *side-channel analysis* (SCA) has been demonstrated on WBC. At CHES 2016, Bos et al. [9, 13] proposed to use *differential fault analysis* (DFA) and *differential computation analysis* (DCA) to attack white-box implementations. DFA induces faults in intermediate values and analyzing the differential equations with faulty and unencoded ciphertexts. DCA adapts a statistical technique of *differential power analysis* (DPA) but uses software computation traces consisting of noise-free intermediate values and accessed data. These attacks perform statistical analysis on the intermediate values such that avoid a time-consuming reverse engineering step and can be implemented automatically. SCA attacks are the major threats to white-box implementations as shown in the security assessment of the submissions of WhibOx 2017/19 competition [11, 23].

Advanced SCA methods on WBC. At CHES 2017, Banik *et al.* [6] developed *zero difference enumeration* attack which records software traces for pairs of selected plaintexts and performs an analysis on the difference of traces. Bock *et al.* [10] analyzed the ineffectiveness of internal encoding on white-box implementations under DCA attack, which pointed out that the Hamming Weight 1 of a row in a matrix are the main cause of key leakage. Lee *et al.* [27] presented an in-depth analysis on the linear encoding and demonstrated that it is the unbalanced distribution of the key-dependent intermediate values that cause the successful DCA. Amadori *et al.* [4] presented a new DFA attack that combines the techniques of DFA and BGE on a class of white-box AES implementation with 8-bit external encodings. At CHES 2019, Rivain and Wang [33] investigated *mutual information analysis* and *collision attack* to defeat the internal encoding. Another DCA-like approach, called *statistical bucketing attack*, was published by Zeyad *et al.* [40] for recovering the key by capturing computation traces based on the cryptanalysis technique introduced by Chow *et al.* [15]. Bogdanov *et al.* [12] and Maghrebi *et al.* [31] evaluated the *higher-order* cases of the computational analysis on the white-box implementations. For mitigating DFA, Lee *et al.* proposed a table redundancy method [28] which replaces the comparison step for fault detection with an exclusive-or (XOR) operation based on the white-box diversity and linearity of encodings. For countering DCA, Lee *et al.* [30] proposed the masking technique to the key-dependent value before applying encodings in the table generating phase. However, Rivain and Wang [33] described a 2-byte key guessing model of DCA and analyzed the first-round output of which the state is unmasked. To thwart the existing DCA attacks, Lee and Kim [29] improved their proposed scheme [30] by extending the masking to the round outputs.

Our contribution. The main contributions of this paper are summarized as follows. (1) Based on the abilities of a white-box attacker and the efficiency of SCA

attacks on white-box implementations, we introduce an adaptive SCA model for WBC. (2) The instantiation of adaptive DFA and DCA attacks are presented to break Lee *et al.*'s table redundancy [28] and improved masking [29] white-box implementations, respectively. The adaptive DFA replaces the ninth-round inputs with the adaptively collected states to bypass the XOR phase for fault detection. And the adaptive DCA exploits the collision of output mask to choose the plaintexts which have the same mask at the first-round output. (3) The higher-order table redundancy and improved masking are also discussed. Moreover, the adaptive higher-order DFA and DCA attacks are extended to defeat such countermeasures. The comparison between the applications of traditional and adaptive SCA models is shown in Table 1.

Table 1. The comparison between the applications of traditional and adaptive SCA models.

Method	SCA	Adaptive SCA
(Higher-order) Table redundancy [28]	(Higher-order) DFA resisted	DFA succeed, adaptively chosen ninth-round inputs (Section 3.2, 5.1)
(Higher-order) Improved masking [29]	(Higher-order) DCA resisted	DCA succeed, adaptively chosen plaintexts (Section 3.3, 5.2)

Organization. The remainder of this paper is organized as follows. Section 2 reviews the table redundancy [28] and improved masking [29] for white-box implementation. Section 3 describes the core idea of the adaptive SCA model on WBC. And then we provide instances of adaptive DFA and DCA attacks against table redundancy and improved masking methods. Afterward, Section 4 shows the experimental results of the adaptive DFA and DCA attacks, and Section 5 extends the attacks to the higher-order countermeasures. Section 6 concludes this paper.

2 Preliminaries

In this section, we briefly recall CEJO-WBAES and its SCA attacks. The state-of-the-art countermeasures on those SCA attacks are also reviewed.

2.1 CEJO-WBAES

In 2002, Chow *et al.* [16] defined the *white-box attack context* and introduced CEJO-WBAES to resist key extraction in cryptographic implementations. The basic idea is to convert the round functions of AES into a series of LUTs and apply secret invertible encodings to protect the intermediate values. Let T denote a LUT, f and g be random bijective mappings. Then the encoded LUT T' is defined as $T' = g \circ T \circ f^{-1}$, where f^{-1} and g are called the input and output

encoding, respectively. To maintain the functionality of AES, the input and output encodings of consecutive rounds should be constructed as pairwise invertible mappings. Therefore, the input encoding can also play a role as input decoding since it decodes the previous encoded output to recover the secret state. Let an encoded LUT R' be defined as $R' = h \circ R \circ g^{-1}$ such that a networked encoding can be depicted as $R' \circ T' = (h \circ R \circ g^{-1}) \circ (g \circ T \circ f^{-1}) = h \circ (R \circ T) \circ f^{-1}$.

The basic principles of CEJO-WBAES are recalled as follows. Note that an AES state is represented by a byte array such that the index is ranked from 0 to 15. By means of *partial evaluation*, for each round, AddRoundKey and SubBytes operations are composed as 16 8-bit bijective key-dependent LUTs which are defined as T-boxes as follows.

$$\begin{aligned} T_i^r(x) &= S(x \oplus \hat{k}_i^{r-1}), \qquad \text{for } i \in [0,15] \text{ and } r \in [1,9], \\ T_i^{10}(x) &= S(x \oplus \hat{k}_i^{9}) \oplus \hat{k}_i^{10}, \ \text{for } i \in [0,15], \end{aligned}$$

where S denotes the Sbox, $\hat{k}^r$ represent the result of applying ShiftRows to the byte array of round key. With *matrix partitioning*, the multiplication of MixColumns can be decomposed into four 32-bit vectors.

$$\begin{aligned} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} &= x_0 \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus x_1 \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus x_2 \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus x_3 \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \\ &= Ty_0(x_0) \oplus Ty_1(x_1) \oplus Ty_2(x_2) \oplus Ty_3(x_3). \end{aligned}$$

Ty_j for $0 \le j \le 3$ denote the 8 bits to 32 bits mappings for the decomposition of MixColumns. For rounds $1 \le r \le 9$, the T-boxes and Ty_j are then merged together to construct 16 TMC_i^r for $0 \le i \le 15$. Each TMC_i^r is defined as follows.

$$\mathrm{TMC}_i^r = Ty_j \circ T_i^r, \ \text{for } 0 \le i \le 15 \text{ and } j = i \bmod 4.$$

To add *diffusion* and *confusion* to each key-dependent intermediate value, the Mixing Bijections and Nibble Encodings are applied to all inputs and outputs of tables. An 8×8-bit linear transformation L_i^r and a 32×32-bit one MB are inserted before and after TMC_i^r, respectively. Subsequently, 4-bit non-linear encodings N are applied to the table inputs and outputs. The resulting encoded TMC_i^r are denoted by *TypeII* which is defined as follows.

$$\textit{TypeII: } N \circ \mathrm{MB} \circ \mathrm{TMC}_i^r \circ L_i^r \circ N^{-1}.$$

To cancel out the effect of MB and convert it to $(L_i^{r+1})^{-1}$ to from the networked encoding, a *TypeIII* table is introduced accordingly and shown in below. Note that the table is generated by the technique of *matrix partitioning* as well.

$$\textit{TypeIII: } N \circ (L_i^{r+1})^{-1} \circ \mathrm{MB}^{-1} \circ N^{-1}.$$

Besides, all the XOR operations between encoded values are conducted by *TypeIV* tables which decode two 4-bit inputs and provides a 4-bit encoded XOR

result. The combination of the outputs of *TypeII* is aptly named *TypeIV_II* while the one of *TypeIII* is aptly named *TypeIV_III*. Due to the linearity of XOR, the encoding and decoding phases of *TypeIV* only consist of non-linear encodings. Combining these tables, one can obtain an encoded fixed-key white-box AES such that $G \circ AES \circ F^{-1}$, where F^{-1} and G denote the external input and output encodings respectively. Since the external encoding lacks of compatibility, most of the theoretical analyses and constructions have not taken it into consideration (e.g., WhibOx 2017/19 competitions [1,2]). We note that every 4 bytes of CEJO-WBAES can also be interpreted as a column vector of 4×4 state matrix. For a detailed description of CEJO-WBAES, please refer to the tutorial paper [32].

2.2 Differential Fault Analysis

Following the ninth-round DFA attack model [20], Teuwen *et al.* [38] applied fault attacks against CEJO-WBAES. The attack injects a byte fault into the steps between the eighth-round and ninth-round MixColumns. Suppose that a fault is injected at the first byte $x \in \mathbb{F}_2^8$ of the ninth-round inputs. Let $\delta, \delta' \in \mathbb{F}_2^8$ denote the difference between the original byte and the faulty one before and after SubBytes, respectively. Such that $\delta' = S(x) \oplus S(x \oplus \delta)$, where S denotes the Sbox of SubBytes. The 4-byte difference after the MixColumn is represented by $(2\delta',\ \delta',\ \delta',\ 3\delta')$, where 2, 1, 1, 3 are the coefficients of MixColumns. Let S^{-1} be the inverse SubBytes. For the fault-free ciphertexts $C_0...C_7...C_{10}...C_{13}$ ($C_i \in \mathbb{F}_2^8,\ i \in [0, 15]$) and the faulty ciphertexts $C_0^*...C_7^*...C_{10}^*...C_{13}^*$ ($C_i^* \in \mathbb{F}_2^8,\ i \in \{0,\ 7,\ 10,\ 13\}$, $C_j \in \mathbb{F}_2^8,\ j \in [0, 15] \backslash \{i\}$), the following equations can be listed to find the tenth-round subkey candidate K_i^* ($i \in \{0,\ 7,\ 10,\ 13\}$). By injecting two such faults, the tenth-round 4-byte subkey can be determined by the differential equations. The other subkeys can be recovered by the similar analysis.

$$
\begin{aligned}
2\delta' &= S^{-1}(C_0 \oplus K_0^*) \oplus S^{-1}(C_0^* \oplus K_0^*),\\
\delta' &= S^{-1}(C_7 \oplus K_7^*) \oplus S^{-1}(C_7^* \oplus K_7^*),\\
\delta' &= S^{-1}(C_{10} \oplus K_{10}^*) \oplus S^{-1}(C_{10}^* \oplus K_{10}^*),\\
3\delta' &= S^{-1}(C_{13} \oplus K_{13}^*) \oplus S^{-1}(C_{13}^* \oplus K_{13}^*).
\end{aligned}
$$

2.3 The Table Redundancy Method Against DFA

As introduced by Lee *et al.* [28], table redundancy method uses multiple branches of LUTs for the vulnerable rounds (e.g., the sixth to ninth rounds) and XOR the outputs to obfuscate the fault injection. The description of the scheme can be shown in three parts as follows. (1) Sharing the LUTs from Round 1 to 5. (2) Transforming independently in parallel with two sets of LUTs from the sixth round to ninth-round *TypeII*. These two computations are constructed with different sets of LUTs built by different encodings. (3) Sharing the LUTs from ninth-round *TypeIV_II* to Round 10. Note that the basic technique for constructing the LUTs is followed by CEJO-WBAES [16].

Let $\mathrm{MB}_l \in \mathbb{F}_2^{32\times 32}$ and $\mathrm{MB}_r \in \mathbb{F}_2^{32\times 32}$ (l (resp. r) represents the left (resp. right) part of the two computations) be the 32-bit Mixing Bijections of the ninth-round *TypeII*. The $x_l \in \mathbb{F}_2^8$ and $x_r \in \mathbb{F}_2^8$ denote the two unencoded bytes of ninth-round inputs. Such that $\mathrm{TMC}(x_l)$ and $\mathrm{TMC}(x_r)$ represent the outputs of SubBytes and MixColumns from x_l and x_r, respectively. The outputs of the two computations XOR with each other by a type of *TypeIV* followed by the shared *TypeIV_II*. Let $x \in \mathbb{F}_2^8$ be the original state at ninth-round inputs of standard AES, the XOR process can be shown as

$$\begin{aligned}\mathrm{MB}_l \cdot \mathrm{TMC}(x_l) \oplus \mathrm{MB}_r \cdot \mathrm{TMC}(x_r) = \\ (\mathrm{MB}_l \oplus \mathrm{MB}_r) \cdot \mathrm{TMC}(x), \text{ iff } x_l = x_r = x.\end{aligned}$$

Hence, the inverse Mixing Bijection $\mathrm{MB}^{-1} \in \mathbb{F}_2^{32\times 32}$ in ninth-round *TypeIII* can be depicted as

$$\mathrm{MB}^{-1} = (\mathrm{MB}_l \oplus \mathrm{MB}_r)^{-1}.$$

The Mixing Bijections can be combined with $\oplus$ because both of them are linear transformations and the underlying state x is fault-free. Once x_l or x_r is injected by a fault, the XOR cannot lead to a valid differential equation.

2.4 Differential Computation Analysis

At CHES 2016, Bos *et al.* [13] introduced DCA as the software counterpart of DPA [26] to break white-box implementations. DCA divides the measurement traces in two distinct sets according to the value of one of the bits of Sbox output. Let x_i be an input, v_i be the collected traces, $b = S_j(x_i \oplus k)$ denote the j-th bit of Sbox. For each j and k, sorting the traces v_i into two sets b_0 and b_1 based on the value of b. The mean trace is defined as

$$\bar{b}_{\{0,1\}} = \frac{\sum_{v \in b_{\{0,1\}}} v}{|b_{\{0,1\}}|}.$$

And the difference of means is calculated as

$$\Delta = |\bar{b}_0 - \bar{b}_1|.$$

Let Δ^j denote the difference of means trace obtained at j-th bit for a key hypothesis k^h. Let denote the best target bit which has the highest peak for k^h by $\Delta^{j'}$. Then $\Delta^{j'}$ is selected as the best difference of means trace for k^h and is denoted as Δ^h. The final step is to select the best difference which has the highest peak among all Δ^h and it is denoted as $\Delta^{h'}$. Such that the hypothesis $k^{h'}$ corresponding to $\Delta^{h'}$ is the most probably correct key.

2.5 The Improved Masking Method against DCA

To thwart DCA, Lee *et al.* presented the improved masking method [29] which applies the random masks to the round outputs. When building *TypeII*, the

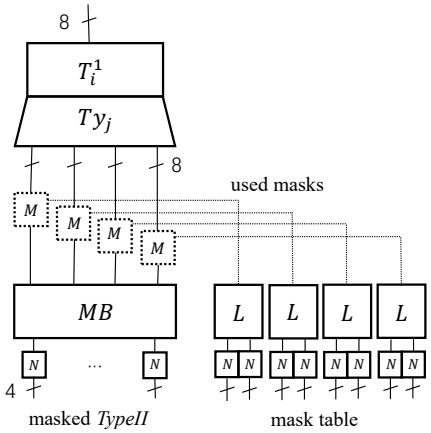


Fig. 1. *TypeII_MO* table in the first round. The input encoding is omitted because of the absence of external encoding.

masks are randomly picked for each input, and each output of Ty_j XOR with the random mask. The 8×8 linear transformations L are applied to the masks to form the mask table. The outputs of masked *TypeII* continue to feed the following tables of the first round (i.e., *TypeII_IV*, *TypeIII*, and *TypeIII_IV*). Both the masked *TypeII* and mask table are combined as *TypeII_MO* as shown in Fig. 1.

For clarity, let *vs* (value state) and *ms* (mask state) denote the round outputs of masked state and mask table itself, respectively. The *TypeII* table in the second round takes each corresponding byte of *vs* and *ms* as inputs. Thus, *vs* and *ms* are combined by XOR to unmask in the input decoding phase of the second round to form the *TypeII_MIMO* table as illustrated in Fig. 2. Since the

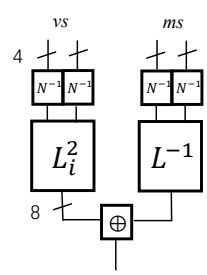


Fig. 2. The input decoding phase of *TypeII_MIMO* table in the second round. The following TMC_i^2 and the output encodings are omitted.

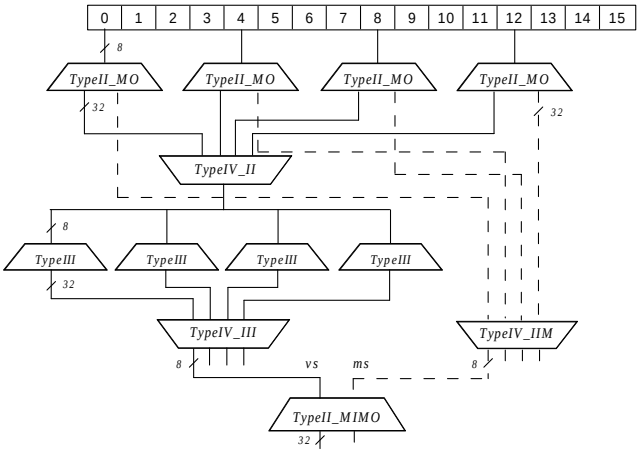


Fig. 3. LUT sequences of Lee *et al.*'s improved masking method.

unmasking is combined with the input decoding phase of *TypeII_MIMO* table in the second round, the intermediate values of the first round are all masked with the random masks. Fig. 3 shows the look-up sequence from the first column of plaintexts to the first entry of *TypeII_MIMO* in the second round. *TypeIV_IIM* represents a type of *TypeIV* table to XOR the outputs of mask table.

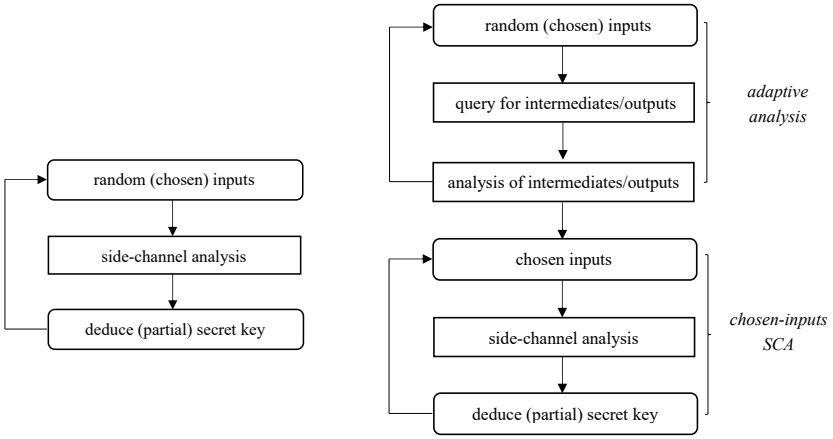
The solid line in Fig. 3 denotes the masked intermediate values in the first round while the dotted line represents the encoded masks. Hence, the collected traces of the encoded first-round Sbox are independent of the hypothetical values due to the presence of random masks. For a detailed description of the improved masking method, one can refer to the original proposal [29].

3 Adaptive Side-Channel Analysis Model and its Applications

In the previous section, we recall the two proposed countermeasures for protecting against DFA and DCA on CEJO-WBAES. The table redundancy method [28] exploits the white-box diversity to combine the redundant computations with fault detection. The improved masking technique [29] randomizes the output value of key-dependent LUTs before encodings. Although the two proposals can mitigate the original DFA and DCA attacks, the designer did not take into account the ability of a white-box adversary. For concerning a white-box attacker with the technique of SCA, an adaptive SCA model on WBC is proposed in this section.

3.1 The Adaptive Side-Channel Analysis Model on WBC

Compared with the algebraic attacks [7,17–19] which need to retrieve the encodings, SCA on WBC has been proved to reduce the time complexity of recovering



(a) SCA model (b) adaptive SCA model

Fig. 4. The flowchart of SCA and adaptive SCA models.

the secret key of the implementation with limited knowledge (e.g., side-channel information). The main benefits of SCA attack are that it do not need knowledge of particular implementation and the effort of reverse engineering. WBC is more vulnerable to SCA attacks even if it is intended to thwart a more powerful attack in the white-box setting (i.e., with full control of the implementation). The steps of an SCA attack in a white-box scenario can be informally described as follows, which are also illustrated in Fig. 4(a).

1. The adversary invokes the white-box implementation many times with random (chosen) inputs.
2. During each execution, the adversary performs a modeling analysis on the side-channel information (e.g., intermediate values) for deducing the (partial) secret key.

Note that the modification and record of side-channel information can be implemented with the help of *dynamic binary instrumentation* tools, such as Intel PIN [3]. For mitigating the SCA attacks, the side-channel countermeasure (e.g., table redundancy and improved masking) prevents the generalized SCA attacks by introducing a newly generated component (e.g., redundant computation and mask table). In practice, an adversary can extend the technique of SCA attacks with the powerful ability in the white-box attack context. An attacker can analyze the correlation between the original and newly generated component to collect the inputs which will contribute to a successful SCA attack. Such a new attack extends the efficiency of generalized SCA and is adapted to the dedicated countermeasure. Now, we introduce an *Adaptive Side-Channel Analysis* model to break the side-channel countermeasure for WBC implementations. The steps

of an adaptive SCA attack are divided into two phases: *adaptive analysis* and *chosen-inputs SCA* as informally described as follows, which are also illustrated in Fig. 4(b).

- *Adaptive analysis.* The adversary pinpoints the entry of a specific function and queries it with random (chosen) inputs for collecting intermediates/outputs. The adversary then performs an analysis on the intermediates/outputs to choose inputs for repeating query or for the following SCA attack.
- *Chosen-inputs SCA.* The adversary makes her choice of the inputs to the cryptographic algorithm and mounts the generalized SCA attacks to retrieve the secret key.

The attacker can pinpoint a dedicated region by the exploitation of *data dependency analysis* [22] and fault attacks [4, 5, 8]. The concrete steps and time complexity are related to the code obfuscation techniques used in the white-box implementation. Since the obscurity of the location of a function in a WBC implementation cannot provide the security on the algorithm itself, our adaptive SCA model assumes that an entry can be pinpointed with affordable complexity. After the adaptive analysis phase, chosen-inputs SCA can be mounted automatically with the help of practical SCA tools [35, 36]. Moreover, we note that an adaptive SCA adversary is capable of all known SCA attacks on WBC.

The main difference between SCA and adaptive SCA model is that an adaptive SCA attacker needs to obtain a set of target inputs before an SCA attack. The previous works [25, 34] described an adversary with the ability of adaptively choosing inputs to induce the faults or probe the intermediate variables. But they studied an attack in the gray-box model instead of the white-box model and are out of scope for this work. The adaptive SCA model in a gray-box setting is left as future work. In the next section, we will show the adaptive DFA and DCA attacks against Lee *et al.*'s table redundancy [28] and improved masking methods [29].

3.2 Adaptive DFA on the Table Redundancy Method

To break the table redundancy method [28], the fault needs to be simultaneously induced at both the original and redundant computations and thus their underlying states are the same value. Concerning the fault-free process of table redundancy method, the decoded states of both sides always keep the equal values since both of the two computations are the same AES encryption. Although the diversity of WBC is considered in the redundant design, the internal encodings are fixed when the implementation is running. So the intermediate input values of the ninth-round *TypeII* always appear pairwise. Let $y_{l_i} \in \mathbb{F}_2^8$ and $y_{r_i} \in \mathbb{F}_2^8$ ($i \in [0, 15]$) be an 8-bit input value of the two ninth-round *TypeII*, respectively. Let $x_i \in \mathbb{F}_2^8$ ($i \in [0, 15]$) be an 8-bit unencoded input state of the ninth round. The P_{l_i} and P_{r_i} ($i \in [0, 15]$) denote an 8-bit input decoding on $\mathbb{F}_2^8$ (i.e., Mixing Bijections and non-linear encodings) of y_{l_i} and y_{r_i}, respectively.

For the decoding process of the ninth-round inputs , we have

$$x_i = P_{l_i}(y_{l_i}) = P_{r_i}(y_{r_i}),\ i \in [0, 15].$$

Since x_i have 256 different values and the mappings of input decoding (i.e., P_{l_i} and P_{r_i}) are bijections, each pair of (y_{l_i}, y_{r_i}) can be collected from the corresponding inputs of the ninth round. Let $\mathcal{T}$ be a set of pairwise values, such that

$$\mathcal{T}_i = \{(y_{l_i}, y_{r_i}) \mid y_{l_i} \in \mathbb{F}_2^8,\ y_{r_i} \in \mathbb{F}_2^8\},\ with\ \#\mathcal{T}_i = 256,\ i \in [0, 15].$$

Based on the ninth-round DFA attack model [20], the fault can be injected at one of the four bytes of a column to recover the four-byte subkey in the tenth round. Such that $\mathcal{T}_j$ for $j = 0, 4, 8, 12$ are sufficient for obtaining the tenth-round key. For simplicity, the index j is discarded and the injection on one byte of the ninth round can be concluded by the following steps. The other locations of the ninth-round inputs are similar to this example.

1. Querying for the pairwise values at the original and redundant ninth-round inputs by repeatedly running the implementation with random plaintexts to form the set $\mathcal{T}$.
2. Getting a fault-free ciphertext by running with a random plaintext and denoting the pairwise values at the ninth-round inputs as (y_l, y_r).
3. Replacing (y_l, y_r) with other pairs in $\mathcal{T} \backslash (y_l, y_r)$ which are denoted as (y_l^*, y_r^*) and collecting the faulty ciphertexts by running the cryptographic program with the same plaintext.
4. Repeating Step 3 and using the DFA tools to perform the analysis between the fault-free and faulty ciphertexts.

Let $x^* \in \mathbb{F}_2^8$ be the underlying state of the faulty pairs (y_l^*, y_r^*). The decoding P_l and P_r are fixed into the parts of LUTs in the encryption, thus x will be tampered as x^* if (y_l, y_r) are replaced with (y_l^*, y_r^*). The modification from x to x^* represents that the faults are injected successfully at the ninth-round inputs. This process simultaneously modifies the underlying states of the two computations into other ones by replacing the pairwise values at the ninth-round inputs with the other pairs in $\mathcal{T}$. In this way, MB can be combined in the XOR process since the decoded states after the ninth-round TMC are identical to each other between two computations. Note that $\mathcal{T}$ can be collected with overwhelming probability because of the splendid confusion and diffusion of the first 8-round AES. Concerning the practical analysis, the subkey can be recovered by only two faults in the same location, thus $\mathcal{T}$ need not be a full set. In this way, the adaptive DFA attack on the table redundancy method exploits the replacement on the ninth-round inputs to bypass the elaborately designed XOR phases of *TypeII*. Thus, the vulnerability of table redundancy under DFA attack is identical to CEJO-WBAES [16]. The adaptive DFA can also be extended to break a table redundant white-box implementation with 8-bit external encodings since we assume that the attacker knows the technique of existing DFA attack [4].

3.3 Adaptive DCA on the Improved Masking Method

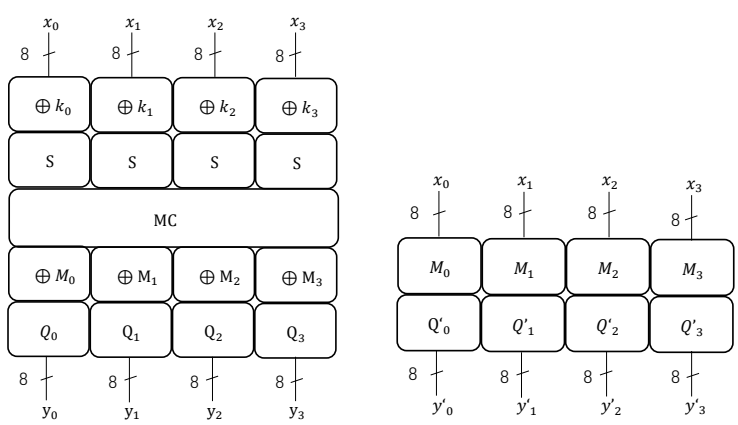
Since the improved masking conceals the key-dependent outputs of the first round by introducing the random masks, DCA fails to analyze the correlation between the intermediate and hypothesis values. In the following text, Q_i and Q'_i for $0 \leq i \leq 3$ denote bijective mapping on $\mathbb{F}_2^8$ and are referred to as output encodings of *vs* and *ms* (refer to Section 2.5), respectively. Let $y_i, y'_i : (\mathbb{F}_2^8)^4 \rightarrow \mathbb{F}_2^8$ for $0 \leq i \leq 3$ be the functions of the mappings from plaintexts to *vs* and *ms*, $k_i \in \mathbb{F}_2^8$ for $0 \leq i \leq 3$ be the subkeys in the first round, and $(x_0, x_1, x_2, x_3) \in (\mathbb{F}_2^8)^4$ denote the first column of plaintexts. Such that the function of *vs* in the first column which is depicted by solid line in Fig. 3 can be shown as follows.

$$\begin{aligned} y_i(x_0, x_1, x_2, x_3) = Q_i \circ (&mc_{i,0} \cdot S(x_0 \oplus k_0) \oplus mc_{i,1} \cdot S(x_1 \oplus k_1) \\ &\oplus mc_{i,2} \cdot S(x_2 \oplus k_2) \oplus mc_{i,3} \cdot S(x_3 \oplus k_3) \oplus M_i). \end{aligned}$$

The corresponding function of *ms* which is depicted by the dotted line in Fig. 3 can be represented in below.

$$y'_i(x_0, x_1, x_2, x_3) = Q'_i \circ (M_i).$$

Note that M_i denote the mask used and take all the values on $\mathbb{F}_2^8$, mc_i denote the MixColumns coefficient, $0 \leq i \leq 3$ denotes the index of states. The functions of *vs* and *ms* are also illustrated in Fig. 5(a) and 5(b), respectively.



(a) $y_i(x_0, x_1, x_2, x_3)$ (b) $y'_i(x_0, x_1, x_2, x_3)$

Fig. 5. The mapping function from plaintexts to (a) *vs* and (b) *ms* in the first column.

Mentioned by Lee *et al.* [29], each M_i is independently generated for different inputs such that its randomness and uniformity help to mask the key-dependent output of y_i. However, since M_i need to be annihilated between the outputs of y_i and y'_i, the underlying M_i used in the function of *vs* and *ms* for the same

(x_0, x_1, x_2, x_3) are identical to each other. Due to the fact that the encodings Q'_i are fixed bijective mapping, a collision in an encoded byte $Q'_i(x)$ corresponds with a collision in the decoded byte x as well. Thus, one can sort the inputs by the identical outputs of y'_i, i.e., to get the set

$$\mathcal{P}_i = \{(x_0, x_1, x_2, x_3) \mid y'_i(x_0, x_1, x_2, x_3) = c\},$$

where c is a constant and i denotes the index of entry. For simplicity, the following analysis takes $i = 0$. Based on the 2-byte key guessing model [33], we suppose that $(\alpha, \beta, 0, 0), (\alpha', \beta', 0, 0) \in \mathcal{P}_0$ such that $y'_0(\alpha, \beta, 0, 0) = y'_0(\alpha', \beta', 0, 0)$, which also implies that $Q'_0(M_0) = Q'_0(M'_0)$. In this way, $M_0 = M'_0$, thus $(\alpha, \beta, 0, 0)$ and $(\alpha', \beta', 0, 0)$ have the same mask value M_0. Once the underlying mask of y_i are constant for different plaintexts, DCA can recover the secret key for analyzing the correlation between the computation traces and hypothesis outputs. The detailed attack can be represented by the following steps on recovering the first two bytes of the first-round key. The steps for other key bytes are similar to this one.

1. Querying for the plaintexts as the set $\mathcal{P}_0$ which have the same output of y'_0 by enumerating the first two bytes of the inputs and fixing the other bytes of y'_0 (e.g., $y'_0(x_0, x_1, 0, 0)$).
2. Collecting the bit traces v which include the values of the outputs of y_0 by choosing the plaintexts in $\mathcal{P}_0$.
3. Selecting the XOR of the first two outputs of MixColumns as the hypothetical value b, that is

$$b = mc_{i,0} \cdot S(x_0 \oplus k'_0) \oplus mc_{i,1} \cdot S(x_1 \oplus k'_1),$$

   where k'_0 and k'_1 are key guesses.
4. Mounting the DCA attack on analyzing the correlation between v and b to recover k_0 and k_1.

For k_2 and k_3, the attack needs to enumerate x_2 and x_3 and fix the other bytes. The key recovery of other columns is similar to this example. Note that the collected y'_0 need not be set as a pre-defined value since any constant on $\mathbb{F}_2^8$ can help to sort the plaintexts. For a practical attack, the number of plaintexts in $\mathcal{P}_0$ depends on the number of traces that are exploited by the original DCA.

Suppose that the mask value is fixed as a constant m, such that

$$y_0(x_0, x_1, 0, 0) = Q_0 \circ (mc_{i,0} \cdot S(x_0 \oplus k_0) \oplus mc_{i,1} \cdot S(x_1 \oplus k_1) \oplus \gamma \oplus m),$$

where the constant $\gamma = mc_{i,2} \cdot S(0 \oplus k_2) \oplus mc_{i,3} \cdot S(0 \oplus k_3)$. Note that the XOR phase of constant can be combined with the linear part of the encoding Q_0 to form a new encoding $\widetilde{Q}_0$. Thus, $y_0(x_0, x_1, 0, 0) = \widetilde{Q}_0 \circ (mc_{i,0} \cdot S(x_0 \oplus k_0) \oplus mc_{i,1} \cdot S(x_1 \oplus k_1))$. Because of the ineffectiveness of internal encodings [10], the strong correlations can be computed between v and b.

The adaptive DCA attack on the improved masking method utilizes the collision on y' to sort the plaintexts which have the same mask. Such that the elaborately designed masking can be bypassed since a constant mask cannot prevent the leakage of a secret key. In this way, the vulnerability of improved masking under DCA attack is identical to CEJO-WBAES [16].

4 Theoretical Analysis and Experimental Results

This section performs the practical attacks on table redundancy and improved masking based on the adaptive SCA model. Following such a model, a white-box adversary can directly locate and modify any intermediate values in an implementation. Such that for efficiency, the experiment collects the target values during the table look-up operations by modifying the corresponding source code of encryption. Our attack is conducted on a PC with Intel Core i5-6200U processor @2.3 GHz, 12GB RAM. The compiler for building a shared library is GCC 8.2.0, while "-O2" optimization is enabled. All the counts have been measured 100,000 times to get the averages. For verifying our results, the crucial components of our experiments are open-sourced in

`https://github.com/scnucrypto/Adaptive-SCA`.

4.1 Results of the Adaptive DFA on the Table Redundancy Method

The experiment firstly generates the sets $\mathcal{T}_j$ for $j = 0, 4, 8, 12$ consisting of pairwise bytes of the ninth-round inputs between which one is the original state and the other is the redundant one. The result shows that each $\mathcal{T}_j$ can be fully filled up by 1,548 executions, encrypting with random plaintexts. Due to the fact that two faults injection at the first byte of a column in the ninth round can help to recover a 4-byte subkey of the tenth round, $\mathcal{T}_j$ can only be collected by three pairwise states instead of a full set. Note that, the first element of $\mathcal{T}_j$ can be found by the first encryption and the corresponding plaintext can be recorded as the one for the fault injection. Such that the replacement for adaptive DFA attack only relies on another two elements of $\mathcal{T}_j$ which can be collected by extra 2 executions with random plaintexts based on our further experiment. The correlation between the number of plaintexts and the number of the adaptively chosen elements of $\mathcal{T}_j$ is illustrated in Fig. 6. Note that each $\mathcal{T}_j$ has 256 elements at most.

Subsequently, the pairwise states at the first byte in each of four columns are collected to form four corresponding $\mathcal{T}_j$. The adaptive replacement between the original pairwise state at ninth-round inputs and the elements in $\mathcal{T}_j$ is performed to yield a result as the fault injection. After the fault-free ciphers and faulty ciphertexts are recorded, the tools Jean Grey [36] is invoked to solve the differential equations and recover the tenth-round key. The main key can be obtained from the tenth-round key by using the key scheduling reversers supported by Stark [37] tools. To sum up, at least 5 executions (3 times for collecting $\mathcal{T}_j$ and 2 times for the replacement of fault injection) of the white-box implementation can recover the 4-byte subkey of the tenth round. Totally at least 20 executions of encryption can extract the 16-byte tenth-round key.

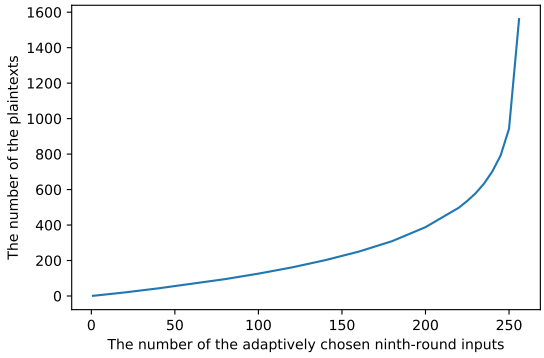


Fig. 6. The number of the plaintexts used to adaptively choose the pairwise ninth-round inputs to form the elements of $\mathcal{T}_j$.

4.2 Results of the Adaptive DCA on the Improved Masking Method

The experiment focuses on collecting $\mathcal{P}_0$ as an example. The collision of y'_0 for $(x_0, x_1, 0, 0)$ can help to choose the plaintexts of which the mask value are identical to each other. Such that DCA can recover k_0 and k_1 based on the 2-byte key guessing model [33]. Similarly, the collision of $y'_0(0, 0, x_2, x_3)$ for all x_2 and x_3 helps to choose the plaintexts to retrieve k_2 and k_3 by DCA. Note that the collision relies on the mapping on $(\mathbb{F}_2^8)^2 \rightarrow \mathbb{F}_2^8$. Based on the randomness and uniformity of the mask, for each constant $c \in \mathbb{F}_2^8$, there are 256 possible inputs for $y'_0(x_0, x_1, 0, 0) = c$. This implies that 256 plaintexts (also 256 traces with the same mask) at most help to recover a 2-byte key. The result shows that 53,884 executions with the enumeration of $(x_0, x_1) \in (\mathbb{F}_2^8)^2$ can collect all 256 inputs which map to a same constant through the function y'_0 to form the set $\mathcal{P}_0$. Subsequently, the elements in $\mathcal{P}_0$ are adaptively chosen as plaintexts for the white-box implementation and thus the DCA attack can be mounted by the tools [21, 35].

In practice, a 2-byte key leakage attack can be captured under DCA by analyzing less than 256 traces. Our experimental result shows that at least 25 plaintexts in $\mathcal{P}_0$ can successfully help to recover the 2-byte key with the traces that are composed of y_0. Fig. 7 shows the relation between the number of the elements in $\mathcal{P}_0$ and the number of required plaintexts. The plaintexts are adaptively chosen by enumerating $(x_0, x_1) \in (\mathbb{F}_2^8)^2$. Note that each $\mathcal{P}_0$ has 256 elements at most because of the uniform distribution with random mask. As illustrated in Fig. 7, nearly 3,240 executions of encryption can help to collect 25 elements of $\mathcal{P}_0$. Note that each 2-byte key is obtained by DCA with chosen 4-byte plaintexts. Such that, in summary, at least about 3,265 executions (3,240 times for obtaining $\mathcal{P}_j$ corresponding to four 4-byte plaintexts and 25 times for collecting computation traces which comprises the outputs of y_j, for $j = 0, 4, 8, 12$) of the

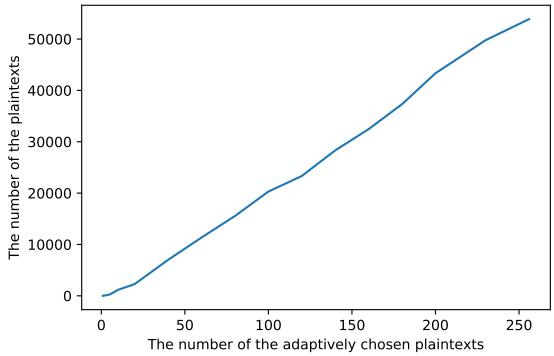


Fig. 7. The number of the plaintexts used to adaptively choose the target plaintexts which have a same mask value in y_0 to form the elements of $\mathcal{P}_0$.

white-box implementation can recover four 2-byte subkeys of the first round. Totally at least about 6,530 executions of encryption can extract the 16-byte first-round key.

5 Adaptive SCA Model on Higher-order Countermeasures

In [28], Lee *et al.* also discussed an enhancement on the security of the table redundancy method for protecting against DFA. The proposal increases the number of redundant computations for reducing the probability of making a fault collision in which the two disturbed bytes will be decoded into the same value. Such an extended redundancy is introduced as the higher-order version of the table redundancy method in this paper. In the following section, we describe the higher-order adaptive DFA against extended redundancy and also evaluate the higher-order adaptive DCA on cryptanalyzing the improved masking in a higher-order manner.

5.1 Adaptive DFA against the Higher-order Table Redundancy Method

A higher-order table redundancy consists of more than one redundant computation to enhance the security for protecting against DFA attack. Let n be the number of redundant computations, x_0 and $x_i \in \mathbb{F}_2^8$ for $i = 1, ..., n$ be the original and redundant states at the first byte of the ninth-round inputs, respectively. $x \in \mathbb{F}_2^8$ denotes the corresponding state of standard AES. The XOR phase of the higher-order table redundancy can be shown as follows, where $\mathrm{MB_i}$, $i = 0, ..., n$ represent the Mixing Bijections of each *TypeII* table.

$$\mathrm{MB}_0 \cdot \mathrm{TMC}(x_0) \oplus ... \oplus \mathrm{MB}_n \cdot \mathrm{TMC}(x_n) = \\ (\mathrm{MB}_0 \oplus ... \oplus \mathrm{MB}_n) \cdot \mathrm{TMC}(x), \text{ iff } x_0 = ... = x_n = x.$$

Note that $\mathrm{MB_i}$ can be combined over $\oplus$ because of their linear property and each x_i is identical to each other. This implies that the faults need to be injected simultaneously on every x_i so that the combination of the original and redundant computations still can induce the available faulty ciphers. The higher-order adaptive DFA attack exploits the sets of pre-collected intermediate values at the ninth-round inputs to implement a replacement between the fault-free and faulty states. The general steps of such an attack are similar to the adaptive DFA on one redundant computation and the differences are twofold. (1) The set $\mathcal{T}$ consists of the $(n+1)$-tuple values of the original and n redundant ninth-round inputs by repeatedly encrypting random plaintexts. (2) The fault is injected through the adaptive replacement between the original $(n+1)$-tuple value and the replaced one from $\mathcal{T}$. Since $\mathcal{T}$ is generated by collecting the fault-free intermediate values, the underlying states of each value among $n+1$ tuples are the same one. In this way, the replacement between each $(n+1)$-tuple value has the same effect as tampering the state to another one. Note that such fault injection takes place in the ninth-round inputs which are not affected by the XOR phase of table redundancy. Thus, the higher-order adaptive DFA attacker can recover the secret key by DFA tools [36] to analyze the faulty ciphertexts obtained by the replacement with $\mathcal{T}$. Consequently, the higher-order redundant computation cannot enhance security since it still cannot protect against the replacement on values at the ninth-round inputs. The required times of encryption for the adaptive DFA on higher-order table redundancy does not increase due to the identical collection as the attack on the one redundant computation.

5.2 Adaptive DCA against Higher-order Improved Masking Method

A higher-order improved masking consists of more than one mask to enhance the security for protecting against DCA attack. In the following texts, the first entries of *vs* and *ms* (refer to Section 2.5) in the first column are still analyzed as an example to recover the first two bytes of the first-round key. Because of the presence of higher-order masking, the functions of *ms* consist of n mask tables. With slight abuse of notation, $y : (\mathbb{F}_2^8)^4 \rightarrow \mathbb{F}_2^8$ denotes the function of *vs*, $y'_i : (\mathbb{F}_2^8)^4 \rightarrow \mathbb{F}_2^8$ for $0 \leq i \leq n-1$ denotes the functions of *ms*, $\mathcal{P}$ represents the set of chosen plaintexts. Note that the index 0 of entry is discarded for y, y', and $\mathcal{P}$. The subscript i of y' is redefined as the index of mask. Let m_i for $0 \leq i \leq n-1$ be the mask used, such that the function of *vs* can be shown as follows.

$$\begin{aligned} y(x_0, x_1, x_2, x_3) = Q \circ (&mc_{0,0} \cdot S(x_0 \oplus k_0) \oplus mc_{0,1} \cdot S(x_1 \oplus k_1) \\ &\oplus mc_{0,2} \cdot S(x_2 \oplus k_2) \oplus mc_{0,3} \cdot S(x_3 \oplus k_3) \oplus m_0 \oplus ... \oplus m_{n-1}). \end{aligned}$$

The corresponding functions of *ms* can be represented in below.

$$y'_i(x_0, x_1, x_2, x_3) = Q'_i \circ (m_i).$$

Note that Q and Q'_i represent the output encodings of *vs* and *ms* on $\mathbb{F}_2^8$, respectively. To bypass the effect of higher-order masking, the bit traces need

to be collected by choosing the plaintexts of which the underlying masks are identical to each other. This implies that for each element $(x_0, x_1, x_2, x_3) \in (\mathbb{F}_2^8)^4$ of the target set $\mathcal{P}$, their m_i in y'_i are identical to each other. Such that $m_0 \oplus ... \oplus m_{n-1}$ of the elements in $\mathcal{P}$ is a same value. Let $c_i \in \mathbb{F}_2^8$ for $0 \leq i \leq n-1$ denote constant values. Now the set S can be redefined as follows.

$$\mathcal{P} = \{(x_0, x_1, x_2, x_3) \mid y'_0(x_0, x_1, x_2, x_3) = c_0, \\ ..., y'_{n-1}(x_0, x_1, x_2, x_3) = c_{n-1}\}.$$

The higher-order adaptive DCA attack needs to successively find the collisions of y'_i to obtain the target plaintexts $(x_0, x_1, x_2, x_3) \in \mathcal{P}$. Subsequently, the attacker can mount a successful DCA attack by the chosen plaintexts in $\mathcal{P}$ and collecting the software traces which include the value of y. Table 2 illustrates the number of the executions of y'_i and the number of corresponding collected plaintexts. The result shows that, to attack a third-order improved masking, $2^{32} + 2^{24} + 2^{16}$ executions of y'_i can obtain 2^8 plaintexts which yield the same masks m_i by making the collision on outputting the same value c_i for $i = 0, 1, 2$. This process requires the adaptive DCA attacker to collect the plaintexts from y'_0 to y'_2. The resulting 2^8 plaintexts have the same masks m_0, m_1, and m_2. Such that DCA can retrieve the secret key with the chosen plaintexts because of the constant mask. It is worth noting that, the higher-order improved masking is not practical for white-box implementations because the multiple inputs of *TypeII_MIMO* (refer to Fig. 2) will result in an exponential increase in the space footprint.

Table 2. The number of executions of $y'_i(x_0, x_1, x_2, x_3)$ to yield the same mask m_i for $i = 0, 1, 2$ and the number of corresponding collected plaintexts.

	m_0 (First-order)	m_1 (Second-order)	m_2 (Third-order)
Executions of y'_i	2^{32}	2^{24}	2^{16}
Collected plaintexts	2^{24}	2^{16}	2^8

6 Conclusion

In this paper, a novel SCA model has been proposed for introducing the *adaptive analysis* and *chosen-inputs SCA* phases to traditional SCA attacks on WBC. The new adaptive model is applied to Lee *et al.*'s improved countermeasures. For the practical security of WBC, our results motivate to explore new SCA countermeasures on WBC by concerning the abilities of adaptive SCA attacker. In future work, it is interesting to build an adaptive algebraic analysis model to break the state-of-the-art countermeasures of algebraic attacks (e.g., BGE attack) on WBC.

Acknowledgments. We are grateful to the anonymous reviewers for their insightful comments. This work was supported in part by National Key R&D Program of China (2020AAA0107700), National Natural Science Foundation of China (62072192, 62072398), and National Cryptography Development Fund (MMJJ20180206). Fan Zhang was also supported by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, by Zhejiang Key R&D Plan (2019C03133).

References

1. CHES 2017 capture the flag challenge - the WhibOx contest, an ecrypt white-box cryptography competition. https://whibox.io/contests/2017/, accessed June 1, 2021
2. CHES 2019 capture the flag challenge - the WhibOx contest edition 2. https://whibox.io/contests/2019/, accessed June 1, 2021
3. Pin - a dynamic binary instrumentation tool. https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html, accessed June 1, 2021
4. Amadori, A., Michiels, W., Roelse, P.: A DFA attack on white-box implementations of AES with external encodings. In: International Conference on Selected Areas in Cryptography. pp. 591–617. Springer (2019). https://doi.org/10.1007/978-3-030-38471-5_24
5. Amadori, A., Michiels, W., Roelse, P.: Automating the BGE attack on white-box implementations of AES with external encodings. In: 2020 IEEE 10th International Conference on Consumer Electronics (ICCE-Berlin). pp. 1–6. IEEE (2020). https://doi.org/10.1109/ICCE-Berlin50680.2020.9352195
6. Banik, S., Bogdanov, A., Isobe, T., Jepsen, M.: Analysis of software countermeasures for whitebox encryption. IACR Transactions on Symmetric Cryptology pp. 307–328 (2017). https://doi.org/10.13154/tosc.v2017.i1.307-328
7. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white box AES implementation. In: International Workshop on Selected Areas in Cryptography. pp. 227–240. Springer (2004). https://doi.org/10.1007/978-3-540-30564-4_16
8. Biryukov, A., Udovenko, A.: Attacks and countermeasures for white-box designs. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 373–402. Springer (2018). https://doi.org/10.1007/978-3-030-03329-3_13
9. Bock, E.A., Bos, J.W., Brzuska, C., Hubain, C., Michiels, W., Mune, C., Gonzalez, E.S., Teuwen, P., Treff, A.: White-box cryptography: don't forget about grey-box attacks. Journal of Cryptology **32**(4), 1095–1143 (2019). https://doi.org/10.1007/s00145-019-09315-1
10. Bock, E.A., Brzuska, C., Michiels, W., Treff, A.: On the ineffectiveness of internal encodings-revisiting the DCA attack on white-box cryptography. In: International Conference on Applied Cryptography and Network Security. pp. 103–120. Springer (2018). https://doi.org/10.1007/978-3-319-93387-0_6
11. Bock, E.A., Treff, A.: Security assessment of white-box design submissions of the CHES 2017 CTF challenge. IACR Cryptol. ePrint Arch. **2020**, 342 (2020). https://eprint.iacr.org/2020/342

12. Bogdanov, A., Rivain, M., Vejre, P.S., Wang, J.: Higher-order DCA against standard side-channel countermeasures. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 118–141. Springer (2019). https://doi.org/10.1007/978-3-030-16350-1_8
13. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: Hiding your white-box designs is not enough. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 215–236. Springer (2016). https://doi.org/10.1007/978-3-662-53140-2_11
14. Bringer, J., Chabanne, H., Dottax, E.: White box cryptography: Another attempt. IACR Cryptology ePrint Archive **2006**(2006), 468 (2006). https://eprint.iacr.org/2006/468
15. Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: ACM Workshop on Digital Rights Management. pp. 1–15. Springer (2002). https://doi.org/10.1007/978-3-540-44993-5_1
16. Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: White-box cryptography and an AES implementation. In: International Workshop on Selected Areas in Cryptography. pp. 250–270. Springer (2002). https://doi.org/10.1007/3-540-36492-7_17
17. De Mulder, Y., Roelse, P., Preneel, B.: Cryptanalysis of the Xiao–Lai white-box AES implementation. In: International Conference on Selected Areas in Cryptography. pp. 34–49. Springer (2012). https://doi.org/10.1007/978-3-642-35999-6_3
18. De Mulder, Y., Roelse, P., Preneel, B.: Revisiting the BGE attack on a white-box AES implementation. IACR Cryptology ePrint Archive **2013**, 450 (2013). https://eprint.iacr.org/2013/450
19. De Mulder, Y., Wyseur, B., Preneel, B.: Cryptanalysis of a perturbated white-box AES implementation. In: International Conference on Cryptology in India. pp. 292–310. Springer (2010). https://doi.org/10.1007/978-3-642-17401-8_21
20. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on AES. In: International Conference on Applied Cryptography and Network Security. pp. 293–306. Springer (2003). https://doi.org/10.1007/978-3-540-45203-4_23
21. Fakub: White-box DPA processing: Scripts for trace acquisition, filtering, processing and displaying results. https://github.com/fakub/White-Box-DPA-Processing, accessed June 1, 2021
22. Goubin, L., Paillier, P., Rivain, M., Wang, J.: How to reveal the secrets of an obscure white-box implementation. Journal of Cryptographic Engineering pp. 1–18 (2019). https://doi.org/10.1007/s13389-019-00207-5
23. Goubin, L., Rivain, M., Wang, J.: Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 454–482 (2020). https://doi.org/10.13154/tches.v2020.i3.454-482
24. Karroumi, M.: Protecting white-box AES with dual ciphers. In: International Conference on Information Security and Cryptology. pp. 278–291. Springer (2010). https://doi.org/10.1007/978-3-642-24209-0_19
25. Keren, O., Polian, I.: IPM-RED: combining higher-order masking with robust error detection. Journal of Cryptographic Engineering pp. 1–14 (2020). https://doi.org/10.1007/s13389-020-00229-4
26. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual international cryptology conference. pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25

27. Lee, S., Jho, N.S., Kim, M.: On the linear transformation in white-box cryptography. IEEE Access **8**, 51684–51691 (2020). https://doi.org/10.1109/ACCESS.2020.2980594
28. Lee, S., Jho, N.S., Kim, M.: Table redundancy method for protecting against fault attacks. IEEE Access **9**, 92214–92223 (2021). https://doi.org/10.1109/ACCESS.2021.3092314
29. Lee, S., Kim, M.: Improvement on a masked white-box cryptographic implementation. IEEE Access **8**, 90992–91004 (2020). https://doi.org/10.1109/ACCESS.2020.2993651
30. Lee, S., Kim, T., Kang, Y.: A masked white-box cryptographic implementation for protecting against differential computation analysis. IEEE Transactions on Information Forensics and Security **13**(10), 2602–2615 (2018). https://doi.org/10.1109/TIFS.2018.2825939
31. Maghrebi, H., Alessio, D.: Revisiting higher-order computational attacks against white-box implementations. IACR Cryptol. ePrint Arch. **2019**, 1405 (2019). https://eprint.iacr.org/2019/1405
32. Muir, J.A.: A tutorial on white-box AES. Advances in network analysis and its applications pp. 209–229 (2012). https://doi.org/10.1007/978-3-642-30904-5_9
33. Rivain, M., Wang, J.: Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 225–255 (2019). https://doi.org/10.13154/tches.v2019.i2.225-255
34. Seker, O., Eisenbarth, T., Liskiewicz, M.: A white-box masking scheme resisting computational and algebraic attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 61–105 (2021). https://doi.org/10.46586/tches.v2021.i2.61-105
35. Side-Channel-Marvels: Deadpool: Repository of various public white-box cryptographic implementations and their practical attacks. https://github.com/SideChannelMarvels/Deadpool, accessed June 1, 2021
36. Side-Channel-Marvels: JeanGrey: A tool to perform differential fault analysis attacks (DFA). https://github.com/SideChannelMarvels/JeanGrey, accessed June 1, 2021
37. Side-Channel-Marvels: Stark: Repository of small utilities related to key recovery. https://github.com/SideChannelMarvels/Stark, accessed June 1, 2021
38. Teuwen, P., Hubain, C.: Differential fault analysis on white-box AES implementations. https://blog.quarkslab.com/differential-fault-analysis-on-white-box-aes-implementations.html, accessed June 1, 2021
39. Xiao, Y., Lai, X.: A secure implementation of white-box AES. In: 2009 2nd International Conference on Computer Science and its Applications. pp. 1–6. IEEE (2009). https://doi.org/10.1109/CSA.2009.5404239
40. Zeyad, M., Maghrebi, H., Alessio, D., Batteux, B.: Another look on bucketing attack to defeat white-box implementations. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 99–117. Springer (2019). https://doi.org/10.1007/978-3-030-16350-1_7