Cryptanalysis of a Perturbated White-Box AES Implementation

Yoni De Mulder[1], Brecht Wyseur[2], and Bart Preneel[1]

[1] Katholieke Universiteit Leuven
Dept. Elect. Eng.-ESAT/SCD-COSIC and IBBT,
Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
{yoni.demulder,bart.preneel}@esat.kuleuven.be
[2] Nagravision S.A.
Route de Genève 22–24, 1033 Cheseaux-sur-Lausanne, Switzerland
brecht.wyseur@nagra.com

Abstract. In response to various cryptanalysis results on white-box cryptography, Bringer *et al.* presented a novel white-box strategy. They propose to extend the round computations of a block cipher with a set of random equations and perturbations, and complicate the analysis by implementing each such round as one system that is obfuscated with annihilating linear input and output encodings. The improved version presented by Bringer *et al.* implements the AEw/oS, which is an AES version with key-dependent S-boxes (the S-boxes are in fact the secret key). In this paper we present an algebraic analysis to recover equivalent keys from the implementation. We show how the perturbations and system of random equations can be distinguished from the implementation, and how the linear input and output encodings can be eliminated. The result is that we have decomposed the white-box implementation into a much more simple, functionally equivalent implementation and retrieved a set of keys that are equivalent to the original key. Our cryptanalysis has a worst time complexity of 2^{17} and a negligible space complexity.

Keywords: White-Box Cryptography, AES, Cryptanalysis, Structural Cryptanalysis.

1 Introduction

In the past decade, we have witnessed a trend towards the use of software applications with strong security requirements. Consider for example online banking and digital multimedia players. Building blocks to enable their security include cryptographic primitives such as the DES or the AES [13]. However, these building blocks are designed to be secure only when they are executed on a trustworthy system, which is typically no longer a valid assumption. White-box cryptography aims to address this issue – it aims to implement a given cryptographic cipher such that it remains 'secure' even when the adversary is assumed to have full access to the implementation and its execution environment (the white-box

G. Gong and K.C. Gupta (Eds.): INDOCRYPT 2010, LNCS 6498, pp. 292–310, 2010.

attack context). We refer to a white-box implementation as an implementation of a cipher to which these techniques are applied.

At SAC 2002, Chow *et al.* introduced the concept of white-box cryptography, applied to the AES [5], and to the DES in [6]. The main idea is to generate a network of re-randomized lookup tables that is functionally equivalent to a key-instantiated primitive. However, subsequent papers have shown that this strategy is prone to differential cryptanalysis [10,11,9,16] and algebraic cryptanalysis [2,12,15]. In [1], Billet and Gilbert proposed a traceable block cipher, by implementing the same instance of a cipher in many different ways. The security is based on the Isomorphisms of Polynomials (IP) problem [14]. Unfortunately, analysis of this IP problem [8] has defeated this approach. Based on the idea to introduce perturbations to reinforce the IP-based cryptosystems [7], Bringer *et al.* [3] reinforced the traceable block cipher, and presented a *perturbated* white-box AES implementation [4]. The main idea of the perturbated AES white-box implementations is to extend the AES rounds with a random system of equations and perturbation functions. The perturbations introduced at the first round are canceled out at the final round with a high probability; to guarantee correct execution, several such instances need to be implemented such that a majority vote can reveal the correct result. The random system of equations is discarded in the final round. As a challenging example, they apply their technique to the AEw/oS – a variant of the AES with non-standard, key-dependent S-boxes. These S-boxes are in fact the secret key. To the best of our knowledge, no attack against this white-box AEw/oS implementation has been proposed so far.

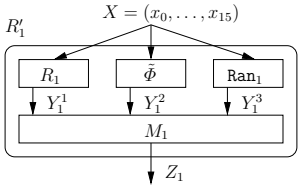
Our contribution. We developed a cryptanalysis of the perturbated white-box AEw/oS implementation; which extends naturally to perturbated white-box AES implementations. In a white-box attack context, the adversary will have access to each of the (obfuscated) rounds – these consist of the composition of random linear input and output encodings, the AES round operation with key-dependent S-boxes, and encompass the random system of equations and perturbated functions. The presence of the (unknown) linear encodings and the extra equations makes it hard to recover the secret information – the S-boxes – from the implementation.

In this paper, we describe the structural analysis of the white-box AEw/oS round operations. We show how to derive a set of equivalent S-boxes and linear encodings that describe a functionally equivalent implementation (due to the construction of the implementation, there are many candidate keys). As a result, we obtain a significantly simpler version of the white-box AEw/oS implementation, which is also invertible, thus defeating the security objective of the original implementation. Our cryptanalysis has a worst time complexity of 2^{17} and a negligible space complexity.

Organization of the paper. To support the cryptanalysis description, in Sect. 2, we present an overview of the perturbated white-box AEw/oS implementation as presented by Bringer *et al.* [4]. The cryptanalysis comprises three main steps, which are presented in Sect. 3.

2 The White-Box AEw/oS Implementation

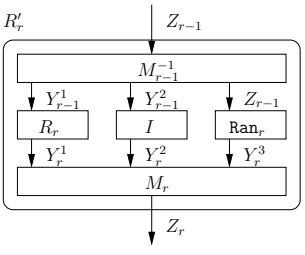
In this section, we describe the perturbated white-box AEw/oS implementation, which is the Advanced Encryption Standard (AES) [13] with non-standard S-boxes – the choice of S-boxes is in fact the secret key, and there are 160 of them comprised in the entire implementation. Fig. 1 depicts the implementation, where X is the plaintext input, Z_r are the intermediate states (outputs of the perturbated round functions R'_r), where Z_{10} is the final output.



1.1: Perturbated first round R'_1.

(a) AddRoundKey K_0
(b) SubBytes $\{S_{1,0}, \ldots, S_{1,15}\}$
(c) ShiftRows
(d) MixColumns

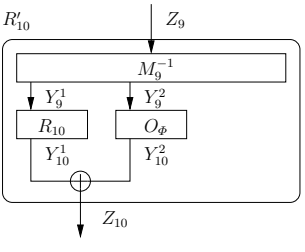
1.2: Original first round R_1.



1.3: Perturbated intermediate rounds R'_r.

for r from 2 to 9:
(a) AddRoundKey K_{r-1}
(b) SubBytes $\{S_{r,0}, \ldots, S_{r,15}\}$
(c) ShiftRows
(d) MixColumns

1.4: Original intermediate rounds R_r.



1.5: Perturbated final round R'_{10}.

(a) AddRoundKey K_9
(b) SubBytes $\{S_{10,0}, \ldots, S_{10,15}\}$
(c) ShiftRows
(d) AddRoundKey K_{10}

1.6: Original final round R_{10}.

Fig. 1. Description of the Perturbated White-box AEw/oS Implementation

The perturbated round functions. Each round R'_r of the perturbated AEw/oS is expressed as a system of 43 multivariate polynomials over GF $\left(2^8\right)$; the final round as a system of 16 multivariate polynomials. Each system is defined over 43 variables (bytes), except for the initial round, which is defined over the 16 bytes of the plaintext. These extra variables and equations are due to the extension of the AES rounds with a perturbation system of 4 polynomials and a system

of 23 random polynomials $\texttt{Ran}_r$. The latter is introduced to dissimulate the perturbation and mask all internal operations.

The perturbation initialization system $\tilde{\Phi}$ is included in the first round R'_1 and comprises of 4 polynomials that "often" take the predefined value $(\varphi_1, \varphi_2, \varphi_3, \varphi_4)$ and is constructed as $\tilde{\Phi}(X) = (\tilde{0}(X) + \varphi_1, \tilde{0}(X) + \varphi_2, \tilde{0}(X) + \varphi_3, \tilde{0}(X) + \varphi_4)$, where the $\tilde{0}$-polynomial "often" vanishes.[1] The 4-byte output of $\tilde{\Phi}(X)$ is then carried through all intermediate rounds to ensure that all intermediate values Z_r are perturbated and all rounds are closely linked. These perturbations are canceled out at the final round R'_{10} by the perturbation cancelation system O_Φ – a function where $O_\Phi(\varphi_1, \varphi_2, \varphi_3, \varphi_4) = 0$. The result of this function is XOR-ed with the output of the original functionality, i.e. the ciphertext Y^1_{10}, to result into Z_{10}: $Z_{10} = Y^1_{10} \oplus Y^2_{10}$.

Linear Encodings $(M_r)_{1 \le r \le 9}$. Annihilating linear input and output encodings M_r over GF $\left(2^8\right)$ between successive rounds ensure that all the variables are interleaved to make analysis hard – e.g. to prevent that an adversary is able to distinguish the system of random equations from the original functionality. These encodings M_r can be represented as a 43×43 diagonal block matrix constructed as follows:

$$M_r = \pi_r \circ \begin{pmatrix} A_r^{(1)} & & & \\ & \ddots & & \\ & & A_r^{(7)} & \\ & & & B_r \end{pmatrix} \circ \sigma_r \ ,$$

where (1) the $A_r^{(l)}|_{l=1,\ldots,7}$ are random invertible 5×5 matrices of which the inverse has exactly 2 non-zero coefficients in GF $\left(2^8\right)$ on each row; (2) B_r is a random invertible 8×8 matrix of which the inverse has at least 7 non-zero coefficients in GF $\left(2^8\right)$ on each row; and (3) π_r and σ_r are random permutations at byte level of $\{1, \ldots, 43\}$ defined such that the matrices $A_r^{(l)}|_{l=1,\ldots,7}$ mix the 16 original polynomials with 19 random polynomials, whereas B_r mixes the 4 perturbation polynomials with the remaining 4 random polynomials. We refer to [4] for determination of the constraints on the matrices. Our cryptanalysis exploits these characteristics of the linear encodings M_r.

Obtaining the Correct Result. Due to the introduction of the perturbation in the first round, there is a probability that the ciphertext is incorrect (when $\tilde{\Phi}(X) \neq (\varphi_1, \varphi_2, \varphi_3, \varphi_4)$ and thus $Y^2_{10} = O_\Phi(\tilde{\Phi}(X)) \neq 0$). Therefore, four correlated instances of the perturbated white-box AEw/oS implementation are generated. Each with a different perturbation function, constructed such that there are always two instances that give the correct result (ciphertext) while the other two result into different random values with an overwhelming probability. A majority vote can then be used to distinguish the correct result. We refer to [3,4] for a discussion on the correlation of the four $\widetilde{0}$-polynomials. Our cryptanalysis requires only one instance of the perturbated white-box AEw/oS implementation.

[1] The construction of the $\tilde{0}$-polynomials is described in [3,4].

Summary. Putting everything together, the perturbated white-box AEw/oS implementation consists of four instances of implementations with different perturbations; each instance comprising of 10 rounds R'_r, defined as follows:

$$\begin{cases} M_1 \circ (R_1 \| \tilde{\Phi} \| \texttt{Ran}_1) & \text{for } R'_1\,, \\ M_r \circ (R_r \| I \| \texttt{Ran}_r) \circ M_{r-1}^{-1} & \text{for } R'_r|_{2 \le r \le 9}, \text{ where } I \text{ is the identity function}\,, \\ \bigoplus \circ (R_{10} \| O_{\Phi}) \circ M_9^{-1} & \text{for } R'_{10}\,. \end{cases}$$

Along the specifications of Bringer *et al.* [4], each instance accounts ≈142 MB, which brings the full size of the white-box implementation to ≈568 MB.

3 Cryptanalysis of the White-Box AEw/oS Implementation

In this section we describe our cryptanalysis, which comprises of the following three steps:

1. Analysis of the final round: distinguish the system of random equations and the perturbations from the AEw/oS round operations, and recover the input encoding M_9^{-1} up to an unknown constant factor s.t. the linear equivalent input of the original final round R_{10} can be observed.
2. Separate the output bytes of the S-boxes: eliminate the `MixColumns` operation from the penultimate round R_9 s.t. the unknown factors of the linear equivalent output of R_9 can be included into the secret S-boxes.
3. Full structural decomposition, i.e. recovering all linear input/output encodings up to an unknown constant factor and eliminating the `MixColumns` operation within all rounds. Recover linear equivalent key-dependent S-boxes.

Note that not all the information of the secret S-boxes and linear mappings can be extracted since there are many equivalent keys which yield the same white-box implementation. Indeed, we can multiply the input/output of an S-box with a fixed constant and compensate for it in the adjacent linear mapping. Our attack recovers an equivalent key, and hence the decomposed implementation can for example be used to decrypt arbitrary ciphertexts although the implementation was only intended to encrypt plaintexts.

Setup. Choose a random 16-byte plaintext X, encrypt it with the four correlated implementations, and select one of both instances that result into the correct ciphertext (using the majority vote). For that instance, store the intermediate and final states $Z_i|_{i=1,\ldots,10}$ (which are clearly readable in a white-box attack context), where the final state equals the ciphertext, i.e. $Z_{10} = Y_{10}^1$. Throughout our cryptanalysis, we will refer to these states as the initial unmodified states.

3.1 Analysis of the Final Round

The first phase of our cryptanalysis focuses on the perturbated final round R'_{10}, which is lossy since the system of random equations is discarded. We will recover

a significant part of the linear input encoding, i.e., the first 16 rows of the linear input encoding M_9^{-1} up to an unknown 16×16 diagonal matrix Λ_9. This enables us to observe the linear equivalent input $\Lambda_9 Y_9^1$ of the original final round R_{10}. This phase consists of several consecutive steps.

Recover pairs of intermediate bytes in Z_9 generating each input byte of Y_9^1 of R_{10}. Due to the specific characteristics of M_9, i.e. the matrices $A_9^{(l)}|_{l=1,\ldots,7}$ mix the 16 original polynomials with 19 random polynomials, the concatenated 35-byte output of $A_9^{(l)-1}|_{l=1,\ldots,7}$ consists of the 16-byte input Y_9^1 of R_{10}, while the remaining 19 bytes are discarded in R'_{10}. Therefore, since these $A_9^{(l)-1}|_{l=1,\ldots,7}$ matrices in M_9^{-1} have exactly 2 non-zero coefficients on each row, each input byte $y_{9,i}^1$ of the original final round R_{10} is a linear combination in GF $\left(2^8\right)$ of exactly two intermediate bytes of Z_9. The pair of intermediate bytes generating $y_{9,i}^1$ is denoted by (z_{9,i_1}, z_{9,i_2}).

In a white-box attack context, the adversary has access to the description of the (obfuscated) system of polynomials, and is able to manipulate the internal states. Hence, he can freely choose to modify bytes of Z_9 and observe the corresponding output Z_{10}. In this context, we present an algorithm to obtain the following sets:

$\mathscr{S}_{Z_9}(y_{10,i}^1)$**:** the set containing the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}) corresponding to each output byte $y_{10,i}^1$ of R_{10}. Due to the lack of the `MixColumns` step in the final round, R_{10} comprises of 16 one-to-one monovariate polynomials, and hence these sets can easily be assigned to the corresponding input byte $y_{9,i}^1$ by applying the inverse `ShiftRows` step;

$\mathscr{S}_{Z_9}(O_\Phi)$**:** the set containing those intermediate bytes of Z_9 which function as input bytes of the B_9^{-1} matrix in M_9^{-1} that only affects the 4-byte input Y_9^2 of O_Φ. This set contains at least 7 and at most 8 bytes.[2]

The setup of the algorithm is to generate for each output byte $z_{10,i}|_{i=0,\ldots,15}$ of the perturbated final round R'_{10} a set $\mathscr{S}_{Z_9}(z_{10,i})$ consisting of those intermediate bytes of Z_9 which influence $z_{10,i}$. Repeat the following steps for each intermediate byte $z_{9,i}$ of Z_9 one at a time:

Step 1: Make the intermediate byte $z_{9,i}$ active by introducing a non-zero difference $\Delta z_{9,i} \in \text{GF}\left(2^8\right) \setminus \{0\}$ while keeping all other bytes of Z_9 fixed to their initial value $(\forall l \neq i : \Delta z_{9,l} = 0)$;

Step 2: Compute R'_{10} and observe its output Z_{10} by comparing with the stored initial final state Y_{10}^1 (ciphertext): if the number of affected output bytes $z_{10,i}$ is larger than 5 bytes, then assign the current active intermediate byte $z_{9,i}$ directly to the set $\mathscr{S}_{Z_9}(O_\Phi)$. Else $z_{9,i}$ is assigned to each set

[2] Since B_9 mixes the 4 perturbation polynomials with 4 random polynomials, the 8-byte output of B_9^{-1} consists of the 4-byte input Y_9^2 of O_Φ and the other 4-byte output is discarded in the perturbated final round R'_{10}, hence we only focus on Y_9^2. If Y_9^2 only depends on 7 intermediate bytes of Z_9 instead of 8 (special case of 8×8 B_9^{-1} matrix), we are only able to identify 7 bytes.

$\mathscr{S}_{Z_9}(z_{10,i})$ of the output bytes $z_{10,i}$ it affects. In case that the number of affected output bytes is zero, $z_{9,i}$ is assigned to no set. Fig. 2 depicts the effect of one active intermediate byte $z_{9,i}$ on the output bytes of Z_{10} and explains the different cases.

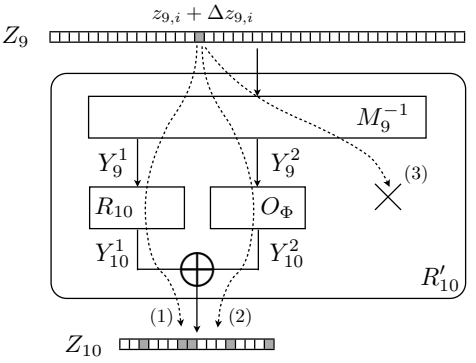


Fig. 2. In case that the active intermediate byte $z_{9,i}$ of Z_9 influences the input Y_9^1 of R_{10} through one of the $A_9^{(l)-1}$ matrices [case (1)], the maximum number of affected input bytes of R_{10} equals 5 since $A_9^{(l)-1}$ are 5×5 invertible matrices. This trivially translates to a maximum of 5 affected ciphertext bytes due to the lack of the `MixColumns` step in R_{10} and accordingly to a maximum of 5 affected output bytes $z_{10,i}$ in Z_{10}. So the case there are more than 5 affected output bytes $z_{10,i}$ only occurs when the active byte $z_{9,i}$ influences the input Y_9^2 of O_Φ through B_9^{-1} [case (2)], which causes the output Y_{10}^2 to change in more than 5 bytes. However, with a very low probability, only 5 or less bytes of Y_{10}^2 are affected which introduces false positives (see below). In case that the active intermediate byte $z_{9,i}$ only affects the input of the system of random polynomials [case (3)] - which has been discarded in R'_{10} - the number of affected output bytes is zero.

Concerning Step 2, false positives can occur, i.e. the incorrect assignment of $z_{9,i}$ to the sets $\mathscr{S}_{Z_9}(z_{10,i})$ instead of the set $\mathscr{S}_{Z_9}(O_\Phi)$. An active intermediate byte $z_{9,i}$ which influences the 4-byte input Y_9^2 of O_Φ through the B_9^{-1} matrix, modifies the initial value $(\varphi_1, \varphi_2, \varphi_3, \varphi_4)$. Since the 16-byte output Y_{10}^2 of O_Φ - which is zero for $Y_9^2 = (\varphi_1, \varphi_2, \varphi_3, \varphi_4)$ and random otherwise - is XOR-ed with the real ciphertext Y_{10}^1 to form the ouput Z_{10}, the probability that the number of affected output bytes $z_{10,i}$ is 5 or less is given by $\sum_{i=1}^{5} \binom{16}{i} (1/2^8)^{16-i} (1-1/2^8)^i \approx 1/2^{76}$. This corresponds also with the probability that $z_{9,i}$ is faulty assigned to each set $\mathscr{S}_{Z_9}(z_{10,i})$ of the affected output bytes $z_{10,i}$ (false positive).

Hence, at the end, the probability that each set $\mathscr{S}_{Z_9}(z_{10,i})|_{i=0,\ldots,15}$ contains exactly the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}) generating the corresponding input byte $y_{9,i}^1$ and that the set $\mathscr{S}_{Z_9}(O_\Phi)$ contains the 7 or 8 intermediate bytes functioning as input bytes of the B_9^{-1} matrix, equals $\approx (1 - 1/2^{76})^a$ with $a = 7$ or 8, which is ≈ 1. In that case $\mathscr{S}_{Z_9}(y_{10,i}^1) = \mathscr{S}_{Z_9}(z_{10,i})$.

The worst case scenario, i.e. the set $\mathscr{S}_{Z_9}(O_\Phi)$ contains less than 7 or 8 intermediate bytes and some or all sets $\mathscr{S}_{Z_9}(z_{10,i})|_{i=0,\ldots,15}$ contain next to the pair

of intermediate bytes (z_{9,i_1}, z_{9,i_2}) also additional intermediate bytes which only influenced the input Y_9^2 of O_Φ (false positives), only occurs with a probability of $\approx 1-(1-1/2^{76})^a$ with $a = 7$ or 8, which is ≈ 0. In that case, after the setup of the algorithm, the sets $\mathscr{S}_{Z_9}(z_{10,i})|_{i=0,\ldots,15}$ need to be reduced to the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}) while completing the set $\mathscr{S}_{Z_9}(O_\Phi)$. This case is fully handled in App. A.1, which is based on pairs of sets of plaintext-ciphertext together with all intermediate states for the same perturbated instance of the cipher we selected during the setup phase of our cryptanalysis.

Note: A side-effect of the above algorithm is the recovery of the set $\mathscr{S}_{Z_9}(O_\Phi)$. Keeping those 7 or 8 intermediate bytes fixed to their initial value ensures that the 4-byte input Y_9^2 of O_Φ remains unmodified, i.e. $(\varphi_1, \varphi_2, \varphi_3, \varphi_4)$, such that the output of O_Φ remains zero and hence we can always observe the real ciphertext: $Z_{10} = Y_{10}^1$. This allows us to circumvent the perturbations.

Decompose the linear input encoding M_9^{-1}. Each unknown input byte $y_{9,i}^1$ of the original final round R_{10} is a linear combination in $\mathrm{GF}\left(2^8\right)$ of a pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}), which has been recovered in the previous step. Thus, there are two non-zero coefficients $c_{i,1}, c_{i,2} \in \mathrm{GF}\left(2^8\right) \setminus \{0\}$ on a row of one of the $A_9^{(l)-1}$ matrices in M_9^{-1} such that $z_{9,i_1} \bullet c_{i,1} + z_{9,i_2} \bullet c_{i,2} = y_{9,i}^1$, where $\bullet$ denotes multiplication in the Rijndael Galois Field [13]. In this step, we recover both coefficients up to an unknown factor $\alpha_{9,i}$, which enables us to observe the linear equivalent input byte $\alpha_{9,i} \bullet y_{9,i}^1$.

Both coefficients $c_{i,1}$ and $c_{i,2}$ can be expressed in terms of a single unknown parameter $\alpha_{9,i}$ as follows: (1) compute the output byte of Z_{10} corresponding to $y_{10,i}^1$ – knowing that $Z_{10} = Y_{10}^1$ (see note above) – where the relevant input bytes (z_{9,i_1}, z_{9,i_2}) are fixed to their initial value in Z_9, and (2) find another pair of values (z'_{9,i_1}, z'_{9,i_2}) by fixing $z'_{9,i_1} = z_{9,i_1} + \texttt{'01'}$ and searching for z'_{9,i_2} which yield the identical output byte in Z_{10}. Hence, since equal output bytes means equal input bytes for R_{10},

$$\begin{aligned} z_{9,i_1} \bullet c_{i,1} + z_{9,i_2} \bullet c_{i,2} &= y_{9,i}^1 \\ (z_{9,i_1} + \texttt{'01'}) \bullet c_{i,1} + z'_{9,i_2} \bullet c_{i,2} &= y_{9,i}^1 \,, \end{aligned}$$

from which we can derive that $c_{i,1} = \varepsilon_i \bullet c_{i,2}$, with $\varepsilon_i = z_{9,i_2} + z'_{9,i_2}$.

By assigning $\texttt{'01'}$ to $c_{i,2}$, only the linear equivalent input byte $y_{9,i}^1$ can be recovered, i.e. $\alpha_{9,i} \bullet y_{9,i}^1 = \varepsilon_i \bullet z_{9,i_1} + z_{9,i_2}$ with $\alpha_{9,i}$ unknown. As a result, we retrieve an expression of the first 16 rows of the linear mapping M_9^{-1} up to (unknown) constants $\alpha_{9,i}$. That is, we obtain the following equation:

$$\begin{pmatrix} \alpha_{9,0} \bullet y_{9,0}^1 \\ \vdots \\ \alpha_{9,15} \bullet y_{9,15}^1 \end{pmatrix} = M_9^{-1}[0..15]' Z_9 = \begin{pmatrix} \alpha_{9,0} & & \\ & \ddots & \\ & & \alpha_{9,15} \end{pmatrix} \begin{pmatrix} L_{9,0} \\ \vdots \\ L_{9,15} \end{pmatrix} Z_9 \,, \qquad (1)$$

where $L_{9,i}$ denotes the i-th row of M_9^{-1} and contains the unknown coefficients $c_{i,1}, c_{i,2}$. The recovered submatrix is denoted by $M_9^{-1}[0..15]' = \Lambda_9 \circ M_9^{-1}[0..15]$

with $\Lambda_9 = diag(\alpha_{9,0}, \ldots, \alpha_{9,15})$, which transforms Z_9 into the linear equivalent input $\Lambda_9 Y_9^1$ of R_{10}. Each row of $M_9^{-1}[0..15]'$ is all '00' except an ε_i and '01' on the relevant columns, i.e. the columns corresponding to the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}).

3.2 Separate the S-Boxes

As a result of the first phase of the cryptanalysis, the adversary is able to derive the input bytes $y_{9,i}^1$ of the original final round R_{10} up to unknown coefficients $\alpha_{9,i}$, i.e. $\Lambda_9 Y_9^1$. Due to the annihilating nature of the linear encodings between successive perturbated rounds, this also corresponds to the linear equivalent output of the preceding, penultimate round R_9. Therefore, R_9 can be expressed as

$$\Lambda_9 \circ \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \{S_{9,0}, \ldots, S_{9,15}\} \circ \bigoplus_{K_8} \circ M_8^{-1}[0..15]\,, \qquad (2)$$

where the set $\{S_{9,i}\}|_{i=0,\ldots,15}$ represents the 16 different invertible 8-to-8 bit original S-boxes of R_9, which together with the round key K_8 are part of the secret key.

The objective in this step of our cryptanalysis is to include the unknown factors of the linear equivalent output of R_9 into the secret S-boxes by separating the output bytes of the S-boxes, which can be achieved by eliminating the `MixColumns` operation from the round. However, due to the presence of the unknown values in Λ_9, this is not trivial since the `MixColumns` step is an invertible linear transformation which operates on four bytes. We address this problem in this section.

The main idea is to search for a transformation such that the matrix Λ_9 has the same factors α for each four bytes of a `MixColumns` operation. Even though this factor remains unknown, such a diagonal matrix can be swapped with the `MixColumns` operation (multiplication with a diagonal matrix with all the same elements is a commutative operation in the group of square matrices). As a result, the `MixColumns` operation is the final operation and can be eliminated by multiplying the result with the inverse `MixColumns` operation.

In total there are four parallel `MixColumns` steps $\texttt{MC}_i|_{i=0,\ldots,3}$ since each step $\texttt{MC}_i$ operates on the output bytes of four different S-boxes. Accordingly, Λ_9 can be divided into four 4×4 diagonal submatrices $\Lambda_{9,i}$, each containing those unknown factors α corresponding to the four output bytes of each $\texttt{MC}_i$: $\Lambda_{9,i} = diag(\alpha_{9,i}, \alpha_{9,i+4}, \alpha_{9,i+8}, \alpha_{9,i+12})$ for $i = 0, \ldots, 3$. Hence out of the requirement $\Lambda_{9,i} \circ \texttt{MC}_i = \texttt{MC}_i \circ \Lambda'_{9,i}$, we seek a 4×4 diagonal matrix $\Lambda'_{9,i} = \texttt{MC}_i^{-1} \circ \Lambda_{9,i} \circ \texttt{MC}_i$. This is the case when all diagonal entries of $\Lambda_{9,i}$ are identical, e.g. $\Lambda_{9,i} = diag(\alpha_{9,i}, \alpha_{9,i}, \alpha_{9,i}, \alpha_{9,i})$, and moreover $\Lambda'_{9,i} = \Lambda_{9,i}$. Here we present an algorithm - which has been successfully implemented in `C++` and confirmed by computer experiments - such that:

Given black-box access to the structure as shown in (2)*, a white-box adversary is able to ensure that all diagonal entries of* $\Lambda_{9,i}$ *become identical and hence*

construct $\Lambda'_{9,i} = diag(\alpha_{9,i}, \alpha_{9,i}, \alpha_{9,i}, \alpha_{9,i})$ *for each of the four parallel* `MixColumns` *steps* $\texttt{MC}_i|_{i=0,\ldots,3}$ *such that* (2) *becomes:*

$$\texttt{MixColumns} \circ \Lambda'_9 \circ \texttt{ShiftRows} \circ \{S_{9,0}, \ldots, S_{9,15}\} \circ \bigoplus_{K_8} \circ M_8^{-1}[0..15] \ . \qquad (3)$$

Since in (3) the `ShiftRows` step is just a permutation on byte level, the unknown diagonal entries in Λ'_9 can be included into the secret S-boxes by applying the inverse `ShiftRows` step.

The setup of the algorithm is to generate for each $\texttt{MC}_i|_{i=0,\ldots,3}$ a set $\mathscr{S}_{Z_8}(\texttt{MC}_i)$ consisting of those intermediate bytes of Z_8 which influence the input of $\texttt{MC}_i$. This is done by making each intermediate byte of Z_8 (the input of (2)) one at a time active and observing the corresponding active output bytes in $\Lambda_9 Y_9^1$ (the output of (2)). Since each input byte of $\texttt{MC}_i$ depends on a pair of intermediate bytes of Z_8 and due to the special structure of the $A_8^{(l)-1}|_{l=1,\ldots,7}$ matrices in M_8^{-1}, modifying one of the bytes in $\mathscr{S}_{Z_8}(\texttt{MC}_i)$ results in making one or two of the four input bytes of $\texttt{MC}_i$ active in most cases. However in the special case when the four input bytes of $\texttt{MC}_i$ are controlled by two distinct pairs of intermediate bytes of Z_8, only exactly two input bytes of $\texttt{MC}_i$ can be made active.

Repeat the following steps for each `MixColumns` step $\texttt{MC}_i|_{i=0,\ldots,3}$:

Step 1: Given the initial unmodified value of the intermediate state Z_8, store the corresponding 4-byte output of $\texttt{MC}_i$ in $\Lambda_9 Y_9^1$, denoted by $Y_{\texttt{MC}_i}$;[3]

Step 2: Modify one byte in $\mathscr{S}_{Z_8}(\texttt{MC}_i)$ and store the corresponding 4-byte output of $\texttt{MC}_i$ in $\Lambda_9 Y_9^1$, denoted by $Y'_{\texttt{MC}_i}$. In the case when less than three bytes between $Y_{\texttt{MC}_i}$ and $Y'_{\texttt{MC}_i}$ have become active and hence at least three of the four input bytes of $\texttt{MC}_i$ have become active,[4] we discard this case and continue with Step 4;

Step 3: Given the pair $(Y_{\texttt{MC}_i}, Y'_{\texttt{MC}_i})$, keep the first factor $\alpha_{9,i}$ fixed while varying the other three factors $(\alpha_{9,i+4}, \alpha_{9,i+8}, \alpha_{9,i+12})$ over $\text{GF}\left(2^8\right) \setminus \{0\}$ by multiplying the second, third and fourth byte within both values $(Y_{\texttt{MC}_i}, Y'_{\texttt{MC}_i})$ with respectively $\beta, \gamma, \delta \in \text{GF}\left(2^8\right) \setminus \{0\}$. For each combination $(\beta, \gamma, \delta)_j$ with the corresponding pair $(Y^{(j)}_{\texttt{MC}_i}, Y'^{(j)}_{\texttt{MC}_i})$ with $j = 1, \ldots, (2^8-1)^3$, invert the `MixColumns` step, i.e. $(Y^{(j)}_{\texttt{MC}_i^{-1}}, Y'^{(j)}_{\texttt{MC}_i^{-1}}) = (\texttt{MC}^{-1}(Y^{(j)}_{\texttt{MC}_i}), \texttt{MC}^{-1}(Y'^{(j)}_{\texttt{MC}_i}))$, and construct the following solution set by comparing both values:

$$\mathscr{S} = \{(\beta, \gamma, \delta)_j \mid \text{one or two active bytes between } (Y^{(j)}_{\texttt{MC}_i^{-1}}, Y'^{(j)}_{\texttt{MC}_i^{-1}})\} \ ;$$

Step 4: Repeat Step 2 and Step 3 for each byte in $\mathscr{S}_{Z_8}(\texttt{MC}_i)$ one at a time;

Step 5: At the end, a solution set $\mathscr{S}$ has been constructed for each modified byte in $\mathscr{S}_{Z_8}(\texttt{MC}_i)$. The triplet $(\beta, \gamma, \delta)_i$ for $\texttt{MC}_i$ is derived as the intersection between all solution sets.

[3] $Y_{\texttt{MC}_i} = (\alpha_{9,i} \bullet y^1_{9,i}, \alpha_{9,i+4} \bullet y^1_{9,i+4}, \alpha_{9,i+8} \bullet y^1_{9,i+8}, \alpha_{9,i+12} \bullet y^1_{9,i+12})$.

[4] The branch number of the `MixColumns` step equals 5.

As can be observed in Step 3, the algorithm only keeps track of single and double active input bytes[5] to each `MixColumns` step for each modified byte in $\mathscr{S}_{Z_8}(\texttt{MC}_i)$ and each combination $(\beta, \gamma, \delta)_j$. When modifying one byte in $\mathscr{S}_{Z_8}(\texttt{MC}_i)$, we distinguish the following two cases:

1. one or two of the four input bytes of $\texttt{MC}_i$ have become active. The resulting solution set $\mathscr{S}$ obtained in Step 3 is considered *valid* and contains the triplet $(\beta, \gamma, \delta)_j$ for which the same one or two bytes have become active between $(Y^{(j)}_{\texttt{MC}_i^{-1}}, Y'^{(j)}_{\texttt{MC}_i^{-1}})$, which only occurs when the triplet made all diagonal entries of $\Lambda_{9,i}$ identical (i.e. all equal to $\alpha_{9,i}$) such that $\Lambda_{9,i}$ could be swapped with the `MixColumns` step $\texttt{MC}_i$. This triplet is contained within all *valid* solution sets;
2. at least three of the four input bytes of $\texttt{MC}_i$ have become active and the case has not been discarded in Step 2. Hence the resulting solution set $\mathscr{S}$ obtained in Step 3 is considered *invalid*.

Hence, in Step 5 only one intersection occurs between all solution sets, i.e. between *valid* sets since there is no intersection with *invalid* sets.

The triplet $(\beta, \gamma, \delta)_i$ as outcome of the above algorithm applied to each $\texttt{MC}_i$ are the factors needed to ensure that all diagonal entries of $\Lambda_{9,i}$ become identical to the first factor $\alpha_{9,i}$, i.e. $diag(\texttt{`01'}, \beta_i, \gamma_i, \delta_i) \circ \Lambda_{9,i} = diag(\alpha_{9,i}, \alpha_{9,i}, \alpha_{9,i}, \alpha_{9,i}) = \Lambda'_{9,i}$. So by multiplying each set of four rows corresponding to each $\texttt{MC}_i$ of the recovered submatrix $M_9^{-1}[0..15]'$ (see (1)) with the derived quartet $(\texttt{`01'}, \beta_i, \gamma_i, \delta_i)$, the adversary is able to construct Λ'_9 in which the diagonal entries corresponding to each $\texttt{MC}_i$ are identical:

$$\begin{pmatrix} I_{4\times4} & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & \Gamma & 0 \\ 0 & 0 & 0 & \Delta \end{pmatrix} M_9^{-1}[0..15]' = \begin{pmatrix} D_9 & 0 & 0 & 0 \\ 0 & D_9 & 0 & 0 \\ 0 & 0 & D_9 & 0 \\ 0 & 0 & 0 & D_9 \end{pmatrix} \begin{pmatrix} L_{9,0} \\ \vdots \\ L_{9,15} \end{pmatrix}, \tag{4}$$

where $B = diag(\beta_0, \beta_1, \beta_2, \beta_3)$, $\Gamma = diag(\gamma_0, \gamma_1, \gamma_2, \gamma_3)$, $\Delta = diag(\delta_0, \delta_1, \delta_2, \delta_3)$, $D_9 = diag(\alpha_{9,0}, \alpha_{9,1}, \alpha_{9,2}, \alpha_{9,3})$. The obtained submatrix of (4) is denoted by $M_9^{-1}[0..15]'' = \Lambda'_9 \circ M_9^{-1}[0..15]$ with $\Lambda'_9 = diag(D_9, D_9, D_9, D_9)$.

3.3 Decomposing the Rounds

The final phase of our cryptanalysis presents the full decomposition of the perturbated white-box AEw/oS implementation and shows how to obtain a set of candidate S-boxes (the secret key). We present an algorithm to recover all remaining linear encodings up to a constant factor and to eliminate the `MixColumns` operation, when applied to all perturbated intermediate rounds $R'_r|_{2\leq r\leq 9}$ from the bottom up.

[5] Keeping track of double active input bytes is necessary due to the special case mentioned in the setup of the algorithm.

Given black-box access to a perturbated intermediate round R'_r which linear output encoding $M_r^{-1}[0..15]'' = \Lambda'_r \circ M_r^{-1}[0..15]$ has already been recovered s.t. the linear equivalent output $\Lambda'_r Y_r^1$ of the original intermediate round R_r can be observed and the `MixColumns` *step can be separated from the S-boxes, i.e.:*

$$\texttt{MixColumns} \circ \Lambda'_r \circ \texttt{ShiftRows} \circ \{S_{r,0}, \ldots, S_{r,15}\} \circ \bigoplus_{K_{r-1}} \circ M_{r-1}^{-1}[0..15]\,,$$

a white-box adversary is able to derive the linear input encoding up to an unknown factor, i.e. $M_{r-1}^{-1}[0..15]'' = \Lambda'_{r-1} \circ M_{r-1}^{-1}[0..15]$, s.t. the linear equivalent input $\Lambda'_{r-1} Y_{r-1}^1$ of R_r can be observed and the `MixColumns` *step of the preceding round R_{r-1} can be separated from the S-boxes, and hence the structure of R_r becomes:*

$$\texttt{MixColumns} \circ \Lambda'_r \circ \texttt{ShiftRows} \circ \{S_{r,0}, \ldots, S_{r,15}\} \circ \bigoplus_{K_{r-1}} \circ \Lambda'^{-1}_{r-1}\,.$$

As a result of the previous two phases of the cryptanalysis – i.e. the recovery of the linear output encoding $M_9^{-1}[0..15]''$ up to a constant factor (see (4)) and the elimination of the `MixColumns` step (see (3)) of the penultimate round R_9 – the above algorithm first applies to R'_9 and then to the remaining perturbated intermediate rounds $R'_r|_{2 \le r \le 8}$ from the bottom up since each linear input encoding matches the linear output encoding of the preceding round. The main steps of the algorithm are:

1. Assign to each input byte $y_{r-1,i}^1$ of the original round R_r a pair of intermediate bytes $(z_{r-1,i_1}, z_{r-1,i_2})$ of Z_{r-1} contained within the set $\mathscr{S}_{Z_{r-1}}(y_{r-1,i}^1)$;
2. Decompose the linear input encoding M_{r-1}^{-1}: recover the first 16 rows up to a 16×16 diagonal linear bijection $\Lambda_{r-1} = diag(\alpha_{r-1,0}, \ldots, \alpha_{r-1,15})$, i.e. $M_{r-1}^{-1}[0..15]' = \Lambda_{r-1} \circ M_{r-1}^{-1}[0..15]$;
3. Eliminate the `MixColumns` step in the preceding round R_{r-1} by converting Λ_{r-1} into Λ'_{r-1} where the diagonal entries corresponding to each $\texttt{MC}_i|_{i=0,\ldots,3}$ are identical: $M_{r-1}^{-1}[0..15]'' = \Lambda'_{r-1} \circ M_{r-1}^{-1}[0..15]$ with $\Lambda'_{r-1} = diag(D_{r-1}, D_{r-1}, D_{r-1}, D_{r-1})$, where $D_{r-1} = diag(\alpha_{r-1,0}, \alpha_{r-1,1}, \alpha_{r-1,2}, \alpha_{r-1,3})$.

We refer to App. A.2 for a detailed description of each step, which are very similar to the ones stated in Sect. 3.1 and 3.2. However, the algorithm for separating the `MixColumns` step from the S-boxes applied to the first round R_1, i.e. the case when $r = 2$, is simplified since the perturbated round R'_1 lacks an input encoding.

As a result of the algorithm mentioned above, the white-box adversary has black-box access to the following structures of each round $R_r|_{r=1,\ldots,10}$:

$$\begin{cases} \texttt{SR} \circ \bigoplus_{K'_{10}} \circ \{S_{10,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_9} \circ \Lambda'^{-1}_9 & \text{for } R_{10}\ , \\ \texttt{MC} \circ \texttt{SR} \circ \Lambda''_r \circ \{S_{r,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_{r-1}} \circ \Lambda'^{-1}_{r-1} & \text{for } R_r|_{2 \le r \le 9}\ , \\ \texttt{MC} \circ \texttt{SR} \circ \Lambda''_1 \circ \{S_{1,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_0} & \text{for } R_1\ , \end{cases} \tag{5}$$

where each set $\{S_{r,i}\}|_{i=0,\ldots,15;r=1,\ldots,10}$ represents the 16 different invertible 8-to-8 bit original S-boxes of R_r, which together with round keys $K_r|_{r=0,\ldots,10}$ are part of the secret key. By altering $\Lambda'_r|_{r=1,\ldots,9}$, its order with the `ShiftRows` step can be reversed, i.e. $\Lambda''_r = \mathtt{SR}^{-1} \circ \Lambda'_r \circ \mathtt{SR} = diag(D_r, D_r^{\gg 1}, D_r^{\gg 2}, D_r^{\gg 3})$ where $D_r^{\gg i}$ denotes the matrix $D_r = diag(\alpha_{r,0}, \alpha_{r,1}, \alpha_{r,2}, \alpha_{r,3})$ whose diagonal entries are cyclically shifted i-times to the right. Note that the first and final rounds respectively lack a linear input and output encoding.

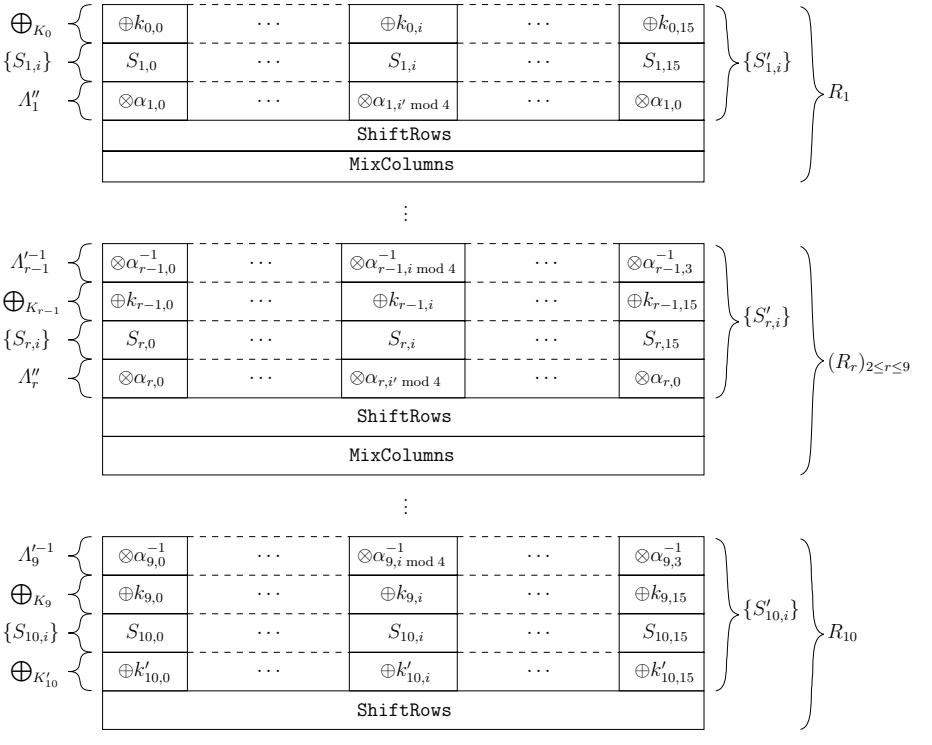


Fig. 3. Our Invertible Functionally Equivalent AEw/oS Implementation

In the structures of (5), only the `ShiftRows` and `MixColumns` steps are known to the adversary. Since all unknown linear bijections $\Lambda'^{-1}_r|_{r=1,\ldots,9}$ and $\Lambda''_r|_{r=1,\ldots,9}$ are 16×16 diagonal matrices, the unknown factors α as diagonal entries can easily be included into respectively the input and output of the secret S-boxes. Hence the linear equivalent key-dependent S-boxes, denoted by $S'_{r,i}$, have the following form for each round $R_r|_{r=1,\ldots,10}$:

$$\begin{aligned} S'_{10,i}|_{i=0,\ldots,15} &= \textstyle\bigoplus_{k'_{10,i}} \circ S_{10,i} \circ \bigoplus_{k_{9,i}} \circ \bigotimes_{\alpha^{-1}_{9,i \bmod 4}} , \\ S'_{r,i}|_{i=0,\ldots,15;r=2,\ldots,9} &= \textstyle\bigotimes_{\alpha_{r,i' \bmod 4}} \circ S_{r,i} \circ \bigoplus_{k_{r-1,i}} \circ \bigotimes_{\alpha^{-1}_{r-1,i \bmod 4}} , \\ S'_{1,i}|_{i=0,\ldots,15} &= \textstyle\bigotimes_{\alpha_{1,i' \bmod 4}} \circ S_{1,i} \circ \bigoplus_{k_{0,i}} . \end{aligned} \tag{6}$$

Although all components in the S-boxes of (6) are unknown to the adversary, each S-box $S'_{r,i}|_{i=0,\ldots,15;r=1,\ldots,10}$ can be defined by varying its input byte $\alpha_{r-1,i \bmod 4} \bullet y^1_{r-1,i}$ (in the case of R_1: the i-th plaintext byte x_i) over $GF(2^8)$ and record the corresponding output byte in $\mathtt{SR}^{-1}(\mathtt{MC}^{-1}(\Lambda'_r Y^1_r))$ (in the case of R_{10}: in $\mathtt{SR}^{-1}(Z_{10})$). In order to vary the input, we keep one of the pair bytes z_{r-1,i_1} fixed and vary the other byte z_{r-1,i_2}.

Fig. 3 depicts an overview of the full decomposition of the perturbated white-box AEw/oS implementation in order to obtain an invertible, functionally equivalent version.

4 Conclusion

In this paper we presented a structural cryptanalysis of the perturbated white-box AEw/oS implementation, presented by Bringer *et al.* [4]. Our attack has a worst time complexity of 2^{17} and negligible space complexity (see App. B). Our cryptanalysis trivially extends to perturbated white-box AES implementations as well.

The technique decomposes the obfuscated round structure of the white-box implementation. After eliminating the system of random equations and perturbations, we show how to distinguish the output bytes of individual S-boxes – by eliminating the `MixColumns` from the round functions. This elimination step is crucial in our cryptanalysis, and a proof of concept has been implemented in `C++`. From the obtained structure, a definition for each S-box can be derived. These S-boxes are linear equivalent to the original (secret) key that was chosen to construct the implementation. Indeed, there are several candidate keys possible that yield the same white-box implementation. This is embodied by the factors α that we meet in our cryptanalysis – these can take any value in GF $\left(2^8\right) \setminus \{0\}$.

Each equivalent key consists of 160 bijective key-dependent 8-bit S-boxes, which can be used to construct a simpler, functionally equivalent version of the white-box AEw/oS implementation (as depicted in Fig. 3 in Sect. 3.3). The S-boxes occupy a total storage space of ≈ 41 kB; hence the total size of the implementation is significantly reduced from several hundred MB to just a few tens of kB. On top of this, the implementation becomes invertible, which renders it useless for many practical implementation where white-box cryptography would be of value.

The cryptanalysis is independent of the definition of the perturbation functions that are introduced in the first round; we exploit the characteristics of the input/output linear encodings and some properties of the AEw/oS block ciphers (such as the `MixColumns` operation). Modifying some specifications in an attempt to mitigate our cryptanalysis, such as the number of non-zero elements on the rows of the $A_r^{(l)-1}$ may turn the white-box implementation useless (its size will increase exponentially).

Although our cryptanalysis is specific to the particular structure of the implementation, some algorithms are of independent interest. In particular for research in structural cryptanalysis. Future research may include research to extend these algorithms to more generic constructions, e.g., where the `MixColumns` operations are also key dependent.

Acknowledgments. This work was supported in part by the IWT-SBO project on Security of Software for Distributed Applications (SEC SODA) and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). Yoni De Mulder was supported in part by IBBT (Interdisciplinary institute for BroadBand Technology) of the Flemish Government and by a research grant of the Katholieke Universiteit Leuven.

References

1. Billet, O., Gilbert, H.: A Traceable Block Cipher. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 331–346. Springer, Heidelberg (2003)
2. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a White Box AES Implementation. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 227–240. Springer, Heidelberg (2004)
3. Bringer, J., Chabanne, H., Dottax, E.: Perturbing and Protecting a Traceable Block Cipher. In: Leitold, H., Markatos, E.P. (eds.) CMS 2006. LNCS, vol. 4237, pp. 109–119. Springer, Heidelberg (2006)
4. Bringer, J., Chabanne, H., Dottax, E.: White box cryptography: Another attempt. Cryptology ePrint Archive, Report 2006/468 (2006), `http://eprint.iacr.org/`
5. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: White-Box Cryptography and an AES Implementation. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 250–270. Springer, Heidelberg (2003)
6. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: Feigenbaum, J. (ed.) DRM 2002. LNCS, vol. 2696, pp. 1–15. Springer, Heidelberg (2003)
7. Ding, J.: A New Variant of the Matsumoto-Imai Cryptosystem through Perturbation. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 305–318. Springer, Heidelberg (2004)
8. Faugère, J.-C., Perret, L.: Polynomial Equivalence Problems: Algorithmic and Theoretical Aspects. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 30–47. Springer, Heidelberg (2006)
9. Goubin, L., Masereel, J.-M., Quisquater, M.: Cryptanalysis of White Box DES Implementations. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 278–295. Springer, Heidelberg (2007)
10. Jacob, M., Boneh, D., Felten, E.W.: Attacking an Obfuscated Cipher by Injecting Faults. In: Feigenbaum, J. (ed.) DRM 2002. LNCS, vol. 2696, pp. 16–31. Springer, Heidelberg (2003)
11. Link, H.E., Neumann, W.D.: Clarifying Obfuscation: Improving the Security of White-Box DES. In: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2005), Washington, DC, USA, vol. 1, pp. 679–684. IEEE Computer Society, Los Alamitos (2005)
12. Michiels, W., Gorissen, P., Hollmann, H.D.L.: Cryptanalysis of a Generic Class of White-Box Implementations. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 414–428. Springer, Heidelberg (2009)
13. National Institute of Standards and Technology. Advanced encryption standard. FIPS publication 197 (2001), `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`
14. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996)

15. Wyseur, B.: White-Box Cryptography. PhD thesis, Katholieke Universiteit Leuven (2009)
16. Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 264–277. Springer, Heidelberg (2007)

A Algorithms

A.1 Algorithm: Recover Pairs of Intermediate Bytes in Z_9 Generating Each Input Byte of Y_9^1 of R_{10} - Worst Case Scenario

The worst case scenario, i.e. the set $\mathscr{S}_{Z_9}(O_\Phi)$ contains less than 7 or 8 intermediate bytes and some or all sets $\mathscr{S}_{Z_9}(z_{10,i})|_{i=0,\ldots,15}$ contain next to the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}) also additional intermediate bytes which only influenced the input Y_9^2 of O_Φ (false positives), occurs with a probability of $\approx 1-(1-1/2^{76})^a$ with $a = 7$ or 8, which is ≈ 0. In that case, after the setup of the algorithm, the sets $\mathscr{S}_{Z_9}(z_{10,i})|_{i=0,\ldots,15}$ need to be reduced to the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}) while completing the set $\mathscr{S}_{Z_9}(O_\Phi)$.

Being able to do so, another triplet consisting of plaintext-ciphertext $\{X, Z_{10} = Y_{10}^1\}$ together with the intermediate state Z_9 is required for the same perturbated instance of the cipher we selected in the setup phase of the cryptanalysis (see Sect. 3). We introduce the index m to refer to each of both triplets, i.e. $\{X_m, (Z_9)_m, (Z_{10})_m\}$ with $m = 1, 2$, where $m = 1$ concerns the triplet stored during the setup phase of the cryptanalysis. By applying the setup of the algorithm mentioned in Sect. 3.1 as well to the newly computed triplet – and in case the worst case scenario occurs again – we obtain the sets $\mathscr{S}_{Z_9}^m(z_{10,i})$ and $\mathscr{S}_{Z_9}^m(O_\Phi)$ for both triplets $m = 1, 2$, where the set $\mathscr{S}_{Z_9}^m(O_\Phi)$ is incomplete and some sets $\mathscr{S}_{Z_9}^m(z_{10,i})$ contain more than 2 bytes for both $m = 1, 2$.

First combine the retrieved information by taking the union $\bigcup_{m=1,2} \mathscr{S}_{Z_9}^m(O_\Phi)$, denoted by $\mathscr{S}_{Z_9}^{\cup}(O_\Phi)$, and removing in all sets $\mathscr{S}_{Z_9}^m(z_{10,i})|_{m=1,2;i=0,\ldots,15}$ those intermediate bytes $z_{9,i} \in \mathscr{S}_{Z_9}^{\cup}(O_\Phi)$. Then repeat the following steps for each output byte $z_{10,i}$ one at a time:

Step 1: Take the intersection $\bigcap_{m=1,2} \mathscr{S}_{Z_9}^m(z_{10,i})$, denoted by $\mathscr{S}_{Z_9}^{\cap}(z_{10,i})$:
If $|\mathscr{S}_{Z_9}^{\cap}(z_{10,i})| = 2$, then this set identifies the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}) corresponding to the output byte $y_{10,i}^1$ and is assigned to the set $\mathscr{S}_{Z_9}(y_{10,i}^1)$. Go to Step 4.
Else take all possible combinations when choosing two bytes out of $\mathscr{S}_{Z_9}^{\cap}(z_{10,i})$, with a total number of $\binom{|\mathscr{S}_{Z_9}^{\cap}(z_{10,i})|}{2}$.

Step 2: Repeat the following for each combination and for both triplets $\{X_m, (Z_9)_m, (Z_{10})_m\}$ with $m = 1, 2$: construct $(Z_9)_m|_{m=1,2}$ in which the chosen two bytes are replaced by `'00'` while the others remain fixed to their original value. Compute the perturbated final round R'_{10} and compare both output bytes $(z_{10,i})_m|_{m=1,2}$;

Step 3: **If** only one collision occurs in Step 2 for all possible combinations, then the two bytes chosen out of $\mathscr{S}^{\cap}_{Z_9}(z_{10,i})$ in the combination for which the collision occurred, identify the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}) corresponding to the output byte $y^1_{10,i}$ and are assigned to the set $\mathscr{S}_{Z_9}(y^1_{10,i})$. Go to Step 4.

Else in the case more than one collision occurs (hence for more than one combination), go back to Step 1 and continue with the next output byte. At the end, repeat all these steps again for the output bytes for which more than one collision occurred. In case multiple collisions continue to occur, start over again with a new and different triplet $m = 2$.

Step 4: Assign all intermediate bytes of both sets $\mathscr{S}^m_{Z_9}(z_{10,i})|_{m=1,2}$ which are not an element of $\mathscr{S}_{Z_9}(y^1_{10,i})$ to the set $\mathscr{S}^{\cup}_{Z_9}(O_\Phi)$ and remove in all remaining sets $\mathscr{S}^m_{Z_9}(z_{10,j})|_{m=1,2;i<j}$ those intermediate bytes $z_{9,i} \in \mathscr{S}^{\cup}_{Z_9}(O_\Phi)$.

At the end, we obtained the sets $\mathscr{S}_{Z_9}(y^1_{10,i})|_{i=0,\ldots,15}$ and trivially $\mathscr{S}_{Z_9}(O_\Phi) = \mathscr{S}^{\cup}_{Z_9}(O_\Phi)$.

Concerning Step 3, always at least one collision occurs, i.e. for the combination in which the two bytes chosen out of $\mathscr{S}^{\cap}_{Z_9}(z_{10,i})$ are in fact the pair of intermediate bytes (z_{9,i_1}, z_{9,i_2}) producing the input byte $y^1_{9,i}$ of the original final round R_{10}. Hence when both bytes are replaced by `'00'` in $(Z_9)_m|_{m=1,2}$ while the other bytes remain fixed to their initial value, the 8-byte input of B_9^{-1} remains unmodified. The latter ensures that the 4-byte input Y^2_9 of O_Φ is unchanged, i.e. $(\varphi_1, \varphi_2, \varphi_3, \varphi_4)$, such that the output of O_Φ remains zero and hence we can observe the real ciphertext for both $m = 1, 2$: $(Z_{10})_m = Y^1_{10} \oplus O_\Phi(\varphi_1, \ldots, \varphi_4) = Y^1_{10}$. The former ensures that the input byte $y^1_{9,i}$ becomes `'00'` for both $m = 1, 2$ which causes that the corresponding output bytes $y^1_{10,i}$ in the ciphertext collide.

The probability that only one collision occurs, equals $(\frac{2^8-1}{2^8})^{[\binom{|\mathscr{S}^{\cap}_{Z_9}(z_{10,i})|}{2}-1]}$. In the worst case, i.e. when $|\mathscr{S}^{\cap}_{Z_9}(z_{10,i})| = 10$, the probability becomes ≈ 0.84 . Hence it is assumed that the algorithm succeeds in the worst case with only 1 or 2 additional triplets.

A.2 Algorithm: Recover the Linear Input Encoding Up to an Unknown Constant Factor of Each Perturbated Intermediate Round $R'_r|_{r=2 \le r \le 9}$

The adversary is able to observe the linear equivalent output $\Lambda'_r Y^1_r$ of the original intermediate round R_r due to the knowledge of the linear output encoding up to an unknown constant factor, i.e. $M_r^{-1}[0..15]'' = \Lambda'_r \circ M_r^{-1}[0..15]$. Moreover, due to the fact that the `MixColumns` step is separated from the S-boxes, he can apply its inverse and observe $Y^{1\prime}_r = \texttt{MC}^{-1}(\Lambda'_r Y^1_r) = \Lambda'_r \circ \texttt{MC}^{-1}(Y^1_r)$. Hereby, the diffusion property of the `MixColumns` step is lost and hence there is again a one-to-one correspondence between single active input and output bytes, which is the `ShiftRows` step. Now we describe each step of the algorithm:

Assign a pair of intermediate bytes $(z_{r-1,i_1}, z_{r-1,i_2})$ *of* Z_{r-1} *to each input byte* $y^1_{r-1,i}$ *of* R_r. By making each intermediate byte $z_{r-1,i}$ one at a time active and

keeping track of the corresponding active bytes in $Y_r^{1\prime}$, a pair $(z_{r-1,i_1}, z_{r-1,i_2})$ is assigned to each output byte $y_{r,i}^{1\prime}$ due to the specific predefined structure of the $A_{r-1}^{(l)-1}|_{l=1,\ldots,7}$ matrices in M_{r-1}^{-1}. By inverting the `ShiftRows` step, these pairs are reassigned to the corresponding input bytes $y_{r-1,i}^1$ of R_r.

Decompose the linear input encoding M_{r-1}^{-1}. This step is completely similar as the one described in Sect. 3.1, but then applied to R_r. Hence search for two different values of the intermediate bytes, i.e. $\{(z_{r-1,i_1}, z_{r-1,i_2}), (z_{r-1,i_1} \oplus \texttt{`01'}, z'_{r-1,i_2})\}$, which both map onto the same value of the input byte $y_{r-1,i}^1$ by indirectly observing the corresponding output byte $y_{r,i}^{1\prime}$ in $Y_r^{1\prime}$. As a result, a relation between both coefficients is derived, i.e. $c_{i,1} = \varepsilon_i \bullet c_{i,2}$, with $\varepsilon_i = z_{r-1,i_2} + z'_{r-1,i_2}$. By assigning `'01'` to $c_{i,2}$, only the linear equivalent input byte $y_{r-1,i}^1$ can be recovered, i.e. $\alpha_{r-1,i} \bullet y_{r-1,i}^1 = \varepsilon_i \bullet z_{r-1,i_1} + z_{r-1,i_2}$ where $\alpha_{r-1,i} \in \mathrm{GF}\left(2^8\right) \setminus \{0\}$ is unknown.

As a result, we obtain the first 16 rows of the linear input encoding M_{r-1}^{-1} up to an unknown 16×16 diagonal linear bijection Λ_{r-1}, i.e. $M_{r-1}^{-1}[0..15]' = \Lambda_{r-1} \circ M_{r-1}^{-1}[0..15]$ with $\Lambda_{r-1} = diag(\alpha_{r-1,0}, \ldots, \alpha_{r-1,15})$. Each row of $M_{r-1}^{-1}[0..15]'$ is all `'00'` except a `'01'` and ε_i on the relevant columns.

Eliminate the `MixColumns` step in the preceding round R_{r-1}. The algorithm to convert Λ_{r-1} into Λ'_{r-1} in which the unknown diagonal entries corresponding to each `MixColumns` step $\texttt{MC}_i|_{i=0,\ldots,3}$ are identical s.t. the order between Λ'_{r-1} and the `MixColumns` step can be reversed, is identical to the one applied to the penultimate round R_9 in Sect. 3.2. However there is one special case, i.e. to generate Λ'_1 when applied to the first round R_1. In contrast to all intermediate rounds, the first round lacks a linear input encoding, and thus it is possible to make only one of the four input bytes to the i-th `MixColumns` step $\texttt{MC}_i$ active by modifying the corresponding byte in the plaintext X. So for $r = 2$, each set $\mathscr{S}_X(\texttt{MC}_i)$ contains exactly four plaintext bytes. Moreover, when constructing the solution sets $\mathscr{S}$ in Step 3 for each modified plaintext byte in $\mathscr{S}_X(\texttt{MC}_i)$, the algorithm only needs to keep track of single active input bytes. This simplifies the algorithm and increases the performance in this special case.

As a result, we recover the new submatrix $M_{r-1}^{-1}[0..15]'' = \Lambda'_{r-1} \circ M_{r-1}^{-1}[0..15]$, where $\Lambda'_{r-1} = diag(D_{r-1}, D_{r-1}, D_{r-1}, D_{r-1})$ with $D_{r-1} = diag(\alpha_{r-1,0}, \alpha_{r-1,1}, \alpha_{r-1,2}, \alpha_{r-1,3})$, which transforms the 43-byte intermediate value Z_{r-1} into the linear equivalent input $\Lambda'_{r-1} Y_{r-1}^1$ of R_r.

B Complexity

The time complexity of our cryptanalysis is expressed in the number of perturbated round evaluations (parsing system of equations over $\mathrm{GF}\left(2^8\right)$). With respect to the round operations, other computations in the attack are negligible and have been ommitted for simplicity.

- *Setup phase of the cryptanalysis:* encryption of one randomly chosen plaintext by all four correlated instances of the perturbated white-box AEw/oS implementation requires $4 \cdot 10$ round executions;
- *Analysis of the final round:* construction of both sets $\mathscr{S}_{Z_9}(y^1_{9,i})$ and $\mathscr{S}_{Z_9}(O_\Phi)$ for the final round needs (1) without the worst case scenario 43 round evaluations or (2) with the worst case scenario – which occurs with a negligible probability – an additional $2 \cdot (8 \cdot 4 \cdot 10 + 43 + 2 \cdot 5 \cdot \binom{10}{2})$ (under the assumption that only eight plaintexts need to be encrypted in order to find an additional triplet with probability $1 - (1/2)^8 \approx 0.996$); the recovery of the coefficients of the linear combination up to an unknown factor α for each pair intermediate bytes demands $16 \cdot 2^8$ round operations;
- *Separate the S-boxes:* elimination of the `MixColumns` step in the penultimate round R_9 requires $43 + 4 \cdot 8$ round evaluations;
- *Decomposing the rounds:* construction of the set $\mathscr{S}_{Z_{r-1}}(y^1_{r-1,i})$ for all intermediate rounds $R_r|_{2 \leq r \leq 9}$ needs $8 \cdot 43$ round evaluations; the recovery of the coefficients of the linear combination up to an unknown factor α for each pair of intermediate bytes for the intermediate rounds $R_r|_{2 \leq r \leq 9}$ demands $8 \cdot 16 \cdot 2^8$ round operations; the elimination of the `MixColumns` step in the rounds $R_r|_{1 \leq r \leq 8}$ requires $7 \cdot (43 + 4 \cdot 8) + (4 \cdot 4)$ round evaluations; finally, in order to define the linear equivalent key, $10 \cdot 16 \cdot 2^8$ perturbated rounds need to be executed.

This brings the total worst time complexity down to $80493 = 2^{16.2966}$ perturbated round evaluations.

The space complexity of our cryptanalysis is negligible.