Another Nail in the Coffin of White-Box AES Implementations

Tancrède Lepoint[1,2] and Matthieu Rivain[1]

[1] CryptoExperts, France
{tancrede.lepoint, matthieu.rivain}@cryptoexperts.com
[2] École Normale Supérieure, France

Abstract. The goal of white-box cryptography is to design implementations of common cryptographic algorithm (*e.g.* AES) that remain secure against an attacker with full control of the implementation and execution environment. This concept was put forward a decade ago by Chow *et al.* (SAC 2002) who proposed the first white-box implementation of AES. Since then, several works have been dedicated to the design of new implementations and/or the breaking of existing ones.

In this paper, we describe a new attack against the original implementation of Chow *et al.* (SAC 2002), which efficiently recovers the AES secret key as well as the private external encodings in complexity 2^{22}. Compared to the previous attack due to Billet *et al.* (SAC 2004) of complexity 2^{30}, our attack is not only more efficient but also simpler to implement. Then, we show that the *last* candidate white-box AES implementation due to Karroumi (ICISC 2010) can be broken by a direct application of either Billet *et al.* attack or ours. Specifically, we show that for any given secret key, the overall implementation has the *exact same* distribution as the implementation of Chow *et al.* making them both vulnerable to the same attacks.

By improving the state of the art of white-box cryptanalysis and putting forward new attack techniques, we believe our work brings new insights on the failure of existing white-box implementations, which could be useful for the design of future solutions.

Key words: White-Box Implementations, Advanced Encryption Standard (AES), Cryptanalysis.

1 Introduction

White-box cryptography was introduced in 2002 by Chow, Eisen, Johnson and van Oorschot [6,7] by proposing a new attack context in which an adversary has a complete access to the implementation of algorithms and the execution environment; thus the dynamic execution and the internal details of the algorithms can both be viewed and altered at will. This attack context is motivated by applications such as Digital Rights Management (DRM): to protect a digital content, the client can be implemented in software and executed on an untrusted end-point such as a PC, a smartphone, or a set-top box. The content provider then wants to avoid any unauthorized distribution of the secret key used for content decryption in the software. The main goal of white-box cryptography is therefore to protect a secret key in a white-box environment.

In [6,7], Chow *et al.* proposed a generic strategy to produce white-box implementations of (symmetric) cryptographic algorithms, and for which key extraction is supposedly hard. From this strategy, they derive two candidate white-box implementations of DES and AES. The followed approach is that look-up tables might be the ideal primitives to hide information, since they allow to implement any given function. They proposed to express the algorithm into a network of look-up tables which, combined all together, yield the complete cryptographic algorithm. To avoid information leakage (and thus key leakage), the basic idea is to compose each table, on input and on output, with random bijections that annihilate one with another when composed.

At SAC 2004, Billet, Gilbert and Ech-Chatbi presented a cryptanalysis of complexity 2^{30} of the white-box AES implementation [3]. Further attacks were subsequently published against white-box DES implementations [11,9,17] and against more general cipher structures implemented with similar strategies [12]. These attacks challenged the research of new white-box implementations. Bringer, Chabanne and Dottax [5] proposed an implementation using an AES with perturbations, which was subsequently broken by De Mulder, Wyseur and Preneel [14]. Two variants of the scheme of Chow *et al.* were also designed by Xiao and Lai [18] based only on (larger) linear encodings, and by Karroumi [10] using AES dual ciphers. The scheme of Xiao and Lai was successfully cryptanalyzed at SAC 2012 [13] by De Mulder, Roelse and Preneel who showed how to extract the AES key with a work factor of about 2^{32}. On the other hand, the scheme of Karroumi has not been publicly broken so far.

In this paper, we present a new attack against the white-box implementation of AES of Chow *et al.* The key idea is to exploit collisions in output of the first round in order to construct sparse linear systems. Solving these systems then reveals the input encoding and secret key byte(s) involved in some target look-up table. Applied to the original scheme, we get an attack of complexity 2^{22}, conceptually simpler than all previous attacks [3,12]. We further show that the last supposedly secure white-box implementation of AES due to Karroumi [10], is actually vulnerable to a direct application of both our attack and that of Billet *et al.* [3]. This comes from the fact that the overall distribution of the look-up tables in [10] is *exactly the same* as in the initial scheme [7], and therefore does not provide any additional security.

Paper Organization. In Section 2 we recall the design principles and the original white-box AES implementation of Chow *et al.* [7]. Next, in Section 3 we present our attack against this scheme. Finally, in Section 4 we explain why the scheme of [10] is vulnerable to both our attack and the attack of Billet *et al.*

2 Original White-Box Implementation of AES

We assume the reader to be familiar with the AES block cipher. A comprehensive description can be found in the AES standard [1], and further information about the design and its security are given in [8]. This section briefly recalls the white-box AES implementation proposed by Chow *et al.* in [7].

2.1 Design Principles

The strategy adopted by Chow *et al.* for their candidate white-box implementations of DES and AES [6,7] was to transform the given block-cipher into a randomized, key-dependent network of look-up tables. This is performed by three main steps:

1. *Partial Evaluation.* Embed the key in an operation (*e.g.*, by transforming the AES S-box S into key-dependent look-up tables $T_i = S(x \oplus k_i)$).
2. *Tabularizing.* Transforming all the components of the block cipher (even the linear transformations) into look-up tables.
3. *Randomization and Delinearization.* The main idea is as follows. Consider a chain of three look-up tables $L_3 \circ L_2 \circ L_1$, where L_2 contains some information on the key (*e.g.*, $L_2(x) = x \oplus k$). Since the implementation is known to the attacker in the white-box model, he may recover this information. This is prevented by inserting random bijections f_1, f_2 into the look-up tables

$$\begin{cases} L_1 \to L_1' = f_1 \circ L_1 \\ L_2 \to L_2' = f_2 \circ L_2 \circ f_1^{-1} \\ L_3 \to L_3' = L_3 \circ f_2^{-1} \end{cases} .$$

Thus the chain $L_3' \circ L_2' \circ L_1'$ is functionally equivalent to the chain $L_3 \circ L_2 \circ L_1$, but L_2' no longer leaks key information, and the attacker needs to analyze more components in order to gain information. Injecting such bijections needs to be unpredictable, and difficult (ideally impossible) to be removed by an adversary.

2.2 Table-Network Representation of AES

We first describe an implementation of AES which consists in a network of look-up tables (Steps 1 and 2). The white-box implementation presented in Section 2.3 is an *encoded* version of this implementation in which look-up tables are composed with random permutations (Step 3).

An AES round takes as input a state $x = (x_0, x_1, \dots, x_{15})$ and a round key $k = (k_0, k_1, \dots, k_{15})$ and compute a new state $y = (y_0, y_1, \dots, y_{15})$. The new state is computed by groups of 4 bytes through the following transformation:

$$(y_0, y_1, y_2, y_3) = f(x_0, x_5, x_{10}, x_{15})$$

with

$$f : (x_0, x_5, x_{10}, x_{15}) \mapsto \begin{pmatrix} \mathsf{02\ 03\ 01\ 01} \\ \mathsf{01\ 02\ 03\ 01} \\ \mathsf{01\ 01\ 02\ 03} \\ \mathsf{03\ 01\ 01\ 02} \end{pmatrix} \otimes \begin{pmatrix} S(x_0 \oplus k_0) \\ S(x_5 \oplus k_5) \\ S(x_{10} \oplus k_{10}) \\ S(x_{15} \oplus k_{15}) \end{pmatrix} \tag{1}$$

where $\otimes$ denotes the matrix-vector product over $\mathbb{F}_{2^8}$ and S denotes the AES S-box. The other 4-byte groups (y_4, y_5, y_6, y_7), $(y_8, y_9, y_{10}, y_{11})$, and $(y_{12}, y_{13}, y_{14}, y_{15})$ are computed from different coordinates x_i (and corresponding subkeys k_i) according to the ShiftRows permutation. For the sake of simplicity, we will only focus on the first 4-byte group (*i.e.* the first column) in the following.

The above transformation can be computed thanks to four 8×32 look-up tables:

$$\begin{aligned}
T_0(w) &= S(w \oplus k_0) \times (\mathsf{02\ 01\ 01\ 03})^T \\
T_5(w) &= S(w \oplus k_5) \times (\mathsf{03\ 02\ 01\ 01})^T \\
T_{10}(w) &= S(w \oplus k_{10}) \times (\mathsf{01\ 03\ 02\ 01})^T \\
T_{15}(w) &= S(w \oplus k_{15}) \times (\mathsf{01\ 01\ 03\ 02})^T
\end{aligned}$$

as we then have

$$(y_0, y_1, y_2, y_3) \; = \; T_0(x_0) \oplus T_5(x_5) \oplus T_{10}(x_{10}) \oplus T_{15}(x_{15}) \ .$$

A fully tabulated implementation can be obtained by performing each 32-bit XOR in the above equation nibble by nibble (storing a table 32×32 being out of reach), with eight calls to a 8×4 XOR table:

$$\mathrm{XOR}(w_0 \| w_1) = w_0 \oplus w_1 \ .$$

2.3 White-Box Implementation

The white-box implementation of Chow *et al.* [7] consists in deriving an *encoded* version of the table network described in the previous section. The basic idea is to encode the output of a table by composing it with a random permutation, called *output encoding*. In order to keep the correctness of the implementation, the next look-up table must be composed in input with the corresponding inverse permutation, called *input decoding*. Since each of the eight nibbles in output of a T_i table enters a different XOR table, these encodings can only be applied to each nibble separately. In order to add more confusion to the implementation, Chow *et al.* then suggest to further use *mixing bijections* which are random linear transformations.

For the sake of clarity, we first describe the implementation obtained by introducing the mixing bijections to the previous table network. The white-box implementation is then obtained by adding *external encoding* in input and output of the overall cipher, and composing all the tables with random nibble encodings/decodings.[3]

[3] As discussed below, the implementation actually computes the function $\mathrm{AES}'_k = G \circ \mathrm{AES}_k \circ F$, where AES_k denotes the AES-128 decryption under the key k and F, G are external encodings. The first reason (see [7] and [16, Section 3.2.3]) is that it will therefore be harder to separate the white-box implementation from its containing applications, especially when F^{-1} and G^{-1} are performed on a platform independent of the white-box implementation platform; this constraint can indeed make sense for DRM applications, that is the first proposed applications of white-box cryptography, since AES modes of operations might be of no use. The second reason is that it is likely to augment the security of the white-box implementation [16, Section 3.2.3]. Indeed, the lookup tables at the beginning of the network would only be protected by means of *output* nibble encodings, and hence could potentially be more vulnerable to cryptanalysis.

Table network with mixing bijections. Mixing bijections are introduced on each byte of the state as well as on each T_i table output. At the beginning of each round, the state is linearly encoded as $(L_0(x_0), L_1(x_1), \ldots, L_{15}(x_{15}))$ where the L_i are random $\mathbb{F}_2^8$-linear bijections.

In order to keep the encryption correctness, one must compose each table T_i with L_i^{-1} in input. A 32-bit mixing bijection is further applied in output of the T_i tables. Namely, each table T_i is replaced by the table T_i' satisfying

$$T_i'(x) = M \circ T_i \circ L_i^{-1} \ , \tag{2}$$

where M is a random $\mathbb{F}_2^{32}$-linear bijection.[4] One can then obtain a linear encoding of the y_i from the $L_i(x_i)$ as we have

$$M(y_0, y_1, y_2, y_3) \ = \ T_0'(L_0(x_0)) \oplus T_5'(L_5(x_5)) \oplus T_{10}'(L_{10}(x_{10})) \oplus T_{15}'(L_{15}(x_{15})) \ . \tag{3}$$

In order to complete the AES round computation, one must translate the above 32-bit linear encoding into a byte-wise linear encoding $(L_0'(y_0), L_1'(y_1), L_2'(y_2), L_3'(y_3))$ for the next round. Namely, one must apply the linear mapping

$$H = (L_0' \| L_1' \| L_2' \| L_3') \circ M^{-1} \tag{4}$$

to $M(y_0, y_1, y_2, y_3)$. This is done by applying the tables

$$\begin{aligned} H_0(x) &= (L_0' \| L_1' \| L_2' \| L_3') \circ M^{-1}(x, 0, 0, 0) \\ H_1(x) &= (L_0' \| L_1' \| L_2' \| L_3') \circ M^{-1}(0, x, 0, 0) \\ H_2(x) &= (L_0' \| L_1' \| L_2' \| L_3') \circ M^{-1}(0, 0, x, 0) \\ H_3(x) &= (L_0' \| L_1' \| L_2' \| L_3') \circ M^{-1}(0, 0, 0, x) \end{aligned}$$

to each byte of $M(y_0, y_1, y_2, y_3)$ and then XORing the outputs. That is, one computes

$$(L_0'(y_0), L_1'(y_1), L_2'(y_2), L_3'(y_3)) \ = \ H_0(m_0) \oplus H_1(m_1) \oplus H_2(m_2) \oplus H_3(m_3) \ ,$$

where $(m_0, m_1, m_2, m_3) = M(y_0, y_1, y_2, y_3)$.

Nibble encodings and XOR tables. The overall white-box implementation is finally obtained by introducing random bijections to encode every nibble. That is, each byte of the state is represented as $(P_{0,i} \| P_{1,i}) \circ L_i(x_i)$ for random bijections $P_{i,0}$ and $P_{i,1}$. The input decoding $(P_{0,i}^{-1} \| P_{1,i}^{-1})$ is then applied to the corresponding T_i' table, as well as an output encoding $(Q_{0,i}, Q_{1,i}, \ldots, Q_{7,i})$ for random bijections $Q_{j,i}$. Then the outputs of the T_i' tables are added thanks to encoded XOR tables. These tables are applied nibble-wise to remove the $Q_{j,i}$ encodings, XOR the nibbles, and encode the output nibble with a new random bijection. Eight such tables are first applied to add the outputs of T_0' and T_5', then to add the outputs of T_{10}' and T_{15}', and eventually to add both results. One proceeds similarly to encode the H_i tables and to XOR their outputs.

[4] Note that a different mixing bijection is involved for each group of four T_i tables.

External encodings. Eventually, in order to encode the input plaintext (and to decode the output ciphertext) without revealing the initial byte-encodings (and their final byte-decoding counterparts), one adds so-called *external encodings*. In other words, the implementation computes the function

$$\mathrm{AES}'_k = G \circ \mathrm{AES}_k \circ F^{-1} \ ,$$

where AES_k denotes the AES-128 decryption under the key k.

As suggested in [7], the external encodings can be defined as the composition of 128×128 mixing bijections and nibble permutations. We let the interested reader refer to [7] for further details. Note that the attack presented in this paper would work with arbitrary external encodings.

Byte Shuffling. Also not explicitly mentioned in Chow *et al.* 's paper, one could use a random shuffling of the byte of the AES state in order to add confusion to the implementation. Indeed, while the designer must ensure that the output of a look-up table well enters in the corresponding look-up table, the order in which these look-ups are done may be randomized. Equivalently, an attacker may not *a priori* know which look-up table corresponds to which byte of the state. Of course he can always group the bytes four-by-four (*i.e.* by columns) since the four bytes of the same column enter the same layer of encoded XOR tables after the T'_i tables. However, the four columns may be treated in a random order and the four bytes within a column may also be treated in a random order. Since such a byte shuffling may be used in the code without changing the definition of the look-up tables of Chow *et al.* 's implementation, we show that our attack can be easily extended to deal with this context in Appendix A. Note that the attack by Billet *et al.* also works assuming such a random byte shuffling.

3 Our Attack

We denote $E_i = (P_{i,0} \| P_{i,1}) \circ L_i$ and $E'_i = (P'_{i,0} \| P'_{i,1}) \circ L'_i$ the byte-encodings of the state in input and in output of the *first round* respectively. According to the previous section, applying a set of successive tables (T'_i, H'_i and XOR tables), one can compute the function

$$f' = (E'_0 \| E'_1 \| E'_2 \| E'_3) \circ f \circ (E_0^{-1} \| E_5^{-1} \| E_{10}^{-1} \| E_{15}^{-1}) \tag{5}$$

where f is the function defined in (1). Let us denote by f'_i the coordinate functions of f' such that $f' = (f'_0, f'_1, f'_2, f'_3)$. Let us further denote by S_j the function defined as

$$S_j(\cdot) = S(k_j \oplus E_j^{-1}(\cdot)) \ . \tag{6}$$

3.1 Recovering the S_j Functions

Our attack consists in finding collisions in output of the coordinate functions f'_i in order to recover functions S_0, S_5, S_{10} and S_{15} and associated key bytes. We start with the recovery of S_0 and S_5 by looking for collision of the form

$$f'_0(\alpha, 0, 0, 0) = f'_0(0, \beta, 0, 0) \ . \tag{7}$$

The above equation can be rewritten as

$$E_0'\big(\mathsf{02} \cdot S_0(\alpha) \oplus \mathsf{03} \cdot S_5(0) \oplus c\big) = E_0'\big(\mathsf{02} \cdot S_0(0) \oplus \mathsf{03} \cdot S_5(\beta) \oplus c\big)$$

where $c = S_{10}(0) \oplus S_{15}(0)$, implying

$$\mathsf{02} \cdot S_0(\alpha) \oplus \mathsf{03} \cdot S_5(0) = \mathsf{02} \cdot S_0(0) \oplus \mathsf{03} \cdot S_5(\beta) \ . \tag{8}$$

Collecting several such equations, we can construct a linear system to recover S_0 and S_5. Let u_0, u_1, …, u_{255} and v_0, v_1, …, v_{255} denote the unknowns associated to the outputs of S_0 and S_5 (*i.e.* $u_i = S_0(i)$ and $v_i = S_5(i)$). Then (8) can be rewritten as

$$\mathsf{02} \cdot (u_0 \oplus u_\alpha) \oplus \mathsf{03} \cdot (v_0 \oplus v_\beta) = 0 \ . \tag{9}$$

Then we can easily obtain a system involving all the u_i and all the v_i. Indeed, the functions $\alpha \mapsto f_0'(\alpha, 0, 0, 0)$ and $\beta \mapsto f_0'(0, \beta, 0, 0)$ are bijections, so we get exactly 256 collisions between $f_0'(\alpha, 0, 0, 0)$ and $f_0'(0, \beta, 0, 0)$ while α and β vary over $\mathbb{F}_{2^8}$. Discarding the irrelevant collision for $(\alpha, \beta) = (0, 0)$, we get 255 pairs (α, β) satisfying $f_0'(\alpha, 0, 0, 0) = f_0'(0, \beta, 0, 0)$ and providing an equation of the form of (9). Moreover, every unknown u_α and v_β appears once for $\alpha, \beta > 0$ and the unknowns u_0 and v_0 appear in each equation. We proceed similarly for coordinates f_i' with $i \in \{1, 2, 3\}$, for which the collisions give rise to similar equations but with different pairs of coefficients in $\{\mathsf{01}, \mathsf{02}, \mathsf{03}\}$. For instance a collision $f_1'(\alpha, 0, 0, 0) = f_1'(0, 0, \beta, 0)$ yields an equation

$$\mathsf{01} \cdot (u_0 \oplus u_\alpha) \oplus \mathsf{02} \cdot (v_0 \oplus v_\beta) = 0 \ .$$

We hence get 4×255 linear equations involving all the 512 unknowns. However, this system is not of full rank. Consider the 2×255 unknowns $u_i' = u_0 \oplus u_i$ and $v_i' = v_0 \oplus v_i$ for $i \in \{1, 2, \ldots, 255\}$. Every equation of the form of (9) can be rewritten as

$$\mathsf{02} \cdot u_\alpha' \oplus \mathsf{03} \cdot v_\beta' = 0 \ .$$

This shows that the system can be rewritten in terms of 510 unknowns and is hence of rank at most 510. But the system has still at least one degree of freedom left, since more than one solution is still possible. For instance, the system is solved by $u_i' = 0$ and $v_i' = 0$ for every i, and it is also solved by the solution we are looking for (*i.e.* $u_i' = S_0(0) \oplus S_0(i)$ and $v_i' = S_5(0) \oplus S_5(i)$), which is such that $u_i' \neq 0$ and $v_i' \neq 0$ by bijectivity of S_0 and S_5. The obtained system is hence of rank at most 509.

In all our experiments, the 4×255 available linear equations always yielded a system of rank 509. From such a system, all the unknowns can be expressed in function of one unknown, say u_1'. And since all the unknowns are linearly linked, there exist coefficients a_i and b_i such that $u_i' = a_i \cdot u_1'$ and $v_i' = b_i \cdot u_1'$. These coefficients can be easily recovered by solving the system for $u_1' = 1$. We then get

$$u_i = a_i \cdot (u_0 \oplus u_1) \oplus u_0 \ , \tag{10}$$

and

$$v_i = b_i \cdot (u_0 \oplus u_1) \oplus v_0 \ . \tag{11}$$

From the a_i coefficients and from Equation (10), we can recover the overall function S_0 by exhaustive search on the pair (u_0, u_1). In order to determine the good solution, we use the particular structure of the function S_0. Specifically, we use the relation

$$S^{-1} \circ S_0(\cdot) = E_0^{-1}(\cdot) \oplus k_0 \ .$$

By definition of E_0, the above function has algebraic degree at most 4. We then use the following lemma.

Lemma 1. *Let g be a function from $\{0,1\}^8$ to itself with algebraic degree at most 4. The map*

$$\varphi : x \mapsto \bigoplus_{\alpha=0}^{15} g(x \oplus \alpha) \ ,$$

is the null function $x \mapsto 0$.

Proof. The map φ defined in Lemma 1 is a 4th-order derivative of the function g (specifically $\varphi = D_1 D_2 D_4 D_8(g)$) and since g has algebraic degree at most 4, all its 4th-order derivatives are null. $\square$

Remark 1. For a wrong pair (u_0, u_1), the candidate function $\hat{S}_0$ obtained from (10) is affine equivalent to S_0. Namely there exist a and b such that $\hat{S}_0(\cdot) = a \cdot S_0(\cdot) \oplus b$, with $a \neq 0$ and $(a, b) \neq (0, 1)$. The function $S^{-1} \circ \hat{S}_0$ then satisfies

$$S^{-1} \circ \hat{S}_0(\cdot) = S^{-1}\big(a \cdot S(k_0 \oplus E_0^{-1}(\cdot)) \oplus b\big) \ ,$$

and it has an algebraic degree greater than 4 with overwhelming probability.[5]

According to Lemma 1 and the above remark, we can easily determine the good pair (u_0, u_1) by computing the 4th-order derivative $\hat{\varphi}$ of the associated function $\hat{g} = S^{-1} \circ \hat{S}_0$, which satisfies

$$\hat{\varphi}(x) = \bigoplus_{\alpha=0}^{15} S^{-1}(a_{x \oplus \alpha} \cdot (u_0 \oplus u_1) \oplus u_0) \ .$$

For the sake of efficiency, we first compute $\hat{\varphi}(0)$ and check whether it equals 0 or not. If we get $\hat{\varphi}(0) = 0$, we step forwards and compute $\hat{\varphi}(x)$ for another x. Note that we only need to compute $\hat{\varphi}$ for 16 inputs at most since for every x we have $\hat{\varphi}(x) = \hat{\varphi}(x \oplus \mathsf{01}) = \cdots = \hat{\varphi}(x \oplus \mathsf{15})$. Getting $\hat{\varphi}(x) = 0$ for a wrong pair (u_0, u_1) should roughly occur with probability 1/256, so wrong guesses are quickly discarded.

Once S_0 has been recovered, we can recover S_5 from (11) by exhaustive search on v_0. Here again, the good solution is determined using Lemma 1 and the above approach. The remaining functions S_{10} and S_{15} are recovered similarly by solving the linear systems arising from collisions of the form $f'_i(\alpha, 0, 0, 0) = f'_i(0, 0, \beta, 0)$ and $f'_i(\alpha, 0, 0, 0) = f'_i(0, 0, 0, \beta)$. Since S_0 is already known, we get the same situation as for the recovery of S_5. Namely, all the elements of S_{10} (resp. S_{15}) can be expressed as affine

[5] We ran a few million tests and never obtained a function with algebraic degree 4 or less.

functions of $S_{10}(0)$ (resp. $S_{15}(0)$), and we can recover the overall function by exhaustive search on this value and with the selection criterion of Lemma 1.

The other functions S_j involved in the other 4-byte transformations can be recovered in the exact same way (only the indices change).

3.2 Recovering the Secret Key

Once the S_j functions have been recovered, one can easily recover the byte-encodings E'_i in output of the first round. For instance evaluating $f'_0(\alpha, 0, 0, 0)$ one gets the value $E'_0\big(\psi(\alpha)\big)$ where

$$\psi : \alpha \mapsto \mathsf{02} \cdot S_0(\alpha) \oplus \mathsf{03} \cdot S_5(0) \oplus S_{10}(0) \oplus S_{15}(0)$$

is a bijective function. We hence get $E'_0(\cdot) = f'_0(\psi^{-1}(\cdot), 0, 0, 0)$ which enables to fully retrieve E'_0 by looping on the 256 input values. Each output byte-encoding E'_i can be recovered in a similar way.

Since the output byte-encodings of the first round are the inverse of the input byte-decodings of the second round, we now show how to retrieve the key bytes in the second round from that knowledge. This is equivalent to retrieving the key bytes involved in the function f' defined in (5) from the E_j, so we keep these notations to describe this stage.

For the recovery of k_0, we use the following distinguisher. Consider the function g associated to k_0 and defined as:

$$g = f'_0(E_0(S^{-1}(\cdot) \oplus k_0), 0, 0, 0) \ .$$

This function satisfies

$$g(x) = E'_0(\mathsf{02} \cdot x \oplus c) \qquad \text{where } c = \mathsf{03} \cdot S_5(0) \oplus S_{10}(0) \oplus S_{15}(0) \ ,$$

and it has algebraic degree at most 4 by definition of E'_0 (since multiplying and adding constant coefficients are linear). Therefore, according to Lemma 1, the 4th-order derivative $\varphi : x \mapsto \bigoplus_{\alpha=0}^{15} g(x \oplus \alpha)$ equals the null function. On the other hand, consider the function $\hat{g}$ associated to a wrong guess $\hat{k}_0 \neq k_0$, that is

$$\hat{g}(x) = f'_0(E_0(S^{-1}(x) \oplus \hat{k}_0), 0, 0, 0) = E'_0(\mathsf{02} \cdot S(S^{-1}(x) \oplus \hat{k}_0 \oplus k_0) \oplus c) \ .$$

This function has algebraic degree greater than 4 with overwhelming probability.[6] This way, we can easily recover k_0 by exhaustive search while testing for every candidate whether the function $\hat{g}$ is of algebraic degree 4 or not. Namely, for every guess $\hat{k}_0$, we test whether the function

$$\hat{\varphi}(x) = \bigoplus_{\alpha=0}^{15} f'_0(E_0(S^{-1}(x) \oplus \hat{k}_0), 0, 0, 0)$$

[6] Here again, we ran a few million tests and never obtained a function with algebraic degree 4 or less.

equals the null function $x \mapsto 0$, or not. As for the previous recovery of the S_j functions, this is done at most for 16 different values of x since we have $\hat{\varphi}(x) = \hat{\varphi}(x \oplus \mathsf{01}) = \cdots = \hat{\varphi}(x \oplus \mathsf{15})$. Moreover, as for the recovery of the S_j, we only need to compute $\hat{\varphi}$ for 16 inputs at most since for every x we have $\hat{\varphi}(x) = \hat{\varphi}(x \oplus \mathsf{01}) = \cdots = \hat{\varphi}(x \oplus \mathsf{15})$. Moreover getting $\hat{\varphi}(x) = 0$ for a wrong guess $\hat{k}_0$ roughly occur with probability 1/256, so wrong guesses are quickly discarded.

The key bytes k_5, k_{10} and k_{15} can be retrieved similarly; only the definition of the function g shall change. For instance, g is defined as $f'_0(0, E_5(S^{-1}(\cdot) \oplus k_5), 0, 0)$ for k_5, and so on for k_{10} and k_{15}. And the other key bytes k_j involved in the other 4-byte transformations can be recovered in the exact same way (only the indices change). Eventually, from the second round key, one can easily recover the full AES secret key by inverting the key schedule process.

3.3 Recovering the External Encodings

From the S_j functions involved in the first round, and the secret key (which is also the first round key), we directly obtain the byte-encodings E_j in input of the first round by the relation $E_j(\cdot) = S_j^{-1} \circ S(\cdot) \oplus k_j$. Then we can cancel these byte-encodings out in output of the initial external encoding in order to recover F^{-1} (under the form of a table-network).

As shown in Section 3.2, one can easily recover the output byte-encodings of a round from the input byte-decodings and the round key bytes. Since we know the byte-decodings in input of the second round, we can thus recover the corresponding output byte-encodings. And since the output byte-encodings of the second round are the inverse input byte-decodings of the third round, we can reiterate such recovery for the fourth round and so on. We can hence sequentially recover the byte-encodings in input/output of each round.[7] Once the output byte-encodings of the last round have been recovered, we can cancel them out in input of the final external encoding in order to recover G (under the form of a table-network).

3.4 Attack Complexity

The bottleneck of our attack is the exhaustive search to recover the function S_0 (resp. S_1, S_2 and S_3 for the three other columns). Indeed, the previous system to solve for the recovery of the a_i and b_i coefficients is very sparse and it can hence be solved with Gaussian elimination in linear complexity (*i.e.* in 512 times a few operations). To recover S_0, one loops on the 2^{16} candidate values for (u_0, u_1), and for each value test whether $\hat{\varphi}(x) = 0$ (which is a XOR over 16 elements) for at most 16 values x. We use laziness, namely we test whether $\hat{\varphi}(0) = 0$ first, if false we stop and if true we step forwards to the next x, and so on and so forth. Now getting $\hat{\varphi}(x) = 0$ for a wrong pair

[7] Note that the recovery of the output byte-encodings of the last round slightly differs from that for the other rounds (the last round being slightly different). Nevertheless we can still apply the same principle for the recovery of the output byte-encoddings; only the mapping ψ shall change (but it is still computable by the attacker).

(u_0, u_1) roughly occurs with probability $1/256$, therefore the expected number of tests is $1 + 1/256 + \cdots + 1/(256^{15}) \leq 1.004$. The complexity of the recovery of S_0 is hence of

$$2^{16} \cdot 1.004 \cdot 2^4 \approx 2^{20} .$$

Then the recovery of S_5 (resp. S_{10}, S_{15}) from S_0 only requires an exhaustive search on v_0, which makes a complexity of $2^8 \cdot 1.004 \cdot 2^4 \approx 2^{12}$. We hence get a complexity of $2^{20} + 3 \cdot 2^{12} \approx 2^{20}$ for the recovery of S_0, S_5, S_{10} and S_{15}. This computation must be performed for each column, which makes a total complexity of $4 \times 2^{20} = 2^{22}$.

The recovery of the key bytes and of the external encodings has a negligible complexity compared to the recovery of the S_j functions. Indeed, according to the above analysis, the recovery of one key byte is roughly of $2^8 \cdot 1.004 \cdot 2^4 \approx 2^{12}$. This must be done 16 times, yielding a complexity of $16 \cdot 2^{12} \ll 2^{22}$. On the other hand, recovering the external encodings requires recovering the input/output byte-encodings of every round. A total 10×16 encodings must hence be recovered (16 per round) and each recovery takes 2^8 times a few operations. This makes a complexity of $160 \cdot 2^8 < 2^{16}$ times a few operations which is negligible compared to 2^{22}.

Comparison with Billet *et al.* attack. The key idea of the attack proposed in [3] is to fix some values $c_1, c_2, c_3, c'_3 \in \mathbb{F}_{2^8}$ and to consider the function

$$f'_0({f'}_0^{-1}(\cdot, c_1, c_2, c_3), c_1, c_2, c'_3) = E'_0({E'}_0^{-1}(\cdot) \oplus \beta) ,$$

where β is a constant depending on c_1, c_2, c_3, c'_3 and on the involved secret key bytes. From this mapping, they first recover the non-linear part of E'_0 in time complexity 2^{24}. They do the same for every i, making a total complexity of $16 \times 2^{24} = 2^{28}$. Then they can recover the affine part of the encodings column by column with smaller complexity. Finally they need to perform such recovery on two successive rounds to be able to extract the key, which makes a total complexity of 2^{29}.

In comparison, our approach works on the first round only by finding collisions on output of f'. This allows us to construct a sparse linear system from which, we can recover the four input encodings E_i and the four key bytes k_i involved in f' in complexity 2^{20}. In addition to having smaller complexity, our attack is conceptually simpler and easier to implement than all previous attacks on white-box AES implementations [3,12,14,13].

4 White-Box Implementation using AES Dual Ciphers

In [10], Karroumi proposed a modification of the Chow *et al.* AES implementation [7] making use of dual representations of the AES cipher [2,4,15], and aiming at improving the resistance of Chow *et al.* scheme to Billet *et al.* attack [3] by a factor 2^{63}. In this section, we briefly present the strategy of [10] and explain that the modified scheme remains vulnerable to the raw versions of both Billet *et al.* attack and our attack presented in Section 3. We indeed show hereafter that, for any given secret key, the distribution of the encoded function f' (c.f. (5)) remains *exactly the same* after applying Karroumi's countermeasure to the original implementation.

Description of the Strategy of [10]. Most of the operations in AES are based on the field $\mathbb{F}_{2^8}$, officially defined as $\mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$. In 2002, Barkan and Biham [2] showed that by changing the polynomial used to define the field, the ciphers produced are dual version of AES. That is, for any such representation $\overline{\text{AES}}$, there exists an isomorphism $\varDelta$ of the field $\mathbb{F}_{2^8}$ such that

$$\boldsymbol{\varDelta}(\text{AES}_k(w)) = \overline{\text{AES}}_{\boldsymbol{\varDelta}(k)}(\boldsymbol{\varDelta}(w)) \ , \qquad \forall \ w, k \in (\mathbb{F}_{2^8})^{16} \ ,$$

where $\boldsymbol{\varDelta} = (\varDelta || \varDelta || \cdots || \varDelta)$ (*i.e.* $\boldsymbol{\varDelta}$ is the bytewise application of $\varDelta$).

In [4], Biryukov *et al.* show that there are at least $61,200$ dual versions of AES. The approach followed by Karroumi in [10] is to make extensive use of these dual representations to further randomize each lookup table in Chow *et al.* implementation. Specifically, a different dual representation is used for each column of the state in each round, which is randomly chosen. The aim of this technique was to modify at each round and for each column the AES S-box and the AES MixColumns matrix (due to the chosen representation), supposedly forcing the attacker to guess the representations and therefore adding a factor $61200^4 \approx 2^{63}$ to the attack.

Let us explain the countermeasure of [10] in more detail. As in Section 2.3 we focus on the group of 4-bytes $(x_0, x_5, x_{10}, x_{15})$, *i.e.* the first column. Assume temporarily that the x_i's follow the classic AES representation, and assume that we want to work with the representation $\varDelta$ on this round and this column. First, one transforms the T_i tables into tables $T_i^{\varDelta}$ as detailed below for T_0:

$$T_0(w) = S(w \oplus k_0) \times (\mathsf{02\ 01\ 01\ 03})^T$$

is transformed into

$$T_0^{\varDelta}(w) = S^{\varDelta}(w \oplus \varDelta(k_0)) \times (\varDelta(\mathsf{02})\ \varDelta(\mathsf{01})\ \varDelta(\mathsf{01})\ \varDelta(\mathsf{03}))^T \ ,$$

where $S^{\varDelta}$ is the AES S-box under the representation $\varDelta$, *i.e.* verifies that

$$S^{\varDelta}(\varDelta(x)) = \varDelta(S(x)) \ .$$

Now, it is easy to see that, applied to some $\varDelta(w)$, the table $T_0^{\varDelta}$ yields

$$T_0^{\varDelta}(\varDelta(w)) = \boldsymbol{\varDelta} \circ T_0(w) \ , \tag{12}$$

which is the table T_0 composed with $\varDelta$ for each of its four bytes, thus the desired result.

The scheme of Karroumi consists in replacing the encoded tables T_i' (see (2)) in the Chow *et al.* implementation, by dual encoded tables. In addition, each table include a switching from one dual representation $\varDelta$ in input, to another representation $\varDelta'$ in output. Namely, the encoded tables T_i' are replaced by the encoded tables $T_i^{\varDelta,\varDelta'}$ satisfying:

$$T_i^{\varDelta,\varDelta'} = M \circ T_i^{\varDelta} \circ \varDelta' \circ \varDelta^{-1} \circ L_i^{-1} = M \circ \boldsymbol{\varDelta}' \circ T_i \circ L_i^{-1} \ ,$$

where L_i and M are the mixing bijections involved in T_i'. The above table is equivalent to the T_i' table, but it includes a switching of dual representations from $\varDelta$ to $\varDelta'$. Then to complete the dual AES round computation, one apply the linear mapping H as defined in (4) via H_i tables as in the original implementation.

Look-up Tables Distribution. Since M is uniformly chosen as a $\mathbb{F}_2^{32}$ linear bijection, $M \circ \boldsymbol{\Delta'}$ follows the uniform distribution over the $\mathbb{F}_2^{32}$ linear bijections, and thus the distribution of ${T_i}^{\Delta,\Delta'}$ is exactly the same as the distribution of T'_i. And for the same reason, the joint distribution of $T_0^{\Delta,\Delta'}$, $T_5^{\Delta,\Delta'}$, $T_{10}^{\Delta,\Delta'}$ and $T_{15}^{\Delta,\Delta'}$ is same as the joint distribution of T'_0, T'_5, T'_{10} and T'_{15}. This implies that the distribution of f' as defined in Equation (5) is exactly the same in the scheme of Chow *et al.* [7] and the scheme of Karroumi [10]. Therefore both our attack and the attack of Billet *et al.*, which use the function f', directly apply without any modification. In particular, working with dual ciphers does not change anything against the existing white-box attacks.

Another way to end up to the same conclusion is to realize that the distributions of every table of the scheme of [10] are exactly the same as the scheme of [7]. Indeed all the representation isomorphisms Δ are composed on the left with *random* linear transformations ℓ, and therefore the distributions of $\{\ell \circ \Delta\}_\ell$ and $\{\ell\}_\ell$ are identical. Since the distributions of the tables does not change between the two implementations, any attack against [7] applies directly to [10].

References

1. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
2. Elad Barkan and Eli Biham. In how many ways can you write Rijndael? In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 160–175. Springer Berlin Heidelberg, 2002.
3. Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In *SAC 2004*, pages 227–240. Springer-Verlag, 2005.
4. Alex Biryukov, Christophe Cannière, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer Berlin Heidelberg, 2003.
5. Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. Cryptology ePrint Archive, Report 2006/468, 2006. http://eprint.iacr.org/.
6. Stanley Chow, Phil Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *DRM 2002*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.
7. Stanley Chow, Phil Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In *SAC 2002*, pages 250–270. Springer-Verlag, 2003.
8. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
9. Louis Goubin, Jean-Michel Masereel, and Michaël Quisquater. Cryptanalysis of white box DES implementations. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *SAC 2007*, volume 4876, pages 278–295. Springer Berlin Heidelberg, 2007.
10. Mohamed Karroumi. Protecting white-box aes with dual ciphers. In *Proceedings of the 13th international conference on Information security and cryptology*, ICISC'10, pages 278–291. Springer-Verlag, 2010.
11. Hamilton E. Link and William D. Neumann. Clarifying obfuscation: improving the security of white-box DES. In *ITCC 2005*, volume 1, pages 679–684, 2005.
12. Wil Michiels, Paul Gorissen, and Henk D. L. Hollmann. Cryptanalysis of a generic class of white-box implementations. In *SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 414–428. Springer, 2009.

13. Yoni Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao - Lai white-box AES implementation. In LarsR. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707, pages 34–49. Springer Berlin Heidelberg, 2013.
14. Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a perturbated white-box aes implementation. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT 2010*, volume 6498, pages 292–310. Springer Berlin Heidelberg, 2010.
15. Håvard Raddum. More dual Rijndaels. In Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors, *Advanced Encryption Standard AES*, volume 3373 of *Lecture Notes in Computer Science*, pages 142–147. Springer Berlin Heidelberg, 2005.
16. Brecht Wyseur. *White-Box Cryptography.* PhD thesis, Katholieke Universiteit Leuven, 2009.
17. Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *SAC 2007*, volume 4876, pages 264–277. Springer Berlin Heidelberg, 2007.
18. Yaying Xiao and Xuejia Lai. A secure implementation of white-box AES. In *CSA 2009*, pages 1–6, 2009.

A Dealing with Byte Shuffling

As mentioned above and although not explicit in Chow *et al.* 's paper, one could use a random shuffling of the byte of the AES state. In such a case the attacker could only group the bytes by column, without knowing the index of each column and the byte indices within a column. In such a case, the attacker has to deal with a function f' as defined in (5) but the four inputs x_i and the four coordinate functions f'_i have a random order. It results that for the recovery of the S_j functions, the MixColumn coefficients may not be the ones expected in the original attack. Specifically, each coordinate function of the attacked function f' is such that

$$f'_i(x_{\sigma(0)}, x_{\sigma(5)}, x_{\sigma(10)}, x_{\sigma(15)}) \\ = E'_i(c_0 \cdot S_0(x_0) \oplus c_5 \cdot S_5(x_5) \oplus c_{10} \cdot S_{10}(x_{10}) \oplus c_{15} \cdot S_{15}(x_{15})) \quad (13)$$

where σ is an unknown permutation of $\{0, 5, 10, 15\}$ and where c_0, c_5, c_{10}, c_{15} are unknown coefficients from $\{\mathtt{01}, \mathtt{02}, \mathtt{03}\}$ (two of them being $\mathtt{01}$). Here, a collision of the form $f'_0(\alpha, 0, 0, 0) = f'_0(0, \beta, 0, 0)$ gives rise to an equation

$$c_k \cdot (u_0 \oplus u_\alpha) \oplus c_\ell \cdot (v_0 \oplus v_\beta) = 0$$

where $k = \sigma(0)$, $\ell = \sigma(5)$, $u_i = S_k(i)$ and $v_i = S_\ell(i)$ for every i. Let us define $u'_i = u_0 \oplus u_i$ and $v'_i = \frac{c_\ell}{c_k}(v_0 \oplus v_i)$, the above equation can be rewritten as

$$u'_i \oplus v'_i = 0 \ . \quad (14)$$

Now a collision for another coordinate function, say $f'_1(\alpha, 0, 0, 0) = f'_1(0, \beta, 0, 0)$, gives rise to an equation

$$d_k \cdot (u_0 \oplus u_\alpha) \oplus d_\ell \cdot (v_0 \oplus v_\beta) = 0 \ ,$$

where d_k and d_ℓ are further unknown coefficients from $\{\mathtt{01}, \mathtt{02}, \mathtt{03}\}$. The above equation can be rewritten as

$$u'_i \oplus \frac{d_\ell c_k}{d_k c_\ell} v'_i = 0 \ . \quad (15)$$

Then we exhaustively guess the coefficient fraction $\frac{d_\ell c_k}{d_k c_\ell}$ among the set of 14 possible candidates (resulting from the MixColumns definition):

$$\left\{\mathsf{02},\mathsf{03},\mathsf{04},\mathsf{06},\frac{\mathsf{03}}{\mathsf{02}},\frac{\mathsf{01}}{\mathsf{02}},\frac{\mathsf{09}}{\mathsf{02}},\frac{\mathsf{01}}{\mathsf{03}},\frac{\mathsf{02}}{\mathsf{03}},\frac{\mathsf{04}}{\mathsf{03}},\frac{\mathsf{01}}{\mathsf{06}},\frac{\mathsf{01}}{\mathsf{09}},\frac{\mathsf{02}}{\mathsf{09}}\right\} .$$

For each guess on the coefficient fraction we try to solve the system arising from the 2×255 equations of the form (14) and (15), with the additional equation $u'_1 = 1$ as in the original attack. If the guess is wrong, we get an unsolvable system. For the good guess we get one or more solutions depending on the system rank. If more than one solutions are possible, this means that the system rank is lower than 510, and we need more equations. In that case, we use collisions from an additional coordinate function, say f'_2, which yield equations with a new unknown coefficient fraction. Here again, the fraction is exhaustively guessed in the above set, and the good candidate yields a solvable system.[8] Once the system solved, we obtained the a_i and b_i coefficients such that

$$u_i = a_i \cdot (u_0 \oplus u_1) \oplus u_0 \ ,$$

and

$$v_i = b_i \cdot \frac{c_k}{c_\ell}(u_0 \oplus u_1) \oplus v_0 \ .$$

Then we proceed as for the original attack by exhaustively guessing (u_0, u_1) with the test of Lemma 1 to recover $S_{\sigma(0)}$. For the recovery of $S_{\sigma(5)}$ we must then guess v_0 as well as the coefficient fraction c_k/c_ℓ (among at most 7 possible candidates). As in the original attack, the remaining functions $S_{\sigma 10}$ and $S_{\sigma(15)}$ are then recovered by exploiting collisions of the form $f'_i(\alpha, 0, 0, 0) = f'_i(0, 0, \beta, 0)$ and $f'_i(\alpha, 0, 0, 0) = f'_i(0, 0, 0, \beta)$.

The recovered coefficient fractions yield the unknown coefficients in the f'_i coordinate functions. From these coefficients and the $S_{\sigma(j)}$ functions, one can recover the output byte-encodings as described in Section 3.2. Then the recovery of the second round-key bytes works as in the original attack (in particular the coefficients in the f' function are not required). Following the same approach, we can recover the key bytes of the third round.

Eventually, it remains to identify the good ordering for the key bytes. As explained in Section 2.3, we can easily group the bytes of each round key four-by-four *i.e.* by column (each group being involved in a different f' function). Then, the key schedule of AES makes it simple to recover the good ordering by checking the correspondence between the first and fourth columns of a round key and the first column of the next one. Speciffically, we have:

$$w_{i+1,1} = \mathsf{SubWord}(\mathsf{RotWord}(w_{i,4})) \oplus w_{i,1} \oplus \mathtt{Rcon}_i$$

where $w_{i,j}$ denotes the jth column of the ith round key and $\mathtt{Rcon}_i$ is some constant value (see [1]). From the recovered shuffled key columns we have $4 \times 3 = 12$ possibilities

[8] In our experiments most systems of $2 \times 255(+1)$ equations arising from the collisions in output of two different coordinate functions are of full rank. However, some underdetermined system occur. In that case, adding the equations arising from the collisions of a third coordinate function always makes the system full rank.

for the pair of columns $(w_{2,1}, w_{2,4})$. Then we have $4! = 24$ possibilities for the order of bytes within each column, making a total of $12 \times 24^2 < 2^{13}$ possibilities. When one of these possibilities yields four bytes corresponding to one group of third round key, we have identified the involved columns with high probability. Then we apply the same principle with the next column following the equation:

$$w_{i+1,j} = w_{i,j} \oplus w_{i+1,j-1} \ ,$$

holing for $j \in \{2, 3, 4\}$. It might occur (still with low probability) that several candidates satisfy one of the above equations. In that case we continue with each candidate until a inconsistency is found.

As a final note, the above attack has the same bottleneck as the original attack, that is the exhaustive search on u_0 and u_1 for the recovery of the first S_j function (here $S_{\sigma(0)}$) in each of the four f' functions of first round. Therefore, although slightly more complicated, the complexity of this extended attack is also of 2^{22}.