Two Attacks on a White-Box AES Implementation[⋆]

Tancrède Lepoint[1,2], Matthieu Rivain[1],
Yoni De Mulder[3], Peter Roelse[4], and Bart Preneel[3]

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

[2] École Normale Supérieure, France

[3] KU Leuven and iMinds, Belgium
{yoni.demulder,bart.preneel}@esat.kuleuven.be

[4] Irdeto B.V., The Netherlands
peter.roelse@irdeto.com

Abstract. White-box cryptography aims to protect the secret key of a cipher in an environment in which an adversary has full access to the implementation of the cipher and its execution environment. In 2002, Chow, Eisen, Johnson and van Oorschot proposed a white-box implementation of AES. In 2004, Billet, Gilbert and Ech-Chatbi presented an efficient attack (referred to as the BGE attack) on this implementation, extracting its embedded AES key with a work factor of 2^{30}. In 2012, Tolhuizen presented an improvement of the most time-consuming phase of the BGE attack. The present paper includes three contributions. First we describe several improvements of the BGE attack. We show that the overall work factor of the BGE attack is reduced to 2^{22} when all improvements are implemented. This paper also presents a new attack on the initial white-box implementation of Chow *et al.* This attack exploits collisions occurring on internal variables of the implementation and it achieves a work factor of 2^{22}. Eventually, we address the white-box AES implementation presented by Karroumi in 2010 which aims to withstand the BGE attack. We show that the implementations of Karroumi and Chow *et al.* are the same, making them both vulnerable to the same attacks.

Keywords. White-box Cryptography, AES Implementation, Dual Cipher, Cryptanalysis.

1 Introduction

In 2002, Chow *et al.* introduced the concept of white-box cryptography by presenting a white-box implementation of AES [5]. White-box cryptography aims to

[⋆] The present paper is a merged abstract of two independent but overlapping works: a paper by De Mulder, Roelse and Preneel [11] and a paper by Lepoint and Rivain [7].

protect the confidentiality of the secret key of a cipher in a white-box model, i.e., where an adversary is assumed to have full access to the implementation of the cipher and its execution environment. For example, in a white-box context the adversary can use tools such as decompilers and debuggers to reverse engineer the implementation of the cipher, and to read and alter values of intermediate results of the cipher during its execution. A typical example of an application in which a cipher is implemented in a white-box environment is a content protection system in which a client is executed on the main processor of a PC, a tablet, a mobile device, or a set-top box.

In 2004, Billet *et al.* [3] presented an attack on the white-box AES implementation of Chow *et al.*. The BGE attack assumes that the order of the bytes of the intermediate AES results is randomized in the white-box implementation, and extracts its embedded AES key with a work factor of 2^{30}. In 2012, Tolhuizen [12] proposed an improvement to the most time-consuming phase of the BGE attack, reducing the work factor of this phase to 2^{19}. If the improvement of Tolhuizen is implemented, then the work factor of the BGE attack is dominated by the other phases of the BGE attack, and equals 2^{29}. This paper presents several improvements to the other phases of the BGE attack, and shows that the work factor of the BGE attack is reduced to 2^{22} when Tolhuizen's improvement and the improvements presented in this paper are implemented.

This paper also presents a new attack on the white-box implementation 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 byte encodings and secret key byte(s) involved in some target look-up tables. Applied to the original scheme, we get an attack of complexity 2^{22}.

The BGE attack triggered the design of new white-box AES implementations, such as the ones proposed by Xiao and Lai in 2009 [13] and by Karroumi in 2010 [6]. In [10], De Mulder, Roelse and Preneel presented a cryptanalysis of Xiao and Lai's white-box AES implementation, showing that this implementation is insecure.

In [6], Karroumi uses the concept of dual ciphers [1,2,4] and the white-box techniques of Chow *et al.* to design a new white-box AES implementation. In [6], Karroumi argues that the additional secrecy introduced by the dual cipher increases the work factor of the BGE attack to 2^{93}. This paper shows that the white-box AES implementations of Chow *et al.* and Karroumi are the same. As a direct consequence, Karroumi's white-box AES implementation is vulnerable to the same attacks, including the original BGE attack and the attacks presented in this paper.

Paper organization. Section 2 describes aspects of AES, the white-box AES implementation of Chow *et al.*, and the BGE attack that are relevant to this paper. The improvements of the BGE attack and their work factor are presented in Section 3. The new attack based on collisions is presented in Section 4. The insecurity of Karroumi's scheme is shown in Section 5. Finally, concluding remarks are provided in Section 6

2 Preliminaries

2.1 AES

AES [8] is a key-iterated block cipher operating on 16-byte blocks. This paper assumes throughout and without loss of generality that the AES variant in [8] with a 128-bit key is used. AES consists of 10 rounds and has 11 round keys which are derived from the secret key using a key scheduling process. Each AES round and the operations within a round update a 16-byte state; the initial and final state are the AES plaintext and ciphertext, respectively. AES can be described elegantly by interpreting the bytes of the state as elements of the finite field $\mathbf{F}_{256}$, and by defining AES operations as mappings over this field (see also [8]). As the final round is not relevant for the discussion in this paper, only the first 9 rounds are considered in the following text. Each round r with $1 \leq r \leq 9$ comprises four operations:

`ShiftRows`: a permutation on the indices of the 16 bytes of the state;

`AddRoundKey`: a byte-wise addition of 16 round key bytes $k_i^{(r,j)}$ $(0 \leq i, j \leq 3)$ and the 16-byte state;

`SubBytes`: applies the AES S-box, denoted by S, to every byte of the 16-byte state;

`MixColumns`: a linear operation on $\mathbf{F}_{256}^{16}$. The `MixColumns` operation is represented by a 4×4 matrix `MC` over $\mathbf{F}_{256}$; the linear operation applies 4 instances of this matrix in parallel to the 16-byte state. The 16 coefficients of `MC` are denoted by mc_{ij} for $0 \leq i, j \leq 3$.

In literature, the boundaries between rounds are defined in different ways. In this paper, `ShiftRows` and `MixColumns` are the first and final operations within a round, respectively. That is, the order of the operations within a round is identical to the order used to describe the operations above. For details about AES, refer to [8].

AES subrounds. The mappings in the following definition will be used to describe the white-box AES implementations and the attacks on the implementations. In the following text, the finite field representation as defined in [8] is referred to as the AES polynomial representation, and $\oplus$ and $\otimes$ denote the addition and multiplication operations in this representation, respectively.

Definition 1. *Let $x_i, y_i \in \mathbf{F}_{256}$ for $0 \leq i \leq 3$ be represented using the AES polynomial representation. The mapping $AES^{(r,j)} \colon \mathbf{F}_{256}^4 \to \mathbf{F}_{256}^4$ for $1 \leq r \leq 9$ and $0 \leq j \leq 3$, called an AES subround, is defined by $(y_0, y_1, y_2, y_3) = AES^{(r,j)}(x_0, x_1, x_2, x_3)$ with*

$$\begin{aligned} y_i = mc_{i0} \otimes S\bigl(x_0 \oplus k_0^{(r,j)}\bigr) \oplus mc_{i1} \otimes S\bigl(x_1 \oplus k_1^{(r,j)}\bigr) \oplus \\ mc_{i2} \otimes S\bigl(x_2 \oplus k_2^{(r,j)}\bigr) \oplus mc_{i3} \otimes S\bigl(x_3 \oplus k_3^{(r,j)}\bigr) \ , \end{aligned}$$

for $0 \leq i \leq 3$.

Observe that an AES subround consists of the key additions, the S-box operations and the `MixColumns` operations in an AES round that are associated with a single `MixColumns` matrix operation, and that one AES round comprises four AES subrounds. The subrounds are indexed by j in Def. 1, and this paper assumes throughout that the four subrounds in a round are numbered left to right. The bytes $k_i^{(r,j)}$ for $0 \leq i, j \leq 3$ are the 16 bytes of the AES round key of round r.

2.2 Chow *et al.*'s White-Box AES Implementation and the BGE Attack

This section describes aspects of Chow *et al.*'s white-box AES implementation [5] and the BGE attack [3] that are relevant to this paper. For an in-depth tutorial on how Chow *et al.*'s white-box AES implementation is constructed, refer to [9].

Encoded AES subrounds. In the following text, $P_i^{(r,j)}$ and $Q_i^{(r,j)}$ for $0 \leq i \leq 3$ denote bijective mappings on the vector space $\mathbf{F}_2^8$, referred to as *encodings* in white-box cryptography. The encodings are generated randomly and are kept secret in a white-box implementation (for details about encodings, refer to [5,9]). A vector of four mappings, such as $\left(P_0^{(r,j)}, P_1^{(r,j)}, P_2^{(r,j)}, P_3^{(r,j)}\right)$ or $\left(Q_0^{(r,j)}, Q_1^{(r,j)}, Q_2^{(r,j)}, Q_3^{(r,j)}\right)$, denotes the mapping defined by applying the i-th element of the vector to its i-th input byte for $0 \leq i \leq 3$. For $a \in \mathbf{F}_2^n$ the mapping $\oplus_a \colon \mathbf{F}_2^n \to \mathbf{F}_2^n$ denotes the addition with a. With slight abuse of notation, an input to $AES^{(r,j)}$ is considered to be an element of $\mathbf{F}_{256}^4$ using the AES polynomial representation in the following definition, and an output of $AES^{(r,j)}$ is considered to be an element of $(\mathbf{F}_2^8)^4$.

Definition 2. *The mapping $AES_{enc}^{(r,j)} \colon (\mathbf{F}_2^8)^4 \to (\mathbf{F}_2^8)^4$ for $1 \leq r \leq 9$ and $0 \leq j \leq 3$, called an encoded AES subround, is defined by*

$$AES_{enc}^{(r,j)} = (Q_0^{(r,j)}, Q_1^{(r,j)}, Q_2^{(r,j)}, Q_3^{(r,j)}) \circ \\ AES^{(r,j)} \circ (P_0^{(r,j)}, P_1^{(r,j)}, P_2^{(r,j)}, P_3^{(r,j)}) \ .$$

In Chow *et al.*'s white-box AES implementation, the output encodings $Q_i^{(r-1,j)}$ and input encodings $P_i^{(r,j)}$ for $0 \leq i, j \leq 3$ of successive AES rounds are pairwise annihilating to maintain the functionality of AES. The data-flow of the white-box implementation between successive AES rounds $r-1$ and r determines the 16 pairs of output/input encodings which are pairwise annihilating.

Remark 1. Although not explicitly mentioned by Chow *et al.* [5], one can use a randomization of the order of the subrounds in an AES round and in the order of the bytes within each subround to add confusion to the implementation. This can be implemented without increasing the size and without decreasing the performance of the white-box implementation. We capture such a randomization in the next definition of encoded subround where permutations $\mathit{\Pi}_i^{(r,j)} \colon (\mathbf{F}_2^8)^4 \to (\mathbf{F}_2^8)^4$ $(i = 1, 2)$ for $1 \leq r \leq 9$ and $0 \leq j \leq 3$ are added to randomize the order of the input bytes and output bytes of an AES subround. Moreover, permutations $\pi^{(r)} \colon \{0, 1, 2, 3\} \to \{0, 1, 2, 3\}$ for $1 \leq r \leq 9$ randomize the order of the four AES subrounds within an AES round. These permutations are randomly chosen and kept secret in a white-box implementation.

Definition 3. *The mapping* $\overline{AES}_{enc}^{(r,j)} \colon (\mathbf{F}_2^8)^4 \to (\mathbf{F}_2^8)^4$ *for* $1 \leq r \leq 9$ *and* $0 \leq j \leq 3$*, called an encoded AES subround with byte permutations, is defined by*

$$\overline{AES}_{enc}^{(r,j)} = (Q_0^{(r,j)}, Q_1^{(r,j)}, Q_2^{(r,j)}, Q_3^{(r,j)}) \circ \\ \overline{AES}^{(r,j)} \circ (P_0^{(r,j)}, P_1^{(r,j)}, P_2^{(r,j)}, P_3^{(r,j)}) \ ,$$

where the mapping $\overline{AES}^{(r,j)}$ *is defined by*

$$\mathit{\Pi}_2^{(r,j)} \circ AES^{(r,\pi^{(r)}(j))} \circ \mathit{\Pi}_1^{(r,j)} = \mathtt{MC}^{(r,j)} \circ (S, S, S, S) \circ \oplus_{[\bar{k}_i^{(r,j)}]_{0 \leq i \leq 3}} \ ,$$

$$\begin{array}{lll} \textit{with} & [\bar{k}_i^{(r,j)}]_{0 \leq i \leq 3} & = (\mathit{\Pi}_1^{(r,j)})^{-1}\big([k_i^{(r,\pi^{(r)}(j))}]_{0 \leq i \leq 3}\big) \\ \textit{and} & \mathtt{MC}^{(r,j)} & = \mathit{\Pi}_2^{(r,j)} \circ \mathtt{MC} \circ \mathit{\Pi}_1^{(r,j)} \ . \end{array}$$

In [3], Billet *et al.* described a cryptanalysis of Chow *et al.*'s white-box AES implementation [5] with byte permutations and subround permutations. The starting point of their attack is that for rounds $1 \leq r \leq 9$, it is possible to compose certain white-box look-up tables in such a way that an adversary has access to the encoded AES subrounds of each round.

BGE attack. As indicated above, the adversary has access to the encoded AES subrounds $\overline{AES}_{\text{enc}}^{(r,j)}$ for $1 \leq r \leq 9$ and $0 \leq j \leq 3$. Next, the BGE attack [3] comprises the following three phases: Phases 1 and 2 retrieve the bytes of the AES round key associated with round r for some r with $2 \leq r \leq 9$, and Phase 3 determines the correct order of the round key bytes and extracts the AES key.

Phase 1 retrieves the encodings $Q_i^{(r,j)}$ $(0 \leq i \leq 3)$ up to an affine part for each encoded AES subround j $(0 \leq j \leq 3)$. Because of the pairwise annihilating property of the encodings between successive rounds, the encodings $P_i^{(r,j)}$ $(0 \leq i, j \leq 3)$ can be retrieved up to an affine part by applying the same technique to the encoded AES subrounds of the previous round.

Phase 2 assumes that all encodings of an encoded AES round are affine mappings (as the other parts have been retrieved in Phase 1). Phase 2 first retrieves the affine encodings $Q_i^{(r,j)}$ $(0 \leq i \leq 3)$ for each encoded AES subround j $(0 \leq j \leq 3)$. During this process, the key-dependent affine mappings $\widetilde{P}_i^{(r,j)}(x) = P_i^{(r,j)}(x) \oplus \bar{k}_i^{(r,j)}$ $(0 \leq i, j \leq 3)$ are obtained as well. As in Phase 1, the affine encodings $P_i^{(r,j)}$ $(0 \leq i, j \leq 3)$ are retrieved by applying the same technique to the encoded

AES subrounds of the previous round. This enables the adversary to compute the round key bytes $\bar{k}_i^{(r,j)} = \widetilde{P}_i^{(r,j)}(0) \oplus P_i^{(r,j)}(0)$ for $0 \le i, j \le 3$.

Phase 3 retrieves the round key bytes of round $r+1$ as discussed above in Phases 1 and 2, and uses the fact that the round key bytes of rounds r and $r+1$ are related to each other via both the data-flow of the white-box implementation and the AES key scheduling algorithm to retrieve the AES round key. Finally, assuming that the AES variant with a 128-bit key is used, the adversary can use the property of the AES key scheduling algorithm that the AES key can be computed if one of the round keys is known.

Work factor of the BGE attack. In [3], the authors claim that the work factor associated with the three phases of the BGE attack is around 2^{30}. As a result, the white-box AES implementation of Chow *et al.* is insecure. For detailed information about the BGE attack, refer to [3].

3 Reducing the Work Factor of the BGE Attack

In this section, an encoded AES subround is defined as in Def. 3. In 2012, Tolhuizen [12] presented an improvement of the first phase of the BGE attack. If the improvement of Tolhuizen is implemented, then the work factor of the BGE attack is dominated by the second phase. In this section we present several improvements to the other phases of the BGE attack:

1. A method to reduce the expected work factor of Phase 2 of the BGE attack;
2. An efficient method to retrieve the round key bytes of round $r+1$ after the round key bytes of round r are extracted;
3. An efficient method to determine the correct order of the round key bytes, given the round key bytes of two consecutive rounds.

As the work factors of Phases 1 and 2 of the BGE attack are reduced by Tolhuizen's improvement and the first improvement above, respectively, it is now important to have an efficient method for Phase 3 of the BGE attack as well, as otherwise the work factor of this phase could dominate the overall work factor. The second and third improvements above comprise such a method for Phase 3. It will be shown that Tolhuizen's improvement to Phase 1 of the BGE attack and the above improvements to the other phases reduce the work factor of the BGE attack to 2^{22}. The improved BGE attack comprises the following four (instead of three) phases:

Phases 1 and 2: retrieve the round key bytes $\bar{k}_i^{(r,j)}$ ($0 \le i, j \le 3$) associated with round r ($2 \le r \le 8$).

The first two phases are the ones of the BGE attack [3] using Tolhuizen's improvement, and retrieve the round key bytes $\bar{k}_i^{(r,j)}$ for $0 \le i, j \le 3$ associated with round r for some r with $2 \le r \le 8$.

Work factor of Phase 1. Tolhuizen's improvement [12] reduces the work factor of Phase 1 to around $2 \cdot 4 \cdot 4 \cdot (35 \cdot 2^8) < 2^{19}$. The first three factors (i.e., $2 \cdot 4 \cdot 4$) denote the number of encodings involved in Phase 1, i.e., four encodings for each of the four subrounds for each of the two consecutive rounds. The fourth factor (i.e., $35 \cdot 2^8$) denotes the work factor required to retrieve one encoding up to an affine part using Tolhuizen's method.

Work factor of Phase 2. The expected work factor F of the second phase as described in [3] equals approximately $2 \cdot 4 \cdot 4 \cdot 2^{15} \cdot 2^8 = 2^{28}$, and is measured in the number of evaluations of mappings on $\mathbf{F}_2^8$. The evaluations are required to determine if a mapping on $\mathbf{F}_2^8$ is affine. The mappings f that need to be tested for being affine are listed in [3, Proposition 3]. Each f is associated with a secret encoding $P_i^{(r,j)}$ $(0 \leq i, j \leq 3)$ of a round r. As Phase 2 needs to be applied to two consecutive rounds, this involves a total of $2 \cdot 4 \cdot 4$ mappings (which corresponds to the first three factors in F). The mappings f are permutations on $\mathbf{F}_2^8$ and have the structure

$$f = S^{-1} \circ Q_{(c,d)}^{-1} \circ Q \circ S \circ \oplus_k \circ P \ , \tag{1}$$

where S denotes the AES S-box mapping (viewed as a permutation on $\mathbf{F}_2^8$), k denotes a key byte, P and Q denote bijective affine mappings on $\mathbf{F}_2^8$, and $Q_{(c,d)}^{-1}$ denotes a bijective affine mapping on $\mathbf{F}_2^8$ for each pair $(c, d) \in \mathbf{F}_{256}^2$. Furthermore, $Q_{(c,d)}^{-1} = Q^{-1}$ for one specific pair $(c, d) \in \mathbf{F}_{256}^2$. An affine-test is performed for each possible pair $(c, d) \in \mathbf{F}_{256}^2$ until the corresponding mapping f is affine. The expected number of pairs for which the test is performed equals approximately 2^{15}, which is the fourth factor in F. The fifth factor in F, i.e., 2^8, is associated with the test used in [3].

Instead of the test used in [3], which requires 2^n evaluations to determine if $f : \mathbf{F}_2^n \to \mathbf{F}_2^n$ is affine, we use the following algorithm to reduce the expected number of evaluations. If e_i $(1 \leq i \leq n)$ denotes the i-th unit vector in $\mathbf{F}_2^n$, then the algorithm first verifies if the equation

$$f(e_1 \oplus e_2) = f(0) \oplus f(e_1) \oplus f(e_2) \tag{2}$$

holds true. If this equation does not hold true, then the algorithm terminates with "f is not affine". Observe that the algorithm requires 4 evaluations of f in this case. If Eq. 2 holds true, then the algorithm applies the method used in [3] to determine if f is affine (with the only difference that f is not re-evaluated for the four input values $0, e_1, e_2$ and $e_1 \oplus e_2$). In this case 2^n evaluations of f are required.

To show the correctness of this algorithm, it is sufficient to show that an affine mapping always satisfies Eq. 2. If f is affine, then $f(x) = A(x) \oplus b$ for some $A \in \mathbf{F}_2^{n \times n}$ and some $b \in \mathbf{F}_2^n$. It follows that $f(0) \oplus f(e_1) \oplus f(e_2) = b \oplus A(e_1) \oplus b \oplus A(e_2) \oplus b = A(e_1 \oplus e_2) \oplus b = f(e_1 \oplus e_2)$.

Lemma 1. *If f is a random permutation on $\mathbf{F}_2^n$ and if $E(n)$ denotes the expected number of evaluations of f required by the algorithm described above, then $E(n) < 5$.*

Proof. Let $p(n)$ denote the probability that Eq. 2 holds true for a random permutation. To determine $p(n)$, note that $f(0), f(e_1), f(e_2)$ and $f(e_1 \oplus e_2)$ are four distinct elements of $\mathbf{F}_2^n$ if f is a permutation. From this it follows that $f(0) \oplus f(e_1) \oplus f(e_2)$ and $f(e_1 \oplus e_2)$ are both elements of $\mathbf{F}_2^n \setminus \{f(0), f(e_1), f(e_2)\}$. Further, as f is a random permutation, $f(e_1 \oplus e_2)$ is a random element of this set. Hence, $p(n) = 1/(2^n - 3)$ and $E(n) = 4(1-p) + 2^n p = 4 + (2^n - 4)/(2^n - 3) < 5$. $\square$

Under the assumption that f in Eq. 1 behaves as a random permutation on $\mathbf{F}_2^8$ for every incorrect guess for (c, d), the expected work factor of the affine-test is reduced from 2^8 to approximately 5 evaluations if f is not affine and the work factor is 2^8 if f is affine. This implies that the fifth factor in F is reduced to approximately 5. That is, the expected work factor of Phase 2 of the BGE attack is now approximately $2 \cdot 4 \cdot 4 \cdot 2^{15} \cdot 5 \approx 2^{22}$.

Phase 3: retrieve the round key bytes $\bar{k}_i^{(r+1,j)}$ $(0 \le i, j \le 3)$ associated with round $r + 1$.

As mentioned in the description of the BGE attack in Section 2.2, [3] obtains the round key bytes of round $r+1$ by applying Phases 1 and 2 to round $r+1$ as well. Here, we present a more efficient method based on the affine-test described above. The method comprises the following three steps for each encoded AES subround j $(0 \le j \le 3)$ associated with round $r + 1$ to retrieve the round key bytes $\bar{k}_i^{(r+1,j)}$ $(0 \le i, j \le 3)$:

Step 1 applies Phase 1 (using Tolhuizen's improvement) to round $r + 1$ in order to retrieve the encodings $Q_i^{(r+1,j)}$ $(0 \le i \le 3)$ up to an affine part.

Step 2 first removes the non-affine part of the output encodings as recovered in Step 1 from the encoded AES subround. Next, Step 2 removes the input encodings $P_i^{(r+1,j)}$ $(0 \le i \le 3)$ from the encoded AES subround (observe that the inverses of these input encodings were obtained in Phases 1 and 2). The resulting mapping $f^{(r+1,j)} : (\mathbf{F}_2^8)^4 \to (\mathbf{F}_2^8)^4$ is given by

$$f^{(r+1,j)} = \left(\hat{Q}_0^{(r+1,j)}, \hat{Q}_1^{(r+1,j)}, \hat{Q}_2^{(r+1,j)}, \hat{Q}_3^{(r+1,j)}\right) \circ \overline{AES}^{(r+1,j)} ,$$

where $\hat{Q}_i^{(r+1,j)}$ $(0 \le i \le 3)$ are affine output encodings.

Step 3 retrieves the round key bytes $\bar{k}_i^{(r+1,j)}$ $(0 \le i \le 3)$. To find a key byte, say $\bar{k}_0^{(r+1,j)}$, fix the other three input bytes to $f^{(r+1,j)}$ (e.g., to zero), search over all possible 2^8 values of the key byte k and verify if

$$g_k(x) = f^{(r+1,j)}\left(k \oplus S^{-1}(x), 0, 0, 0\right)$$

is affine using the test described above. In case $g_k(x)$ is affine, then $\bar{k}_0^{(r+1,j)} = k$. Repeat this for $\bar{k}_i^{(r+1,j)}$ $(i = 1, 2, 3)$.

The correctness of Step 3 uses the fact that the mapping $S\left(c \oplus S^{-1}(x)\right)$ is non-affine for all non-zero values of c. This has already been proven in [3, proof of Proposition 3].

Work factor of Phase 3. The work factor of Step 3 equals $4 \cdot 4 \cdot 2^7 \cdot 5 \approx 2^{13}$, where $4 \cdot 4$ denotes the number of round key bytes, 2^7 denotes the expected number of key values for which the affine-test is performed and 5 denotes the expected number of evaluations of the affine-test if g_k is not affine. The work factor of Step 1 is $4 \cdot 4 \cdot (35 \cdot 2^8) < 2^{18}$, where the first two factors denote the number of output encodings involved in Step 1. As a result, the work factor of Phase 3 is dominated by Step 1 and is less than 2^{18}.

Phase 4: determine the correct order of the round key bytes and extract the secret AES key.

After Phases 1 - 3, the values of the round key bytes of two consecutive rounds r and $r+1$ are known. However, for each round, the order of the round key bytes of each subround and the order of the four subrounds are still unknown. Notice that there are still $(4!)^5 \approx 2^{23}$ possibilities for the round key if only the bytes of that round key are considered. In [3], it is indicated how the correct order can be determined given the "shuffled" round key bytes of rounds r and $r+1$. However, [3] does not contain an explicit description of such a method. As the work factor of the first three phases equals 2^{22}, it is desirable to have a method to determine the correct order of the round key bytes with a work factor that is less than 2^{22}. Below we present such a method, comprising the following three steps:

Step 1 retrieves $\mathtt{MC}^{(r,j)}$ associated with each subround j $(0 \le j \le 3)$ of round r. Recall that the encodings $P_i^{(r,j)}$ and $Q_i^{(r,j)}$ $(0 \le i,j \le 3)$ were obtained in Phases 1 and 2. Together with the knowledge of the round key bytes $\bar{k}_i^{(r,j)}$ $(0 \le i,j \le 3)$, compute

$$\begin{aligned}\mathtt{MC}^{(r,j)} = \left(Q_0^{(r,j)}, Q_1^{(r,j)}, Q_2^{(r,j)}, Q_3^{(r,j)}\right)^{-1} \circ \overline{AES}_{enc}^{(r,j)} \circ \\ \left(P_0^{(r,j)}, P_1^{(r,j)}, P_2^{(r,j)}, P_3^{(r,j)}\right)^{-1} \circ \oplus_{[\bar{k}_i^{(r,j)}]_{0 \le i \le 3}} \circ (S,S,S,S)^{-1} \ ,\end{aligned}$$

for $j = 0,1,2,3$.

Step 2 computes for each $\mathtt{MC}^{(r,j)}$ $(0 \le j \le 3)$ the permutations $\Pi_1, \Pi_2 : (\mathbf{F}_2^8)^4 \to (\mathbf{F}_2^8)^4$ such that

$$\mathtt{MC}^{(r,j)} = \Pi_2 \circ \mathtt{MC} \circ \Pi_1 \ . \tag{3}$$

Let $(\Pi^{(1)}, \Pi^{(2)})$ denote the pairs of permutations for which $\mathtt{MC}$ remains invariant, i.e., $\mathtt{MC} = \Pi^{(2)} \circ \mathtt{MC} \circ \Pi^{(1)}$. It is easily verified that there are exactly four such pairs. The four permutations $\Pi^{(1)}$ are the four different circular shifts on the indices of a 4-byte vector, and $\Pi^{(2)} = (\Pi^{(1)})^{-1}$ for each of these pairs. This implies that there are also exactly four different pairs of permutations satisfying Eq. 3, given by

$$\left(\Pi^{(1)} \circ \Pi_1 \ , \ \Pi_2 \circ \Pi^{(2)}\right) \ . \tag{4}$$

As a consequence, finding one pair of permutation matrices satisfying Eq. 3 suffices to find the remaining three as well. Notice that exactly one of these four pairs of permutations equals the pair $(\Pi_1^{(r,j)}, \Pi_2^{(r,j)})$ of the encoded subround (see also Def. 3); in other words, one of these pairs is the correct pair.

After this, the order of the round key bytes associated with each subround is known up to an uncertainty of four possibilities (circular shifts). Observe that the order of the four subrounds is still unknown.

Step 3 determines the correct order of the round key bytes. For each of the possible orderings of the four AES subrounds of round r and the round key bytes within these subrounds (as determined in Step 2), obtain a candidate for the $(r+1)^{th}$ round key using the following two methods: (i) the AES key scheduling algorithm and (ii) the data-flow of the white-box AES implementation between the encoded subrounds of rounds r and $r+1$. Notice that once an order of the round key bytes of round r is selected, the order of the round key bytes of round $r+1$ can be determined using the corresponding pair of permutations of each of the subrounds of round r (see also Eq. 4) and the data-flow of the white-box implementation. With overwhelming probability, only one ordering of round key bytes of round r results in the same $(r+1)^{th}$ round key; this ordering corresponds to the correct round key of round r. Finally, use the property of the AES key scheduling algorithm that the AES key can be computed if one of the round keys is known.

Work factor of Phase 4. A naive approach yields an expected work factor of $(4!)^2 \approx 2^9$ for Step 2 by searching over all possible pairs of permutations. Step 2 reduces the number of possible orderings of the round key bytes from 2^{23} to $4^4 \cdot 4! < 2^{13}$ (where the first and second factor denote the possible orderings of round key bytes within each subround and of the four subrounds, respectively), which equals the work factor of Step 3. As a result, the overall work factor of Phase 4 is dominated by the work factor of Step 3 and hence is less than 2^{13}.

Conclusion

The work factor of the improved BGE attack is dominated by the work factor of the second phase and equals 2^{22}.

Note that the uncertainty in the order of the round key bytes results in the need to retrieve key bytes of two consecutive rounds. This affects the work factor of the original BGE attack. In the improved BGE attack this is no longer the case, as the work factors of the phases that determine the correct order (i.e. Phases 3 and 4) are negligible compared to the work factor of Phase 2. A consequence of Tolhuizen's improvement is that the use of non-affine white-box encodings has a negligible impact on the overall work factor of the improved BGE attack.

4 A New Attack Exploiting Internal Collisions

In this section we propose a new attack on the initial Chow *et al.* implementation exploiting collisions in output of the first AES round. Note that unlike the

BGE attack, the description below only considers the basic implementation, i.e., without byte permutations. In this section, an encoded AES subround is defined as in Def. 2.

According to Section 2, applying a set of successive look-up tables, one can compute the first encoded AES subround $AES_{enc}^{(1,0)}$, which is denoted by f' in the following for the sake of clarity (and in accordance to notations in [7]):

$$f' = (Q_0^{(1,0)}, Q_1^{(1,0)}, Q_2^{(1,0)}, Q_3^{(1,0)}) \circ AES^{(1,0)} \circ (P_0^{(1,0)}, P_1^{(1,0)}, P_2^{(1,0)}, P_3^{(1,0)}) \ . \quad (5)$$

Let us denote by f'_ℓ the coordinate functions of f' such that $f' = (f'_0, f'_1, f'_2, f'_3)$. Let us further denote by S_i the function defined as

$$S_i(\cdot) = S(k_i^{(1,0)} \oplus (P_i^{(1,0)})(\cdot)) \ , \quad (6)$$

for $0 \le i \le 3$.

4.1 Recovering the S_i Functions

Our attack consists in finding collisions in output of the coordinate functions f'_ℓ in order to recover functions S_0, S_1, S_2 and S_3 and associated key bytes. For the sake of clarity, we drop all the surperscripts $(1,0)$ in the following. We start with the recovery of S_0 and S_1 by looking for collision of the form

$$f'_0(\alpha, 0, 0, 0) = f'_0(0, \beta, 0, 0) \ . \quad (7)$$

By definition of the `MixColumns` transformation, the above equation can be rewritten as

$$Q_0\big(\mathsf{02} \otimes S_0(\alpha) \oplus \mathsf{03} \otimes S_1(0) \oplus c\big) = Q_0\big(\mathsf{02} \otimes S_0(0) \oplus \mathsf{03} \otimes S_1(\beta) \oplus c\big)$$

where $c = S_2(0) \oplus S_3(0)$, implying

$$\mathsf{02} \otimes S_0(\alpha) \oplus \mathsf{03} \otimes S_1(0) = \mathsf{02} \otimes S_0(0) \oplus \mathsf{03} \otimes S_1(\beta) \ . \quad (8)$$

Collecting several such equations, we can construct a linear system to recover S_0 and S_1. Let u_0, u_1, $\ldots$, u_{255} and v_0, v_1, $\ldots$, v_{255} denote the unknowns associated to the outputs of S_0 and S_1 (*i.e.* $u_i = S_0(i)$ and $v_i = S_1(i)$). Then (8) can be rewritten as

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

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 $\mathbf{F}_{256}$. 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'_ℓ with $\ell \in \{1,2,3\}$, 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

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

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

$$\mathsf{02} \otimes u'_\alpha \oplus \mathsf{03} \otimes v'_\beta = 0 \ .$$

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 $v'_i = S_1(0) \oplus S_1(i)$), which is such that $u'_i \neq 0$ and $v'_i \neq 0$ by bijectivity of S_0 and S_1. 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 $u'_i = a_i \otimes u'_1$ and $v'_i = b_i \otimes u'_1$. These coefficients can be easily recovered by solving the system for $u'_1 = 1$. We then get

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

and

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

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

$$S^{-1} \circ S_0(\cdot) = P_0(\cdot) \oplus k_0 \ .$$

By definition of P_0, the above function has algebraic degree at most 4. We then use the following lemma.

Lemma 2. *Let g be a function from $\{0,1\}^8$ to itself with algebraic degree at most 4. The map*

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

is the null function $x \mapsto 0$.

Proof. The map φ 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. □

Remark 2. 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 $\hat{S}_0(\cdot) = a \otimes S_0(\cdot) \oplus b$, with $a \neq 0$ and $(a,b) \neq (0,1)$. The function $S^{-1} \circ \hat{S}_0$ then satisfies

$$S^{-1} \circ \hat{S}_0(\cdot) = S^{-1}\big(a \otimes S(k_0 \oplus P_0(\cdot)) \oplus b\big) ,$$

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

According to Lemma 2 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

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

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 $\hat{\varphi}(x) = \hat{\varphi}(x \oplus 01) = \cdots = \hat{\varphi}(x \oplus 15)$. 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_1 from (11) by exhaustive search on v_0. Here again, the good solution is determined using Lemma 2 and the above approach. The remaining functions S_2 and S_3 are recovered similarly by solving the linear systems arising from collisions of the form $f'_\ell(\alpha, 0, 0, 0) = f'_\ell(0, 0, \beta, 0)$ and $f'_\ell(\alpha, 0, 0, 0) = f'_\ell(0, 0, 0, \beta)$. Since S_0 is already known, we get the same situation as for the recovery of S_1. Namely, all the elements of S_2 (resp. S_3) can be expressed as affine functions of $S_2(0)$ (resp. $S_3(0)$), and we can recover the overall function by exhaustive search on this value and with the selection criterion of Lemma 2.

4.2 Recovering the Secret Key

Once the S_i functions have been recovered, one can easily recover the byte-encodings Q_i in output of the first round. For instance evaluating $f'_0(\alpha, 0, 0, 0)$ one gets the value $Q_0\big(\psi(\alpha)\big)$ where

$$\psi \colon \alpha \mapsto 02 \otimes S_0(\alpha) \oplus 03 \otimes S_1(0) \oplus S_2(0) \oplus S_3(0)$$

is a bijective function. We hence get $Q_0(\cdot) = f'_0(\psi^{-1}(\cdot), 0, 0, 0)$ which enables to fully retrieve Q_0 by looping on the 256 input values. Each byte-encoding $Q_i^{(1,j)}$ in output of the first round 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

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

key bytes in the second round from that knowledge. In what follows, we shall slightly change the definition of f' and the S_i's given in (5) and (6). Namely, f' shall denote the first encoded subround of the second round (rather that of the first round), and S_i the associated functions, that is $f' = AES_{enc}^{(2,0)}$ and $S_i(\cdot) = S(k_i^{(2,0)} \oplus (P_i^{(2,0)})(\cdot))$ for $0 \leq i \leq 3$. As in the previous section, we shall further drop all the superscripts $(2,0)$ for the sake of clarity.

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

$$g = f_0'(P_0^{-1}(S^{-1}(\cdot) \oplus k_0), 0, 0, 0) \ .$$

This function satisfies

$$g(x) = Q_0(\mathsf{02} \otimes x \oplus c) \qquad \text{where } c = \mathsf{03} \otimes S_1(0) \oplus S_2(0) \oplus S_3(0) \ ,$$

and it has algebraic degree at most 4 by definition of Q_0 (since multiplying and adding constant coefficients are linear). Therefore, according to Lemma 2, the 4th-order derivative $\varphi\colon x \mapsto \bigoplus_{\alpha=\mathsf{0}}^{\mathsf{15}} g(x \oplus \alpha)$ 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

$$\hat{g}(x) = f_0'(P_0^{-1}(S^{-1}(x) \oplus \hat{k}_0), 0, 0, 0) = Q_0(\mathsf{02} \otimes S(S^{-1}(x) \oplus \hat{k}_0 \oplus k_0) \oplus c) \ .$$

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

$$\hat{\varphi}(x) = \bigoplus_{\alpha=\mathsf{0}}^{\mathsf{15}} f_0'(P_0^{-1}(S^{-1}(x) \oplus \hat{k}_0), 0, 0, 0)$$

equals the null function $x \mapsto 0$, or not. As for the previous recovery of the S_i 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_i, 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_1, k_2 and k_3 can be retrieved similarly; only the definition of the function g shall change. For instance, g is defined as $f_0'(0, P_1^{-1}(S^{-1}(\cdot) \oplus k_1), 0, 0)$ for k_1, and so on for k_2 and k_3. And the other key bytes $k_i^{(2,j)}$ for $j \geq 1$ can be recovered in the exact same way. Eventually, from the second round key, one can easily recover the full AES secret key by inverting the key schedule process.

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

4.3 Attack Complexity

The bottleneck of our attack is the exhaustive search to recover the functions S_i in the first round. 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 (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_1 (resp. S_2, S_3) 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_1, S_2 and S_3. This computation must be performed for each subround of the first AES round, which makes a total complexity of $4 \times 2^{20} = 2^{22}$.

The recovery of the key bytes has a negligible complexity compared to the recovery of the S_i functions in the first round. 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 $16 \cdot 2^{12} \ll 2^{22}$.

5 Karroumi's White-Box AES Implementation

Karroumi's method to generate a white-box AES implementation [6] can be divided into two phases; *Phase 1* generates a dual AES cipher from a key-instantiated AES cipher, and *Phase 2* applies the white-box techniques presented by Chow *et al.* to the dual AES cipher. Below, aspects of these phases that are relevant to this paper are described.

Phase 1: Dual AES cipher

In this section we give a description of the set of dual AES ciphers used by Karroumi in [6]. First, we define a dual AES subround. The following notation is used: $m_\alpha : \mathbf{F}_{256} \to \mathbf{F}_{256}$ with $\alpha \in \mathbf{F}_{256}^*$ is defined by $m_\alpha(x) = \alpha \otimes x$, and $f_t : \mathbf{F}_{256} \to \mathbf{F}_{256}$ defined by $f_t(x) = x^{2^t}$ for $0 \leq t \leq 7$ are the automorphisms of $\mathbf{F}_{256}$ over $\mathbf{F}_2$. Further, $R_l : \mathbf{F}_{256} \to \mathbf{F}_{256}$ are the isomorphisms mapping elements in the AES polynomial representation to field elements in one of the polynomial representations of $\mathbf{F}_{256}$. There are 30 irreducible polynomials of degree 8 over $\mathbf{F}_2$, each one resulting in a unique polynomial representation of $\mathbf{F}_{256}$ (one of these representations being the AES polynomial representation), hence in total there are 30 distinct isomorphisms R_l ($1 \leq l \leq 30$). The addition and multiplication operations in the polynomial representation associated with R_l are denoted by $\oplus_l$ and $\otimes_l$, respectively ($\oplus_l$ and $\otimes_l$ being equal to $\oplus$ and $\otimes$ for exactly one value of l with $1 \leq l \leq 30$). Finally, the definition of a dual AES subround uses a set of mappings, denoted by $\mathcal{T}$, and defined by

$$\mathcal{T} = \{R_l \circ m_\alpha \circ f_t \mid 1 \leq l \leq 30, \alpha \in \mathbf{F}_{256}^* \text{ and } 0 \leq t \leq 7\} \ .$$

Observe that an element of $\mathcal{T}$ maps elements in the AES polynomial representation to elements in one of the 30 polynomial representations of $\mathbf{F}_{256}$.

Definition 4. *Let $\Delta_{r,j} \in \mathcal{T}$ with $\Delta_{r,j} = R_l \circ m_\alpha \circ f_t$ for some triple (l, α, t) with $1 \leq l \leq 30, \alpha \in \mathbf{F}_{256}^*$ and $0 \leq t \leq 7$, and let $\delta_{r,j} = R_l \circ f_t$. Further, let $v_i, w_i \in \mathbf{F}_{256}$ for $0 \leq i \leq 3$ be represented using the polynomial representation associated with R_l. The mapping $AES^{(r,j,\Delta_{r,j})} : \mathbf{F}_{256}^4 \to \mathbf{F}_{256}^4$ for $1 \leq r \leq 9$ and $0 \leq j \leq 3$, called a dual AES subround, is defined by $(w_0, w_1, w_2, w_3) = AES^{(r,j,\Delta_{r,j})}(v_0, v_1, v_2, v_3)$ with*

$$\begin{aligned}
w_i = {} & \delta_{r,j}(mc_{i0}) \otimes_l \Delta_{r,j} \circ S \circ \Delta_{r,j}^{-1}\big(v_0 \oplus_l \Delta_{r,j}(k_0^{(r,j)})\big) \\
& \oplus_l \delta_{r,j}(mc_{i1}) \otimes_l \Delta_{r,j} \circ S \circ \Delta_{r,j}^{-1}\big(v_1 \oplus_l \Delta_{r,j}(k_1^{(r,j)})\big) \\
& \oplus_l \delta_{r,j}(mc_{i2}) \otimes_l \Delta_{r,j} \circ S \circ \Delta_{r,j}^{-1}\big(v_2 \oplus_l \Delta_{r,j}(k_2^{(r,j)})\big) \\
& \oplus_l \delta_{r,j}(mc_{i3}) \otimes_l \Delta_{r,j} \circ S \circ \Delta_{r,j}^{-1}\big(v_3 \oplus_l \Delta_{r,j}(k_3^{(r,j)})\big) \ ,
\end{aligned}$$

for $0 \leq i \leq 3$.

The following lemma presents a property that is required to show that a dual AES cipher maintains the functionality of AES. As the lemma is also used in the cryptanalysis in this paper, and as a formal proof of this property is omitted in [4] and [6], we include a proof as well.

Lemma 3. *If $\Delta_{r,j} \in \mathcal{T}$, then*

$$AES^{(r,j,\Delta_{r,j})} \circ (\Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}) = (\Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}) \circ AES^{(r,j)} \ ,$$

for $1 \leq r \leq 9$ and $0 \leq j \leq 3$.

Proof. Let x_i for $0 \leq i \leq 3$ be elements of $\mathbf{F}_{256}$ using the AES polynomial representation, let w_i for $0 \leq i \leq 3$ be elements of $\mathbf{F}_{256}$ using the polynomial representation associated with R_l (assuming that $\Delta_{r,j} = R_l \circ m_\alpha \circ f_t$), and let

$$(w_0, w_1, w_2, w_3) = \mathrm{AES}^{(r,j,\Delta_{r,j})} \circ (\Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j})(x_0, x_1, x_2, x_3) \ .$$

Substituting $v_i = \Delta_{r,j}(x_i)$ for $0 \leq i \leq 3$ in the equation in Def. 4 yields

$$w_i = \bigoplus_{z=0}^{3}{}_l \, \delta_{r,j}(mc_{iz}) \otimes_l \Delta_{r,j} \circ S \circ \Delta_{r,j}^{-1}\big(\Delta_{r,j}(x_z) \oplus_l \Delta_{r,j}(k_z^{(r,j)})\big) \ ,$$

for $0 \leq i \leq 3$. Next, observe that $\Delta_{r,j}(a) \oplus_l \Delta_{r,j}(b) = R_l \circ m_\alpha \circ f_t(a) \oplus_l R_l \circ m_\alpha \circ f_t(b) = R_l(m_\alpha \circ f_t(a) \oplus m_\alpha \circ f_t(b)) = R_l(m_\alpha(f_t(a) \oplus f_t(b)) = R_l(m_\alpha(f_t(a \oplus b))) =$

$\varDelta_{r,j}(a \oplus b)$ for all $a, b \in \mathbf{F}_{256}$ and all $\alpha \in \mathbf{F}^*_{256}$; the second equality holds true since R_l is an isomorphism, the third equality holds true as $\alpha(a \oplus b) = \alpha(a) \oplus \alpha(b)$ for all $a, b \in \mathbf{F}_{256}$ and the fourth equality holds true since f_t is an automorphism. It follows that

$$w_i = \bigoplus_{z=0}^{3}{}_l\, \delta_{r,j}(mc_{iz}) \otimes_l \varDelta_{r,j} \circ S\big(x_z \oplus k_z^{(r,j)}\big) \ ,$$

for $0 \le i \le 3$. Next, note that $\delta_{r,j}(a) \otimes_l \varDelta_{r,j}(b) = R_l \circ f_t(a) \otimes_l R_l \circ m_\alpha \circ f_t(b) = R_l(f_t(a) \otimes m_\alpha \circ f_t(b)) = R_l(m_\alpha(f_t(a \otimes b))) = \varDelta_{r,j}(a \otimes b)$ for all $a, b \in \mathbf{F}_{256}$; the second equality holds true since R_l is an isomorphism and the third equality uses the fact that $a^{2^t} \otimes \alpha b^{2^t} = \alpha(ab)^{2^t}$ for all $a, b \in \mathbf{F}_{256}$ and all $\alpha \in \mathbf{F}^*_{256}$. It follows that

$$w_i = \bigoplus_{z=0}^{3}{}_l\, \varDelta_{r,j}\Big(mc_{iz} \otimes S\big(x_z \oplus k_z^{(r,j)}\big)\Big) \ ,$$

for $0 \le i \le 3$. From this, $\varDelta_{r,j}(a) \oplus_l \varDelta_{r,j}(b) = \varDelta_{r,j}(a \oplus b)$ for all $a, b \in \mathbf{F}_{256}$, and the definition of y_i in Def. 1, it follows that $w_i = \varDelta_{r,j}(y_i)$ for $0 \le i \le 3$. $\square$

Now, Karroumi [6] obtains a dual AES cipher as follows:

Step 1 assigns a randomly chosen $\varDelta_{r,j} \in \mathcal{T}$ to each AES subround $AES^{(r,j)}$ $(1 \le r \le 9$ and $0 \le j \le 3)$. Based on $\varDelta_{r,j}$, the corresponding dual AES subround $AES^{(r,j,\varDelta_{r,j})}$ is implemented as specified by Def. 4. The mappings $\varDelta_{r,j}$ and $\delta_{r,j}$ (and the implementation of the dual cipher) are kept secret.

Step 2 ensures that the functionality of AES is maintained by including an additional operation (referred to as `ChangeDualState`) between `ShiftRows` and `AddRoundKey` operations of round r for $1 \le r \le 9$. If the inverse `ShiftRows` operation is defined by the mapping $sr(i,j) = (j+i) \bmod 4$ for $0 \le i,j \le 3$, then the `ChangeDualState` operation of round r applies the mapping $C_i^{(r,j)} : \mathbf{F}_{256} \to \mathbf{F}_{256}$ to the byte of the state associated with the i-th input byte of $\mathrm{AES}^{(r,j,\varDelta_{r,j})}$ for $0 \le i,j \le 3$, defined by $C_i^{(1,j)} = \varDelta_{1,j}$ and $C_i^{(r,j)} = \varDelta_{r,j} \circ \varDelta^{-1}_{r-1,\mathrm{sr}(i,j)}$ if $2 \le r \le 9$. Observe that for $2 \le r \le 9$, $C_i^{(r,j)}$ maps elements from $\mathbf{F}_{256}$ using the polynomial representation associated with $\varDelta_{r-1,\mathrm{sr}(i,j)}$ to elements of $\mathbf{F}_{256}$ using the polynomial representation associated with $\varDelta_{r,j}$.

Karroumi presents two different but equivalent methods (from a security point of view) in [6] to perform the `ChangeDualState` operation, and specifies the white-box AES implementation using one of these methods. In this paper we use the specification as in [6]; the cryptanalysis can easily be adapted if the other method is used.

Phase 2: Apply the techniques of Chow *et al.*

The following description of Karroumi's white-box AES implementation is equivalent to the description in [6]:

Step 1 applies the techniques of Chow *et al.* to write the dual AES cipher (with a fixed key) obtained in Phase 1 as a series of lookup tables. In particular, the dual AES key addition operations and the dual S-box operations are merged into key-dependent bijective mappings $T_i^{(r,j,\Delta_{r,j})}$ for $0 \leq i,j \leq 3$ and $1 \leq r \leq 9$. These mappings are referred to as dual T-boxes and are defined by

$$T_i^{(r,j,\Delta_{r,j})} = \Delta_{r,j} \circ S \circ \Delta_{r,j}^{-1} \circ \oplus_{\Delta_{r,j}(k_i^{(r,j)})} \circ C_i^{(r,j)} \quad ,$$

where each dual T-box mapping is implemented as a table mapping 8 input bits to 8 output bits. Recall that the mappings $C_i^{(r,j)}$ define the `ChangeDualState` operation. Next, write the other part of the dual AES cipher as a series of lookup tables as indicated by Chow *et al.* in [5]. The number and types of tables (including the tables representing the dual T-boxes) and the data-flow between tables are the same as in the lookup table implementation of AES in [5]. The only difference is that the values of the table entries of the dual AES implementation are likely to be different from the values of the corresponding entries in the AES implementation in [5] due to the dual version of the AES operations.

Step 2 applies the white-box encoding techniques of Chow *et al.* in [5] to this lookup table implementation of dual AES. As these white-box encoding techniques do not depend on the values of the table entries, the number and types of white-box tables, and the data-flow of Karroumi's white-box AES implementation are the same as in the white-box AES implementation of Chow *et al.* in [5].

In [6], Karroumi argues that the secrecy of the mappings $\Delta_{r,j}$, randomly selected from the set $\mathcal{T}$ and used to generate the dual cipher, increases the work factor of the BGE attack to 2^{93}.

5.1 Insecurity

This section shows that Karroumi's white-box AES implementation [6] is insecure. Recall that Karroumi's white-box AES implementation uses the same number and types of white-box tables, and that the data-flow of the implementation is the same as in Chow *et al.*'s white-box AES implementation in [5]. As a result, the techniques of Billet *et al.* can be applied directly to compose lookup tables in Karroumi's implementation to obtain access to the encoded dual AES subrounds (instead of the encoded AES subrounds in case of Chow *et al.*'s implementation) for rounds $1 \leq r \leq 9$. In the following definition, $A_i^{(r,j)}$ and $B_i^{(r,j)}$ for $0 \leq i \leq 3$ denote bijective mappings (or encodings) on the vector space $\mathbf{F}_2^8$. Further, with slight abuse of notation, an output of $A_i^{(r,j)}$ is considered to be an element of $\mathbf{F}_{256}$ using the polynomial representation associated with the mapping R_l as defined by $\Delta_{r-1,\mathrm{sr}(i',j')}$, and an output of $AES^{(r,j,\Delta_{r,j})}$ is considered to be an element of $(\mathbf{F}_2^8)^4$. In the following definition, $\Pi_1^{(r,j)}, \Pi_2^{(r,j)}$ and $\pi^{(r)}$ are the permutations as used in Def. 3.

Definition 5. *The mapping* $\overline{AES}_{enc}^{(r,j,\Delta_{r,j})} : (\mathbf{F}_2^8)^4 \to (\mathbf{F}_2^8)^4$ *for* $1 \le r \le 9$ *and* $0 \le j \le 3$*, called an encoded dual AES subround, is defined by*

$$\overline{AES}_{enc}^{(r,j,\Delta_{r,j})} = (B_0^{(r,j)}, B_1^{(r,j)}, B_2^{(r,j)}, B_3^{(r,j)}) \circ \overline{AES}^{(r,j,\Delta_{r,j})} \circ (A_0^{(r,j)}, A_1^{(r,j)}, A_2^{(r,j)}, A_3^{(r,j)}) \ , \qquad (12)$$

where the mapping $\overline{AES}^{(r,j,\Delta_{r,j})}$ *is defined by*

$$\Pi_2^{(r,j)} \circ AES^{(r,j',\Delta_{r,j'})} \circ (C_0^{(r,j')}, C_1^{(r,j')}, C_2^{(r,j')}, C_3^{(r,j')}) \circ \Pi_1^{(r,j)} \ , \qquad (13)$$

with $j' = \pi^{(r)}(j)$*.*

The next lemma shows that an encoded dual AES subround can be represented by an encoded AES subround using the same key bytes:

Lemma 4. *An encoded dual AES subround* $\overline{AES}_{enc}^{(r,j,\Delta_{r,j})}$ *is an encoded AES subround* $\overline{AES}_{enc}^{(r,j)}$ *as in Def. 3 with*

$$P_i^{(1,j)} = A_i^{(1,j)} \quad \text{and} \quad P_i^{(r,j)} = \Delta^{-1}_{r-1,\mathrm{sr}(i',j')} \circ A_i^{(r,j)} \quad \text{if} \quad 2 \le r \le 9 \ ,$$

and

$$Q_i^{(r,j)} = B_i^{(r,j)} \circ \Delta_{r,j'} \ ,$$

for $0 \le i, j \le 3$ *and* $1 \le r \le 9$*, with* $i' = (\pi_1^{(r,j)})^{-1}(i)$ *and* $j' = \pi^{(r)}(j)$ *where* $(\pi_1^{(r,j)})^{-1}$ *denotes the permutation on the indices of a 4-byte vector as a result of the application of* $\left(\Pi_1^{(r,j)}\right)^{-1}$*.*

Proof. The proof is given for the case $2 \le r \le 9$; similar reasoning applies to the case $r = 1$. From the definition of the `ChangeDualState` operation (see Step 2 of Phase 1 of Karroumi's implementation) it follows that

$$(C_0^{(r,j')}, C_1^{(r,j')}, C_2^{(r,j')}, C_3^{(r,j')}) = (\Delta_{r,j'}, \Delta_{r,j'}, \Delta_{r,j'}, \Delta_{r,j'}) \circ (\Delta^{-1}_{r-1,\mathrm{sr}(0,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(1,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(2,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(3,j')}) \quad \text{if } 2 \le r \le 9 \ ,$$

for $0 \le j \le 3$. Substituting the above expression for the `ChangeDualState` operation in Eq. 13 and applying Lemma 3 gives

$$\overline{AES}^{(r,j,\Delta_{r,j})} = \Pi_2^{(r,j)} \circ (\Delta_{r,j'}, \Delta_{r,j'}, \Delta_{r,j'}, \Delta_{r,j'}) \circ AES^{(r,j')} \circ (\Delta^{-1}_{r-1,\mathrm{sr}(0,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(1,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(2,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(3,j')}) \circ \Pi_1^{(r,j)} \ .$$

Observe that $\Pi_2^{(r,j)}$ and $(\Delta_{r,j'}, \Delta_{r,j'}, \Delta_{r,j'}, \Delta_{r,j'})$ commute and thus can be swapped. By applying the equation

$$(\Delta^{-1}_{r-1,\mathrm{sr}(0,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(1,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(2,j')}, \Delta^{-1}_{r-1,\mathrm{sr}(3,j')}) \circ \Pi_1^{(r,j)} = \Pi_1^{(r,j)} \circ (\Delta^{-1}_{r-1,\mathrm{sr}(0',j')}, \Delta^{-1}_{r-1,\mathrm{sr}(1',j')}, \Delta^{-1}_{r-1,\mathrm{sr}(2',j')}, \Delta^{-1}_{r-1,\mathrm{sr}(3',j')}) \ ,$$

where $i' = (\pi_1^{(r,j)})^{-1}(i)$ for $i = 0, 1, 2, 3$ where $(\pi_1^{(r,j)})^{-1}$ denotes the permutation on the indices of a 4-byte vector as a result of the application of $\left(\mathit{\Pi}_1^{(r,j)}\right)^{-1}$, one gets the result of Lemma 4. $\square$

From the discussion above it follows that Karroumi's white-box AES implementation and the white-box AES implementation of Chow *et al.* are the same. As a consequence, Karroumi's white-box AES implementation is vulnerable to the original BGE attack and the attacks presented in this paper.

6 Conclusion

The BGE attack on the white-box AES implementation of Chow *et al.* extracts the AES key from such an implementation with a work factor of 2^{30}. Taking Tolhuizen's improvement to the most time-consuming phase of the BGE attack as the starting point, Section 3 presented several improvements to the other phases of the BGE attack. It was shown that the overall work factor of the BGE attack is reduced to 2^{22} when all improvements are implemented. Unlike the original BGE attack, the use of non-affine white-box encodings and the randomization in the order of the bytes of the intermediate results in AES have a negligible contribution to the overall work factor of the improved BGE attack.

Section 4 presented a new attack on the white-box implementation of Chow *et al.* based on collisions occurring in the output bytes of an encoded AES round. It was shown that the new attack also has a work factor of 2^{22}.

Karroumi's white-box AES implementation was designed to withstand the BGE attack. Section 5 showed that the white-box AES implementations of Chow *et al.* and Karroumi are the same. As a result, the original BGE attack and the attacks presented in this paper can be applied directly to extract the key from Karroumi's white-box AES implementation, implying that this implementation is insecure.

Acknowledgements. This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). In addition, this work was supported by the Flemish Government, FWO WET G.0213.11N and IWT GBO SEC SODA. Yoni De Mulder was supported in part by a research grant of iMinds of the Flemish Government.

References

1. Elad Barkan and Eli Biham. In How Many Ways Can You Write Rijndael? In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2002.
2. Elad Barkan and Eli Biham. The Book of Rijndaels. *IACR Cryptology ePrint Archive*, 2002:158, 2002. `http://eprint.iacr.org/2002/158`.

3. Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2004.
4. Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2003.
5. Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-Box Cryptography and an AES Implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.
6. Mohamed Karroumi. Protecting White-Box AES with Dual Ciphers. In Kyung Hyune Rhee and DaeHun Nyang, editors, *ICISC*, volume 6829 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2010.
7. Tancrède Lepoint and Matthieu Rivain. Another Nail in the Coffin of White-Box AES Implementations. Cryptology ePrint Archive, Report 2013/455, 2013. `http://eprint.iacr.org/2013/455.pdf`.
8. National Institute of Standards and Technology. Advanced Encryption Standard. Federal Information Processing Standard (FIPS), Publication 197, U.S. Department of Commerce, Washington D.C., November 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.
9. James A. Muir. A Tutorial on White-box AES. *Mathematics in Industry*, 2012. To appear. `http://www.ccsl.carleton.ca/~jamuir/papers/wb-aes-tutorial.pdf`.
10. Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao - Lai White-Box AES Implementation. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2012.
11. Yoni De Mulder, Peter Roelse, and Bart Preneel. Revisiting the BGE Attack on a White-Box AES Implementation. Cryptology ePrint Archive, Report 2013/450, 2013. `http://eprint.iacr.org/2013/450.pdf`.
12. Ludo Tolhuizen. Improved Cryptanalysis of an AES implementation. *33rd WIC Symposium on Information Theory in the Benelux*, 2012.
13. Yaying Xiao and Xuejia Lai. A Secure Implementation of White-Box AES. In *2nd International Conference on Computer Science and its Applications (CSA 2009)*, pages 1–6. IEEE, 2009.