On the Ineffectiveness of Internal Encodings - Revisiting the DCA Attack on White-Box Cryptography

Estuardo Alpirez Bock[1], Chris Brzuska[1,4], Wil Michiels[2,3], and Alexander Treff[1]

[1] Hamburg University of Technology
[2] Technische Universiteit Eindhoven
[3] NXP Semiconductors
[4] Aalto University

Abstract. The goal of white-box cryptography is to implement cryptographic algorithms securely in software in the presence of an adversary that has complete access to the software's program code and execution environment. In particular, white-box cryptography needs to protect the embedded secret key from being extracted. As for today, all publicly available white-box implementations turned out susceptible to key extraction attacks. In the meanwhile, white-box cryptography is widely deployed in commercial implementations that claim to be secure. Bos, Hubain, Michiels and Teuwen (CHES 2016) introduced differential computational analysis (DCA), the first automated attack on white-box cryptography. The DCA attack performs a statistical analysis on execution traces. These traces contain information about the execution, such as memory addresses or register values, that is collected via binary instrumentation tooling during the encryption process. The white-box implementations that were attacked by Bos et al., as well as white-box implementations that have been described in the literature, protect the embedded key by using internal encodings techniques that have been introduced by Chow, Eisen, Johnson and van Oorschot (SAC 2002). In this paper, we prove rigorously that such internal encodings are too weak to protect against the DCA attack and thereby explain the experimental success of the DCA attack of Bos et al.

Keywords: white-box cryptography, differential computational analysis, software execution traces, mixing bijections

1 Introduction

When an application for mobile payments runs in software on Android phones or other open platforms, it needs to protect itself as it cannot rely on platform security. In particular, the cryptographic algorithms used within an application need to be secured against adversaries that have a high degree of control over the environment. Cryptography that remains secure even when the adversary

has full control over the execution environment is known as *white-box cryptography*. The white-box attack model was introduced in 2002 by Chow, Eisen, Johnson and van Oorschot [9,8], originally in the context of Digital Rights Management (DRM). However, the white-box attack model may be a serious threat for many modern applications such as, for instance, mobile payment applications. Originally, mobile payment applications would rely on secure hardware to implement near field communication (NFC) protocols, but since the addition of host-card emulation (HCE) in Android 4.4, NFC protocols can be implemented in software-only. White-box cryptography has thus become an attractive tool to help increase the security of such applications [20].

A necessary requirement for secure white-box cryptography is that an adversary cannot extract the embedded secret key from the implementation. Chow, Eisen, Johnson and van Oorschot [9,8] suggest to implement a symmetric cipher with a fixed key as a network of look-up tables (LUT). The key is compiled into a table instead of being stored in plain in the implementation. In order to protect a network of lookup tables against reverse-engineering, Chow et al. propose to obfuscate the lookup tables and the intermediate results via a combination of linear and non-linear encodings. The idea of implementing symmetric ciphers as such an obfuscated network of LUTs has caught on in the white-box community since then, see, e.g., [7,23,10]. However, although the LUT-based white-box designs hide the key in obfuscated lookup tables, it turns out that all aforementioned LUT-based designs are succeptible to key extraction attacks performed via differential and algebraic cryptanalysis (see [4,17,16,13]). Specifically, these attacks invert the obfuscation process by deriving the applied encoding functions after which the key can easily be recovered.

In real-life applications, mounting cryptanalysis and reverse engineering attacks requires abundant skills and time from an adversary. A larger practical approach for an adversary are *automated* key extraction attacks which were introduced by Bos, Hubain, Michiels and Teuwen [6] and Sanfelix, de Haas and Mune [18]. Their method, known as the differential computational analysis (DCA), is described by the authors as the software counterpart of the differential power analysis (DPA) applied for attacking cryptographic hardware implementations [12]. Bos et al. [6] monitor the memory addresses accessed by a program during the encryption process and display them in the form of *software execution traces*. These software execution traces can also include other information that can be monitored using binary instrumentation, such as stack reads or register values. These traces serve the following three goals. (1) They can help to determine which cryptographic algorithms was implemented. (2) The traces provide hints to determine where roughly the cryptographic algorithm is located in the software implementation. (3) Finally and importantly, the traces can be statistically analyzed to extract the secret key. The automated DCA attack turned out to be successful against a large number of publicly available white-box implementations. It has since then become a popular method for the evaluation of newly proposed white-box implementations [5] and software countermeasures for white-box cryptography [1].

While Bos et al. [6] explain (1) and (2) in detail, their exposition of (3) is rather laconic. In addition, we need a further study to understand why step (3) of the attack actually works for the type of white-box techniques that are currently in use in order to improve upon them. The work of Sasdrich, Moradi and Güneysu [19] takes a first step towards this understanding. They use the Walsh transform to show that the encodings used by their white-box AES design are not balanced correlation immune and thus are susceptible to the DCA attack. In this paper, we aim at giving a structured exposition to improve our understanding of the power of the DCA attack which can then guide the search for new and more effective countermeasures.

Our contribution In this paper we provide an annotated step-by-step graphical presentation of the key-extraction step of the DCA attack, which relies on a difference of means distinguisher, and explain how to interpret the results. Our presentation follows the style that Kocher [11] and Messerges [14] used for the (analogous) differential power analysis on hardware implementations.

Further, we analyse how the presence of internal encodings on white-box implementations affects the effectiveness of the DCA attack. Thereby, we focus on the encodings suggested by Chow et. al. [9,8], which are a combination of linear and non-linear transformations. We start by studying the effects of a single linear transformation. We derive a sufficient and necessary condition for the DCA attack to successfully extract the key from a linearly encoded look-up table. Namely, if the outputs of a key-dependent look-up table are encoded via an invertible matrix that contains at least one row with Hamming weight (HW) $= 1$, then the DCA will be successful. In this same line, we show that an invertible matrix whose rows all have HW > 1, provides an effective masking for the look-up table outputs, such that a standard DCA attack is not effective any more. However, we explain later in the paper how the DCA attack can be modified in such way that it is successful on any linearly encoded key-dependent look-up table. Next, we consider the effect that non-linear nibble encodings have on the outputs of key-dependent look-up tables and prove that the use of nibble encodings firstly provides conditions so that the DCA attack succeeds. Namely, when we attack a key dependent look-up table encoded via non-linear nibble encodings, we always obtain a difference of means curve with values equal to either 0, 0.25, 0.5, 0.75 or 1 for the correct key guess. The results obtained from these analyses help us determine why the DCA attack also works in the presence of both linear and non-linear nibble encodings. Hereby we focus first on the invertible matrix performing the linear transformation. We prove that, if one half of the matrix has at least one column whose value is not contained in the space spanned by the other columns, then we always obtain a difference of means curve with values equal to either 0, 0.25, 0.5, 0.75 or 1 for the correct key guess. These values on the difference of means curves are caused by the effect of the nibble encodings applied *after* the linear transformations.

Throughout the paper, we also present experimental results of the DCA attack when performed on single key-dependent look-up tables and on complete

white-box implementations. In all cases, we see that the results match completely to the observations pointed out in our analyses.

2 White-Box Cryptography Implementations

White-box cryptography can be seen as special-purpose obfuscation, but is usually not discussed in this way. In particular, *general*-purpose obfuscation with perfect security is known to be impossible [2] and the hope is that achieving weaker-than-perfect security for a *specific* algorithm is still feasible. The most popular approach in academic literature (and perhaps also beyond) for white-box implementations of symmetric encryption is to encode the underlying symmetric cipher with a fixed key as a networks of look-up tables (LUT). In particular, the LUTs depend on the secret key used in the cipher. An additional protection technique is to apply linear and non-liner *internal* encodings which are used to encode the intermediate state between LUTs. Another popular technique are *external* encodings which are applied on the outside of the cipher and help to bind the white-box to an application. In this paper, we focus solely on internal encodings, because, as Bos et al. point out in [6], applying external input and output encodings yields an implementation of a function that is not functionally equivalent to AES anymore and thus, some of its security can be shifted to other programs. Moreover, this paper focusses on using internal encodings for LUT-based white-box constructions of AES. We will focus on the encodings and refer to the LUT-based construction as an abstract design. The interested reader might find the work by Muir [15] a useful read for a more detailed description on how to construct an LUT-based white-box AES implementation. In this section, we first provide a short description of AES, recalling the operations performed during its execution rounds. We later introduce the concepts of linear and non-linear encodings.

2.1 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is a symmetric-key block cipher introduced by Vincent Rijmen and Joan Daemen [22]. It is the most widely deployed cipher since it replaced its successor Data Encryption Standard (DES) [21]. In the following, we focus on AES-128 which is an AES variant that uses 128 bit keys and, for any fixed key, implements a bijective function on 128 bit blocks. In the description that follows, we will briefly recall the iterative structure of AES. AES takes a 128 bit long plaintext p as input and sets it as its *input state* z. It is convenient to think of the state as a 4×4 matrix of bytes which we display as a 2-dimensional array. Each round of the algorithm provides a new state. It is also useful to think of the AES operations as being performed over the finite field $GF(2^8)$. AES-128 consists of a `KeyExpansion` operation and 10 rounds of operations which we describe in the following. The first 9 rounds consist of the `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` operations. The 10th round consists only of the `SubBytes`, `ShiftRows` and `AddRoundKey` operations, without performing `MixColumns`.

`KeyExpansion` takes the secret key k as input and computes *round keys* k_i, with $i = 0, \ldots, 10$ and where the 0th round key k_0 is equal to k. k_0 is directly added to the input state before the first round of AES-128.

`AddRoundKey` adds a round key k_i *bytewise* to the current state by using an exclusive-or operation.

`SubBytes` is an invertible, non-linear transformation that substitutes each state byte by another byte chosen from a substitution table, called `S-Box`.

`ShiftRows` performs a cyclical shift of the state rows. Each row is shifted differently depending on its row number. The first row is not shifted, the second row is shifted by one position to the left, the third row is shifted by two positions to the left and the last row is shifted by three positions to the left.

`MixColumns` treats each column of the state as a polynomial $b_i(x) = s_{3,i}x^3 + s_{2,i}x^2 + s_{1,i}x + s_{0,i}$ for columns $i = 0, \ldots, 3$. Then, $b_i(x)$ is multiplied by $a(x) = \texttt{03}x^3 + \texttt{01}x^2 + \texttt{01}x + \texttt{02} \mod x^4 + 1$ over $GF(2^8)$. Commonly, this is represented as a vector-matrix multiplication, where each state column is multiplied by a matrix in $GF(2^8)$. The operation returns a new state consisting of the resulting columns of each multiplication.

2.2 Internal encodings

Consider an LUT-based white-box implementation of AES, where the LUTs depend on the secret key. Internal encodings can now help to re-randomize those LUTs to make it harder to recover secret-key information based on the LUTs. Such internal encodings were first suggested by Chow et. al [9,8]. We now discuss two types of encodings. Random bijections a.k.a. non-linear encodings are used to achieve *confusion* on the tables, i.e. hiding the relation between the content of the table and the secret key. In turn, linear transformations a.k.a. mixing bijections can be applied at the input and output of each table to achieve *diffusion* such that if one bit is changed in the input of a table, then several bits of its corresponding output are changed too.

Non-linear encodings Recall that the secret key is hard-coded in the LUTs. When non-linear encodings are applied, each LUT in the construction becomes statistically independent from the key and thus, attacks need to exploit key dependency across several LUTs. A table T can be transformed into a table T' by using the input bijections I and output bijections O as follows:

$$T' = O \circ T \circ I^{-1}.$$

As a result, we obtain a new table T' which maps encoded inputs to encoded outputs. Note that no information is lost as the encodings are bijective. If table T' is followed by another table R', their corresponding output and input encodings

can be chosen such that they cancel out each other. Say, that for two tables T and R we have the following input and output encodings

$$T' = O_{T'} \circ T \circ I_{T'}^{-1} \text{ and } R' = O_{R'} \circ R \circ I_{R'}^{-1}.$$

We can choose them such that $I_{T'}^{-1} \circ O_{R'}$ is the identity function and thus,

$$R' \circ T' = (O_{R'} \circ R \circ I_{R'}^{-1}) \circ (O_{T'} \circ T \circ I_{T'}^{-1}) = O_{R'} \circ (R \circ T) \circ I_{T'}^{-1}.$$

Considering a complex network of LUTs of an AES implementation, we have input- and output encodings on almost all look-up tables. The only exceptions are the very first and the very last tables of the AES implementation, which take the input of the algorithm and correspondingly return the output data. The first tables omit the input encodings and the last tables omit the output encodings. As the internal encodings cancel each other out, the encodings do not affect the input-output behaviour of the AES implementation.

Size requirements Descriptions of uniformly random bijections (which are non-linear with overwhelming probability) are exponential in the input size of the bijection. Therefore, a uniformly random encoding of the 8-bit S-box takes space $2^8 \cdot 2^8$. If we were to encode a 16-bit function, the required space would already be 2^{32}. Therefore, one usually splits longer values in nibbles of 4 bits and then only needs two tables of size 2^{16} rather than one table of size 2^{32}. However, by moving to a split non-linear encoding we introduce a vulnerability since a bit in one nibble does no longer influence the encoded value of another nibble in the same encoded word. To (partly) compensate for this, Chow et al. propose to apply linear encodings whose size is merely quadradic in the input size and thus, they can be implemented on larger words. We explain next when linear encodings on large words can be used.

Linear encodings A convenient feature of linear encodings is that, by definition of linearity, they commute with the XOR-operation. Therefore, Chow suggests to apply linear encodings to words that are input or output of an XOR-network. These linear encodings have as width the complete word and are applied before the non-linear encodings discussed above. While the non-linear encodings need to be removed before performing an XOR, one can perform the XOR on linearly encoded values (due to commutativity). Therefore, one usually refers to linear encodings as *mixing bijections*.

The linear encodings are invertible and selected uniformly at random. For example, we can select L and A as a mixing bijections for inputs and outputs of table T respectively:

$$A \circ T \circ L^{-1}.$$

As stated above, it is not necessary to cancel the effect of the linear encodings before an XOR-operation. However, after the XOR-operation we obtain an output which is still dependent on the linear function A and the effect of A needs to be eventually removed, e.g. at the end of an AES round. In this case, dedicated tables in the form of

$$L_n \circ A^{-1}$$

are introduced, where L_n is the corresponding linear encoding needed for the next LUT. In the white-box designs of Chow et al. we have 8-bit and 32-bit mixing bijections. The former encode the 8-bit S-box inputs, while the latter obfuscate the MixColumns outputs.

3 Differential Computational Analysis

We now revisit the DCA attack by Bos et al. [6]. As the white-box attacker has full control over the platform, the adversary can execute the binary and simultaneously use a Dynamic Binary Instrumentation framework such as Pin and Valgrind to record the addresses that are accessed (for more details on the acquisition of the software traces, see the original DCA paper by Bos et al. [6]). Note that unlike in side-channel analysis, software-based memory-tracking is noise-free. To display the tracked memory-information in a so-called *software execution trace*, one proceeds as follows: One fixes one bit of information of the bit string that describes the memory address and displays whether the bit was 0 or 1 at each memory access performed during the execution. In this section we provide a detailed description of one statistical method to analyse such software execution traces, namely the *difference of means* method. The two attack capabilities required to perform the DCA attack are as follows:

- execute the white-box program under attack several times in a controlled environment with different input messages.
- knowledge of the plaintext[5] values given to the program as input.

The goal of the attack is to determine the first-round key of AES as it allows to recover the entire key. The first-round key of AES is 128 bits long and the attack aims to recover it byte-by-byte. For the remainder of this section, we focus on recovering the first byte of the first-round key, as the recovery attack for the other bytes of the first round key proceeds analogously. For the first key byte, the attacker tries out all possible 256 key byte hypotheses k^h, with $1 \leq h \leq 256$, uses the traces to test how good a key byte hypothesis is, and eventually returns the key hypothesis that performs best according to a metric that we specify shortly. For sake of exposition, we focus on one particular key-byte hypothesis k^h.

[5] The attack works analogously when having access to the ciphertexts. The attacker needs access to either plaintexts or ciphertexts.

The adversary starts by collecting memory access traces s_e which are associated with some plaintext p_e. To test the key byte hypothesis k^h, the adversary first specifies a selection function (detailed in Step **2** below) `Sel` that calculates one state-byte depending on the plaintext p_e and the the key byte hypothesis k^h. `Sel` returns only the j-th bit of the state-byte, which we denote as b. For each pair (s_e, p_e), the adversary groups the trace s_e in a set A_b, where $b = \texttt{Sel}(p_e, k^h, j) \in \{0, 1\}$. The adversary then performs the difference of means test (explained from Step **4** on) which, essentially, measures correlations between a bit of the memory address and the bit $b = \texttt{Sel}(p_e, k^h, j)$. If those correlations are strong, then the attack algorithm considers the key byte hypothesis k^h good. We now explain the analysis steps performed in the DCA attack.

1. Collecting Traces We first execute the white-box program n times, each time using a different plaintext p_e, $1 \leq e \leq n$ as input. For each execution, one software trace s_e is recorded during the first round of AES. Fig. 1 shows a single software trace consisting of 300 samples. Each sample corresponds to one bit of the memory addresses accessed during execution.

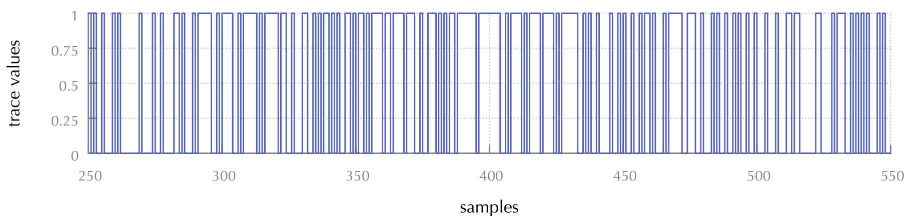


Fig. 1: Single software trace consisting of 300 samples

2. Selection Function We define a selection function for calculating an intermediate state-byte of the calculation process of AES. More precisely, we calculate a state-byte which depends on the key-byte we are analysing in the actual iteration of the attack. For the sake of simplicity, we refer to this state-byte as z. The selection function returns only one bit of z, which we refer to as our *target bit*. The value of our target bit will be used as a distinguisher in the following Steps.

In this work, our selection function $\texttt{Sel}(p_e, k^h, j)$ calculates the state z after the `SBox` substitution in the first round. The index j indicates *which* bit of z is returned, with $1 \leq j \leq 8$.

$$\texttt{Sel}(p_e, k^h, j) := \texttt{SBox}(p_e \oplus k^h)[j] = b \in \{0, 1\}. \quad (1)$$

Depending on the white-box implementation being analysed, it may be the case that strong correlations between b and the software traces are only observable for some bits of z, i.e. depending on which j we choose to focus on. Thereby,

we perform the following Steps 3, 4 and 5 for each bit j of z.

3. Sorting of Traces We sort each trace s_e into one of the two sets A_0 or A_1 according to the value of $\texttt{Sel}(p_e, k^h, j) = b$:

$$\text{For } b \in \{0,1\}\ A_b := \{s_e | 1 \leq e \leq n,\ \texttt{Sel}(p_e, k^h, j) = b\}. \tag{2}$$

4. Mean Trace We now take the two sets of traces obtained in the previous step and calculate a *mean trace* for each set. We add all traces of one set sample wise and divide them by the total number of traces in the set. For $b \in \{0,1\}$, we define

$$\bar{A}_b := \frac{\sum_{s \in A_b} s}{|A_b|}, \tag{3}$$

For each of the two sets, we obtain a mean trace such as the one shown in Fig. 2.

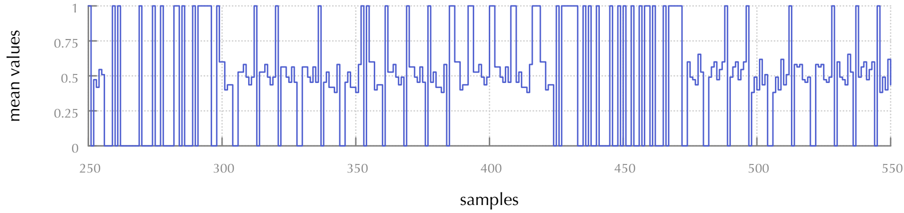


Fig. 2: Mean trace for the set A_0

5. Difference of Means We now calculate the difference between the two previously obtained mean traces sample wise. Fig. 3 shows the resulting difference of means trace:

$$\Delta = |\bar{A}_0 - \bar{A}_1|. \tag{4}$$

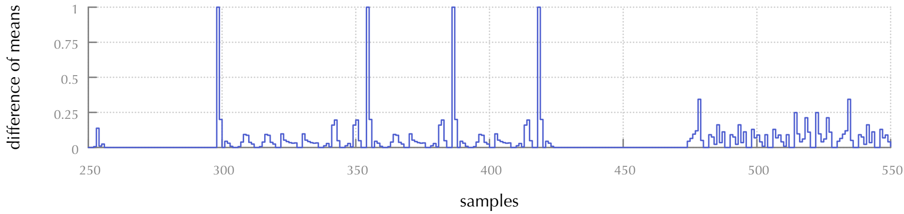


Fig. 3: Difference of means trace for correct key guess

6. Best target bit We now compare the difference of means traces obtained for all target bits j for a given key hypothesis k^h. Let Δ^j be the difference of

means trace obtained for target bit j, and let $H(\Delta^j)$ be the highest peak in the trace Δ^j. Then, we select Δ^j as the best difference of means trace for k^h, such that $H(\Delta^j)$ is maximal amongst the highest peaks of all other difference of means traces, i.e.

$$\forall\ 1 \leq j' \leq 8,\ H(\Delta^{j'}) \leq H(\Delta^j).$$

In other words, we look for the highest peak obtained from any difference of means trace. The difference of means trace with the highest peak $H(\Delta^j)$ is assigned as the difference of means obtained for the key hypothesis k^h analysed in the actual iteration of the attack, such that $\Delta^h := \Delta^j$. We explain this reasoning in the analysis provided after Step 7.

7. Best Key Byte Hypothesis Let Δ^h be the difference of means trace for key hypothesis h, and let $H(\Delta^h)$ be the highest peak in the trace Δ^h. Then, we select k^h such that $H(\Delta^h)$ is maximal amongst all other difference of means traces Δ^h, i.e.

$$\forall\ 1 \leq h' \leq 256,\ H(\Delta^{h'}) \leq H(\Delta^h).$$

Analysis The higher $H(\Delta^h)$, the more likely it is that this key-hypothesis is the correct one, which can be explained as follows. The attack partitions the traces in sets A_0 and A_1 based on whether a bit in z is set to 0 or 1. First, suppose that the key hypothesis is correct and consider a region R in the traces where (an encoded version of) z is processed. Then, we expect that the memory accesses in R for A_0 are slightly different than for A_1. After all, if they would be the same, the computations would be the same too. We know that the computations are different because the value of the target bit is different. Hence, it may be expected that this difference is reflected in the mean traces for A_0 and A_1, which results in a peak in the difference of means trace. Next, suppose that the key hypothesis was not correct. Then, the sets A_0 and A_1 can rather be seen as a random partition of the traces, which implies that z can take any arbitrary value in both A_0 and A_1. Hence, we do not expect big differences between the executions traces from A_0 and A_1 in region R, which results in a rather flat difference of means trace.

To illustrate this, consider the difference of means trace depicted in Fig. 3. This difference of means trace corresponds to the analysis performed on a white-box implementation obtained from the hack.lu challenge [3]. This is a public table-based implementation of AES-128, which does not make any use of internal encodings. For analysing it, a total of 100 traces were recorded. The trace in Fig. 3 shows four spikes which reach the maximum value of 1 (note that the sample points have a value of either 0 or 1). Let ℓ be one of the four sample points in which we have a spike. Then, having a maximum value of 1 means that for all traces in A_0, the bit of the memory address considered in ℓ is 0 and that this bit is 1 for all traces in A_1 (or vice versa). In other words, the target bit $z[j]$ is either directly or in its negated form present in the memory address accessed in the implementation. This can happen if z is used in non-encoded form as input to a lookup table or if it is only XORed with a constant mask. For sake of

completeness, Fig. 4 shows a difference of means trace obtained for an incorrect key-hypothesis. No sample has a value higher than 0.3.

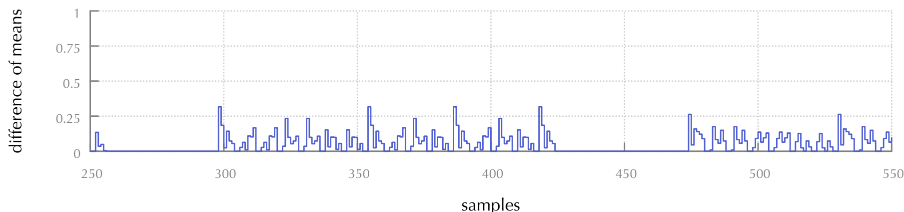


Fig. 4: Difference of means trace for incorrect key guess

The results of the DCA attack shown in this section correspond to the attack performed using software traces which consist of the memory addresses accessed during the encryption process. The attack can also be performed using software traces which consist of other type of information, e.g., the stack writes and/or reads performed during encryption. In all cases, the analysis is performed in an analogous way as explained in this section.

3.1 Successful Attack

Throughout this paper, considering the implementation of a cipher, we refer to the DCA attack as being *successful for a given key k*, if this key is ranked number 1 among all possible keys for a large enough number of traces. It may be the case that multiple keys have this same rank. If DCA is not successful for k, then it is called *unsuccessful for key k*. Remark that in practice, an attack is usually considered successful as long as the correct key guess is ranked as one of the best key candidates. We use a stronger definition as we require the correct key guess to be ranked as the best key candidate.

Alternatively when attacking one precise n-bit to n-bit key dependent look-up table, we consider the DCA attack as being *successful for a given key k*, if this key is ranked number 1 among all possible keys for exactly 2^n traces. Thereby, each trace is generated by giving exactly 2^n different inputs to the look-up table, i.e. all possible inputs that the look-up table can obtain. To get the correlation between a look-up table output and our selection function, the correlation we obtain by evaluating all 2^n possible inputs is exactly equal to the correlation we obtain by generating a large enough number of traces for inputs chosen uniformly at random. We use this property for the experiments we perform in the following section.

4 Effect of the Encodings

In this section we analyse the use of internal encodings on white-box implementations as suggested by Chow et. al. [8]. Moreover, we analyse how the presence

of such encodings affects the vulnerability of a white-box implementation to the DCA attack. If intermediate values in an implementation are encoded, it becomes more difficult to re-calculate such values using our selection function as defined in Step 2 of the DCA (see Sec. 3). Namely, `Sel` does not consider the transformations used to encode these intermediate values, but only calculates a value *before* it is encoded. Thus, what we calculate with `Sel` in Step 2 does not necessarily match with the actual encoded value computed by the white-box, even if the correct key hypothesis is used.

In this section we discuss linear encodings, non-linear encodings and a combination of both as means to protect key dependent look-up tables in a white-box implementation. These types of encodings are the methods usually applied in the literature and in several open white-box implementations. For our analyses in this section, we first build single look-up tables which map an 8-bit long input to an 8-bit long output. More precisely, these look-up tables correspond to the key addition operation merged with the S-box substitution step performed on AES (see Sec. 2.1). As common in the literature, we refer to such look-up tables as *T-boxes*. We apply the different encoding methods to the outputs of the look-up tables and obtain encoded T-boxes. Following our definition for a successful DCA attack on an n-to-n look-up table given in Sec. 3.1, we generate exactly 256 different software traces for attacking a T-box. Our selection function is defined the same way as in Step 2 of Sec. 3 and calculates the output of the T-boxes *before* it is encoded. The output of the T-box is a typical vulnerable spot for performing the DCA on white-box implementations as this output can be calculated based on the known plaintext and a key guess. As we will see in this section, internal encodings as suggested by Chow et. al. cannot effectively add a masking countermeasure to the outputs of the S-box.

4.1 Linear Encodings

The outputs of a T-box can be *linearly* encoded by applying linear transformations. To do this, we randomly generate an 8-to-8 invertible matrix such as, for instance,

$$A = \begin{bmatrix} 1&1&1&0&1&0&0&0 \\ 0&0&1&1&0&1&0&1 \\ 1&0&1&0&0&0&1&0 \\ 0&1&1&1&0&1&0&1 \\ 0&1&1&1&0&1&1&1 \\ 1&1&0&0&0&0&1&1 \\ 0&0&1&0&0&0&1&0 \\ 1&0&0&0&0&1&1&0 \end{bmatrix}. \quad (5)$$

For each output y of the T-box T, we perform a matrix multiplication $A \cdot y$ and obtain an encoded output m. We obtain a new look-up table lT, which maps each input x to a linearly encoded output m. Fig. 5 displays this behaviour.

We now compute the DCA on the outputs of lT. The difference of means analysis is performed in the same way as described in Sec. 3. Fig. 6 shows the

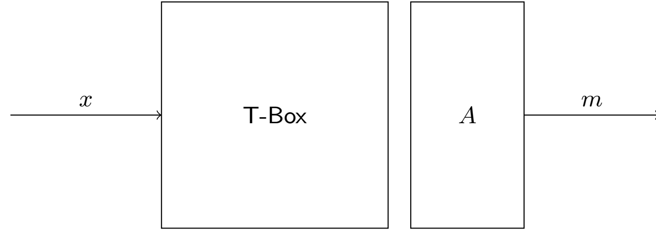


Fig. 5: An lT-box maps each input x to a linearly encoded output m.

results of the analysis when using the correct key guess. Since we are attacking only an 8×8 look-up table, the generated software traces consist only of 8 samples which correspond to the 8 output bits of the lT-box. As it can be seen, all samples in the difference of means curve have a value of zero or almost zero. No correlations can be identified and thus, the analysis is not successful if the output of the original T-box is encoded using the matrix A.

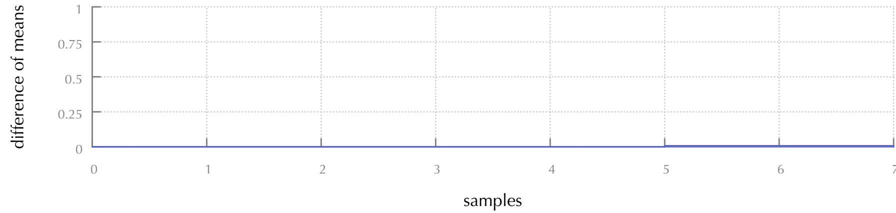


Fig. 6: Difference of means trace for the lT-box

The results shown in Fig. 6 correspond to the DCA performed on a look-up table constructed using one particular linear transformation to encode the output of one look-up table. We observe that the DCA as described in Sec. 3 is not effective in the presence of this particular transformation. However in practice, linear transformations are randomly chosen and some may not effectively hide information about a target bit, such that the DCA attack is successful. The theorem below gives a necessary and sufficient condition under which the DCA attack is successful in the presence of linear transformations.

Theorem 1. *Given a T-box encoded via an invertible matrix A. The DCA attack returns a difference of means value equal to 1 for the correct key guess if and only if the matrix A has at least one row i with Hamming weight $(HW) = 1$. Otherwise, the DCA attack returns a difference of means value equal to 0 for the correct key guess.*

Proof. For all $1 \leq j \leq 8$ let $y[j]$ be the jth bit of the output y of a T-box. Let $a_{ij} \in GF(2)$ be the entries of an 8×8 matrix A, where i denotes the row and j

denotes the column of the entry. We obtain each encoded bit $m[i]$ of the lT-box via

$$m[i] = \sum_j a_{ij} \cdot y[j] = \sum_{j:a_{ij}=1} y[j]. \tag{6}$$

Suppose that row i of A has $HW(i) = 1$. Let j be such that $a_{ij} = 1$. It follows from Equation (6) that $m[i] = y[j]$. Let k^h be the correct key hypothesis and let bit $y[j]$ be our target bit. With our selection function $\texttt{Sel}(p_e, k^h, j)$ we calculate the value for $y[j]$ and sort the corresponding trace in the set A_0 or A_1. We refer to these sets as sets consisting of encoded values m, since a software trace is a representation of the encoded values. Recall now that $y[j] = m[i]$. It follows that $m[i] = 0$ for all $m \in A_0$ and $m[i] = 1$ for all $m \in A_1$. Thus, when calculating the averages of both sets, for $\bar{A}[i]$, we obtain $\bar{A}_0[i] = 0$ and $\bar{A}_1[i] = 1$. Subsequently, we obtain a difference of means curve with $\Delta[i] = 1$, which leads us to a successful DCA attack.

What's left to prove is that if row i has $HW(i) > 1$, then the value of bit $y[j]$ is masked via the linear transformation such that the difference of means curve obtained for $\Delta[i]$ has a value converging to zero. Suppose that row i of A has $HW(i) = l > 1$. Let j be such that $a_{ij} = 1$ and let $y[j']$ denote one bit of y, such that $a_{ij'} = 1$. It follows from Equation (6) that the value of $m[i]$ is equal to the sum of at least two bits $y[j]$ and $y[j']$. Let k^h be the correct key hypothesis and let $y[j']$ be our target bit. Let $\vec{v}$ be a vector consisting of the bits of y, for which $a_{ij} = 1$, excluding bit $y[j']$. Since row i has $HW(i) = l$, vector $\vec{v}$ consists of $l - 1$ bits. This means that $\vec{v}$ can have up to 2^{l-1} possible values. Recall that each non-encoded T-box output value y occurs with an equal probability of $1/256$ over the inputs of the T-box. Thus, all 2^{l-1} possible values of $\vec{v}$ occur with the same probability over the inputs of the T-box. The sum of the $l - 1$ bits in $\vec{v}$ is equal to 0 or 1 with a probability of 50%, independently of the value of $y[j']$. Therefore, our target bit $y[j']$ is masked via $\sum_{j:a_{ij}=1, j\neq j'} y[j]$ and our calculations obtained with $\texttt{Sel}(p_e, k^h, j')$ only match 50% of the time with the value of $m[i]$. Each set A_b consists thus of an equal number of values $m[i] = 0$ and $m[i] = 1$. The difference between the averages of both sets is thus equal to zero and the DCA is unsuccessful for k^h. □

One could be tempted to believe that using a matrix which does not have any identity row serves as a good countermeasure against the DCA attack. However, we could easily adapt the DCA attack such that it is also successful in the presence of a matrix without any identity row. In Step 2, we just need to define our selection function such that, after calculating an 8-bit long output state z, we calculate all possible linear combinations LC of the bits in z. Thereby, in Step 3 we sort according to the result obtained for an LC. This means that we perform Steps 3 to 5 for each possible LC ($2^8 = 256$ times per key guess). For at least one of those cases, we will obtain a difference of means curve with peak values equal to 1 for the correct key guess as our LC will be equal to the LC defined by row i of matrix A. Our selection function calculates thus a value equal to the encoded value $m[i]$ and we obtain perfect correlations.

Note that Theorem 1 also applies in the presence of affine encodings. In case we add a 0 to a target bit, traces $\bar{A}_0$ and $\bar{A}_1$ do not change and in case we add a 1 the entries in $\bar{A}_0$ and $\bar{A}_1$ that relate to the target bit change to 1 minus their value. In both cases, the difference of means value does not change.

To illustrate how the effect of linear encodings is shown on complete white-box implementations, we now perform the DCA attack on our white-box implementation of AES which only makes use of linear encodings. This is a table based implementation which follows the design strategy proposed by Chow et. al., but only uses linear encodings. We collect 200 software traces, which consist of the memory addresses accessed during the encryption process. We use our selection function $\texttt{Sel}(p_e, k^h, j) = z[j]$. Fig. 7 shows the difference of means trace obtained for the correct key guess.

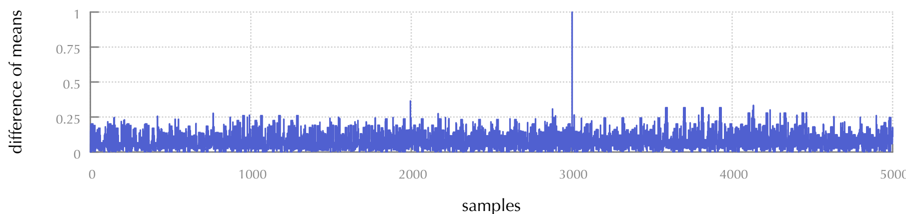


Fig. 7: DCA results for our white-box implementation with linear encodings

Fig. 7 shows one peak reaching a value of 1 (see sample 3001). Since the peak reaches the value of 1, we can again say that our selection function is perfectly correlated with the targeted bit $z[j]$, even though the output z was encoded using a linear transformation. Since our partition was done with our selection function calculating the output of the T-box, our results tell us that the 8×8 matrix used to encode the T-box outputs contains at least one identity row.

4.2 Non-Linear Encodings

Next, we consider the effect that non-linear encodings have on the outputs of a T-box. For this purpose, we randomly generate bijections, which map each output value y of the T-box to a different value f and thus obtain a non-linearly encoded T-box, which we call OT-box. Recall that a T-box is a bijective function. If we encode each possible output of a T-box T with a randomly generated byte function O and obtain the OT-box OT, then OT does not leak any information about T. Namely, given OT, *any* other T-box T' could be a candidate for constructing the same OT-box OT, since there always exists a corresponding function O' which could give us OT' such that $OT' = OT$. Chow et. al. refer to this property as *local security* [9]. Based on this property, we could expect

resistance against the DCA attack for a non-linearly encoded T-box. For practical implementations, unfortunately, using an 8-to-8 bit encoding for each key dependent look-up table is not realistic in terms of code size (see Sec. 4.1 of [15] for more details). Therefore, non-linear *nibble encodings* are typically used to encode the outputs of a T-box. The output of a T-box is 8-bits long and each half of the output is encoded by a different 4-to-4 bit transformation and both results are concatenated. Fig. 8 displays the behaviour of an OT-box constructed using two nibble encodings.

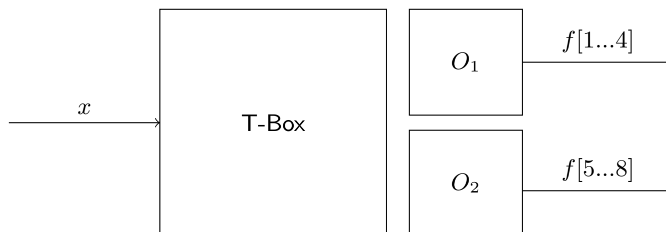


Fig. 8: Non-linear encodings of the T-Box outputs

Encoding the outputs of a T-box via non-linear nibble encodings does not hide correlations between the secret key of the T-box and its output bits as proved in the theorem below. When collecting the traces of an OT-box to perform a DCA using the correct key hypothesis, each (encoded) nibble value is returned a total of 16 times. Thereby, all encoded nibbles that have the same value are always grouped under the same set A_b in Step 3. Therefore, we always obtain a difference of means curve which consists of only 5 possible correlation values.

Theorem 2. *Given an OT-box which makes use of nibble encodings, the difference of means curve obtained for the correct key hypothesis k^h consists only of values equal to 0, 0.25, 0.5, 0.75 or 1.*

Proof. We first prove that the mean value of the set A_0 is always a fraction of 8 when we sort the sets according to the correct key hypothesis. The same applies for the set A_1 and the proof is analogous. For all $1 \leq j \leq 8$ let $y_d[j]$ be the jth bit of the output y of a T-box, where $d \in \{1, 2\}$ refers to the nibble of y where bit j is located. Let k^h be the correct key hypothesis. With our selection function $\texttt{Sel}(p_e, k^h, j)$ we calculate a total of 128 nibble values y_d, for which $y_d[j] = 0$. As there exist only 8 possible nibble values y_d for which $y_d[j] = 0$ holds, we obtain each value y_d a total of 16 times. Each time we obtain a value y_d, we group its corresponding encoded value f_d under the set A_0. Recall that an OT-box uses one bijective function to encode each nibble y_d. Thus, when we calculate the mean trace $\bar{A}_0$ and focus on its region corresponding to f_d, we do the following:

$$\bar{A}_0[f_d] = \frac{16f_d}{128} + \dots + \frac{16f'_d}{128}$$
$$= \frac{f_d}{8} + \dots + \frac{f'_d}{8},$$

with $f_d \neq f'_d$. We now prove that the difference between the means of sets A_0 and A_1 is always equal to the values 0, 0.25, 0.5, 0.75 or 1. Let $f_d[j]$ be one bit of an encoded nibble f_d.

- If $f_d[j] = 0$ is true for all nibbles in set A_0, then this implies that $f_d[j] = 1$ is true for all nibbles in set A_1, that is $\bar{A}_0[j] = \frac{8}{8}$ and $\bar{A}_1[j] = \frac{0}{8}$. The difference between the means of both sets is thus $\Delta[j] = |\frac{0}{8} - \frac{8}{8}| = |0 - 1| = 1$.
- If $f_d[j] = 1$ is true for 1 nibble in set A_0, then $f_d[j] = 1$ is true for 7 nibbles in set A_1, that is, the difference between both means is $\Delta[j] = |\frac{1}{8} - \frac{7}{8}| = |\frac{6}{8}| = 0.75$.
- If $f_d[j] = 1$ is true for 2 nibbles in set A_0, then $f_d[j] = 1$ is true for 6 nibbles in set A_1, that is, the difference between both means is $\Delta[j] = |\frac{2}{8} - \frac{6}{8}| = |\frac{4}{8}| = 0.5$.
- If $f_d[j] = 1$ is true for 3 nibbles in set A_0, then $f_d[j] = 1$ is true for 5 nibbles in set A_1, that is, the difference between both means is $\Delta[j] = |\frac{3}{8} - \frac{5}{8}| = |\frac{2}{8}| = 0.25$.
- If $f_d[j] = 1$ is true for 4 nibbles in set A_0, then $f_d[j] = 1$ is true for 4 nibbles in set A_1, that is, the difference between both means is $\Delta[j] = |\frac{4}{8} - \frac{4}{8}| = |\frac{0}{8}| = 0$.

The remaining 4 cases follow analogously and thus, all difference of means traces consist of only the values 0, 0.25, 0.5, 0.75 or 1. □

Seeing these values in a difference of means trace can help us recognise if our key hypothesis is correct. Moreover, a peak value of 0.5, 0.75 or 1 is significantly high, such that its corresponding key candidate will very likely be ranked as the correct key.

We now argue that, when we use an incorrect key candidate, nibbles with the same value may be grouped in different sets. Therefore, we cannot say as for the correct key hypothesis, that each encoded nibble value f_d is repeated exactly 16 times in a set. If we partition according to an incorrect key hypothesis k^h, the value we calculate for $y_d[j]$ does not always match with what is really calculated by the T-box and afterwards encoded by the non-linear function. It is not the case that for each nibble value y_d for which $y_d[j] = 0$, we group its corresponding encoded value f_d in the same set. Therefore, our sets A_b consist of up to 16 different encoded nibbles, whereby each nibble value is repeated a different number of times. This applies for both sets A_0 and A_1 and therefore, both sets have similar mean values, such that the difference between both means is a value closer to zero.

To get practical results corresponding to Theorem 2, we now construct 10 000 different OT-boxes following the idea displayed in Fig. 8. Thereby, each OT-box

is based on a different T-box, i.e. each one depends on a different key, and is encoded with a different pair of functions O_1 and O_2. We now perform the DCA attack on each OT-box. The DCA attack is successful on almost all of the 10 000 OT-boxes with the exception of three. In all cases, the difference of means curves obtained when using the correct key hypotheses return a highest peak value of 0.25, 0.5, 0.75 or 1. The table below summarizes how many OT-boxes return each peak value for the correct key hypotheses.

Peak value for correct key	Nr. of OT-boxes
1	55
0.75	2804
0.5	7107
0.25	34

Fig. 9 compares the results obtained for all key candidates when analysing one particular OT-box. In this case, the correct key is key candidate 119, for which we obtain a peak value of 0.5. The peak values returned for all other key candidates are notably lower.

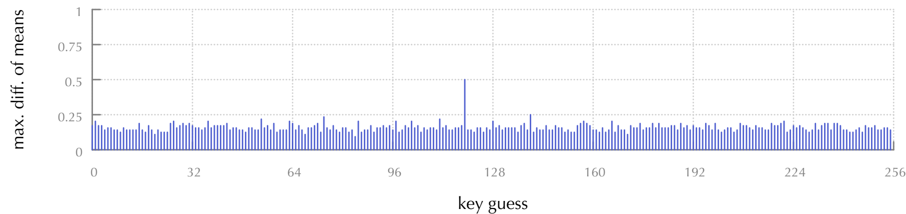


Fig. 9: Difference of means results for all key candidates when attacking one particular OT-box. Key guess 119 is the correct one.

Fig. 10 summarizes the results obtained when analysing one of the three OT-boxes which could not be successfully attacked. In this case, the correct key is key candidate 191. The peak corresponding to this candidate has a value of 0.25 and is not the highest peak obtained. For instance, the peak for key candidate 89 has a value of 0.28. Therefore, our DCA ranks key candidate 89 as the correct one. Similarly, when analysing the other OT-boxes which could not be successfully attacked, the peaks obtained for the correct key hypotheses have a value of 0.25 and there exists at least one other key candidate with a peak value slightly higher or with the same value of 0.25.

To illustrate how this effect is shown on complete white-box implementations, we now perform the DCA attack on our table-based white-box implementation of AES which only makes use of non-linear nibble encodings. We collect 2000 software traces, which consist of the memory addresses accessed during the encryption process. Fig. 11 shows the difference of means trace obtained when using the correct key byte with our selection function.

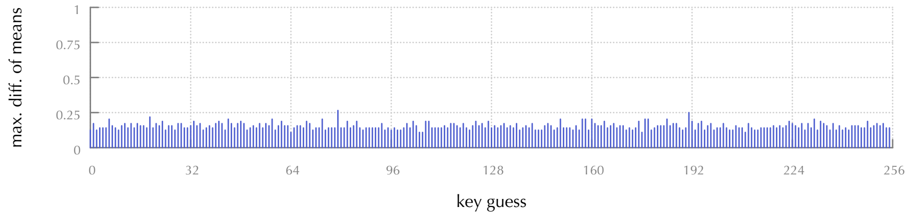


Fig. 10: Difference of means results for all key candidates when attacking one particular OT-box. Key guess 191 is the correct one.

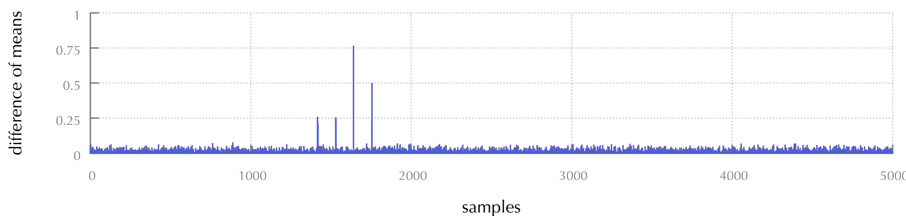


Fig. 11: DCA results for our white-box implementation with non-linear encodings

Fig. 11 is flat with one peak with a value very close to 0.75 (see sample 1640), another peak with a value very close to 0.5 (see sample 1750). Additionally, the value of two peaks is very close to 0.25. This result corresponds to the difference of means results obtained with our OT-box examples and to Theorem 2. Based on the results shown in this section we can conclude that randomly generated nibble encodings do not effectively work as a countermeasure for hiding correlations between a target bit and a selection function when performing the difference of means test. When using the correct key hypothesis, we do not always have perfect correlations such as those shown in Fig. 3, but the correlations are still high enough in order to allow a key extraction. Moreover, we learn one way to increase our success probabilities when performing the DCA attack. Namely, when ranking the key hypotheses, we could start by considering only the keys for which we obtain a difference of means curve with values very close to those described in Theorem 2. After that, we could rank our key hypotheses according to the height of these values.

4.3 Combination of Linear and Non-Linear Encodings

We now discuss the effectiveness of the DCA when performed on white-box implementations that make use of both linear and non-linear encodings to protect their key-dependent look-up tables. The combination of both encodings is the approach proposed by Chow et. al. in order to protect the content of the look-up tables from reverse engineering attempts. The output of each key-dependent look-up table, such as a T-box, is first encoded by a linear transformation and

afterwards by the combination of two non-linear functions as shown in Fig. 12. In the following, we refer to lOT-boxes as T-boxes that are encoded using both types of encodings.

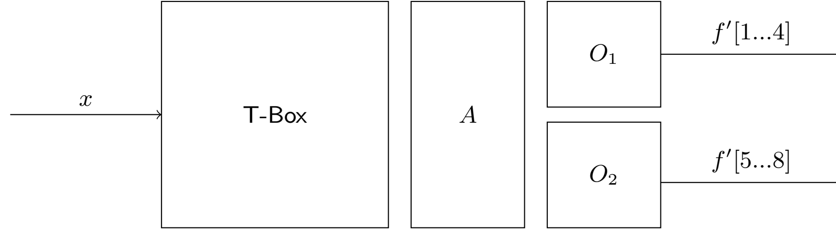


Fig. 12: Linear and non-linear encodings of the T-Box outputs

We now explain why the DCA attack can be successful even in the presence of both types of encodings. In this case, the success of the DCA attack depends on the matrix performing the linear transformation, which returns linearly encoded values m. The values m are split into two nibbles $m_1 = m[1, ..., 4]$ and $m_2 = m[5, ...8]$, to afterwards be encoded by the non-linear functions O_1 and O_2 respectively. The values of m_1 are obtained from the linear combinations of the first 4 bits of the columns of A. Analogously, the values of m_2 are obtained from the linear combinations of the last 4 bits of the columns of A. In the following, we refer to Sub_d as the upper (or lower) part of the columns of A, with $d \in \{1, 2\}$. For instance, when considering the matrix A described in 5 in Sec. 4.1, its corresponding Sub_1 looks as follows

$$Sub_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \quad (7)$$

In the following theorem, we derive a condition of a Sub_d matrix, such that our DCA returns values equal to those mentioned in Theorem 2 when we use the correct key hypothesis. As for the OT-boxes, the reason for this lies on how the encoded nibble values f_d are grouped in the sets A_b.

Theorem 3. *Given an lOT-box which makes use of a matrix A and nibble encodings. The difference of means curve obtained for the correct key hypothesis k^h consist only of values equal to 0, 0.25, 0.5, 0.75 or 1, if A has at least one Sub_d with one column $\vec{a}_j$, such that $rank[Sub_d \setminus \vec{a}_j] = 3$.*

Proof. Let $\vec{a}_j \in GF(2^4)$ be the columns of Sub_d, where j denotes the column number. We obtain each encoded nibble m_d via

$$m_d = \sum_j \vec{a}_j \cdot y[j] = \sum_{j:y[j]=1} \vec{a}_j. \quad (8)$$

Suppose that the value of $\vec{a}_j$ is not contained in the space spanned by the other columns of Sub_d. If bit $y[j] = 0$, then it follows from Equation 8 that the value m_i is obtained as a linear combination of at most 7 vectors with rank 3. These combinations do not span $GF(2^4)$, but only a 3-dimensional subspace $S \subseteq GF(2^4)$, which consists of exactly 8 different 4-bit values. If $y[j] = 1$, the value of m_i is obtained as a linear combination that includes the value of $\vec{a}_j$. These combinations thus span an affine subspace $S' := \{\vec{a}_j + \vec{s} | \vec{s} \in S\}$, i.e. $S \cup S' = GF(2^4)$ and $S \cap S' = \emptyset$.

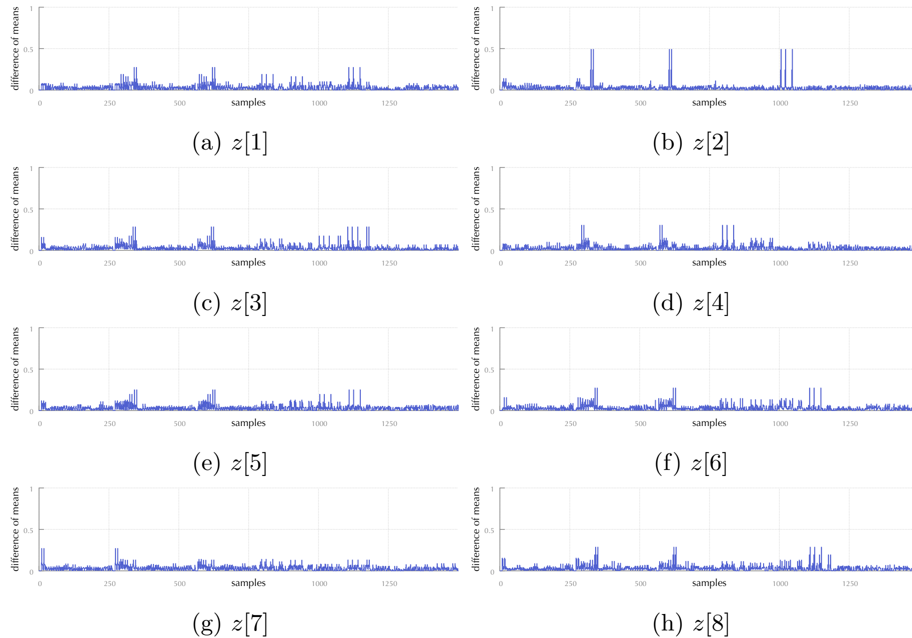
We now prove that when we use the correct key hypothesis, our sets A_b consist each of eaxtly 8 different encoded nibbles, whereby each nibble is repeated a total of 16 times. Let k^h be the correct key hypothesis and let bit $y_d[j]$ be our target bit, with $d \in \{1, 2\}$ referring to the nibble of y where bit j is located. With our selection function $\texttt{Sel}(p_e, k^h, j)$ we calculate a total of 128 values for which $y_d[j] = 0$. We know that nibble m_i is always one of the 8 possible values in S. Analogously, we know that for each time $y_d[j] = 1$, $m_i \in S'$. Therefore, our set A_0 consists only of encoded nibble values from a set $F := \{O_i(m_i) | m_i \in S\}$. Our set A_1 consists only of encoded nibbles from a set $F' := \{O_i(m_i) | m_i \in S'\}$. Since O_i is a bijective function, it follows that $F \cap F' = \emptyset$, i.e. all elements in A_0 differ from the elements in A_1. Moreover, each set A_b consists of 8 different nibble values, whereby each nibble is repeated a total of 16 times. Analogous to the proof of Theorem 2, it follows that the difference of means curve calculated for $\texttt{Sel}(p_e, k^h, j)$ consists only of the values 0, 0.25, 0.5, 0.75 or 1. □

As for the case with the OT-boxes, a key candidate corresponding to correlation values of 0.5, 0.75 and 1 is very likely to be ranked as the best key in the DCA. We now discuss what happens in the DCA attack if the pre-condition described in Theorem 3 is not satisfied. If for Sub_1 and Sub_2 of matrix A, the value of any column vector $\vec{a}_j$ *is* contained in the space spanned by the other columns, we see from Equation 8 that even when $y[j] = 0$, the value of m_i is still obtained as a linear combination of 7 vectors with rank 4. Thereby, m_i can be any value in $GF(2^4)$. The sets A_b can contain thus any 4-bit long value. As explained in Sec. 4.2, the average of both sets are more or less equal and thus, their difference converges to a value close to 0.

We now perform the DCA attack on the OpenWhiteBox challenge by Chow.[6] This AES implementation was designed based on the work described in [8] and [15]. To perform the attack, we collect 2000 software traces. These software traces consist of values read *and* written to the stack during the first round. We define our selection function the same way as in Sec. 3, $\texttt{Sel}(p_e, k^h, j) = z[j]$. Note that based on Theorem 3, the correlations between our selection function and a correct key hypothesis may be shown depending on *which* target bit $z[j]$

[6] https://github.com/OpenWhiteBox/AES/tree/master/constructions/chow

we use. When we use the correct key byte `0x69` with our selection function we obtain the difference of means traces shown in Fig. 13 for each target bit.



(a) $z[1]$ (b) $z[2]$ (c) $z[3]$ (d) $z[4]$ (e) $z[5]$ (f) $z[6]$ (g) $z[7]$ (h) $z[8]$

Fig. 13: Difference of means results for the OpenWhiteBox Challenge for bits $z[1]$ to $z[8]$

Not all difference of means traces show significant peaks that reveal correlations between our selection function $sel(p_e, \texttt{0x69}, j)$ and the targeted bit $z[j]$. Fig. 13b nevertheless, shows a flat trace with 7 peaks reaching a value of almost 0.5 (see samples 327, 335, 607, 615, 1007, 1023, 1047). Due to this trace, the key byte `0x69` is ranked as the best key candidate and the DCA attack is successful on this implementation. The peak values shown in Fig. 13b correspond to those described in Theorem 3. We can conclude that the matrix used for performing the linear transformation of the state byte z, fulfils the condition described in Theorem 3.

5 Generalized DCA

Traditionally, the DCA has been described as a software counterpart of the differential power analysis (DPA). The key candidates are ranked according to the results obtained from a difference of means test. Thereby, the difference of means graph showing the highest peak corresponds to the highest ranked key. In this paper we review how this approach also works for attacking many white-box

implementations. Nevertheless, the results shown in Sec. 4.2 show that we can increase our success rate by modifying our method for ranking the key candidates. Namely, if none of our key candidates returns a difference of means curve with a peak value significantly high, we can rank the key candidates according to the convergence of their peaks to the values 0.25 or 0. Thereby, we expect that by increasing the number of traces used for the analysis, the peak values for the correct key guess get closer to 0.25 or 0. In the following, we refer to the steps performed in Sec. 3.

1. Collecting Traces We perform this step the same way as described in Sec. 3. We remark that when attacking a complete white-box implementation, we need to generate a large enough number of traces in order to increase our success probability.

Steps 2 to 5. For each key candidate k^h and for each target bit $z[j]$, we perform Steps 2 to 5 as described in Sec.3.

6. Best target bit ranking In this step we compare the difference of means traces obtained for all target bits j for a given key hypothesis k^h. If there is a trace with a peak value high enough that it stands out among the other traces, then we select that as the best one for the key candidate we are analysing. Otherwise, we look for a trace with values converging to 0.25 (or 0) and select that one for the key candidate we are analysing.

This means that we perform Step 6 as in Sec. 3 but make a revision of the value $H(\Delta^j)$ before assigning Δ^j to Δ^h:

- If $H(\Delta^j) > 0.3$, then we select Δ^j such that, $\Delta^h := \Delta^j$.
- If $0.2 \leq H(\Delta^j) \leq 0.3$, then we look for a trace with values converging to 0.25. The trace $\Delta^{j'}$ with the peak values closest to 0.25 is thus selected such that $\Delta^h := \Delta^{j'}$.
- Otherwise if $H(\Delta^j) < 0.2$, then we look for a trace with values converging to 0. The trace $\Delta^{j'}$ with the peak values closest to 0 is thus selected such that $\Delta^h := \Delta^{j'}$.

7 Best Key Byte Hypothesis Analogous to the previous step, we perform Step 7 the same way as in Sec. 3, but if no key candidate stands out, we look for a candidate with the values closest to 0.25 or 0.

- If $H(\Delta^h) > 0.3$, then we select the key hypothesis corresponding to Δ^h as our best key candidate.
- If $0.2 \leq H(\Delta^h) \leq 0.3$, then we look for a trace with values converging to 0.25. The key hypothesis corresponding to the trace $\Delta^{h'}$ with the peak values closest to 0.25 is thus selected as our best key candidate.
- Otherwise if $H(\Delta^h) < 0.2$, then we look for a trace with values converging to 0. The key hypothesis corresponding to the trace $\Delta^{h'}$ with the peak values closest to 0 is thus selected as our best key candidate.

6 Conclusions

In this paper we observe how internal encodings as suggested by Chow et. al. do not effectively hide information regarding the outputs of a key dependent look-up table. Therefore, the use of such encodings makes a white-box implementation very vulnerable against the DCA attack. Our experiments performed with T-boxes encoded only via linear transformations show us that this method could provide a good masking for the output bits of a look-up table as long as the matrix used for performing the linear transformation does not have an identity row (see Theorem 1). However as explained, the DCA can easily be modified in order to be successful even in the presence of such matrices. Our selection function only needs to calculate all possible linear combinations of the bits in the calculated state z. Thereby, at least one linear combination will be equal to the linear combination defind by one of the matrix rows. We will thus calculate one bit of the encoded output state correctly when using the correct key guess and this will lead us to a successful DCA attack.

We prove that the DCA attack performed on nibble encoded look-up tables always returns a difference of means curve with values that converge to 0, 0.25, 0.5, 0.75 or 1 when using the correct key hypothesis (see Theorem 2). Therefore nibble encodings alone are vulnerable against a DCA attack. These observations help us understand why the DCA attack is successful even in the cases that intermediate results are encoded using both linear and non-linear nibble encodings. We prove that, if the upper or lower part of a matrix (denoted as Sub_d) performing the linear transformation of a look-up table output has at least one column $\vec{a}$ such that its column rank is $rank[Sub_d \setminus \vec{a}_j] = 3$, the DCA returns a difference of means curve with the values 0, 0.25, 0.5, 0.75 or 1 for the correct key guess. This leads us to a successful DCA attack as peak values of 0.5, 0.75 or 1 are very likely to be ranked as the best key candidates. These results help us improve our success rate when performing a DCA attack. Namely, when ranking the key candidates, if no candidate has a difference of means curve with significantly high peaks, we can rank the key candidates according to the convergence of their difference of means peaks to the values 0.25 or 0 (see Sec. 5).

This paper helps understand the reasons why the DCA was successful on many white-box implementations analysed in [6] and helps as a step towards more secure white-box designs in the light of the DCA attack. Using non-linear nibble encodings is not sufficient, but one could consider using linear and non-linear byte encodings only on the look-up tables that are accessed during the first and the last rounds of an implemented cipher. Namely, only the outputs of these look-up leak information useful for performing a DCA and, if they are encoded via non-linear byte encodings, we efficiently hide correlations between the look-up table outputs and the secret key (see Sec. 4.2). The rest of the look-up tables in the implementation could be protected as proposed by Chow et. al., as far as the DCA attack is concerned, with a combination of linear and non-linear nibble encodings. In this case, our implementation would still remain with a considerable code size.

Acknowledgments

The authors would like to acknowledge the contribution of the COST Action IC1306. Chris Brzuska is grateful to NXP for supporting his chair for IT Security Analysis.

References

1. S. Banik, A. Bogdanov, T. Isobe, and M. Jepsen. Analysis of software countermeasures for whitebox encryption. volume 2017, 2017.
2. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, Aug. 2001.
3. J.-B. Bdrune. Hack.lu 2009 reverse challenge 1. 2009.
4. O. Billet, H. Gilbert, and C. Ech-Chatbi. Cryptanalysis of a white box AES implementation. In H. Handschuh and A. Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 227–240. Springer, Heidelberg, Aug. 2004.
5. A. Bogdanov, T. Isobe, and E. Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 126–158. Springer, Heidelberg, Dec. 2016.
6. J. W. Bos, C. Hubain, W. Michiels, and P. Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In B. Gierlichs and A. Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 215–236. Springer, Heidelberg, Aug. 2016.
7. J. Bringer, H. Chabanne, and E. Dottax. White box cryptography: Another attempt. Cryptology ePrint Archive, Report 2006/468, 2006. `http://eprint.iacr.org/2006/468`.
8. S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. White-box cryptography and an AES implementation. In K. Nyberg and H. M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, Heidelberg, Aug. 2003.
9. S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. A white-box DES implementation for DRM applications. In J. Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002*, volume 2696 of *LNCS*, pages 1–15. Springer, 2003.
10. M. Karroumi. Protecting white-box AES with dual ciphers. In K. H. Rhee and D. Nyang, editors, *ICISC 10*, volume 6829 of *LNCS*, pages 278–291. Springer, Heidelberg, Dec. 2011.
11. P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, pages 5–27, 2011.
12. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.
13. T. Lepoint, M. Rivain, Y. D. Mulder, P. Roelse, and B. Preneel. Two attacks on a white-box AES implementation. In T. Lange, K. Lauter, and P. Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 265–285. Springer, Heidelberg, Aug. 2014.

14. T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Investigations of power analysis attacks on smartcards. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 17–17, Berkeley, CA, USA, 1999. USENIX Association.
15. J. A. Muir. A tutorial on white-box aes. *IACR Cryptology ePrint Archive*, 2013:104, 2013.
16. Y. D. Mulder, P. Roelse, and B. Preneel. Cryptanalysis of the Xiao-Lai white-box AES implementation. In L. R. Knudsen and H. Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 34–49. Springer, Heidelberg, Aug. 2013.
17. Y. D. Mulder, B. Wyseur, and B. Preneel. Cryptanalysis of a perturbated white-box AES implementation. In G. Gong and K. C. Gupta, editors, *INDOCRYPT 2010*, volume 6498 of *LNCS*, pages 292–310. Springer, Heidelberg, Dec. 2010.
18. E. Sanfelix, J. de Haas, and C. Mune. Unboxing the white-box: Practical attacks against obfuscated ciphers. Presentation at BlackHat Europe 2015, 2015. `https://www.blackhat.com/eu-15/briefings.html`.
19. P. Sasdrich, A. Moradi, and T. Güneysu. *White-Box Cryptography in the Gray Box*, pages 185–203. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.
20. Smart Card Alliance Mobile and NFC Council. Host card emulation 101. white paper, 2014. `http://www.smartcardalliance.org/downloads/HCE-101-WP-FINAL-081114-clean.pdf`.
21. U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology. Data encryption standard (des). Federal information processing standards publication (fips pub) 46-3, 10 1999.
22. U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology. Announcing the advanced encryption standard (aes). Federal information processing standards publication (fips pub) 197, 11 2001.
23. Y. Xiao and X. Lai. A secure implementation of white-box AES. In *Computer Science and its Applications, 2009. CSA '09. 2nd International Conference on*, pages 1–6, 2009.