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

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

[1] Hamburg University of Technology, Hamburg, Germany
{estuardo.alpirezbock,brzuska,alexander.treff}@tuhh.de
[2] Aalto University, Espoo, Finland
[3] NXP Semiconductors, Eindhoven, The Netherlands
wil.michiels@nxp.com
[4] Technische Universiteit Eindhoven, Eindhoven, The Netherlands

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. Bos et al. (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 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 introduced by Chow et al. (SAC 2002). Thereby, a combination of linear and non-liner nibble encodings is used to protect the secret key. In this paper we analyse the use of such internal encodings and prove rigorously that they are too weak to protect against DCA. We prove that the use of non-linear nibble encodings does not hide key dependent correlations, such that a DCA attack succeeds with high probability.

Keywords: White-box cryptography
Differential computational analysis · Software execution traces
Mixing bijections

1 Introduction

When an application for mobile payment runs in software on Android 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

© Springer International Publishing AG, part of Springer Nature 2018
B. Preneel and F. Vercauteren (Eds.): ACNS 2018, LNCS 10892, pp. 103–120, 2018.
https://doi.org/10.1007/978-3-319-93387-0_6

be secured against adversaries who have a high degree of control over the environment. In 2002, Chow et al. [9,10] introduced *white-box cryptography*, which aims at remaining secure even when the adversary has full control over the execution environment. As mobile payment became widely used and as its security nowadays often relies on software security only, Visa and Mastercard made the use of white-box cryptography for mobile payment applications mandatory [15].

A necessary requirement for secure white-box cryptography is that an adversary cannot extract the embedded secret key from the implementation. However, hiding the secret key is not always enough to achieve security in the white-box attack scenario. For example, if a mobile payment application uses a secret key for authentication by encrypting a challenge, then an adversary may simply try to copy the white-box program performing the encryption and run it on another device. The adversary could successfully use the functionality of the white-box program without knowing the value of its embedded secret key.

While it seems clear that a white-box program needs to achieve more than just security against key extraction, hiding the secret key remains a difficult task to achieve for real-life applications. Chow et al. [9,10] 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. To achieve robustness 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., [7,11]. While the LUT-based white-box designs only store the keys obfuscated in lookup tables, all aforementioned LUT-based designs turn out to be susceptible to key extraction attacks performed via differential and algebraic cryptanalysis (see [4,14,16,17]). 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. Thus, Bos et al. [6] and Sanfelix et al. [20] introduced *automated* key extraction attacks that are substantially simpler and faster to carry out. The authors call their method differential computational analysis (DCA) and describe it as the software counterpart of the differential power analysis (DPA), a method for attacking cryptographic hardware implementations [13]. 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 most 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 [2].

In this paper, we analyze why step (3) of the attack by Bos et al. [6] actually works and show which types of encodings are susceptible to the DCA attack. The work of Sasdrich et al. [21] 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.

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 [12] and Messerges [18] 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. Here, we focus on the encodings suggested by Chow et al. [9,10], which are a combination of linear and non-linear transformations. We start by studying the effects of a single linear transformation. We show that the DCA attack can successfully extract the key from a look-up table when it only uses linear or affine encodings. Next, we consider the effect of non-linear nibble encodings and prove that the use of nibble encodings 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 as we discuss shortly in the end of the paper and in more detail in the extended version [1]. 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, the experimental results align with the theoretical observations.

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 [3] and the hope is that achieving perfect security or at least a good level of 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 may find the work by Muir [19] a useful read for a more detailed description on how to construct an LUT-based white-box AES implementation. In the following, we introduce the concept of 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. [9,10]. We now discuss two types of encodings.

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. 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 requires a storage of 2^8 bytes. Although this may still be acceptable, the problem arises when two values with a byte encoding need to be XORed. An encoded XOR has a storage requirement of 2^{16} nibbles. As we need many of them, this becomes an issue. Therefore, one usually splits longer values in nibbles of 4 bits. When XORing those, we only need a lookup table of 2^8 nibbles. 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 quadratic in the input size and thus, they can be implemented on larger words.

Linear Encodings. Chow et al. suggest 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-operation, 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 on white-box implementations, which aims at finding key dependent correlations by analysing memory access information recorded during the encryption process. To display the tracked memory-information in so called *software execution traces*, 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. For more details on the acquisition of software traces, see the original DCA paper by Bos et al. [6]. In this section we provide a detailed description of one statistical method to analyse such software execution traces, namely the *difference of means* method. Note that this method corresponds 1-to-1 to the difference of means method as presented by Kocher using power traces [12]. Nevertheless we now show the results obtained from a difference of means analysis when performed using a group of software traces. 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[1] values given to the program as input.

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

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. The analysis steps on a DCA attack are performed as follows.

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. Figure 1 shows a single software trace consisting of 300 samples. Each sample corresponds to one bit of the memory addresses accessed during execution.

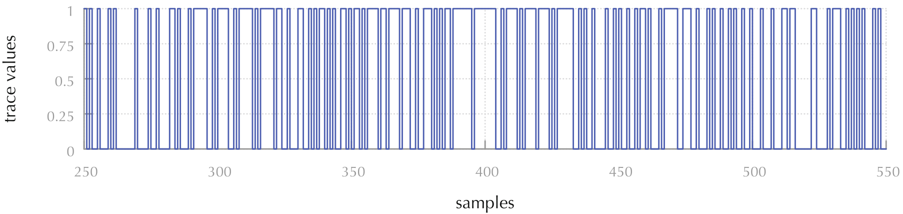

Fig. 1. Single software trace consisting of 300 samples

2. Selection Function: We define a selection function for calculating an intermediate state-byte z 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. 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 $1 \leq j \leq 8$.

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

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

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

5. Difference of Means: We now calculate the difference between the two previously obtained mean traces sample wise. Figure 2 shows the resulting difference of means trace:

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

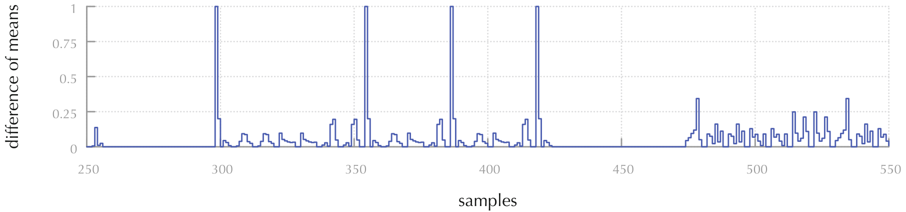

Fig. 2. Difference of means trace for correct key guess

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 is 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. 2. This difference of means trace corresponds to the analysis performed on a white-box implementation obtained from the hack.lu challenge [8]. 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. 2 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. 3 shows a difference of means trace obtained for an incorrect key-hypothesis. No sample has a value higher than 0.3.

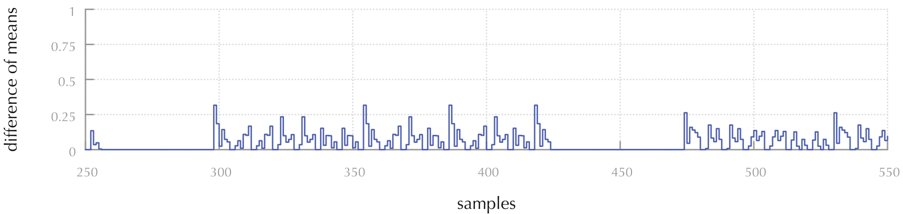

Fig. 3. Difference of means trace for incorrect key guess

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.

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 a single 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

Chow et al. [9] recommend a combination of linear and non-linear encodings 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 publicly available white-box implementations. In this section we analyse how these types of encodings affect the effectiveness of the DCA attack. Namely, if intermediate values in an implementation are encoded, it becomes more difficult to re-calculate such values using a selection function as defined in Step 2 of the DCA, as this selection function does not consider any encodings (see Sect. 3). 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. 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. Note that Chow et al. merge the T-box and the MixColumns operation into one 8-to-32 bit look-up table and encode the look-up table output via a 32-bit linear transformation. However, an 8-to-32 bit look-up table can be split into four 8-to-8 bit lookup tables, which correspond to the look-up tables used for our analyses.[2]

Following our definition for a successful DCA attack on an n-to-n look-up table given in Sect. 3, 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 Sect. 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 a known plaintext and a key guess. As we will see in this section, internal encodings as suggested by

[2] It can be the case that the four lookup tables are, in isolation, not bijective. In that case, our results do not apply directly. It is left as an exercise to adapt them to this setting.

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 A. For each output y of a 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. Figure 4 displays this behaviour.

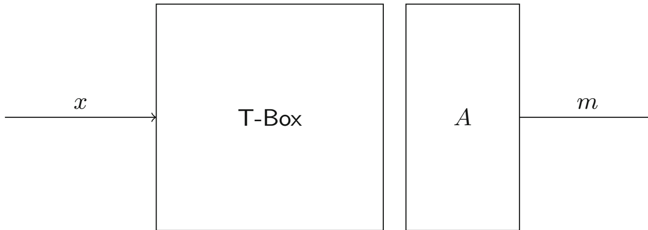

Fig. 4. An lT-box maps each input x to a linearly encoded output m.

We now compute the DCA on the outputs of an lT, constructed with a randomly generated invertible matrix A. Figure 5 shows the 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 24 samples. No high peaks can be seen in the difference of means trace, i.e., 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.

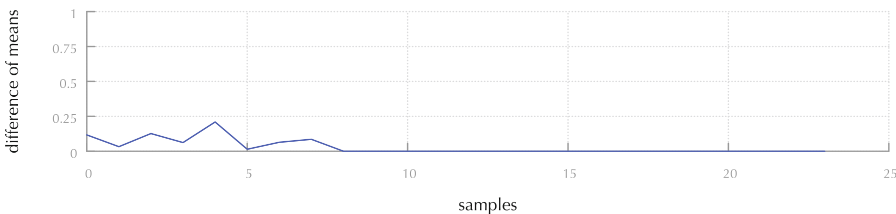

Fig. 5. Difference of means trace for the lT-box

The results shown in Fig. 5 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 Sect. 3 is not effective in the presence of this particular transformation. The theorem below

gives a necessary and sufficient condition under which linear transformations provide protection against the DCA attack.

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

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

$$m[i] = \sum_j a_{ij} \cdot y[j] = \sum_{j:a_{ij}=1} y[j]. \tag{5}$$

Suppose that row i of A has $HW(i) = 1$. Let j be such that $a_{ij} = 1$. It follows from Eq. (5) 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 equal 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 Eq. (5) 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 $m[i] = 1$ and the difference between the averages of both sets is equal to zero. $\square$

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. 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 $\texttt{Sel}(p_e, k^h, j) = z[j]$. Figure 6 shows the difference of means trace obtained for the correct key guess.

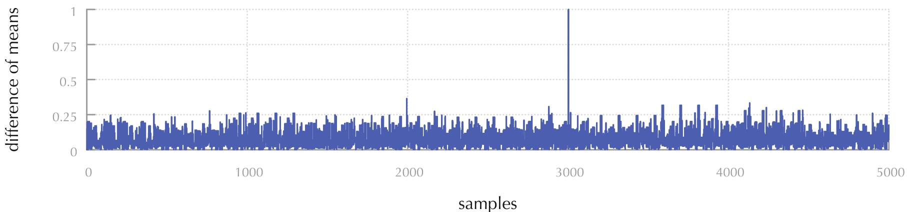

Fig. 6. DCA results for our white-box implementation with linear encodings

Figure 6 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 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* [10]. 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 Sect. 4.1 of [19] 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. Figure 7 displays the behaviour of an OT-box constructed using two nibble encodings.

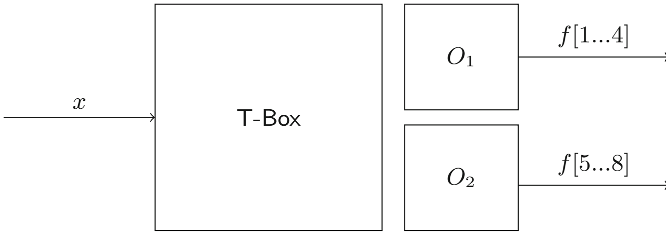

Fig. 7. Non-linear encodings of the T-Box outputs

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:

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

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. □

A peak value of 0.5, 0.75 or 1 is high enough to ensure that its corresponding key candidate will be ranked as the correct one. We now argue that, when we use an incorrect key guess, nibbles with the same value may be grouped in different sets. 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 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. 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 three OT-boxes which cannot be successfully attacked return peak values of 0.25 for the correct key guess. For each of the three cases, the correct key guess is not ranked as the best key candidate because there exists at least one other key candidate with a peak value slightly higher or with the same value of 0.25. 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

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. Figure 8 shows the difference of means trace obtained when using the correct key byte with our selection function.

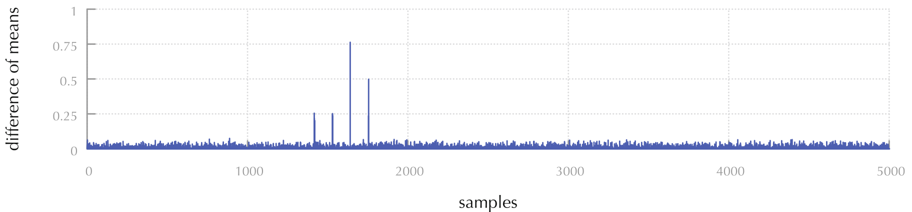

Fig. 8. DCA results for our white-box implementation with non-linear encodings

Figure 8 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 key dependent correlations when performing the difference of means test. Additionally, we learn one way to increase our success probabilities when performing the DCA: when

ranking the key hypotheses, if no key candidate returns a peak value which really stands out (0.5 or higher), we could rank our key candidates according to the convergence of their peaks to the values 0.25 or 0. In the extended version of this paper, we describe this generalization of the DCA attack in more detail [1].

4.3 Combination of Linear and Non-linear Encodings

We now discuss shortly 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. For a more detailed description of the effect of this type of encodings, we refer the reader to the extended version of this paper [1]. 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.

We now perform the DCA attack on the OpenWhiteBox challenge by Chow.[3] This AES implementation was designed based on the work described in [9,19]. We collect 2000 software traces which consist of values read *and* written to the stack during the first round. We define our selection function the same way as in Sect. 3, $\texttt{Sel}(p_e, k^h, j) = z[j]$. For the correct key byte `0x69` we obtain the difference of means trace shown in Fig. 9.

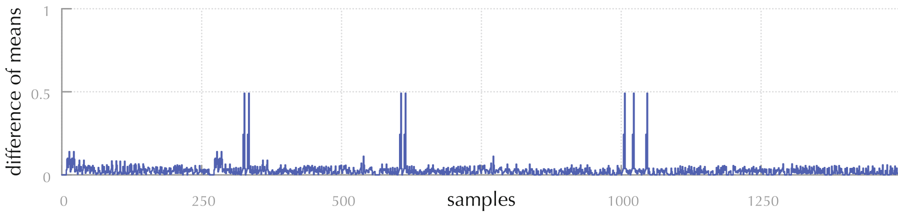

Fig. 9. Difference of means results for the OpenWhiteBox Challenge when using the correct key guess and targeting bit $z[2]$

Figure 9 shows a flat trace with 7 peaks reaching a value of almost 0.5 (see e.g. sample 327). Due to this trace, the key byte `0x69` is ranked as the best key candidate and the DCA attack is successful. The peak values shown in Fig. 9 correspond to those described in Theorem 2. We discuss these results shortly based on Theorems 1 and 2. From Theorem 1 we can conclude that, when considering an output bit of a T-box, it is important that all bits can still be transformed in all possible values (i.e., 0 and 1) for achieving resistance against the DCA. When a white-box uses the combination of linear and non-linear encodings and we target an output bit, we need to consider the output of a T-box as two individual

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

nibbles. Thereby, it is important that each nibble can be transformed into each possible value in $GF(2^4)$. If that is the case, we can avoid correlation values such as those mentioned in Theorem 2 caused by the use of non-linear nibble encodings.

5 Conclusions

As automated attacks on white-box implementations become more popular, it is important to understand the experimental success of the original DCA attack in order to aim for resistance against such attacks. 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 DCA. In this work we focused on analysing these types of encodings due to their popularity amongst the white-box community and hope that our results motivate the further research on efficient alternatives for internal encodings in white-box cryptographic designs.

Acknowledgments. The authors would like to thank the anonymous referee for his/her helpful comments. 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. Alpirez Bock, E., Brzuska, C., Michiels, W., Treff, A.: On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography (2018). https://eprint.iacr.org/2018/301
2. Banik, S., Bogdanov, A., Isobe, T., Jepsen, M.: Analysis of software countermeasures for whitebox encryption. IACR Trans. Symmetric Cryptol. **2017**(1), 307–328 (2017)
3. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_1
4. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white box AES implementation. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 227–240. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30564-4_16
5. Bogdanov, A., Isobe, T., Tischhauser, E.: Towards practical whitebox cryptography: optimizing efficiency and space hardness. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 126–158. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_5
6. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: hiding your white-box designs is not enough. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 215–236. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_11

7. Bringer, J., Chabanne, H., Dottax, E.: White box cryptography: another attempt. Cryptology ePrint Archive, Report 2006/468 (2006). http://eprint.iacr.org/2006/468
8. Bédrune, J.-B.: Hack.lu 2009 reverse challenge 1 (2009). https://2017.hack.lu/
9. Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 250–270. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36492-7_17
10. Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: Feigenbaum, J. (ed.) DRM 2002. LNCS, vol. 2696, pp. 1–15. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-44993-5_1
11. Karroumi, M.: Protecting white-box AES with dual ciphers. In: Rhee, K.-H., Nyang, D.H. (eds.) ICISC 2010. LNCS, vol. 6829, pp. 278–291. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24209-0_19
12. Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**, 5–27 (2011)
13. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
14. Lepoint, T., Rivain, M., De Mulder, Y., Roelse, P., Preneel, B.: Two attacks on a white-box AES implementation. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 265–285. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_14
15. Mastercard Mobile Payment SDK: Security guide for MP SDK v1.0.6. White paper (2017). https://developer.mastercard.com/media/32/b3/b6a8b4134e50bfe53590c128085e/mastercard-mobile-payment-sdk-security-guide-v2.0.pdf
16. De Mulder, Y., Roelse, P., Preneel, B.: Cryptanalysis of the Xiao – Lai white-box AES implementation. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 34–49. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_3
17. De Mulder, Y., Wyseur, B., Preneel, B.: Cryptanalysis of a perturbated white-box AES implementation. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 292–310. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17401-8_21
18. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Investigations of power analysis attacks on smartcards. In: Proceedings of the USENIX Workshop on Smartcard Technology, WOST 1999, Berkeley, CA, USA, p. 17. USENIX Association (1999)
19. Muir, J.A.: A tutorial on white-box AES (2013). https://eprint.iacr.org/2013/104.pdf
20. Sanfelix, E., de Haas, J., Mune, C.: Unboxing the white-box: practical attacks against obfuscated ciphers. In: Presentation at BlackHat Europe 2015 (2015). https://www.blackhat.com/eu-15/briefings.html
21. Sasdrich, P., Moradi, A., Güneysu, T.: White-box cryptography in the gray box. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 185–203. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_10