Revisiting Higher-Order Computational Attacks against White-Box Implementations – Extended version –

Houssem Maghrebi and Davide Alessio

UL Identity Management & Security, France
firstname.lastname@ul.com

Abstract. White-box cryptography was first introduced by Chow *et al.* in 2002 as a software technique for implementing cryptographic algorithms in a secure way that protects secret keys in an untrusted environment. Ever since, Chow *et al.*'s design has been subject to the well-known Differential Computation Analysis (DCA). To resist DCA, a natural approach that white-box designers investigated is to apply the common side-channel countermeasures such as masking. In this paper, we suggest applying the well-studied leakage detection methods to assess the security of masked white-box implementations. Then, we extend some well-known side-channel attacks (*i.e.* the bucketing computational analysis, the mutual information analysis, and the collision attack) to the higher-order case to defeat higher-order masked white-box implementations. To illustrate the effectiveness of these attacks, we perform a practical evaluation against a first-order masked white-box implementation. The obtained results have demonstrated the practicability of these attacks in a real-world scenario.

Keywords: white-box cryptography, masking, higher-order computational attacks, leakage detection, AES.

1 Introduction

White-box Implementations and Computational Attacks. In 2002, Chow *et al.* introduced the first white-box implementations of AES and DES block ciphers [5, 6]. The main idea behind was to embed the secret key in the implementation using a network of precomputed Look-Up Tables (LUTs) composed with some linear and non-linear random encodings to protect the intermediate states between the LUTs. The knowledge of one or all of these LUTs shall not give any information about the embedded secret key. To implement this design, two types of encodings shall be considered:

– Internal encodings: are non-linear bijections applied to the input and/or the output of each LUT to hide its entries and/or its outputs. This category encompasses the so-called *mixing bijections* which are linear transformation applied to the input and output of each LUT to add more confusion to the implementation and ensure the cryptographic diffusion property.

– External encodings: are bijective mappings applied to decode the plaintext from the sending process and to encode the resulting ciphertext to the receiving process.

To defeat white-box implementations, Bos *et al.* proposed in [4] the Differential Computational Analysis (DCA). This attack is the software adaptation of the well-known Differential Power Analysis (DPA) [10]. Specifically, the idea of the DCA consists in monitoring the memory addresses (as well as the stack, the CPU instructions, ...) accessed during the encryption process and recording them in the so-called *computation traces* (aka software execution traces). Then, a statistical analysis is performed to compute the correlation between a prediction of the targeted sensitive variable (that depends on a key guess) and each sample of the collected computation traces. The secret key corresponds to the key guess for which the highest correlation peak is obtained.

Since the publication of the DCA, several researchers have investigated the adaptation of either the well-studied side-channel attacks [14] or the algebraic cryptanalysis techniques [16] to perform computational attacks in the white-box context. For instance, authors in [14] proposed a software version of the collision attack and the Mutual Information Analysis (MIA). The experimental results performed on several publicly available white-box AES implementations have shown significant improvements in terms of trace complexity compared to the DCA. Recently, Zeyad *et al.* have suggested the Computational Bucketing Attack (BCA) in [16]. This attack is inherently inspired by a cryptanalysis technique named statistical bucketing attack. The authors have demonstrated that this attack is very efficient to defeat some sophisticated white-box AES implementations (*e.g.* the WhibOx 2016 contest) with a fixed amount of traces (precisely 1024 traces to break a white-box AES implementation).

Masking and Higher-Order Computational Attacks. Obviously, the well-studied side-channel countermeasures can be adapted and applied to protect white-box implementations. One common countermeasure is to apply masking which consists in sharing the intermediate variable into several mutually independent shares. In [3], Bogdanov *et al.* investigated the approach of applying higher-order masking to resist DCA attack. Furthermore, the authors introduced, for the first time, the extension of the DCA to the higher-order case and analyzed the security of the masking countermeasure against these attacks in the context of white-box implementation.

Our Contributions. Following the investigations done in [3], we propose in this work:

– **The use of leakage detection for white-box assessment:** We discuss in Sec. 3 how the side-channel leakage detection techniques can be applied to assess the leakage of white-box implementations. These techniques are of great interest from an adversary's perspective and from a security developer's perspective as well.

- **A higher-order BCA attack:** We extend in Sec. 4 the BCA attack to the second-order. Then, we demonstrate how this attack can defeat an internally encoded masked white-box implementation with exactly the same low trace complexity as for the first-order version studied in [16].
- **A higher-order MIA attack:** We suggest in Sec. 5 two fashions of applying higher-order MIA in the context of white-box implementation. Both approaches are compared through practical experiments in terms of key-recovery efficiency and performance.
- **A higher-order collision attack:** We study in Sec. 6 the higher-order version of collision attacks to defeat masked implementations. Then, we provide in Sec. 7 a comparison of the efficiency of the proposed attacks.

All our analyses are validated through practical experiments on the same first-order masked white-box AES implementation. Furthermore, we made the computation traces collected on this reference implementation publicly available [1]. The goal is twofold: (1) to ease the reproduction of our results by the white-box community and (2) to provide a commonly masked database (as no such traces from a masked implementation are available so far). In addition, the source code of some of our proposed attacks is publicly available as well.

2 Preliminaries and Study Framework

2.1 Notations and Definitions

Along this paper, we use the following notations. The bold block capitals $\mathbf{X}$ denote matrices. The i^{th} column vector of a matrix $\mathbf{X}$ is denoted by $\mathbf{X}[i]$. The random variables are denoted by uppercase Latin letters, like X, while the lowercase letter x denotes a particular realization of X. The entropy $\mathbb{H}[X]$ of a random variable X aims at measuring the amount of information provided by an observation of X.

The intersection of two sets of values A and B is denoted by $A \cap B$ and is defined as $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$. Two sets are disjoint if they have no elements in common, that is, A and B are disjoint if $A \cap B = \emptyset$.

To perform his attack, the adversary targets an intermediate sensitive variable which is a function of a plaintext x and a guessable secret key k^*. Then, for each key guess k, he computes a prediction of the target sensitive variable denoted $\phi(x, k)$ (or $\phi(k)$ for short) and measures the dependency between this prediction and the acquired traces. For our practical experiments, we target the Sbox output of the first round of an AES, *i.e.* $\phi(x, k) = \text{Sbox}(x \oplus k)$.

2.2 Assumptions

The analyses and conclusions drawn in this work are done under the three following assumptions:

Assumption 1 (Nibble-encoded naked white-box implementations) *The targeted white-box implementations are nibble-encoded using an internal encoding. No external encoding is applied.*

Remark 1. Nibble-encoding is the most common encoding size used for white-box AES implementations. We stress the fact that Assumption 1 is mandatory (according to [16]) for the study of the BCA described in Sec. 4. However, for the other investigated attack techniques in this work this assumption can be relaxed (*i.e.* the results can be generalized to any encoding size).

Assumption 2 (Similarity encoding) *The most significant (respectively the least significant) nibbles of the masks used to protect the sensitive variable are encoded with exactly the same encoding function applied on the most significant (respectively the least significant) nibbles of the masked sensitive variable.*

Assumption 3 (Perfect synchronization) *The targeted white-box implementations are only protected with higher-order masking. No shuffling or any random delays is introduced to de-synchronize the acquired computation traces.*

2.3 Targeted White-box Implementation

As introduced earlier, we study in this work the extension of some well-known side-channel attacks to the higher-order case when applied on masked white-box implementations. When a d^{th}-order masking is applied, each sensitive variable Z is split into $d+1$ shares $s_0, s_1, \ldots, s_d$ such that $s_0 \oplus s_1 \oplus \cdots \oplus s_d = Z$. Usually, the d shares $s_1, \ldots, s_d$ (called the masks) are randomly picked up and the last one s_0 (called the masked variable) is processed such that it satisfies the previous equality. Under assumption 3, we only focus on higher-order masking as a unique countermeasure applied to ensure protection.

For our experimental validation, we restrict the assessment of our proposed higher-order attacks against first-order masked white-box AES implementations. That is, we only evaluated the second-order versions. To do so, we implement a first-order white-box AES implementation under the three assumptions formulated in Sec. 2.2. This implementation is based on the classical Chow *et al.*'s white-box design based on an internal encoding but with the major difference that every Tbox input and output is protected with an independent random mask. Then, we collect the computation traces using an internal tool that monitors the read memory access during the execution of this first-order masked implementation. Each collected value during the acquisition phase is decomposed into several nibbles and then stored in the computation trace (Assumption 1). The collected traces are publicly available in [1] to ease the reproducibility of our results. This trace database will serve as a reference for evaluating our proposed attacks in practice.

To check that this reference masked implementation is well protected against first-order attacks (*i.e.* no obvious first-order leakage can be detected), we suggest in the following section a study of the well-known leakage detection methods in the white-box context.

3 Leakage Detection

3.1 Background

Leakage detection (*aka* leakage assessment) methods are commonly used in side-channel context to perform preliminary evaluations of the resistance of the targeted implementation. The Welchs t-test (*aka* Test Vector Leakage Assessment (TVLA)) is a popular method that consists (in its most popular form, *aka* non-specific test or fixed *vs.* random approach) in comparing the leakages of a cryptographic implementation with fixed plaintexts to the leakages of the same implementation with random plaintexts, both captured for the same fixed key [7]. Other methods exist such that the Signal-to-Noise Ratio (SNR) [11] and the Pearson's $\mathcal{X}^2$-test [12]. The main advantages of leakage detection are its simplicity, its efficiency (in time and data complexity) and its ability to be used with minimum implementation knowledge.

To the best of our knowledge, the leakage detection methods have never been applied in the white-box context despite their great interest from an adversary's perspective (*i.e.* to detect the leakiest time samples and hence to reduce the dimensionality of processed traces) and from a security developer's perspective (to detect vulnerabilities and hence to add the appropriate countermeasures). In this section, we emphasize the use of the leakage detection methods as a primordial step in the evaluation roadmap of a white-box implementation.

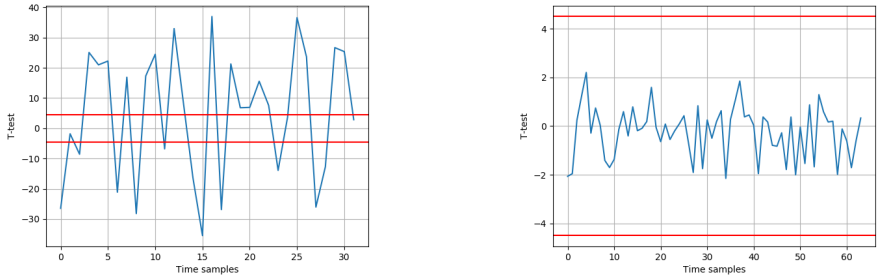
3.2 Simulation Results

For our simulation, we mainly focus on two leakage detection methods. Namely, we implement the Welchs t-test and the Pearson's $\mathcal{X}^2$-test. Then, we generate 1.000 simulated traces for the first-round SubBytes step of a first-order masked and an unprotected nibble-encoded white-box AES implementation[1]. That is, we obtain 500 simulated traces with a fixed plaintext and 500 simulated traces with random plaintexts on which we run the t-test and the Pearson's $\mathcal{X}^2$-test. The obtained results are plotted in Fig. 1 where the red horizontal curves represent the threshold above which the implementation is leaking information on the sensitive data.

As expected, when the implementation is masked, no first-order leakage was detected[2]. It is worth noting that higher-order versions of t-test and Pearson's $\mathcal{X}^2$-test exist [12, 15] and allow to detect higher-order leakage (*e.g.* to detect the second-order leakage of a masked implementation). These extended versions of leakage detection methods are not studied in this work for lack of room.

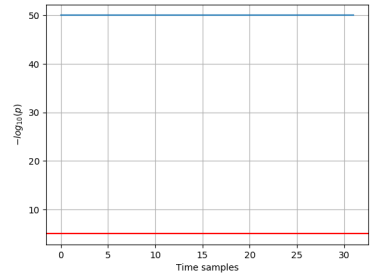
[1] The used scripts to generate the simulated traces are available in [1].

[2] For the $\mathcal{X}^2$-test, the obtained $-log_{10}(p)$ values for the unprotected implementation are equal to the infinity (since the simulated computation traces are unnoisy). For clarity reasons, we fixed the $-log_{10}(p)$ at 50 for this particular case.

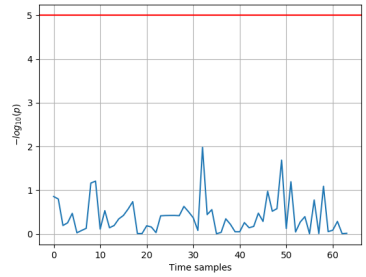


(a) T-test result.

(b) T-test result.



(c) $\mathcal{X}^2$-test result.



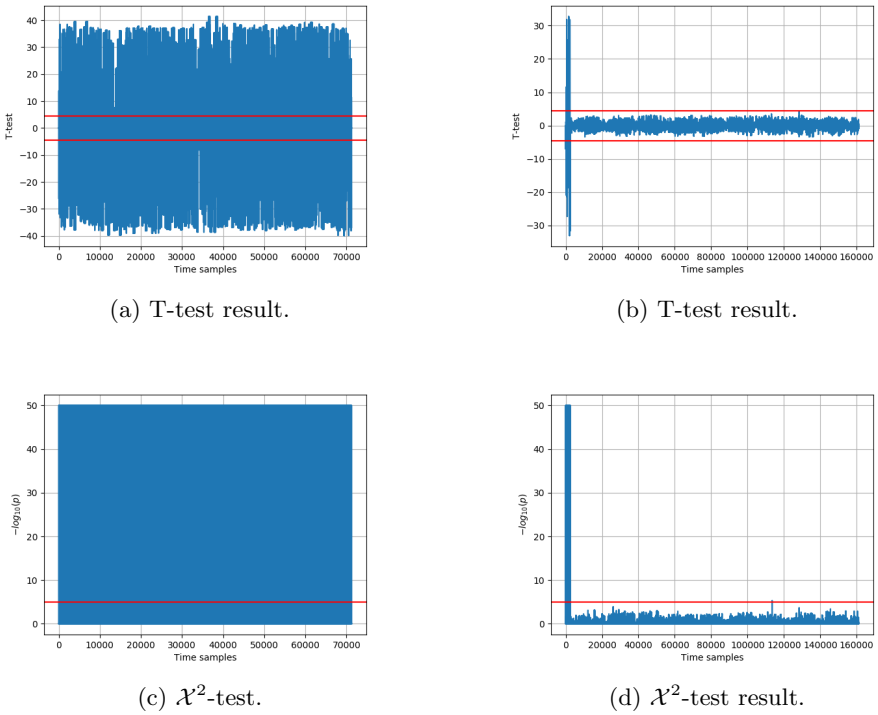
(d) $\mathcal{X}^2$-test result.

Fig. 1: Results of the fixed-vs-random t-test and $\mathcal{X}^2$-test on the simulated traces: unprotected (left-hand side) and masked (right-hand side).

3.3 Experimental Results

To validate the efficiency of the leakage detection methods in a real-world scenario. We target two white-box implementations. The first one is Chow's white-box AES implementation. The second one is our reference masked AES implementation described in Sec. 2.3. The used traces for this assessment are publicly available in [1]. The obtained are results are depicted in Fig. 2.

From Fig. 2, it is noticeable that the unprotected white-box leaks information on the sensitive data which is easily detected by both tests. Regarding our masked implementation, one can identify some leakage (for the t-test and the $\mathcal{X}^2$-test) at the very beginning of the AES execution. In fact, this leakage corresponds to the loading of the plaintexts. To confirm this claim, we run a first-DCA and a first-order BCA when targeting the area where this leakage is detected and both attacks fail to recover the used AES key. For the remaining area, one can conclude that no first-order leakage can be identified for the masked AES implementation.



(a) T-test result.

(b) T-test result.

(c) $\mathcal{X}^2$-test.

(d) $\mathcal{X}^2$-test result.

Fig. 2: Results of the fixed-vs-random t-test and $\mathcal{X}^2$-test on white-box AES implementations: unprotected (left-hand side) and masked (right-hand side).

4 Bucketing Computational Attack

4.1 Background

In [16], Zeyad *et al.* introduced a new computational attack called *Bucketing Computational Analysis* (BCA) to defeat unprotected white-box implementation. The core idea of BCA is that if two sensitive intermediate variables for two different plaintexts do not collide, then their encodings (using a deterministic bijection) should not collide as well. We recall in Algorithm 1 the pseudo-code describing the different steps of the BCA attack when applied on white-box AES implementations [16].

The BCA attack consists of 3 phases. During the pre-computation phase, for each key guess k, the attacker split the 256 plaintexts x (each corresponds to a different Sbox input) into two sets ($I_{0,k}$ and $I_{1,k}$) according to the resulting bucketing nibble $d = S'(x \oplus k) = S(x \oplus k)\&0\text{xF}$; *i.e.* x in $I_{0,k}$ (respectively in $I_{1,k}$) if $d = d_0$ (respectively if $d = d_1$). Regarding the choice of the values d_0 and d_1, the authors in [16] emphasized the use of $d_0 = 0$ and $d_1 = 0\text{xF}$. In the sequel, we will use these two values as recommended. Then, during the acquisition phase,

Algorithm 1 BCA on white-box AES implementations.

Inputs: a targeted AES Sbox S of the first round and its corresponding S' (*s.t.* $\forall x \in GF(2^8)$, $S'(x) = S(x)\&0\text{xF}$)
Output: good guess of the sub-key

∗ ∗ ∗ Pre-computation phase ∗ ∗ ∗
Compute a set I of 256 plaintexts each corresponding to a different input of S
Pick two values d_0 and d_1 such that: $0 \leq d_0 < d_1 \leq 15$
for each key guess $k \in [0, 255]$ **do**
    Split the plaintexts into two sets $I_{0,k}$ and $I_{1,k}$ w.r.t to the nibble d of S'
end for
∗ ∗ ∗ Acquisition phase ∗ ∗ ∗
Acquire a set of 256 traces $\mathbf{T} = (t_{i,j})_{\substack{0 \leq i \leq 255 \\ 0 \leq j \leq n}}$
∗ ∗ ∗ Key-recovery phase ∗ ∗ ∗
Initialize a result vector R with 256 zeros
for each key guess $k \in [0, 255]$ **do**
    Group the traces into $\mathbf{V_0}$ and $\mathbf{V_1}$ *w.r.t.* to the sorted plaintexts in $I_{0,k}$ and $I_{1,k}$
    for each sample j in the trace **do**
        if $\mathbf{V_0}[j] \cap \mathbf{V_1}[j] = \emptyset$ **then** : $R[k] = R[k] + 1$
        end if
    end for
end for
The good sub-key guess corresponds to $k \in [0, 255]$ that maximizes $R[k]$

the attacker acquires 256 computation traces. Finally, the key recovery phase consists in sorting the computation traces into two *buckets* denoted $\mathbf{V_0}$ and $\mathbf{V_1}$ depending on the bucketing nibble d (whose value depend on a key guess). Then, it counts the number of disjoint columns in $\mathbf{V_0}$ and $\mathbf{V_1}$. The good key value corresponds to the key guess for which the number of disjoint columns is maximal.

Throughout several experiments, the authors have demonstrated that this attack is an efficient alternative to the DCA. Indeed, the required amount of traces to recover 4 bytes of the key is fixed (256 traces for a white-box AES implementation). However, as stated by the authors in [16], when masking is properly applied as a countermeasure, then the BCA attack fails to recover the key. Our goal is to extend the BCA to the second-order case to defeat masked white-box implementations. We keep the study of the generalization of BCA to an order greater than two as a future work.

4.2 Extension to the Higher-Order Case

Let's consider a first-order masked white-box AES implementation for which c computation traces of n samples each were acquired $\mathbf{T} = (t_{i,j})_{\substack{0 \leq i \leq c \\ 0 \leq j \leq n}}$. Then, according to Sec. 4 of [3] and under Assumption 3 there exists a fixed couple (j_1^*, j_2^*) such that $(t_{i,j_1^*}, t_{i,j_2^*})$ are the shares (*i.e.* the mask and the masked value)

of the target secret variable $S(x_i \oplus k^*)$ for any i in $[0, c]$. To check if the BCA can be extended to the second-order case, the arising question is whether $(\mathbf{V_0}[j_1^*] \oplus \mathbf{V_0}[j_2^*])$ and $(\mathbf{V_1}[j_1^*] \oplus \mathbf{V_1}[j_2^*])$ are disjoint sets only for the correct key guess k^*?

To answer this question, let's consider a couple of plaintexts (x_0, x_1) such that $x_0 \in I_{0,k^*}$ and $x_1 \in I_{1,k^*}$, B a 4-bit encoding function, and m_0 and m_1 the masks used during the encryption of x_0 and x_1 respectively. Then, on one hand, we have $x_0 \in I_{0,k^*}$ implies (under Assumption 2) that:

$$\begin{aligned}
\mathbf{V_0}[x_0][j_1^*] \oplus \mathbf{V_0}[x_0][j_2^*] &= B\Big(m_0\&0\text{xF}\Big) \oplus B\Big((S(x_0 \oplus k^*) \oplus m_0)\&0\text{xF}\Big) \\
&= B\Big(m_0\&0\text{xF}\Big) \oplus B\Big(\underbrace{(S(x_0 \oplus k^*)\&0\text{xF})}_{=0} \oplus (m_0\&0\text{xF})\Big) \\
&= B\Big(m_0\&0\text{xF}\Big) \oplus B\Big(m_0\&0\text{xF}\Big) \\
&= 0 \ . \qquad (1)
\end{aligned}$$

On the other hand, $x_1 \in I_{1,k^*}$ implies that:

$$\begin{aligned}
\mathbf{V_1}[x_1][j_1^*] \oplus \mathbf{V_1}[x_1][j_2^*] &= B\Big(m_1\&0\text{xF}\Big) \oplus B\Big((S(x_1 \oplus k^*) \oplus m_1)\&0\text{xF}\Big) \\
&= B\Big(m_1\&0\text{xF}\Big) \oplus B\Big(\underbrace{(S(x_1 \oplus k^*)\&0\text{xF})}_{=0\text{xF}} \oplus (m_1\&0\text{xF})\Big) \\
&= B\Big(m_1\&0\text{xF}\Big) \oplus B\Big(0\text{xF} \oplus (m_1\&0\text{xF})\Big) \\
&= B\Big(m_1\&0\text{xF}\Big) \oplus B\Big(\overline{m_1\&0\text{xF}}\Big) \ . \qquad (2)
\end{aligned}$$

Since B is bijective then $\mathbf{V_1}[x_1][j_1^*] \oplus \mathbf{V_1}[x_1][j_2^*]$ is non-null. Consequently, Eq. (1) and Eq. (2) prove that the sets $(\mathbf{V_0}[j_1^*] \oplus \mathbf{V_0}[j_2^*])$ and $(\mathbf{V_1}[j_1^*] \oplus \mathbf{V_1}[j_2^*])$ are disjoint for the good key guess. For a wrong key guess k, it is obvious that these sets have common values. Indeed, for any $x_0 \in I_{0,k}$ (respectively $x_1 \in I_{1,k}$) such that $k \neq k^*$, $S(x_0 \oplus k^*)$ (respectively $S(x_1 \oplus k^*)$) is a random value. Now, as the intersection between two sets containing random values is non-null (for some cardinality), then the sets $(\mathbf{V_0}[j_1^*] \oplus \mathbf{V_0}[j_2^*])$ and $(\mathbf{V_1}[j_1^*] \oplus \mathbf{V_1}[j_2^*])$ are only disjoint for the good key guess k^* which prove the soundness of our proposal. It is worth noting that the same soundness proof can be generalized to handle any value of the bucketing nibble pair (d_0, d_1).

So, the core idea of the second-order BCA is to search for two time samples (j_1^*, j_2^*) in the traces such that $(\mathbf{V_0}[j_1^*] \oplus \mathbf{V_0}[j_2^*])$ and $(\mathbf{V_1}[j_1^*] \oplus \mathbf{V_1}[j_2^*])$ are disjoints. To further discard the false positives, we emphases the use of the two following additional criteria that should be only fulfilled for the good key guess: (1) the set $(\mathbf{V_0}[j_1^*] \oplus \mathbf{V_0}[j_2^*])$ is null and (2) the set $(\mathbf{V_1}[j_1^*] \oplus \mathbf{V_1}[j_2^*])$ contains non-constant values (as the mask m_1 in Eq. (2) should be different from one encryption to another).

The complexity of the second-order BCA attack (as for any higher-order attack investigated in this work) highly depends on the size of the targeted area

of interest to detect the leakages of the mask and the masked variable. To reduce this complexity, we recommend filtering the computation traces (*i.e.* removing the common constant values) as suggested in [16] and applying the different hints proposed in [9] to reduce the dimensionnality of the traces.

Finally, we suggest an improvement for the first-order BCA reported in [16] to avoid the appearance of some false positive in some practical white-box evaluations. The idea is that, for each disjoint columns in $\mathbf{V_0}$ and $\mathbf{V_1}$, the attacker has to check if $\mathbf{V_0}$ and $\mathbf{V_1}$ are constant sets. Indeed, following the reasoning in Eq. (1) and Eq. (2), $\mathbf{V_0}$ and $\mathbf{V_1}$ should contain respectively $B(0)$ and $B(0\text{xF})$ for the good key guess.

4.3 Experimental Results

To check the effectiveness of the extended BCA in a real-world scenario, we develop the second-order BCA. The source code of our implementation is publicly available in [1]. Then, we run the attack on the acquired traces of our masked white-box AES implementation described in Sec. 2.3. The obtained results are shown in Fig. 3.

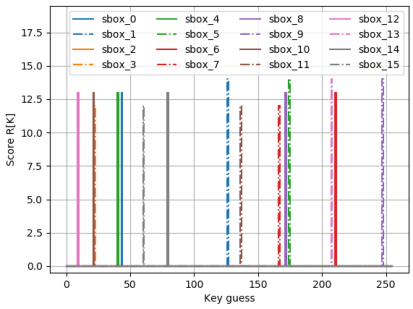


Fig. 3: Second-order BCA results when targeting the 16 AES Sboxes.

The results demonstrate that the 16 bytes of the AES key were recovered. It is worth noting that thanks to the new criteria suggested in this work, no false positives were detected for the false key guesses (*i.e.* $R[k]$ remains equal to zero when k is different from k^*).

5 Mutual Information Attack

5.1 Background

In 2008, Gierlichs *et al.* have proposed a new side-channel distinguisher called Mutual Information Analysis (MIA) [8]. It is an attractive alternative to the Correlation Power Analysis as it exploits any kind of dependency (linear or non-linear) between the leakage measurements and the predicted data. The MIA has been largely studied and tested on several implementations [2, 8, 13].

The core idea consists in estimating the mutual information between the leakage measurements L and the predictions $\phi(k)$ for every key guess k, that is: $\Delta_{\text{MIA}}(k) = \mathbb{H}[L] - \mathbb{H}[L|\phi(k)]$. The correct guess of the key k^* corresponds to the key for which $\Delta_{\text{MIA}}(k)$ is maximum. Since $\mathbb{H}[L]$ does not depend on the key guesses, then the adversary can equivalently look for the key that minimizes the conditional entropy $\mathbb{H}[L|\phi(k)]$. The major practical issue of mutual information is the estimation of the statistical distribution of the leakages. Several methods have been proposed in the literature: histograms, kernel density function, parametric estimation [13]. We discuss in Sec. 5.2 how we dealt with this problematic to conduct our practical attacks.

To the best of our knowledge, the first report on the use of MIA in the context of white-box evaluation was provided by Rivain *et al.* in [14]. Indeed, the authors have assessed the publicly available white-box AES implementation NoSuchCon 2013) against an improved version of the MIA. The obtained results have proven that this attack is efficient to break internally-encoded implementation with only few traces (60 traces) compared to the DCA (4000 traces). This is expected as the dependency between the leakage and the predictions in the white-box context is non-linear (due to the usage of the internal-encoding to hide the implementation intermediate values). In the following section, our goal is to extend the MIA to higher-order context to target masked white-box implementations.

5.2 Extension to the Higher-Order Case

In 2009, Prouff *et al.* have proposed in [13] a generalization of the MIA to higher-orders. Let's consider a d^{th}-order masked implementation and assume that the adversary knows exactly the manipulation times of the used masks and masked data. Hence, he is able to recover the corresponding $(d+1)$-tuples of leakages $\mathbf{L} = (L_0, L_1, \ldots, L_d)$. Then, the higher-order MIA consists in finding the key guess that minimizes the conditional entropy $\mathbb{H}[\mathbf{L}|\phi(k)]$. It is noticeable that the probability density function (pdf) of the variable $\mathbf{L}|\phi(k)$ is often assumed to be a multivariate Gaussian mixture whose entropy can be estimated as for the first-order case using histograms, kernel density function or parametric estimation.

To efficiently apply the higher-order MIA [13] in the context of masked white-box implementation, we focus on the two following practical challenges:

The choice of the pdf estimation method. In side-channel context, the estimation of the mutual information is a major practical issue as it involves some complex pdf estimation methods. It is worth noting that the results of these methods have a strong impact on the efficiency of the MIA [2]. In white-box context, the computation traces contains non-noisy values. Hence, the estimated pdfs are discrete which makes the practical evaluation simpler (as also argued in [14]). In such context, the histogram estimation seems to be the easier (natural) method to consider. That said, the optimal choice of the bin width is also an issue in statistical theory. For simple distribution, reasonable choices of the bin width are the Scott rule and the Freedman-Diaconis rule. However, under Assumption 1, the computation traces contain values in the range $[0, 15]$.

Thus, the natural choice of the number of bins we consider is 16 (*i.e.* bin width equals to 1)[3].

The choice of the leakage combination function. To apply a d^{th}-order MIA in the white-box context, one can apply directly the side-channel approach described in [13]. That is, the adversary has to consider d-tuples of leakages and look for the key guess that minimizes the conditional entropy of the multivariate pdf $(\mathbf{L}|\phi(k))$. Another approach, that we investigate in this work, consists in combining the d-tuples of leakages by applying the XOR function. Said differently, the idea is to minimize the conditional entropy of the uni-variate pdf $(\bigoplus_{i=0}^{i=d} L_i|\phi(k))$. Indeed, under Assumption 3, the computation traces (and hence the leakages L_i) contains the exact value of the manipulated variable. Thus, by applying the XOR combination, the adversary converts the leakages from multivariate to uni-variate context where the pdf estimation is much easier. In the following section, we compare both approaches when targeting our reference masked white-box AES implementation.

5.3 Experimental Results

We implement two versions of a second-order MIA: the first one is based on an estimation of a bi-variate pdf and the second one is based on the XOR combination of the leakages and then an estimation of the resulting uni-variate pdf. The pdf estimation is based on the histogram method with a fixed number of bins (16) as discussed previously. Then, we target our masked white-box implementation when using 150 computation traces.

The obtained results have proven that both attack versions have succeeded to recover the complete AES key. In the meantime, and as expected, the XOR combination based second-order MIA is two times faster than the multivariate one. We provide the source code of our second-order MIA in [1].

6 Collision Attack

6.1 Background

Recently, Rivain *et al.* have proposed a collision attack to defeat internally-encoded white-box implementations [14]. The proposed attack is inspired by the DCA. The major difference is that the collision attack is based on the processing of a pair of plaintexts (instead of one plaintext as for DCA) to build the prediction vector and the corresponding Collision Computation Traces (CCT). That is, for each key guess k and each pair of inputs (x_1, x_2), the adversary computes the Pearson correlation coefficient $\rho(\delta_{L(x_1)L(x_2)}, \delta_{\phi(x_1,k)\phi(x_2,k)})$ where δ_{xy} is the "vector adaptation" of the well-known Kronecker delta function and $L(x)$ is the computation trace collected while processing the input x. The secret key corresponds to the key guess for which the highest correlation peak is obtained.

[3] Commonly, the number of bins should be equal to $2^{(\text{encoding size})}$.

So, the core idea of this DCA-like collision attack is that if some sensitive variable collides for a pair of inputs, so does the corresponding encoded variable in the computation (as the encoding functions are bijective). One improvement of this attack we suggest from a performance perspective and which we validate through simulation and practical experiments is to consider the *equality distinguisher* instead of computing the correlation. Indeed, the adversary can count, for each key guess, the number of times the prediction vector $\delta_{\phi(x_1,k)\phi(x_2,k)}$ and the vector of the targeted samples in the CCT $\delta_{L(x_1)L(x_2)}$ are equal. This equality counter is maximum for the good key guess.

The authors in [14] have successfully applied this attack on some publicly available white-box implementations and have demonstrated that the trace complexity is quite low compared to DCA. In the following section, we study the extension of this DCA-like collision attack to target higher-order masked white-box implementations.

6.2 Extension to the Higher-Order Case

Let's assume that an adversary recovers the d-tuples of leakages $(L_1, \ldots, L_d)$ from a $(d-1)^{\text{th}}$-order masked white-box implementation. Then, to apply the d^{th}-order collision attack, he has to compute for each key guess and each pair of inputs (x_1, x_2) the following distinguisher:

$$\rho(\bigoplus_{i=1}^{d} L_i(x_1) \oplus \bigoplus_{i=1}^{d} L_i(x_2), \delta_{\phi(x_1,k)\phi(x_2,k)}) \ . \tag{3}$$

The soundness of our proposed d^{th}-order collision attack is inherently based on the soundness of the d^{th}-order DCA attack under Assumption 3. In fact, the d^{th}-order DCA is defined as $\rho(\bigoplus_{i=1}^{d} L_i(x), \phi(x,k))$ for every input x. When considering any pair of inputs (x_1, x_2), the d^{th}-order DCA rewrites $\rho(\bigoplus_{i=1}^{d} L_i(x_1) \oplus \bigoplus_{i=1}^{d} L_i(x_2), \phi(x_1,k) \oplus \phi(x_2,k))$. Now, since we are focusing on the study of the collision during the processing of two different inputs x_1 and x_2, the relevant values of the prediction vector are when $\phi(x_1,k) \oplus \phi(x_2,k) = 0$. Thus, one can transform the prediction vector from $\phi(x_1,k) \oplus \phi(x_2,k)$ to $\delta_{\phi(x_1,k)\phi(x_2,k)}$ to obtain the distinguisher described in Eq. (3). Said differently, the correlation is only computed for the pair of plaintexts for which a collision occurs for a key guess k (*i.e.* $\phi(x_1,k) = \phi(x_2,k)$).

6.3 Experimental Results

We validate first the soundness of the extended collision attack through simulation and we provide the source code of its second-order version along with the used simulated traces in [1]. Then, we run the attack on the computation traces

collected from our reference masked white-box implementation. The obtained results have demonstrated the practicability of the attack. Indeed, we succeed to recover the AES key using 500 computation traces.

7 Attack Comparison

We provide in Tab. 1 a comparison of the results obtained for the studied attacks in this work. For the sake of comparison, we perform as well the second-order DCA attack as described in [3]. For a fair comparison, we target the same areas of interest to search for the leaking points of the mask and the masked sensitive variable. All the attacks are executed on a Linux machine with an Intel Core i7 processor at 3.60GHz and 16 GB of RAM.

	DCA	BCA	MIA (XOR)	MIA (multivariate)	Collision
Execution time (s)	4.28	5.67	15.58	34.11	70.15
Number of traces	310	256	150	150	500

Table 1: Comparison of the studied second-order attacks when targeting a masked white-box implementation.

The most efficient attack in terms of traces complexity is the MIA. This is expected due to its ability to capture the non-linear dependency between the leakage and the predictions. In terms of performance, the MIA based on the XOR combination is faster than the multivariate version. This could be explained, as already discussed in Sec. 5, by the fact that the estimation of the conditional entropy of multivariate random variables is more time-consuming compared to the univariate case. The second-order BCA offers the best trade-off between the execution time and trace complexity and hence it is a good alternative to the second-order MIA attack. Finally, when masking is involved then the DCA is better than the collision attack. Indeed, the collision attack (*w.r.t.* to Eq. (3)) can be seen as a particular case of the DCA attack where the correlation is only computed when a collision is detected. However, the collision attack remains a good candidate to consider in an unmasked context as demonstrated in [14].

8 Conclusion

In this work, we considered the evaluation of higher-order masked white-box implementations. In particular, we proposed first to apply the well-known leakage detection methods to help a security evaluator to assess the security level of the targeted implementation. Then, to exploit a detected higher-order leakage, if any, we extended some well-known computational attacks to the higher-order context. The practical evaluation of these attacks had shown their efficiency to defeat masked white-box implementations.

As a future work, we intend to study these higher-order computational attacks when relaxing the assumptions formulated in Sec. 2.2. Namely, we plan to evaluate some internally-encoded and masked white-box implementation combined with shuffling countermeasure and such that the encoding functions applied on the masks and the masked variable are different.

References

1. Supporting materials. https://github.com/Bucketing/HO-attacks-on-WB.
2. L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon. Mutual Information Analysis: a Comprehensive Study. *J. Cryptology*, 24(2):269–291, 2011.
3. A. Bogdanov, M. Rivain, P. S. Vejre, and J. Wang. Higher-order DCA against standard side-channel countermeasures. In *COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, pages 118–141, 2019.
4. J. W. Bos, C. Hubain, W. Michiels, and P. Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In *CHES 2016*, pages 215–236, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
5. S. Chow, P. Eisen, H. Johnson, and P. C. van Oorschot. A white-box des implementation for drm applications. In J. Feigenbaum, editor, *Digital Rights Management*, pages 1–15, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
6. S. Chow, P. A. Eisen, H. Johnson, and P. C. v. Oorschot. White-box cryptography and an aes implementation. In *SAC 2002*, pages 250–270, London, UK, UK, 2003. Springer-Verlag.
7. J. Cooper, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. Test Vector Leakage Assessment (TVLA) Methodology in Practice, Sept 24–26 2013. International Cryptographic Module Conference, Gaithersburg, MD, USA.
8. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In *CHES 2010*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, August 10-13 2008. Washington, D.C., USA.
9. L. Goubin, P. Paillier, M. Rivain, and J. Wang. How to reveal the secrets of an obscure white-box implementation. *Journal of Cryptographic Engineering*, Apr 2019.
10. 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.
11. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. http://www.springer.com/Springer, December 2006. ISBN 0-387-30857-1, http://www.dpabook.org/.
12. A. Moradi, B. Richter, T. Schneider, and F.-X. Standaert. Leakage detection with the x2-test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):209–237, Feb. 2018.
13. E. Prouff and M. Rivain. Theoretical and Practical Aspects of Mutual Information Based Side Channel Analysis. In Springer, editor, *ACNS*, volume 5536 of *LNCS*, pages 499–518, June 2-5 2009. Paris-Rocquencourt, France.
14. M. Rivain and J. Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):225–255, Feb. 2019.
15. T. Schneider and A. Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In *CHES 2015, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 495–513, 2015.
16. M. Zeyad, H. Maghrebi, D. Alessio, and B. Batteux. Another look on bucketing attack to defeat white-box implementations. In I. Polian and M. Stöttinger, editors, *COSADE*, pages 99–117, Cham, 2019. Springer International Publishing.