Higher-Order DCA against Standard Side-Channel Countermeasures

Andrey Bogdanov[1], Matthieu Rivain[2], Philip S. Vejre[1] and Junwei Wang[2,3,4]

[1] Technical University of Denmark {anbog,psve}@dtu.dk
[2] CryptoExperts, France {matthieu.rivain,junwei.wang}@cryptoexperts.com
[3] University of Luxembourg
[4] University Paris 8, France

Abstract. At CHES 2016, Bos *et al.* introduced *differential computational analysis* (DCA) as an attack on white-box software implementations of block ciphers. This attack builds on the same principles as DPA in the classical side-channel context, but uses traces consisting of plain values computed by the implementation during execution. This attack was shown to be able to recover the key of many existing AES white-box implementations.
The *DCA adversary* is *passive*, and so does not exploit the full power of the white-box setting, implying that many white-box schemes are insecure even in a weaker setting than the one they were designed for. An important problem is therefore how to develop implementations which are resistant to this attack. A natural approach is to apply standard side-channel countermeasures such as *masking* and *shuffling*. In this paper, we study the security brought by this approach against the DCA adversary. We show that under some necessary conditions on the underlying randomness generation, these countermeasures provide resistance to standard (first-order) DCA. Furthermore, we introduce *higher-order DCA*, and analyze the security of the countermeasures against this attack. This attack is enhanced by introducing a *multivariate* version based on the maximum likelihood approach. We derive analytic expressions for the complexity of the attacks which are backed up through extensive attack experiments. As a result, we can quantify the security level of a masked and shuffled implementation in the (higher-order) DCA setting. This enables a designer to choose appropriate implementation parameters in order to obtain the desired level of protection against passive DCA attacks.

Keywords: White-box · DCA · Higher-order DCA · Masking · Shuffling

1 Introduction

In the classical cryptanalytic setting, the adversary faces the challenge of breaking the security of *e.g.* an encryption algorithm while only being able to consider the algorithm as a *black box*; she can query the box with inputs and receive the corresponding outputs. While the design of the algorithm is known, the adversary cannot observe the internal state of the algorithm, or affect the execution of the algorithm.

In practice, a cryptographic algorithm has to be implemented somewhere to be useful, *i.e.* in hardware or software. Thus, the adversary has the option of physically interacting with the encryption device. In this case, the adversary has access to implementation specific *side-channel information* which most real world implementations will leak. If an implementation is not sufficiently protected, information such as the execution time or the power consumption can be used to extract secret information, *e.g.* encryption keys. Limited modification of a device as it executes the algorithm is also possible, *i.e.* in the case of fault attacks. The widespread use and success of side-channel attacks show that a cryptographer has to be very careful when operating in this *grey-box* model.

1.1 Shades of Grey

For hardware implementations, the grey-box model is often the limit of what an adversary can achieve. While it is possible to interfere with the execution of a hardware implementation, getting accurate control over the execution is often prohibitively expensive, even in physical possession of the targeted device.

This is not the case for software implementations that are executed in untrusted environments. If an adversary is given full access to the execution environment of the cryptographic software, she can easily observe and manipulate the execution of the primitive, instantiated with some secret key. This setting, introduced by Chow *et al.* in [CEJvO02b], is called the *white-box* model or the white-box attack context.

In the white-box setting, the adversary can study the logic flow of the implementation, observe any tables the implementation uses, observe intermediate values computed during execution, alter the implementation at run-time, etc. Indeed, *"when the attacker has internal information about a cryptographic implementation, choice of implementation is the sole remaining line of defence"* [CEJvO02b]. Ideally, the aim of a white-box implementation would be to leave the white-box adversary with at most the same advantage as a black-box adversary, but this seems to be a very difficult goal to achieve. Instead, the current white-box paradigm aims to provide *practical security*, in the sense that the implementation is difficult enough to attack so that an adversary is forced to attempt other attack vectors.

The AES has been a popular choice for white-box implementations. Shortly after its standardisation, the first proposals for such implementations appeared in the literature. Perhaps the most used approach to this problem, first introduced by Chow *et al.*, is to translate the operations of the AES into a network of lookup tables, and use randomly chosen external and internal encodings on these tables, embedding the secret key in the randomised tables [CEJvO02b, CEJvO02a, XL09, Kar10]. A radically different approach, based on perturbations of multivariate polynomial equations, was given by Bringer *et al.* [BCD06a, BCD06b]. Sadly, all these designs have been broken by structural attacks [BGE04, MRP12, MGH08, Tol12, LRM+13, MWP10]. As such, all current white-box AES implementations presented in the literature are insecure.

Although no provably secure solutions have been reported in the public literature, there exists an industrial need for the protection of cryptographic implementations executed in untrusted environments. A traditional use case is the distribution of multimedia content (video/music on demand) with conditional access control relying on *digital rights management* (DRM). In this context, the user herself might try to recover the underlying keys to get full (and unauthorized) access to some content. More recently, white-box cryptography has been widely applied to protect applications running on smart devices (which cannot be considered as trusted environments) in particular with the advent of mobile payment and host card emulation technology.

As explained in [DLPR13] the minimal requirement for white-box cryptography in such contexts is the resistance to key-extraction (also called *unbreakability*). This property alone it is not *enough* and further security features should be provided to restrict what can be obtained from *code-lifting*, *i.e.* using the implementation as an oracle of the underlying algorithm. It has for instance been suggested to rely on *external encodings* and/or to lock the implementation with a password or some biometric data. One could also make the implementation large and *incompressible* to harden its extraction, make it *one-way* to prevent either encryption or decryption, or make it traceable to counteract unauthorized sharing (see for instance [CEJvO02b, DLPR13, BIT16]).

The lack of provably-secure white-box implementations has driven the industry to develop home-made solutions relying on *obscurity* (i.e. secrecy of the underlying obfuscation techniques). In this paradigm, white-box cryptography acts as a moving target based on regular security updates and/or short-term key tokens and is considered a building block of wider security solutions.

1.2 Differential Computational Analysis

The mentioned attacks against *public* white-box implementations exploit flaws in the underlying white-box techniques. They are, however, not fit to address the obscure-design paradigm. Indeed, secret variants of known designs that change a few parameters or combine different techniques would likely thwart these attacks, given that the exact design is not known to the adversary. To attack such an implementation, the adversary would have to perform reverse engineering, which can take considerable time and effort if various layers of obfuscation have been applied.

A more generic approach was given recently by Bos *et al.*; [BHMT16] introduced the white-box equivalent of DPA, namely *differential computational analysis* (DCA), and demonstrated how this technique is able to recover the encryption key of several existing white-box implementations of the AES. Notably, the *DCA adversary* is extremely powerful as it does not require full knowledge of the white-box techniques involved in a given implementation – the adversary is *implementation agnostic* and therefore does not need to exert expensive reverse engineering efforts. Thus, DCA has been devastating

for industrial solutions that leverage design secrecy to develop white-box implementations consisting of a mix of various techniques (common white-box techniques, code obfuscation, home-made encodings, etc.).

Moreover, the DCA adversary does not take advantage of the full power of the white-box model. All that is required of the adversary is that she is able to observe the addresses and values of memory being accessed during the execution of the implementation. The adversary does not need to reason about the implementation details, or modify the functionality of the code in any way, *e.g.* by disabling RNG functionalities – tasks that, depending on the implementation could required considerable effort. Thus, the DCA attack is a *passive* and *non-invasive* attack, and the adversary operates in a setting which is closer to a grey-box than a white-box. Additionally, the attacks presented in [BHMT16] have very low complexities.

Following these observations, the current white-box AES implementations are not even secure in a weaker attack context than the one they were designed for, and as a consequence, designing secure white-box implementation seems out of reach. Indeed, DCA seems to be the biggest hindrance to designing practically secure white-box implementations. It is therefore of importance to first explore the design of cryptographic implementations which are secure against DCA.

1.3 Our Contributions

When attempting to mitigate the threat of DCA attacks on white-box implementations, a natural approach is to apply known countermeasures from the side-channel literature. However, it is not clear how well these countermeasures carry over to the white-box context and what level of security can be achieved by such countermeasures against a DCA adversary. To address these issues, we achieve the following:

- **Side-channel countermeasures in the white-box setting:** In Section 3 we discuss the application of well known side-channel countermeasures in the white-box setting with a special focus to the (passive) DCA adversary. Specifically, we focus on *higher-order masking* along with *shuffling of operations* to introduce noise in the DCA traces. We show that if the source of randomness used in the implementation satisfies some specific security properties, then this approach is sufficient to achieve security against standard first-order DCA.

- **Higher-order DCA:** In order to analyze the security of the proposed protection against more advanced attacks, we develop *higher-order DCA* in Section 4. This attack combines several coordinates of a computation trace into a higher-order trace to defeat the masking countermeasure. We show that in the absence of noise in the computation traces, higher-order DCA breaks a masked implementation of any order using a couple traces. However, by introducing noise in the form of shuffling, the security of the implementation can be dramatically increased. As a demonstration, a typical AES implementation with 2^{nd} order masking (and shuffling degree of 16) requires 2^{21} traces to break with 3^{rd} order DCA.

- **Multivariate higher-order DCA:** We extend the above attack by introducing a multivariate version in Section 5. This improved attack exploits many points of the higher-order traces simultaneously and uses a maximum likelihood approach, making it theoretically optimal. This improvement allows us to decrease the number of traces required for a successful attack, and consequently reduce the computational complexity. Using multivariate higher-order DCA, the number of traces required to successfully attack the AES implementation mentioned above can be reduced to 2^{10}.

- **Formal analysis:** We derive analytic expressions for the success probability and attack complexities of both the higher-order DCA and its multivariate variant. Using these expressions, we are able to give estimates for the security level of a masked and shuffled implementation in the DCA setting. As an example, an AES implementation with 7^{th} order masking would have a security level of about 85 bits in this setting.

- **Experimental verification:** We thoroughly verify the expressions for the success probability of the multivariate higher-order DCA in Section 5.4. Through extensive experiments, we verify the accuracy of our estimates for a wide range of implementation and attack parameters. 2 000 attacks of up to order 4 were simulated, using as many as 30 000 traces per attack.

In summary, our result provides formal ground to the study of standard side-channel defenses in the white-box setting. We have analyzed the widely used masking and shuffling countermeasures with respect to the DCA adversary and we have quantified their security against advanced DCA attacks. From our analysis, a designer can chose an appropriate set of implementation parameters to achieve a given security level with respect to DCA, which is a first step towards building security against a stronger white-box adversary.

1.4 Related Works

Two independent and related works have been published since the first version of the present paper which both address masking in the white-box context.

First, Biryukov and Udovenko [BU18] broadly overview how a white-box adversary could attack masked white-box implementations in several aspects. They present a comprehensive complexity analysis from which they deduce several design criteria for masking countermeasure to prevent some of these attacks. In particular, they consider an *active* (fault injection) attack which reveals the position of masking shares in the implementation.

Second, Goubin *et al.* [GPRW18] proposed a method to attack an *obscure* white-box implementation (in the sense that the adversary has no/limited knowledge on the design), which was successfully applied to break the winning challenge in the recent WhibOx 2017 contest [whi]. Particularly, they introduced *linear decoding analysis* (LDA): this attack technique can break a white-box implementation in which intermediate variables are encoded in a way that a linear decoding can be applied to recover the plain variables (which is the case with masking of any order). LDA could hence break a noise-free masked implementation with complexity approximately cubic in the size of the computation trace. Both LDA and the higher-order DCA analysed in Section 4 demonstrate that the sole masking countermeasure (regardless of the number of shares) is insufficient against a passive DCA adversary.

Compared to these works, the adversary we consider is *passive* (she cannot tamper with the execution); as such she would not be impacted by, *e.g.*, fault detection/correction measures. Moreover, masking is known to be a weak countermeasure unless it is composed with some kind of *noise*. Therefore, we focus on white-box implementations protected with both masking and shuffling; we introduce and analyze advanced DCA techniques to deal with such a setting.

2 Differential Computation Analysis

The *DCA adversary* is capable of querying a software implementation of a cryptographic primitive with arbitrary input to obtain a *computational trace* of the execution. The computational trace consists of: any value calculated, written, or read by the implementation, and the address of any memory location read from or written to during execution. Each data point in the computational trace is furthermore annotated with the time it occurred in the execution. When a number of traces have been collected, the adversary runs these through a tool to calculate correlations between a prediction of a key-dependent intermediate value and computed values.

2.1 DCA Setting

As for hardware side-channel attacks, DCA exploits that the (software) implementation leaks some information about intermediate variables involved in the execution of the cryptographic algorithm. Some of these intermediate variables depend on the plaintext and (part of) the secret key, and knowledge of such variables can therefore reveal the key. We will denote by s such a *secret variable* and express it

$$s = \varphi(x, k^*),$$

where φ is a deterministic function, x is a public value, *e.g.* (part of) the plaintext, and $k^* \in \mathcal{K}$ is a (secret) subkey over some subkey space $\mathcal{K}$. For instance, k^* could be a byte of the secret key and $\mathcal{K}$ would then be $\{0,1\}^8$.

The DCA attack itself consists of first obtaining a number of computational traces from the execution of the cipher implementation with secret subkey k^* for several (random) plaintexts. We denote a computational trace consisting of t time points by the ordered t-tuple $\boldsymbol{v} = (v_1, v_2, \ldots, v_t)$, with $v_i \in \mathcal{V}$

for some set $\mathcal{V}$. In principle, the traces can be any value exposed to a dynamic binary analysis tool, as explained above. Usually, an attacker will obtain N computational traces, $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N$, representing N executions of the implementation with different inputs, each corresponding to a value $s_i = \varphi(x_i, k^*)$ of the target secret variable. These traces could *e.g.* arise from the encryption of N different plaintexts. The attacker then performs a classic DPA, in which a *distinguisher* is used to indicate a correct guess of k^*. The distinguisher is a function D which maps the set of computational traces $(\boldsymbol{v}_i)_i$, and corresponding inputs $(x_i)_i$, to a *score vector*:

$$(\gamma_k)_{k\in\mathcal{K}} = \mathsf{D}\big((\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N); (x_1, \ldots, x_N)\big) \ .$$

The adversary then selects the key guess k with the highest score γ_k as candidate for the correct value of k^*. We define the success probability of the attack as

$$p_{\text{succ}} = \Pr(\text{argmax}_{k\in\mathcal{K}}\gamma_k = k^*),$$

where this probability is taken over any randomness supplied to the implementation (including the randomness of the inputs).

2.2 Standard First-Order DCA

The description above does not specify the distinguisher D. Here, we briefly describe the distinguisher used in [BHMT16], which we will call the *standard first-order DCA*. Let us denote by $v_{i,j}$ the value at the j'th time point of the i'th trace. The standard first-order DCA attack consists of calculating a correlation coefficient between a vector of predicted values of the secret variable, $(s_1^k, \ldots, s_N^k)$, where $s_i^k = \varphi(x_i, k)$, and the vector $(v_{1,j}, \ldots, v_{N,j})$, for every time index $1 \le j \le t$. Then the score γ_k is defined as the maximum correlation obtained over the different time indices, i.e.

$$\gamma_k = \max_j C\big((v_{1,j}, \ldots, v_{N,j}), (\psi(s_1^k), \ldots, \psi(s_N^k))\big),$$

for some correlation measure C and some *pre-processing function* ψ. For example, C could be the Pearson correlation coefficient and ψ either the Hamming weight function or the selection of one bit of the predicted variable. If there exists a statistical correlation between the secret variable and the values of the computational trace, we would expect a large absolute value of C for some index j and the correct prediction of the secret variables, *i.e.* the vector $(s_1^{k^*}, \ldots, s_N^{k^*})$. On the other hand, if $k \neq k^*$, we expect a low correlation between all s_i^k and any point in the computational trace. It was shown in [BHMT16] that this approach is very effective against a range of different AES and DES white-box implementations.

3 Side-Channel Countermeasures against DCA

The DCA adversary is highly reminiscent of the standard side-channel adversary. It is therefore natural to apply traditional side-channel countermeasures to a white-box implementation, and to evaluate their performance against a DCA adversary. In the following, we specifically study the common software countermeasures of *higher-order masking* and *operation shuffling*.

3.1 DCA is a Passive and Non-Invasive Grey-Box Attack

We start by noting that applying the mentioned countermeasures in a strict white-box context might be hazardous – indeed classical countermeasures such as masking and shuffling use fresh randomness throughout the execution of the protected implementation which is usually provided by an external random number generator (RNG). Since a white-box adversary has full control over the execution environment, such an RNG could be detected and disabled, shuffled operations could be resynchronised (*e.g.* using memory addresses, program counter, etc.), and/or masks could be cancelled (if masked variables and corresponding masks are easily identified).

In order to make such disabling difficult, the used randomness should rely on some internal PRNG (this issue is discussed in details in Section 3.4) and one should further add some layers of obfuscation countermeasures in top of it. The adversary would then have to invest some reverse engineering effort to

bypass the countermeasures. Nevertheless, we stress that this is exactly the type of analysis an adversary performing a DCA attack is trying to avoid. As discussed in Section 1.2, the major strength of the DCA attack is that it is highly *non-invasive*, allowing an attacker to avoid any deep analysis and reverse engineering of the targeted implementation. While DCA attacks might not be optimal in terms of time and/or data complexity, they are very powerful due to their genericness and the fact that they can be applied in a *black-box* way, *i.e.* without requiring reverse engineering effort. This is an essential property of these attacks in the current white-box cryptography paradigm where designers aim at practical security (as provable security seems out of reach) and use the secrecy of the design as a leverage towards this goal. The purpose of protecting against DCA is therefore to force an adversary to employ more complicated and dedicated attack techniques, which might take a long time to develop and apply, which is beneficial when combined with a moving target strategy.

In the following, we recall the principle of masking and shuffling and discuss their application in the white-box setting. We further discuss the source of randomness necessary to feed these countermeasures and state a few security properties that it should satisfy in this setting. We then show that this approach achieves security against standard first-order DCA. The rest of the paper is dedicated to the study of advanced DCA attacks against this kind of countermeasures.

3.2 Masking

A widely used countermeasure to standard DPA of hardware implementations is *masking* [CJRR99, GP99]. Since the DCA attack relies on the same ideas as DPA, the prospect of applying masking to secure a software implementation against DCA is promising. The general idea of masking is to split each secret variable into several parts that are then processed independently. Specifically, each secret variable s occurring in the execution of the implementation is split into d *shares* $s_1, \ldots, s_d$ such that

$$s_1 \oplus \ldots \oplus s_d = s.$$

The masking must be done such that any subset of less than d shares are statistically independent of s. A simple way to achieve this is by picking $s_1, \ldots, s_{d-1}$ uniformly at random (*the masks*), and setting $s_d = s \oplus s_1 \oplus \ldots \oplus s_{d-1}$ (*the masked variable*). The masking is then said to be of *order* $d-1$. One important aspect of a masked implementation is therefore that of randomness: the implementation has to use a (P)RNG to generate these $d-1$ masks.

Knowledge of all d shares is required to recover s, but if the implementation at any point combines the d shares, the DCA adversary will be able to observe this combination. Thus, the implementation must be able to perform computations on the secret variable s without combining the shares and revealing the value of s. For a function f used in the cryptographic algorithm, we want to compute shares r_i such that $r_1 \oplus \ldots \oplus r_d = f(s)$, from the original shares $s_1, \ldots, s_d$. The computation is then said to be *secure at the order* τ if no τ-tuple of intermediate variables is statistically dependent on a key-dependent variable. Usually, d'th order masking aims to provide security at the order $\tau = d-1$. To compute any $\mathbb{F}_2$-linear function on a masked variable s, we simply compute the function on each share separately. Thus, calculation of the linear components of a typical SPN can be easily implemented on the masked state. Computing the non-linear components (*i.e.* typically the S-boxes) is more involved but several masking schemes exist that achieve $(d-1)$'th order security (see for instance [RP10, CPRR13, Cor14]).

3.3 Shuffling

Usually, masking is not used as the only countermeasure. Indeed, the strength of a masked implementation is directly related to how noisy the adversary's observation of the shares is. However, the DCA adversary can observe the shares without any noise. In fact, we will show in Section 4 that if masking is the only countermeasure, the DCA adversary can easily recover the key.

Several approaches for introducing and increasing noise in masked implementations have been proposed and analysed, *e.g.* in [VMKS12, SP11, CK10, RPD09]. One such approach is shuffling: instead of processing the calculations of the cipher in some fixed order, the order of execution is randomly chosen for each run of the implementation based on the value of the input (*e.g.* the plaintext). The situation is slightly more complicated in the DCA setting. Here, the adversary can make observations in two dimensions, namely *time* and *memory*. Even if the order of execution is shuffled in time, an adversary

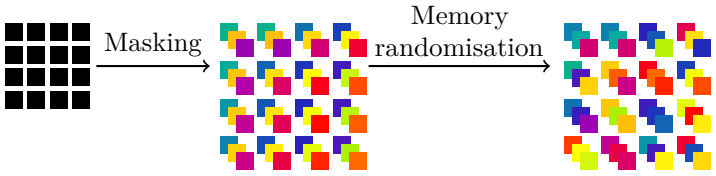

Figure 1: An illustration of the memory shuffling used in our masked implementation. Here, second-order masking is used, splitting each state element into three shares. To avoid that *e.g.* the share $s_{1,1}$ is always located at the first memory location of the state, the location of each share in memory is randomised for each execution.

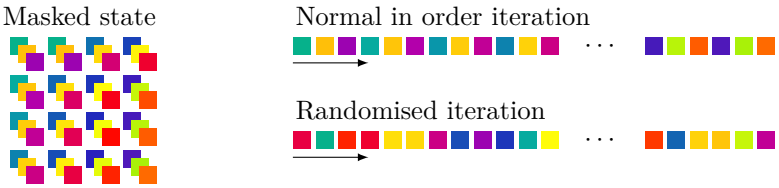

Figure 2: An illustration of the time shuffle used in our masked implementation. A second order-masked state is shown. The normal iteration processes each share of each state element in order. To avoid that e.g the share $s_{1,1}$ is always processed first, the order of iteration is randomised for each execution.

can choose to order the traces by the memory addresses accessed. Thus, we need to shuffle in both the time and memory dimension.

3.3.1 Memory Shuffle

In a masked implementation, we will typically have some state in which each element is shared as described in Section 3.2. The idea of the memory shuffle is to randomly rearrange the shares of the state in memory. Consider a state consisting of S elements. We assume that the shares $s_{i,j}$, $1 \leq i < S$, $1 \leq j \leq d$, are stored in an array, initially in order. That is, the implementation uses the array

$$[s_{1,1}, s_{2,1}, \ldots, s_{S-1,d}, s_{S,d}].$$

Then, we randomly pick a permutation $P : [1, S] \times [1, d] \rightarrow [1, S] \times [1, d]$, based on the value of the input. Note that this can be done efficiently using the Fisher-Yates shuffle [FY+38]. Now, instead of using the in-order array, we rearrange the array such that the implementation uses the array

$$[s_{P(1,1)}, s_{P(2,1)}, \ldots, s_{P(S-1,d)}, s_{P(S,d)}].$$

The situation is shown in Figure 1. Whenever the implementation needs to access share $s_{i,j}$, it simply looks up the element in position $P^{-1}(i, j)$ of the array. A similar randomisation is performed for any key shares.

3.3.2 Time Shuffle

In a typical SPN, there will be several operations that operate on each element of the state in each round. The order of these operations is typically suggested by the cipher designers. As an example, consider the case where we want to apply a linear operation A to each element of the state separately. Since the operation is linear, we can apply it to each share of the masked elements individually. This will normally be done in some "natural" order, *e.g.*:

1: **for** i from 1 to S **do**
2: **for** j from 1 to d **do**
3: Calculate $A(s_{i,j})$
4: **end for**
5: **end for**

However, the order of each execution of line 3 does not matter. Thus, we can shuffle in the time dimension by randomly ordering these $S \cdot d$ operations. In general, if a set of λ independent operations exists, we can freely shuffle the order in which we processes the $\lambda \cdot d$ shares. Formally, we randomly pick a permutation $Q : [1, \lambda] \times [1, d] \to [1, \lambda] \times [1, d]$. Then, when we normally would have processed share $s_{i,j}$, we instead process share $s_{Q(i,j)}$. Thus, the probability that a specific share is processed in a given step is $1/(\lambda \cdot d)$. The situation is depicted in Figure 2. We will denote the size of the smallest maximal set of independent operations the *shuffling degree*.

3.4 On the Source of Randomness

As mentioned above, a potential issue while applying side-channel countermeasures to the white-box context is randomness generation. Since a white-box adversary can easily get rid of an external RNG, the randomness used by a white-box implementation must be pseudo-randomly generated from the single available source of variation: the input plaintext. In other words, the white-box implementation should embed some kind of pseudo random number generator (PRNG) seeded by the input plaintext.

We now (informally) state a few security properties that should be fulfilled by such a PRNG in the white-box setting:

1. *Pseudorandomness:* The output stream of the PRNG should be hard to distinguish from true randomness.

2. *Obscurity:* The design of the PRNG should be kept secret.

3. *Obfuscation:* The PRNG should be mixed with the white-box implementation so that its output stream is hard to distinguish from other intermediate variables.

The pseudorandomness property is required to ensure that the PRNG does not introduce a statistical flaw in the implemented countermeasures. It is well known that a flawed RNG can be a disaster for the security of masking (see for instance [MOP07]). The pseudorandomness property further implies that the generated randomness is *unpredictable* provided that the obscurity property also holds. The unpredictability of the generated randomness is necessary to get DCA resistance, since without it all the intermediate variables can be expressed as (known) deterministic functions of the plaintext and the secret key, which enables the application of standard first-order DCA.[1]

Indeed, if the PRNG design was known to the adversary, then she could predict all the generated randomness from the plaintext. Therefore in order to provide unpredictability, some part of the design must be secret, even if the obscurity concept clearly clashes with the adage of Kerckhoffs's Principle [Ker83]. Nevertheless, it seems almost impossible to provide any security if the full design is known, and we stress that this does not imply that one should forego all good cryptographic engineering practices. One could use a keyed PRNG (or PRNG with secret initial state), but even then if the design was known to the adversary she could mount a DCA attack to recover the PRNG key and we would then face a chicken and egg problem. Alternatively, an implementation could use a known strong PRNG with some sound changes to design parameters, in order to have some confidence in its security. Another approach, which aligns with the moving target strategy, would be to have a set of different PRNG designs that are often changed.

Finally the obfuscation property is required to prevent easy detection of the PRNG output which could facilitate a DCA attack. It is for instance described in [GPRW18] how the generated randomness can be easily detected by switching the values of intermediate variables and checking whether this affects the final result. Such a detection is an *active* attack that tampers with the execution (in the same way as *fault attacks*) and is hence out of scope of the DCA adversary. However it should be made difficult (in the same way as fault attacks should be made difficult) to achieve some level of resistance in practice.

In the following, we shall consider that the above security properties are satisfied by the used PRNG so that the DCA adversary cannot easily remove or predict the generated randomness. We will then analyse which level of security is achievable by using masking and shuffling in this context.

[1] Following the DCA setting described in Section 2.1, the only impact of the countermeasures in presence of a known PRNG is to change the deterministic function φ in the expression of the secret variable s.

3.5 Resistance to First-Order DCA

In Section 2 we described the capabilities of the DCA adversary and the standard first-order DCA. For the masked implementation described above with $d > 1$, the d'th order security of the underlying masking scheme implies that any d-tuple of variables from the computation trace $\boldsymbol{v} = (v_1, v_2, \ldots, v_t)$ is statistically independent of any key-dependent variable s. Assuming that the PRNG embedded in the target implementation outputs strong and unpredictable pseudorandomness (as required in Section 3.4) the distribution of any d-tuple of variables from $\boldsymbol{v}$ is indistinguishable from the same distribution with perfect randomness, which makes it (computationally) independent of any key-dependent variable s. From this observation, standard first-order DCA as described in Section 2.2 is doomed to fail.

4 Introducing Higher-Order DCA

While masking has been proven to be an effective defense against standard DPA, and we have argued for its effectiveness against standard first-order DCA, there are ways to attack such masked implementations. For hardware implementations, it is well known that a $(d-1)$'th-order masked implementation, such as the one described above, can be defeated by d'th-order DPA, if no other protection is employed. We will therefore develop a higher-order version of DCA.

A d'th-order DCA consists of a pre-processing step followed by a first-order DCA. The adversary first pre-processes each computational trace $\boldsymbol{v}$ to obtain a d'th-order computational trace $\boldsymbol{w}$ by applying a so-called (d'th-order) *combination function* ψ. Specifically, the d'th-order computational trace $\boldsymbol{w}$ consists of $q = \binom{t}{d}$ points $(w_1, \ldots, w_q)$ given by

$$w_j = \psi(v_{j_1}, v_{j_2}, \ldots, v_{j_d}), \quad \{j_1, \ldots, j_d\} = \phi(j),$$

where $\phi(j)$ is the j'th subset of $\{1, \ldots, t\}$ of size d (for some ordering). After computing the set of d'th order traces $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N$, the adversary proceeds as for first-order DCA, using the $\boldsymbol{w}_i$'s as input to the distinguisher D. Specifically, the adversary computes the score vector $(\gamma_k)_{k \in K} = \mathsf{D}\big((\boldsymbol{w}_i)_i; (x_i)_i\big)$ in order to determine a candidate for k^*.

For side-channel analysis of hardware implementations, it has been shown that a good combination function for higher-order DPA is the centered product $\psi : (v_1, \ldots, v_d) \mapsto \prod_j (v_j - \mu_j)$, where μ_j is the average of the leakage point v_j over several encryptions. Nevertheless, since the measurements in this setting are inherently noisy, a larger masking degree will require a larger number of traces to obtain a good success probability. Note that this is not the case in the DCA context, if no noise is introduced in the implementation, *e.g.* by using shuffling as described in Section 3.3. In this case, the exact value of each variable that appears in an execution of the implementation appears at the same position of every computational trace $\boldsymbol{v}_i$. Then there exists a fixed j^*, such that for $\phi(j^*) = (j_1^*, \ldots, j_d^*)$, the elements $v_{j_1^*}, \ldots, v_{j_d^*}$ of the trace are the shares of the target secret variable s. In that case, an optimal choice for the combination function is the XOR sum of the trace values, that is

$$\psi(v_{j_1}, v_{j_2}, \ldots, v_{j_d}) = v_{j_1} \oplus v_{j_2} \oplus \cdots \oplus v_{j_d} \ .$$

For this combination function, we have that $\psi(v_{j_1^*}, v_{j_2^*}, \ldots, v_{j_d^*}) = s$ for all the d'th-order traces. By counting the number of times this equality holds, we can easily determine the correct key. That is, we set

$$\gamma_k = \max_j \big(C_k(\boldsymbol{v}_{\phi(j)}, (x_i)_i)\big),$$
$$C_k(\boldsymbol{v}_{\phi(j)}, (x_i)_i) = \left| \left\{ i \ ; \ \bigoplus_{l \in \phi(j)} v_{i,l} = \varphi(x_i, k) \right\} \right|.$$

For the correct key k^*, we deterministically have that $\gamma_{k^*} = N$. Thus, if no noise is present, the higher-order DCA is successful when $\gamma_{k^\times} < N$ for all $k^\times \neq k^*$. The probability of this happening is quite close to 1, even for small N. Thus, the introduction of some noise in the traces is required to secure a masked white-box implementation against DCA.

4.1 Higher-Order DCA against Masking and Shuffling

We now consider how well the masked and shuffled implementation resists the higher-order DCA attack described above. Due to the shuffling, the adversary is no longer guaranteed that her prediction for the correct key guess will correspond to a single time point for all traces. Thus, she must compensate by increasing the number of traces. The higher the degree of shuffling, the more traces needs to be collected.

4.1.1 Attack Analysis

In the following, we assume that the adversary knows exactly where in the computational trace to attack. That is, for a masking order $d-1$ and a shuffling degree λ, she knows the range of the $t = \lambda \cdot d$ time points that can contain the shares of the target secret variable. In other words, the length of each computational trace $\boldsymbol{v}$ is t. This, intuitively, represents the optimal situation for the adversary.[2] We seek an expression for the success probability of the attack, *i.e.* the probability that the correct key has a higher score than all other key candidates.

The adversary proceeds as above and computes the d'th-order computational trace. However, there will no longer be a single value j such that $C_{k^*}(\boldsymbol{v}_{\phi(j)}, (x_i)_i) = N$ deterministically for the correct key k^*. Thus, we need to know the distribution of γ_k, both for a wrong and a right guess of the key.

Theorem 1. *Consider a masked white-box implementation of order $d-1$ with shuffling degree λ. Let $p = \binom{t}{d}^{-1}$ where $t = \lambda \cdot d$, and let $F(x; n, q)$ be the CDF (cumulative distribution function) of the binomial distribution with parameters n and q. Let $|\mathcal{K}|$ be the number of possible key values and define*

$$F^{\times}_{\max}(x) = F(x; N, (1-p)\tfrac{1}{|\mathcal{K}|})^{\binom{t}{d}},$$
$$F^{*}_{\max}(x) = F(x; N, p + (1-p)\tfrac{1}{|\mathcal{K}|})^{\binom{t}{d}}.$$

Then the probability of recovering a key using d'th-order DCA with N traces is

$$p_{succ} = \left(\sum_{i=0}^{N} (F^{*}_{\max}(i) - F^{*}_{\max}(i-1)) \cdot F^{\times}_{\max}(i-1) \right)^{|\mathcal{K}|-1}.$$

Proof. Consider a specific value w_j of the higher-order trace $\boldsymbol{w}$. Denote by $\mathcal{A}$ the event that w_j corresponds to the combination of the correct shares. The probability of $\mathcal{A}$ occurring, *i.e.* of choosing the correct d shares out of the t elements of the original computational trace $\boldsymbol{v}$, is $p = \binom{t}{d}^{-1}$.

Fix some plaintext and the corresponding trace. By the law of total probability, the probability that a value w_j of the d'th order trace is equal to a prediction $s = \varphi(x, k)$ for some key guess k is

$$\Pr(w_j = s) = \Pr(w_j = s|\mathcal{A}) \cdot \Pr(\mathcal{A}) + \Pr(w_j = s|\neg\mathcal{A}) \cdot \Pr(\neg\mathcal{A}).$$

For a wrong key guess, $k^{\times} \neq k^*$, $\Pr(w_j = s^{\times}|\mathcal{A}) = 0$, while for a right key guess $\Pr(w_j = s^*|\mathcal{A}) = 1$. In both cases, we have $\Pr(w_j = s|\neg\mathcal{A}) = 1/|\mathcal{K}|$. In total:

$$p^{\times} = \Pr(w_j = s^{\times}) = (1-p)/|\mathcal{K}|,$$
$$p^{*} = \Pr(w_j = s^{*}) = p + (1-p)/|\mathcal{K}|.$$

Thus, for N traces,

$$C_{k^{\times}}(\boldsymbol{v}_{\phi(j)}, (x_i)_i) \sim \mathrm{Bin}(N, p^{\times})$$

for a wrong key guess, and

$$C_{k^{*}}(\boldsymbol{v}_{\phi(j)}, (x_i)_i) \sim \mathrm{Bin}(N, p^{*})$$

for a right key guess. Note that $|\boldsymbol{w}| = \binom{t}{d}$. Let $X_1, \ldots, X_{|\boldsymbol{w}|}$ be distributed as $\mathrm{Bin}(N, p^{\times})$. Then $\gamma_{k^{\times}} \sim \max X_i$, and we denote the CDF by $F^{\times}_{\max}(x)$. If the X_i were independent, we would have

$$F^{\times}_{\max}(x) = F(x; N, p^{\times})^{\binom{t}{d}}.$$

[2] In practice, the adversary could exhaustively search the correct location of the $(\lambda \cdot d)$-length subtrace in the full computation trace of length t_{full}, which increases the complexity at most t_{full} times.

While the X_i are pairwise independent, they are not mutually independent. However, we find that in practice, the dependence is so weak that $\gamma_{k^\times}$ approximately has CDF $F^{\times}_{\max}$, even for small values of $|\boldsymbol{w}|$ and N. We define $F^{*}_{\max}(x)$ similarly.

The attack is successful if $\gamma_{k^*} > \gamma_{k^\times}$ for all $k^\times$. As there are $|\mathcal{K}| - 1$ wrong keys, and all $\gamma_{k^\times}$ are independent and identically distributed, we have $p_{\text{succ}} = \Pr(\gamma_{k^*} > \gamma_{k^\times})^{|\mathcal{K}|-1}$, where

$$\Pr(\gamma_{k^*} > \gamma_{k^\times}) = \sum_{i=0}^{N} (F^{*}_{\max}(i) - F^{*}_{\max}(i-1)) \cdot F^{\times}_{\max}(i-1).$$

which concludes the proof. □

We can use this formula to calculate the required N to obtain a desired probability of success. The number of traces required to obtain 90% success probability for a range of parameters is show in Table 1. Here, $|\mathcal{K}| = 256$, and the parameters would be typical choices for *e.g.* a protected AES implementation.

Table 1: The number of traces N and the time needed to successfully attack an implementation with $(d-1)$-order masking and shuffling of degree λ with d'th-order DCA. Here, $|\mathcal{K}| = 256$, and we fix the success probability at 90%. The parameters chosen would be typical for a protected AES implementation.

d	λ	$\log_2 N$	$\log_2$ time	d	λ	$\log_2 N$	$\log_2$ time	d	λ	$\log_2 N$	$\log_2$ time
2	8	8.6	23.5	3	8	15.7	34.7	4	8	23.6	46.7
2	16	11.0	28.0	3	16	21.6	43.7	4	16	31.7	59.0

4.1.2 Attack Complexity

We consider the time complexity of recovering the secret key k^* using the higher-order DCA attack. For a fixed probability of success p_{succ}, let N_d be the number of computational traces required to obtain this probability for a d'th-order implementation. We again assume that $t = \lambda \cdot d$. The cost of computing the higher-order trace is $N_d \cdot \binom{t}{d}$. Then, for each key guess k and each time point in the higher-order trace, the adversary computes C_k. The complexity of this is $|\mathcal{K}| \cdot N_d \cdot \binom{t}{d}$. Thus, the time complexity is $\mathcal{O}\left(|\mathcal{K}| \cdot N_d \cdot \binom{t}{d}\right)$. Table 1 shows the time complexity of the attack for a range of parameters.

5 Multivariate Higher-Order DCA

In the higher-order DCA, presented in Section 4, the adversary tries to correlate each sample of the higher-order trace with the predicted variable independently, finally taking the maximum over the obtained correlation scores. Such an approach is not optimal, as successive samples may carry joint information on the secret. As in the side-channel context, one can take advantage of this joint information by performing a *multivariate attack*, namely an attack in which the distinguisher exploits the multivariate distribution of different samples in the higher-order trace. Emblematic multivariate attacks in the classical side-channel context are the so-called *template attacks* [CRR02]. In the following section, we describe a similar attack in the setting of the DCA adversary.

5.1 Multivariate Higher-Order DCA against Masking and Shuffling

Our proposed multivariate higher-order DCA attack is based on the principle of maximum likelihood. Similar techniques have been adopted in side-channel template attacks. Let K, $(X_i)_i$, and $(\boldsymbol{V}_i)_i$ be random variables representing the subkey k^*, the public inputs $(x_i)_i$, and the computational traces $(\boldsymbol{v}_i)_i$. The likelihood distinguisher is then defined as

$$\begin{aligned} &\mathsf{L} : \left((\boldsymbol{v}_i)_i, (x_i)_i\right) \mapsto (\ell_k)_{k \in \mathcal{K}}, \\ &\ell_k \propto \Pr\left(K = k \mid (\boldsymbol{V}_i)_i = (\boldsymbol{v}_i)_i \wedge (X_i)_i = (x_i)_i\right), \end{aligned} \tag{1}$$

where $\propto$ means equal up to some factor constant w.r.t. k. To evaluate this likelihood function, we need a model for the distribution of the traces (also called a *template* in the side-channel context). It is well

known that if Equation 1 is evaluated from the true distributions of $(X_i)_i$ and $(\boldsymbol{V}_i)_i$, then the above distinguisher is optimal. This is sounds, as in this case, the score is the exact probability that the target subkey equals a key guess k, for all $k \in \mathcal{K}$.

In the following, we will assume that $\boldsymbol{V}_i$ is composed of t uniformly distributed random variables $V_{i,1}$, $V_{i,2}$, ..., $V_{i,t}$, with the constraint that for a uniformly chosen j, we have $\bigoplus_{l \in \phi(j)} V_{i,l} = \varphi(X_i, K)$. This assumption matches the setting of a masked and shuffled implementation. The public inputs X_i and the subkey K are also assumed to be uniformly distributed and mutually independent. Under this model, we have the following result (see proof in Appendix A):

Proposition 1. *The likelihood distinguisher, Equation 1, satisfies:*

$$\ell_k \propto \prod_{i=1}^{N} C_k(\boldsymbol{v}_i, x_i),$$

where $C_k(\boldsymbol{v}, x)$ is the number of d-tuples in a trace $\boldsymbol{v}$ with bitwise sum equals to $\varphi(x, k)$, that is

$$C_k(\boldsymbol{v}, x) = \left|\left\{(v_{j_1}, \ldots, v_{j_d}) \ ; \ v_{j_1} \oplus \cdots \oplus v_{j_d} = \varphi(x, k)\right\}\right| .$$

Remark 1. For practical reasons, it is more convenient to evaluate the log-likelihood, that is

$$\log \ell_k = \sum_{i=1}^{N} \log C_k(\boldsymbol{v}_i, x_i) \ .$$

Note that this does not affect the ranking of the key guesses (as the logarithm is a monotonically increasing function) and therefore has no impact on the success probability of the attack.

5.2 Analysis of the Likelihood Distinguisher

In this section we analyse the success probability of the likelihood distinguisher. For the sake of simplicity, we only consider two key guesses, namely a right key guess k^* and a wrong key guess $k^\times$. We then consider their likelihood scores ℓ_{k^*} and $\ell_{k^\times}$ random variables, since

$$\ell_k = \prod_{i=1}^{N} C_k(\boldsymbol{V}_i, X_i) \ ,$$

for $k \in \{k^*, k^\times\}$, where $(\boldsymbol{V}_i)_i$ and $(X_i)_i$ are the random variables defined above for the computational traces and the corresponding public inputs. We then consider the probability $p_{\text{succ}} = \Pr\left(\ell_{k^*} > \ell_{k^\times}\right)$. In the following, we give a proof of the following result.

Theorem 2. *For a multivariate d'th-order DCA attack using the likelihood distinguisher on N traces of length t, the probability that a correct key guess is ranked higher than an incorrect key guess is approximately given by*

$$p_{succ} \approx p_{\mathcal{U}} + (1 - p_{\mathcal{U}}) \left(\frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{\sqrt{N|\mathcal{V}|}}{2\sqrt{q}}\right)\right)$$

where $q = \binom{t}{d}$ and $p_{\mathcal{U}} = 1 - \left(1 - \left(1 - |\mathcal{V}|^{-1}\right)^q\right)^N$.

The total success probability of the attack $p_{\text{full-succ}}$, *i.e.* the probability that the correct key guess has the largest likelihood, is then heuristically $p_{\text{full-succ}} \approx p_{\text{succ}}^{|\mathcal{K}|-1}$. Moreover, it can be checked that $p_{\mathcal{U}} \approx N \cdot \left(1 - |\mathcal{V}|^{-1}\right)^q$ becomes negligible as q grows. Theorem 2 then implies

$$p_{\text{succ}} = \Theta\left(\text{erf}\left(\frac{\sqrt{N|\mathcal{V}|}}{2\sqrt{q}}\right)\right) \ ,$$

from which we deduce that the data complexity of the attack is

$$N = \Theta(q) \ .$$

Namely, the number of required traces N to achieve certain p_{succ} is linear in the number of combinations $q = \binom{t}{d}$. We also have $N = \Theta(q/|\mathcal{V}|)$ to make appear the impact of the definition set $\mathcal{V}$.

In order to prove Theorem 2, we introduce the concept of the *zero-counter event*. Denoted by $\mathcal{U}_k$, this is the event that $C_k(\boldsymbol{v}_i, x_i) = 0$ for at least one $i \in [1, N]$ for a key guess k. Note that this event can never happen for $k = k^*$, since for all i, there exists a j such that $\bigoplus_{l \in \phi(j)} \boldsymbol{v}_{i,l} = \varphi(x_i, k^*)$. Thus, $\Pr(\ell_{k^*} > \ell_{k^\times} \mid \mathcal{U}_{k^\times}) = 1$, since in this case the likelihood $\ell_{k^\times}$ equals zero (or equivalently, the log-likelihood equals $-\infty$). This is intuitively sound, as the right key guess could not give rise to a zero counter for any of the N computational traces. Then, by the law of total probability, we can write

$$p_{\text{succ}} = \Pr(\mathcal{U}_{k^\times}) + \Pr(\neg\, \mathcal{U}_{k^\times}) \cdot \Pr(\ell_{k^*} > \ell_{k^\times} \mid \neg\, \mathcal{U}_{k^\times}). \tag{2}$$

We are therefore interested in the probabilities $\Pr(\mathcal{U}_{k^\times})$ and $\Pr(\ell_{k^*} > \ell_{k^\times} \mid \neg\, \mathcal{U}_{k^\times})$. These are given in the following lemmata.

Lemma 1. *Given N traces of length t, the probability of the zero-counter event for a wrong key guess $k^\times$ in a d'th-order attack is approximately given by*

$$\Pr(\mathcal{U}_{k^\times}) \approx 1 - \left(1 - \left(1 - |\mathcal{V}|^{-1}\right)^q\right)^N,$$

where $q = \binom{t}{d}$.

Lemma 2. *Given N traces of length t, let $q = \binom{t}{d}$, and assume that the zero-counter event does not occur. The probability that a correct key guess has a higher likelihood score than a wrong key guess in a d'th-order attack is approximately*

$$\Pr(\ell_{k^*} > \ell_{k^\times} \mid \neg\, \mathcal{U}_{k^\times}) \approx \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{\sqrt{N|\mathcal{V}|}}{2\sqrt{q}}\right).$$

We prove Lemma 1 in Section 5.3 and Lemma 2 in Section 5.4. Theorem 2 then follows directly from Equation 2 and these two results.

5.3 Probability of the Zero-Counter Event (Proof of Lemma 1)

We first define $\mathcal{Z}_k$ as the zero-counter event for key k for a *single* computational trace $\boldsymbol{V}$. Formally,

$$\mathcal{Z}_k \;=\; \text{“}\;\; \forall j \subseteq \left\{1, \ldots, \binom{t}{d}\right\} \;:\; \bigoplus_{i \in \phi(j)} V_i \neq \varphi(X, k) \;\text{”}.$$

The zero-counter event $\mathcal{Z}_k$ occurs if and only if none of the $q = \binom{t}{d}$ combinations $\bigoplus_{i \in \phi(j)} V_i$ match the predicted value $\varphi(X, k)$. As discussed, $\mathcal{Z}_{k^*}$ never occurs for the correct key guess k^*. For the incorrect key guess $k^\times$, intuitively, the zero-counter probability $\Pr(\mathcal{Z}_{k^\times})$ should quickly become negligible as the number of combinations q grows. While all q combinations are not strictly independent, we can approximate the probability of $\mathcal{Z}_{k^\times}$ by:

$$\Pr(\mathcal{Z}_{k^\times}) \approx \left(1 - \frac{1}{|\mathcal{V}|}\right)^q. \tag{3}$$

We verified this approximation by estimating the zero-counter probability over some sampled computation traces. As illustrated it Table 2, the obtained estimations match the approximation pretty well.

Table 2: Approximation and estimation of the zero-counter probability.

(t, d)	(16,2)	(16,3)	(16,4)	(24,2)	(24,3)	(32,2)	(32,3)
Approximation (3)	0.625	0.112	$8{\cdot}10^{-4}$	0.340	$4{\cdot}10^{-4}$	0.144	$4{\cdot}10^{-9}$
Estimation (prec. $\sim 10^{-3}$)	0.628	0.135	$< 10^{-3}$	0.342	$< 10^{-3}$	0.145	$< 10^{-3}$

Then, by definition, the zero-counter event for N traces is the union

$$\mathcal{U}_k = \mathcal{Z}_k^{(1)} \vee \mathcal{Z}_k^{(2)} \vee \cdots \vee \mathcal{Z}_k^{(N)},$$

where $\mathcal{Z}_k^{(i)}$ denotes the zero-counter event for k on trace $\boldsymbol{V}_i$. Taking the negation we obtain $\neg\,\mathcal{U}_{k^\times} = (\neg\mathcal{Z}_k^{(1)}) \wedge (\neg\mathcal{Z}_k^{(2)}) \wedge \cdots \wedge (\neg\mathcal{Z}_k^{(N)})$, and since the zero events $\mathcal{Z}_{k^\times}^{(i)}$ are mutually independent, we get

$$\Pr(\mathcal{U}_{k^\times}) = 1 - \prod_{i=1}^{N} \Pr(\neg\mathcal{Z}_{k^\times}^{(i)}) = 1 - \big(1 - \Pr(\mathcal{Z}_{k^\times})\big)^N.$$

This finishes the proof of Lemma 1.

5.4 Success Probability with no Zero Counters (Proof of Lemma 2)

If the zero counter event does not occur, we can think of each trace $\boldsymbol{V}_i$ as a random variable uniformly distributed over $\mathcal{V}^t$. Since the public input X_i is also random, the counters $C_k(\boldsymbol{V}, X)$ follow some probability distribution. In order to prove Lemma 2, we first prove the following result regarding these distributions.

Lemma 3. *Let k^* and $k^\times$ be a right and wrong key guess. Let $q = \binom{t}{d}$ and $\kappa = (q-1)\frac{1}{|\mathcal{V}|}$. Then for a trace of length t and a d'th-order attack,*

$$C_{k^*}(\boldsymbol{V}, X) \sim \mathcal{N}(\kappa + 1, \kappa) \quad \text{and} \quad C_{k^\times}(\boldsymbol{V}, X) \sim \mathcal{N}(\kappa, \kappa),$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes the normal distribution with mean μ and variance σ^2.

Proof. Let $\delta : \mathcal{V}^2 \to \{0, 1\}$ be the function defined as

$$\delta(v_1, v_2) = \begin{cases} 1 & \text{if } v_1 = v_2, \\ 0 & \text{otherwise.} \end{cases}$$

The counter $C_k(\boldsymbol{V}, X)$ can be rewritten as a sum $C_k(\boldsymbol{V}, X) = \sum_{j=1}^{q} \delta(W_j, \varphi(X, k))$, where the variables $(W_j)_j$ are defined as the $q = \binom{t}{d}$ combinations $\bigoplus_{i\in\phi(j)} V_i$. We recall that for one index j we have $W_j = \varphi(X, k^*)$, whereas for the other indices the W_j are randomly distributed independently of X. The counter expectation then satisfies

$$\mathrm{E}\big(C_k(\boldsymbol{V}, X)\big) = \sum_{j=1}^{q} \mathrm{E}\big(\delta(W_j, \varphi(X, k))\big) = \begin{cases} (q-1)\frac{1}{|\mathcal{V}|} & \text{if } k \neq k^*, \\ (q-1)\frac{1}{|\mathcal{V}|} + 1 & \text{if } k = k^*. \end{cases}$$

On the other hand, the counter variance can be expressed as:

$$\begin{aligned} \mathrm{Var}\big(C_k(\boldsymbol{V}, X)\big) = &\sum_{j=1}^{q} \mathrm{Var}\big(\delta(W_j, \varphi(X, k))\big) \\ &+ 2 \sum_{1\leq j<j'\leq q} \mathrm{Cov}\big(\delta(W_j, \varphi(X, k)), \delta(W_{j'}, \varphi(X, k))\big). \end{aligned}$$

It can be checked that the covariances will be equal to 0 most of the time. Indeed, the covariances are non-zero only when $W_j \oplus W_{j'} = \varphi(X, k^*)$, which never happens when d is odd and which happens for few pairs (j, j') when d is even. Therefore these covariance terms will only have a small impact on the overall variance. Moreover, it can be checked that this impact is negative, *i.e.* it reduces the variance.[3] Therefore we will ignore the sum of covariances, which yields a correct result when d is odd and a slight overestimation when d is even. We then have

$$\mathrm{Var}\big(\delta(W_j, \varphi(X, k))\big) = \begin{cases} \frac{1}{|\mathcal{V}|}\big(1 - \frac{1}{|\mathcal{V}|}\big) & \text{if } j \neq j^*, \\ 0 & \text{if } j = j^*, \end{cases}$$

[3] Most of the time we have $\varphi(X, k^*) \neq 0$ so that the pairs (j, j') with $W_j \oplus W_{j'} = \varphi(X, k^*)$ are such that $W_j \neq W_{j'}$ with high probability. In that case $\delta(W_j, \varphi(X, k)) = 1$ implies $\delta(W_{j'}, \varphi(X, k)) = 0$ and conversely which yields a negative covariance.

where j^* denotes the index of the right combination matching $\varphi(X, k^*)$. Combining the two above equations gives:

$$\mathrm{Var}\big(C_k(\boldsymbol{V}, X)\big) = (q-1)\frac{1}{|\mathcal{V}|}\Big(1 - \frac{1}{|\mathcal{V}|}\Big) \approx (q-1)\frac{1}{|\mathcal{V}|}.$$

Since the counter is defined as a sum of *somewhat independent* random variables, we can soundly approximate its distribution by a Gaussian, and setting $\kappa = (q-1)\frac{1}{|\mathcal{V}|}$ concludes the proof. □

In the above proof, we use that the $\delta(W_j, \varphi(X, k))$ are somewhat independent. By *somewhat independent* we mean that these variables are pairwise independent (for most or all of them, as discussed). Note that variants of the central limit theorem exist that take some form of dependence between the summed variables into account. We have experimentally verified that the Gaussian approximation is sound for various parameters (t, d).

Using Lemma 3, we can now prove Lemma 2. Following Remark 1, we will focus on the log-likelihood, *i.e.* we consider

$$\begin{aligned}\Pr(\ell_{k^*} > \ell_{k^\times} \mid \neg\, \mathcal{U}_{k^\times}) &= \Pr\big(\log \ell_{k^*} - \log \ell_{k^\times} > 0 \mid \neg\, \mathcal{U}_{k^\times}\big),\\ \log \ell_{k^*} - \log \ell_{k^\times} &= \sum_{i=1}^{N} \underbrace{\log C_{k^*}(\boldsymbol{V}_i, X_i) - \log C_{k^\times}(\boldsymbol{V}_i, X_i)}_{Y_i}\ .\end{aligned}$$

As introduced above, we denote by Y_i the difference between the log-counters for the trace $\boldsymbol{V}_i$. Since the Y_i are mutually independent and identically distributed, the central limit theorem implies that, for N sufficiently large,

$$\frac{1}{N}(\log \ell_{k^*} - \log \ell_{k^\times}) \sim \mathcal{N}\big(\mu_Y, \sigma_Y^2 N^{-1}\big) \quad \text{with} \quad \begin{cases}\mu_Y = \mathrm{E}(Y),\\ \sigma_Y^2 = \mathrm{Var}(Y),\end{cases}$$

for $Y = \log C_{k^*}(\boldsymbol{V}, X) - \log C_{k^\times}(\boldsymbol{V}, X)$. Thus

$$\Pr(\ell_{k^*} > \ell_{k^\times} \mid \neg\, \mathcal{U}_{k^\times}) = 1 - \Phi_{\mu_Y, \sigma_Y^2/N}(0) = \frac{1}{2} + \frac{1}{2}\mathrm{erf}\Big(\frac{\sqrt{N}\,\mu_Y}{\sqrt{2}\,\sigma_Y}\Big), \tag{4}$$

where $\Phi_{\mu,\sigma}$ is the CDF of $\mathcal{N}(\mu, \sigma^2)$. By the heuristic assumption that $C_{k^*}(\boldsymbol{V}, X)$ and $C_{k^\times}(\boldsymbol{V}, X)$ are mutually independent, and using the Taylor expansion of the logarithm at $\mathrm{E}(C)$, as well as Lemma 3, we have

$$\mu_Y \approx \log(\kappa+1) - \frac{\kappa}{2(\kappa+1)^2} - \log\kappa + \frac{\kappa}{2\kappa^2} \approx \frac{1}{\kappa}, \quad \text{and} \quad \sigma_Y^2 \approx 2\frac{\kappa}{\kappa^2} = \frac{2}{\kappa},$$

where the approximation of the mean is sound if κ is large enough (*e.g.* $\kappa > 10$). Inserting these approximations into Equation 4, remembering that $\kappa = (q-1)\frac{1}{|\mathcal{V}|} \approx \frac{q}{|\mathcal{V}|}$, finishes the proof.

6 Experimental Verification and Security Evaluation

The proof of Theorem 2 relies on a number of approximations. We therefore verified the accuracy of the estimate by simulating the multivariate higher-order DCA attack for various choices of the parameters d and t. We chose to simulate traces of a masked and shuffled AES implementation, that is, the target secret variable was taken to be $\varphi(x, k^*) = \mathsf{Sbox}_{\mathrm{AES}}(x \oplus k^*)$. The computational traces were generated according to the model described at the beginning of Section 5.1, namely by sampling random values v_j over $\mathcal{V} = \mathbb{F}_{2^8}$ with the constraint that one randomly chosen d-tuple of each trace has XOR-sum $\varphi(x, k^*)$.

We generated traces for $d \in \{2, 3, 4\}$ and $t \in \{8, 16, 24, 32, 40, 48, 56, 64, 72\}$, and calculated the log-likelihood scores for the correct key and a randomly chosen wrong key. This was repeated 2 000 times, and the probability $\Pr(\ell_{k^*} > \ell_{k^\times})$ was calculated for varying values of N. The results are show in Figure 3. The figure shows that the estimate of Theorem 2 is quite accurate in most cases, only deviating from the experimental measurements for very small values of $q = \binom{t}{d}$ ($q < 300$). Note that in practice,

this would rarely be a problem. For example, if all shares of the full AES state were shuffled in a first order masked implementation, as described in Section 3.3, the smallest trace that would always contain the correct shares would have $q = \binom{2 \cdot 16}{2} = 496$.

The attack complexity of the multivariate higher-order DCA is the same as that of the higher-order DCA, namely $\mathcal{O}\left(|\mathcal{K}| \cdot N_d \cdot \binom{t}{d}\right)$. Using this and Theorem 2 we can provide an estimate of the security level obtained by a masked and shuffled implementation against the DCA adversary. Table 3 shows the complexities of attacking *e.g.* a protected AES implementation where the operations are shuffled amongst all 16 state bytes (the shuffling degree is $\lambda = 16$ implying $t = 16 \cdot d$). When fixing $p_{\text{full-succ}}$ at 90%, we see that an implementation which uses 7^{th} order masking will obtain an estimated security level of 85 bits.

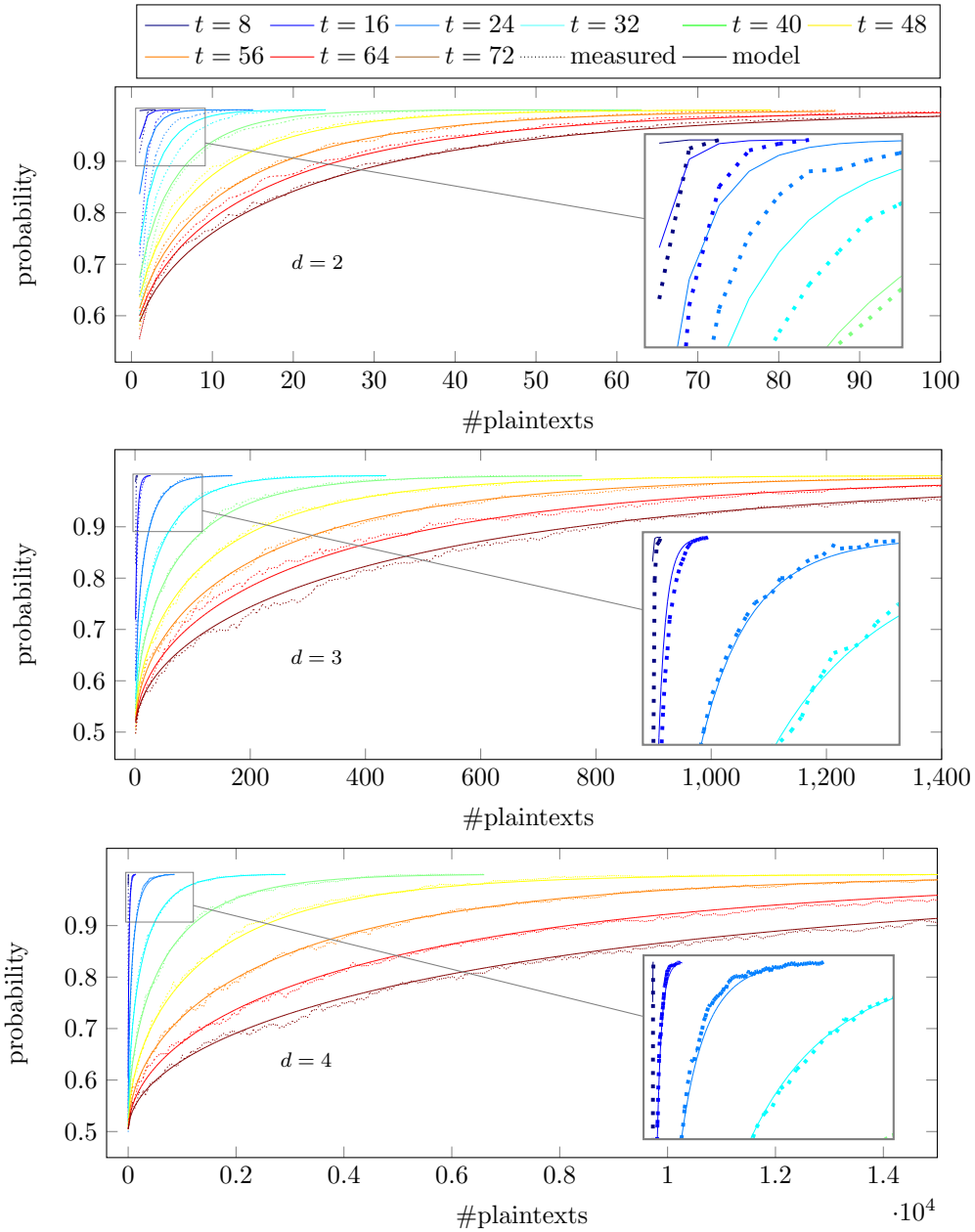

Figure 3: The measured probability of ranking a correct key higher than an incorrect key in the multivariate higher-order DCA attack, compared to Theorem 2. The measurement is based on 2 000 simulations of the attack. Here, d is the attack order and t is the length of the obtained traces.

Table 3: The number of traces N and the time needed to successfully attack an implementation with $(d-1)$-order masking and shuffling of degree $\lambda = 16$ ($t = 16 \cdot d$) using multivariate d'th-order DCA . Here, $|\mathcal{K}| = 256$, and we fix the success probability at 90%.

d	$\log_2 N$	$\log_2$ time	d	$\log_2 N$	$\log_2$ time	d	$\log_2 N$	$\log_2$ time
3	10.6	32.7	5	21.0	53.5	7	31.6	74.6
4	15.8	43.1	6	26.3	64.1	8	36.9	85.3

References

[BCD06a] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. Perturbing and Protecting a Traceable Block Cipher. In *Communications and Multimedia Security, 10th IFIP TC-6 TC-11 International Conference, CMS 2006*, pages 109–119, 2006.

[BCD06b] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White Box Cryptography: Another Attempt. *IACR Cryptology ePrint Archive*, 2006:468, 2006.

[BGE04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In *Selected Areas in Cryptography, 11th International Workshop, SAC 2004*, pages 227–240, 2004.

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough. In *Cryptographic Hardware and Embedded Systems - CHES 2016*, pages 215–236, 2016.

[BIT16] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards Practical Whitebox Cryptography: Optimizing Efficiency and Space Hardness. In *Advances in Cryptology - ASIACRYPT 2016*, pages 126–158, 2016.

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. Cryptology ePrint Archive, Report 2018/049, 2018. https://eprint.iacr.org/2018/049.

[CEJvO02a] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A White-Box DES Implementation for DRM Applications. In *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002*, pages 1–15, 2002.

[CEJvO02b] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-Box Cryptography and an AES Implementation. In *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002*, pages 250–270, 2002.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology - CRYPTO '99*, pages 398–412, 1999.

[CK10] Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and Improvement of the Random Delay Countermeasure of CHES 2009. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 95–109, 2010.

[Cor14] Jean-Sébastien Coron. Higher Order Masking of Look-Up Tables. In *Advances in Cryptology - EUROCRYPT 2014*, pages 441–458, 2014.

[CPRR13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-Order Side Channel Security and Mask Refreshing. In *Fast Software Encryption - 20th International Workshop, FSE 2013*, pages 410–424, 2013.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, 2002.

[DLPR13] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-box security notions for symmetric encryption schemes. In *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, pages 247–264, 2013.

[FY+38] Ronald Aylmer Fisher, Frank Yates, et al. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research.*, 1938.

[GP99] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis (The "Duplication" Method). In *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99*, pages 158–172, 1999.

[GPRW18] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. Cryptology ePrint Archive, Report 2018/098, 2018. https://eprint.iacr.org/2018/098.

[Kar10] Mohamed Karroumi. Protecting White-Box AES with Dual Ciphers. In *Information Security and Cryptology - ICISC 2010*, pages 278–291, 2010.

[Ker83] Auguste Kerckhoffs. La Cryptographic Militaire. *Journal des Sciences Militaires*, pages 5–38, 1883.

[LRM+13] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two Attacks on a White-Box AES Implementation. In *Selected Areas in Cryptography - SAC 2013*, pages 265–285, 2013.

[MGH08] Wil Michiels, Paul Gorissen, and Henk D. L. Hollmann. Cryptanalysis of a Generic Class of White-Box Implementations. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008*, pages 414–428, 2008.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[MRP12] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao - Lai White-Box AES Implementation. In *Selected Areas in Cryptography, 19th International Conference, SAC 2012*, pages 34–49, 2012.

[MWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a Perturbated White-Box AES Implementation. In *Progress in Cryptology - INDOCRYPT 2010*, pages 292–310, 2010.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 413–427, 2010.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 171–188, 2009.

[SP11] Daehyun Strobel and Christof Paar. An Efficient Method for Eliminating Random Delays in Power Traces of Embedded Software. In *Information Security and Cryptology - ICISC 2011*, pages 48–60, 2011.

[Tol12] LMG Tolhuizen. Improved cryptanalysis of an aes implementation. In *Proceedings of the 33rd WIC Symposium on Information Theory, 2012*. WIC (Werkgemeenschap voor Inform.-en Communicatietheorie), 2012.

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note. In *Advances in Cryptology - ASIACRYPT 2012*, pages 740–757, 2012.

[whi] CHES 2017 Capture the Flag Challenge - The WhibOx Contest, An ECRYPT White-Box Cryptography Competition. https://whibox.cr.yp.to/. Accessed: October 2017.

[XL09] Yaying Xiao and Xuejia Lai. A secure implementation of white-box aes. In *Computer Science and its Applications, 2009. CSA'09*, pages 1–6. IEEE, 2009.

A Proof of Proposition 1

Proof. By applying the Bayes' rule, one gets (we skip random variables for the sake of clarity):

$$\Pr\left(k \mid (\boldsymbol{v}_i)_i \wedge (x_i)_i\right) = \frac{\Pr\left((\boldsymbol{v}_i)_i \mid k \wedge (x_i)_i\right) \cdot \Pr\left(k \wedge (x_i)_i\right)}{\Pr\left((\boldsymbol{v}_i)_i \wedge (x_i)_i\right)} \tag{5}$$

By mutual independence of the X_i's and K, we have $\Pr\left(k \wedge (x_i)_i\right) = \frac{1}{|\mathcal{K}|}\left(\frac{1}{|\mathcal{X}|}\right)^N$ for every $k \in \mathcal{K}$. Moreover, $\Pr\left((\boldsymbol{v}_i)_i \wedge (x_i)_i\right)$ is constant with respect to k. We hence get

$$\Pr\left(k \mid (\boldsymbol{v}_i)_i \wedge (x_i)_i\right) \propto \Pr\left((\boldsymbol{v}_i)_i \mid k \wedge (x_i)_i\right) . \tag{6}$$

By mutual independence of the $\boldsymbol{V}_i$'s and the X_i's we further deduce

$$\Pr\left((\boldsymbol{v}_i)_i \mid k \wedge (x_i)_i\right) = \prod_{i=1}^{N} \Pr(\boldsymbol{v}_i \mid k \wedge x_i) . \tag{7}$$

For the sake of simplicity we skip the index i in the following. By the law of total probability, we have

$$\Pr(\boldsymbol{v} \mid k \wedge x) \ = \sum_{\phi(j)} \Pr(\mathcal{S}_{\phi(j)}) \cdot \Pr(\boldsymbol{v} \mid k \wedge x \wedge \mathcal{S}_{\phi(j)}) , \tag{8}$$

where $\mathcal{S}_{\phi(j)}$ denotes the event that the set $\phi(j)$ is selected for the sharing of $\varphi(X, K)$. By definition, we have

$$\Pr(\mathcal{S}_{\phi(j)}) = \frac{1}{\binom{t}{d}} \tag{9}$$

and

$$\Pr(\boldsymbol{v} \mid k \wedge x \wedge \mathcal{S}_{\phi(j)}) = \begin{cases} \left(\frac{1}{|\mathcal{V}|}\right)^{t-1} & \text{if } \bigoplus_{l \in \phi(j)} v_l = \varphi(x, k) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

which finally gives

$$\Pr(\boldsymbol{v} \mid k \wedge x) \propto C_k(\boldsymbol{v}, x) . \tag{11}$$