Dummy Shuffling against Algebraic Attacks in White-box Implementations*†‡

Alex Biryukov[1] and Aleksei Udovenko[2]

[1] DCS and SnT, University of Luxembourg
alex.biryukov@uni.lu
[2] CryptoExperts, Paris, France
aleksei@affine.group

October 8, 2021

Abstract. At CHES 2016, Bos, Hubain, Michiels and Teuwen showed that most of existing white-box implementations are easily broken by standard side-channel attacks. A natural idea to apply the well-developed side-channel countermeasure - linear masking schemes - leaves implementations vulnerable to linear algebraic attacks which exploit absence of noise in the white-box setting and are applicable for any order of linear masking. At ASIACRYPT 2018, Biryukov and Udovenko proposed a security model (BU-model for short) for protection against linear algebraic attacks and a new *quadratic* masking scheme which is provably secure in this model. However, countermeasures against higher-degree attacks were left as an open problem.

In this work, we study the effectiveness of another well-known side-channel countermeasure - shuffling - against linear and higher-degree algebraic attacks in the white-box setting. First, we extend the classic shuffling to include dummy computation slots and show that this is a crucial component for protecting against the algebraic attacks. We quantify and prove the security of dummy shuffling against the linear algebraic attack in the BU-model. We introduce a *refreshing* technique for dummy shuffling and show that it allows to achieve close to optimal protection in the model for arbitrary degrees of the attack, thus solving the open problem of protection against the algebraic attack in the BU-model.

Furthermore, we describe an interesting proof-of-concept construction that makes the slot function public (while keeping the shuffling indexes private). A variant of this construction was used, among other countermeasures, in the challenge #100, one of the three white-box AES challenges from the CHES 2019 CTF / WhibOx 2019 contest that proved to be challenging for the attackers.

Keywords: White-box · Obfuscation · Provable Security · Shuffling · Algebraic Attack

†This work was partly supported by the French FUI-AAP25 IDECYS+ project, by the French ANR-AAPG2019 SWITECH project and by the Luxembourg National Research Fund (FNR) project FinCrypt (C17/IS/11684537).

‡Supporting code for this work is available at github.com/CryptoExperts/EC21-dummy-shuffling

Contents

1 Introduction

White-box model studies security of cryptographic implementations under full control of an adversary. In the seminal works, Chow, Eisen, Johnson and van Oorschot [CEJv03,CEJv02] proposed first white-box implementations of the AES and DES block ciphers, which were later broken with practical attacks [BGEC04, WMGP07]. Further attempts at fixing the implementations did not succeeded. The main idea behind these implementations is to implement the cipher as a network of lookup tables (LUTs) and obfuscate tables by composing them with random encodings. In 2016, Bos, Hubain, Michiels and Teuwen [BHMT16] showed that most existing white-box implementations can be defeated with classic correlation attacks known from side-channel analysis. The adaptation of the attack to the white-box model was called *Differential Computation Analysis* (DCA). More recently, Rivain and Wang [RW19] showed that any table-based encoding of LUTs is always susceptible to the DCA attack, possibly applied to a later round.

The DCA attack can be fully automated and is easy to mount. Therefore, a natural question is how to protect white-box implementations against the DCA attack. A well-studied countermeasure against correlation attacks is *masking*. The idea is to split sensitive variables in the implementation into pseudorandom shares and perform computations without recombining the shares explicitly. The classic masking schemes are *linear*. While this is not a problem in the side-channel setting (*e.g.* analyzing power measurements) because of large amounts of noise in measurements, it becomes an issue in the white-box setting. Recently, Biryukov and Udovenko [BU18] and Goubin, Paillier, Rivain and Wang [GPRW20] showed that the linear masking countermeasure in the white-box setting can be easily and generically broken using elementary linear algebra. The attack was called *algebraic DCA* in the former and *linear decoding analysis (LDA)* in the latter and was used in a sophisticated multi-stage cryptanalysis of the winning challenge from the CHES 2017 CTF / WhibOx Contest 2017 [PCY+17, GPRW20]. The authors of [BU18] further developed a security model and a *quadratic* masking scheme achieving provable security against the linear algebraic attack. Seker, Eisenbarth and Liskiewicz [SEL21] combined the nonlinear masking scheme with a linear scheme and extended it to a *cubic* masking scheme, offering protection against degree-2 algebraic attacks.

Another known side-channel countermeasure is *shuffling*, inspired by hardware randomization techniques and described by Herbst, Oswald and Mangard [HOM06] and later analyzed in [THM07, RPD09, VMKS12]. The idea is to shuffle the evaluation of identical components (mainly S-boxes) to introduce more noise into measurements. It provides limited security against the correlation attacks by itself and is usually combined with the masking countermeasure. Security of shuffling against the correlation DCA attack in the white-box setting was recently studied by Bogdanov, Rivain, Vejre and Wang [BRVW19]. In addition, Goubin, Rivain and Wang [GRW20] developed *data-dependency higher-order DCA* and used it to cryptanalyze the winning challenges of the CHES 2019 CTF / WhibOx Contest 2019 [BGK+19]. One of the challenges included a shuffling countermeasure, which was defeated by a fault attack.

It can be expected that shuffling provides security against the algebraic attack due to its nonlinearity. However, the algebraic security of shuffling has not yet been evaluated. This work aims to fill this gap and analyzes shuffling rigorously and extensively.

Our contribution

- We show that *classic shuffling* provides weak security against the linear algebraic attack, especially against chosen-plaintext attacks. We describe a simple generalization of the attack called *differential algebraic attack*, which defeats the classic shuffling countermeasure by analyzing *pairs of executions* with well-chosen differences in the inputs. However, we show that the model of [BU18] guarantees protection against

the new *differential* algebraic attack as well, highlighting rigidity of the model.

- We define *dummy shuffling*, which extends the classic shuffling by adding dummy "random" inputs. While the idea of adding dummy operations was already present in previous works, our new definition is the first to emphasize the importance of dummy slots. In addition, we distinguish *hidden* and *public* shuffling, the property which is relevant in the white-box model.

- We prove and quantify security of dummy shuffling against the degree-1 algebraic attack, in the model of [BU18]. We show that it depends on a particular property of the implementation being protected, however this property is hard to evaluate. To overcome this problem, we introduce a novel *refreshing* technique, that transforms any implementation into an equivalent one, but with the relevant property being known and optimal, leading to provable security against linear algebraic attacks.

- We prove that such "refreshed" implementations in fact provide protection against algebraic attacks of *any degree* up to the amount of dummy slots used. The degree bound is tight as shown by our generic higher-degree attack. As a result, we obtain the first provable method of protection against algebraic attacks of arbitrary (predetermined) degree. Our main result is stated in Theorem 3. Surprisingly, our new protection has quite low complexity, as illustrated in Table 1.

- We describe an interesting proof-of-concept construction of *uniform public dummy shuffling*. In this construction, shuffling is done implicitly by calling a single slot function with an extra "index" argument. This construction shows that a white-box designer needs only to obfuscate a single slot function, rather than the whole shuffling process and evaluation of all the slots. A variant of this construction was used, among other countermeasures, in challenge #100, one of the only three challenges from the CHES 2019 CTF / WhibOx 2019 contest [BGK+19] that stayed unbroken during the competition, although cryptanalyzed later by Goubin, Rivain and Wang [GRW20] using non-algebraic attacks.

To summarize, our work provides extensive analysis of the dummy shuffling as a countermeasure against algebraic attacks. This proves useful as it turns out to be a solid provably secure protection. We believe that it is a useful tool for protecting white-box implementations against generic attacks.

We remark that this work studies dummy shuffling strictly in the gray-box model of algebraic security of [BU18] and white-box related problems such as white-box-secure pseudorandomness generation, structure hiding, fault protection, etc. are out of scope for this paper.

2 The framework

In this section, we fix the notation, recall necessary preliminaries and the framework of white-box algebraic attacks.

We write $:=$ to note that the equation holds by definition. For $a \leq b$ integers, the sequence $(a, a+1, \ldots, b-1, b)$ is denoted by $[a \ldots b]$. The finite field of size 2 is denoted by $\mathbb{F}_2$, and the n-dimensional vector space over $\mathbb{F}_2$ is denoted by $\mathbb{F}_2^n$. Vectors/sequences are written as $v = (v_1, v_2, \ldots, v_n)$. The symbol $||$ denotes concatenation of vectors/sequences. $|X|$ denotes the size of the vector/set X, or weight of the Boolean function X, or the number of computed functions in the implementation X. $\mathbf{0}, \mathbf{1}$ denote constant Boolean functions. The *bias* of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is given by $\mathcal{E}(f) := |f|/2^n - 1/2$, and the *error* of f is given by $\mathsf{err}(f) := \min(|f|, |f \oplus 1|)/2^n = 1/2 - |\mathcal{E}(f)|$. The Kronecker delta function $[x = y] : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2$ is a Boolean function that is equal to 1 if and only

Table 1: Estimation of gate complexity for protections against algebraic attacks per original AND/XOR gate. \$ stands for one random bit generation. The error bound τ is a security parameter (larger is more secure). Instances from [SEL21] are created with minimal order of linear masking ($n = 1$). The parameter t is an arbitrary integer greater or equal than the protection degree.

Protection degree	XOR	AND	Error τ	Ref.
1	$33 + 6\$$	$43 + 6\$$	1/16	[BU18, Alg. 3]
1	7	$16 + 2\$$	1/16	[SEL21]
1 $(t = 1)$	2	$8 + 1\$$	1/8	Section 5
2	16	$46 + 3\$$	1/4096	[SEL21]
2 $(t = 2)$	3	$14 + 2\$$	1/48	Section 5
d $(t \geq d)$	$t + 1$	$(6t + 2) + t\$$	$\frac{t+1-d}{t+1} \cdot \frac{1}{2^{2d}}$	Section 5

if $x = y$; its complement is denoted by $[x \neq y]$. For a Boolean function $f(x_1, \ldots, x_t)$ we denote its restriction to $x_i = c$ by $f|_{x_i=c}$. Every Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be uniquely written in the algebraic normal form (ANF): $f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u x^u$, where $a_u \in \mathbb{F}_2$ and x^u is a shorthand for $x_1^{u_1} \ldots x_n^{u_n}$. The *algebraic degree* (or simply *degree*) of f, denoted $\deg f$, is the maximum Hamming weight of all u with $a_u = 1$.

2.1 Implementations and computational traces

In this work, we do not restrict our *analysis* to any particular type of implementations (*e.g.* Boolean circuits or programs), even though our *constructions* are most naturally and generally expressed as Boolean circuits. The only requirement for analysis is that an implementation represents a finite sequence of Boolean functions, which can be efficiently evaluated on arbitrary inputs (resulting in a *computational trace*). Note that not all programs are easily expressed in this form due to possibly varying control flow paths on different inputs. However, various techniques for recording and processing (e.g. aligning) computational traces of (compiled) programs are described in the literature [BHMT16, BKMS18]. Our setting is formalized as follows.

Definition 1 (Implementation)**.** An *implementation* is a vectorial Boolean function $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ together with an associated sequence of efficiently computable Boolean functions

$$\mathcal{F}(C) = (\mathcal{F}_i(C) : \mathbb{F}_2^n \to \mathbb{F}_2 \;\mid\; i \in [1 \ldots |C|]) \,.$$

The functions $x \mapsto x_i$ representing the input variables $x \in \mathbb{F}_2^n$ and the output coordinates of C are included in $\mathcal{F}(C)$.

Remark 1. For ease of understanding one can think of C as a Boolean circuit and $\mathcal{F}_i(C)$ as nodes of this circuit. Note that our definition omits data-dependency relations. While out of scope for this work, they can be used to aid higher-order correlation or algebraic attacks by selecting nearby nodes and thus reducing the combinatorial complexity, as was recently shown in [GRW20].

In the context of white-box attacks, an adversary typically analyzes a part of the implementation, for example the first 10% of operations to target the first round of a block cipher. We call such part a *window*.

Definition 2 (Window)**.** Let C be an implementation. A *window* $\mathcal{W}$ is a subsequence of $\mathcal{F}(C)$.

For the correlation/algebraic attacks, an adversary runs the analyzed implementation on a chosen input and records all intermediate computed values inside the chosen window, producing a so-called *computational trace*.

Definition 3 (Computational trace)**.** A *computational trace* of an implementation $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ on a window $\mathcal{W} \subseteq \mathcal{F}(C)$ and on input $x \in \mathbb{F}_2^n$ is the vector $\mathcal{W}(x) := (f(x) \mid f \in \mathcal{W}) \in \mathbb{F}_2^{|\mathcal{W}|}$.

After recording a certain amount of computational traces, the adversary is trying to check whether a chosen *sensitive function* is computed in the implementation. This analysis can be done statistically (correlation attacks) or algebraically (algebraic attacks). A standard example of a *sensitive function* that we will use throughout the paper is an output bit of the S-box in the first round of AES. This function depends on one key byte and the adversary recovers the key byte by matching the correct sensitive function with the traces. More generally, one may also consider an obfuscation-related scenario, where an adversary's goal is to decide whether a given protected implementation computes internally a certain function or not. In order to develop generic protection against such adversaries, we will consider *every* function in the original unprotected implementation to be sensitive. The protection is then required to "hide" all original computations and anything related to them. This is also a standard requirement in the side-channel context of correlation attacks.

2.2 Algebraic attack

We now recall and restate formally the notion of an algebraic attack. In the degree-1 (linear) algebraic attack, the idea is to find a linear combination of functions computed in the analyzed implementation that results in a sensitive function. For example, in an implementation protected by a linear masking scheme, the shares of a sensitive value describe such a linear combination. By utilizing elementary linear algebra, the shares can be located efficiently, given a sufficient amount of computational traces. This allows to avoid the step of *guessing* the locations of shares and thus avoid the combinatorial explosion in the complexity.

Note that it may be possible to find the shares by other methods, for example, by analyzing the implementation structure. Indeed, the attacks against winning challenges of the WhibOx 2017/2019 competitions included analysis of the data-dependency graphs of the implementations [GPRW20, GRW20]. Nonetheless, the current state-of-the-art of white-box implementations struggles to provide security even against *generic, automated* attacks. Thus achieving security against the powerful algebraic attack is already an ambitious goal.

The linear algebraic attack can be naturally extended to higher degrees. The idea is to include products of 2, 3 or more computed functions in the allowed linear combinations. This extension can break *nonlinear* masking schemes, such as quadratic masking proposed in [BU18]. In addition, it can also defeat table-based encodings, since in that case a sensitive value can be computed as a higher-degree function of the exposed encoded value.

We first define the degree-d expansion of a vector, which captures the idea of including products of degree up to d.

Definition 4 (Degree-d expansion and closure)**.** Let x be an n-dimensional vector over a ring K. For an integer $d \geq 1$ define the *degree-d expansion* of x, denoted $\pi_d(x)$, as a concatenation of all products of $0, 1, 2, \ldots, d$ coordinates of x in a fixed order:

$$\begin{aligned}\pi_d(x) := (1) \,||\, x \,||\, (x_{i_1}x_{i_2} \mid 1 \leq i_1 < i_2 \leq n) \,||\, \ldots \\ ||\, (x_{i_1}x_{i_2}\ldots x_{i_d} \mid 1 \leq i_1 < i_2 < \ldots < i_d \leq n)\,.\end{aligned}$$

Let $\mathcal{V}$ be a sequence of Boolean functions with the same domain $\mathbb{F}_2^n$. The degree-d closure of $\mathcal{V}$ [BU18] is defined as:

$$\mathcal{V}^{(d)} := \operatorname{span} c(\pi_d(\mathcal{V})) = \operatorname{span}\left(\{\mathbf{1}\} \cup \{f_1 f_2 \cdots f_d \mid f_1, f_2, \ldots, f_d \in \mathcal{V}\}\right),$$

where c maps a vector to the set of its coordinates[1].

Example 1. Let $\mathcal{V} = (f_1, f_2, f_3)$ for some Boolean functions $f_1, f_2, f_3 : \mathbb{F}_2^n \to \mathbb{F}_2$. Then $\mathcal{V}^{(2)}$ is a vector space of Boolean functions spanned by $\mathbf{1}$, f_1, f_2, f_3, $f_1 f_2$, $f_1 f_3$, $f_2 f_3$.

Example 2. We will usually consider $\mathcal{F}^{(d)}(C)$ for an implementation $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$. This set consists of all degree-d combinations of intermediate functions computed in C. Elements of this set are Boolean functions f mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2$.

Let $\binom{n}{\leq d} := \sum_{i=0}^{d} \binom{n}{i}$. It is easy to see that the length of $\pi_d(x)$ is equal to $\binom{|x|}{\leq d}$. When $n \gg d$, $\binom{n}{\leq d} = n^d/d! + \mathcal{O}(n^{d-1})$. We are now ready to formalize the algebraic attack.

Definition 5 (Algebraic attack)**.** A *degree-d algebraic attack* against an implementation $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ targeting a sensitive function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ consists of the following steps :

1. choose a *window* $\mathcal{W} \subseteq \mathcal{F}(C)$;
2. choose an input vector $\mathbf{x} := (\mathbf{x}_1, \ldots, \mathbf{x}_t) \in (\mathbb{F}_2^n)^t$, where $t := \binom{|\mathcal{W}|}{\leq d} + \epsilon$ for some small integer ϵ;
3. compute on these inputs the t traces $\mathcal{W}(\mathbf{x}_i)$ and their degree-d expansion;
4. compute on these inputs the sensitive function $f(\mathbf{x}_i)$;
5. solve the following linear system in z:

$$\begin{pmatrix} \pi_d(\mathcal{W}(\mathbf{x}_1)) \\ \vdots \\ \pi_d(\mathcal{W}(\mathbf{x}_t)) \end{pmatrix} \times z = \begin{pmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_t) \end{pmatrix}. \tag{1}$$

The attack succeeds if at least one non-trivial solution is found. It is further required that $\mathbf{x}$ is such that the right-hand side of the equation is non-zero.

Remark 2. It is of course trivial to mount a successful algebraic attack on any implementation by choosing f to be one of the computed functions. What we are actually interested in are attacks on particular sensitive functions f, which in protected implementations should not appear in clear. As we will describe further, the BU-model allows to protect *all* (non-trivial) intermediate functions at the cost of introducing a black-box encoding step and access to randomness.

Example 3. Consider an AES implementation protected with a Boolean masking of an arbitrarily large order (for example, the ISW scheme [ISW03]). An adversary may choose f as a coordinate of an S-box output in the first round. Then, the degree-1 algebraic attack succeeds, as f can be expressed as a linear combination of shares which are computed in the implementation. Note that in order to compute f (for the right part of the Equation 1), the adversary has to guess a subkey byte.

The time complexity of the attack on a single window $\mathcal{W}$ with $|\mathcal{W}| \gg d$ is

$$\mathcal{O}\left(\binom{|\mathcal{W}|}{\leq d}^{2.8}\right) = \mathcal{O}\left(\frac{|\mathcal{W}|^{2.8d}}{d!^{2.8}}\right), \tag{2}$$

where 2.8 is the matrix multiplication exponent using the Strassen algorithm. We leave out the discussion about the choice of the window(s). For a relevant analysis we refer to [BU18, GPRW20].

[1] Products of degrees less than d are included by setting, for example, $f_1 = f_2$.

2.3 Security model

We now recall the security model introduced in [BU18] and reformulate it concisely. The authors proposed a game-based notion of *prediction security*, which aimed to motivate the security goals. Furthermore, the authors defined *algebraically secure* circuits and encoding functions, which together implied a stronger notion [BU18, Def. 3] sufficient for achieving prediction security. In this work, we concentrate on this strongest combined notion, which we equivalently reformulate as an *algebraically secure scheme*.

The model is a variant of the gray box model allowing a particular type of leakage. Roughly speaking, the implementation may leak a degree-d function of intermediate inputs, whereas in t-probing security, the implementation may leak t intermediate wires. The model relies on the use of *randomness*, which in the white-box setting has to be derived pseudorandomly from the inputs. The model formally defines security of a *scheme*, containing an *encoding function*, an *implementation* and a *decoding function*.

Definition 6 (Scheme)**.** Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a function. A *scheme* S *computing* f consists of

1. an *encoding function* $\mathsf{S.enc}(x, r_e) : \mathbb{F}_2^n \times \mathbb{F}_2^{|r_e|} \to \mathbb{F}_2^{n'}$;
2. an *implementation* $\mathsf{S.comp}(x', r_c) : \mathbb{F}_2^{n'} \times \mathbb{F}_2^{|r_c|} \to \mathbb{F}_2^{m'}$;
3. a *decoding function* $\mathsf{S.dec}(y') : \mathbb{F}_2^{m'} \to \mathbb{F}_2^m$.

It is required that for all $r_e \in \mathbb{F}_2^{|r_e|}, r_c \in \mathbb{F}_2^{|r_c|}$ $\quad \mathsf{S.dec}(\mathsf{S.comp}(\mathsf{S.enc}(x, r_e), r_c)) = f(x)$.

The encoding step is considered as a black-box and its implementation is not analyzed. However, it is important that it has access to the random bits r_e. The output of the encoding step $\mathsf{S.enc}$ is passed to the implementation $\mathsf{S.comp}$, which may access additional random bits r_c. The output of $\mathsf{S.comp}$ is then decoded by the black-box function $\mathsf{S.dec}$ to obtain the final output. Full computation process can be described as

$$x' \leftarrow \mathsf{S.enc}(x, r_e), \quad y' \leftarrow \mathsf{S.comp}(x', r_c), \quad y \leftarrow \mathsf{S.dec}(y').$$

Remark 3. The randomness r_c used in $\mathsf{S.comp}$ can always be generated in $\mathsf{S.enc}$ and included in the "encoded" input x'. The schemes that we propose in this work in fact do not use any randomness in $\mathsf{S.comp}$ *by construction*. A downside of this is that the intermediate state x' may become very large because of the included randomness, which otherwise could be computed "on the fly".

The algebraic security model requires that the implementation $\mathsf{S.comp}$ provides security against the algebraic attacks. In the attacks, the adversary controls the input $x \in \mathbb{F}_2^n$ to $\mathsf{S.enc}$ and is mounting the algebraic attack on $\mathsf{S.comp}$ as described in Definition 5. The security goal is to prevent the algebraic attack from succeeding on *any* function computed in $\mathsf{S.comp}$ and *any* set of inputs chosen by the adversary. This becomes possible due to the use of (pseudo)randomness.

Note that functions $\mathcal{F}(\mathsf{S.comp})$ computed in the implementation are functions of the "encoded" input (that is, of the output of $\mathsf{S.enc}$), which is not directly controlled by the adversary. This requirement can be captured by composing each function from $\mathcal{F}(\mathsf{S.comp})$ with $\mathsf{S.enc}$.

We are now ready to reformulate the main security definition given in [BU18]. Recall that $\mathcal{F}^{(d)}(\mathsf{S.comp})$ contains all degree-d combinations of intermediate functions from $\mathsf{S.comp}$. The idea is to require *every* non-trivial function from $\mathcal{F}^{(d)}(\mathsf{S.comp}(\mathsf{S.enc}))$ and *restricted to any fixed input* x to have a non-negligible error (as a function of random bits r_e, r_c)[2].

[2]In a real white-box implementation r_e, r_c would be constant for fixed x (i.e., derived from x pseudorandomly), but in our definitions we allow a more powerful adversary with ability to re-randomize for the same x.

Then, any such function would be hard to predict and target in the attack *even when the input is fully controlled*. Such security requirement guarantees hardness of launching an algebraic attack even when the adversary knows all the intermediate values computed in the original implementation (for example, knows the secret key if the scheme implements a white-box AES). While such an adversary would not need anymore to launch such an attack, this property highlights the universality of the protection.

We define the algebraic security in terms of the *error* (τ-error-d-AS scheme) instead of the *bias* as in [BU18] ($(1/2-\tau)$-d-AS circuits and encoding functions), as it simplifies the notation. Indeed, the error in our cases is small, especially for the higher-degree case but sufficient to thwart an attacker. Furthermore, it highlights the link with the *Learning Parity with Noise* (LPN) problem, where a linear system with errors has to be solved. Indeed, if some equations in Equation 1 from Definition 5 are erroneous, the attack might still succeed if the fraction of erroneous equations is small enough for LPN-solving algorithms to be applicable. For example, in the case of an extremely small error, the constructed linear system may be error-free and then even the basic algebraic attack succeeds.

Definition 7 (τ-error-d-AS scheme)**.** Let S be a scheme and let $d \geq 1$ be an integer. Let τ be the minimum error among all non-trivial functions from $\mathcal{F}^{(d)}(\mathsf{S.comp})$ composed with $\mathsf{S.enc} = \mathsf{S.enc}(x, r_e)$ for any *fixed* $x = \tilde{x} \in \mathbb{F}_2^n$:

$$\tau \coloneqq \min \left\{ \mathsf{err}\left(f(\mathsf{S.enc}(\tilde{x}, \cdot), \cdot) \right) \;\middle|\; f(x, r_c) \in \mathcal{F}^{(d)}(\mathsf{S.comp}) \setminus \{\mathbf{0}, \mathbf{1}\}, \quad \tilde{x} \in \mathbb{F}_2^n \right\},$$

where the error is computed over r_e, r_c. If $\tau > 0$, the scheme S is said to be *degree-d algebraically secure with error τ (τ-error-d-AS).*

Remark 4. The larger is the error bound τ, the more secure the scheme is against LPN attacks. As noted above, an extremely low error may even allow the basic algebraic attack to succeed with non-negligible probability.

Remark 5. We emphasize that each $f(\mathsf{S.enc}(\tilde{x}, \cdot), \cdot)$ is a function *only* of r_e, r_c used in the scheme.

Remark 6. The algebraic security definition does not cover the decoding function $\mathsf{S.dec}$, which is defined for completeness and to restrict the analysis to *useful* schemes - schemes that indeed compute the desired function $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$.

Remark 7. The requirement $f \notin \{\mathbf{0}, \mathbf{1}\}$ can in principle be weakened to $f(\mathsf{S.enc}(\cdot, \cdot), \cdot) \notin \{\mathbf{0}, \mathbf{1}\}$. The former considers the function on all possible inputs to $\mathsf{S.comp}$, while the latter considers the function restricted only to the image of $\mathsf{S.enc}$. As less functions would be considered, the security notion would be weaker (though not immediately implying an attack).

Remark 8. This definition imposes a very strong requirement, which is *sufficient* but not *necessary* for security against algebraic attacks. For example, consider a function $zr_1r_2 \dots r_t$ for a large integer t, where z is a sensitive function and r_i are random bits. This function is equal to zero when $z = 0$ (the error is zero) and to $r_1r_2 \dots r_t$ when $z = 1$ (the error is 2^{-t}). This function would prevent the definition to hold, but would be impossible to attack algebraically since for large enough t the function $r_1r_2 \dots r_t$ is equal to zero with overwhelming probability.

A major goal is to develop a method of embedding any given implementation into a τ-error-d-AS scheme with a *constant* $\tau > 0$ (i.e. independent of the circuit size) and with the encoding function independent of the circuit structure. The work [BU18] proposed a quadratic masking scheme that achieves 1/16-error-1-AS (i.e. based on 7/16-1-AS circuit gadgets), but didn't provide schemes for degree $d > 1$. The aim of this work is to evaluate shuffling techniques as such a protection method.

What is the maximum value of τ that could possibly be achieved by a scheme? Consider a Boolean circuit-based scheme and consider d independent functions computed in the scheme for some fixed input x. Their product has error 2^{-d} if the functions are balanced and less otherwise (for large-weight functions f we multiply $f \oplus 1$). As a linear computation would not be universal, we assume that d AND gates with independent balanced inputs are present. Since each computed function in such gate has error $1/4$, a well-chosen degree-d product of these functions has error 2^{-2d}. We conclude that in Boolean circuit implementations the error lower bound close to 2^{-2d} would be optimal to achieve. In other implementation models, such as lookup table (LUT) networks, a larger error bound may be achievable. Indeed, absence of intermediate nodes in pure LUT-based implementation gives less variables to use for an attack. As an extreme case, consider a scheme where the implementation consists of a single big LUT, with the input and the output being encoded with some simple and balanced algebraically secure encoding. Since inputs and outputs are balanced and are the only available values in the computational trace, the best error bound to get is 2^{-d}, which is better than 2^{-2d} for circuits.

Remark 9. The discussed upper bounds on the minimum error are tied to the BU-model of algebraic security and can not necessarily be translated into bounds for LPN-based attacks on concrete implementations (not schemes). Hypothetically, better error bounds may be achievable in models with weaker guarantees.

LPN complexity. Complexity of the LPN problem grows with increasing the error τ and the number of unknowns k. In a recent exposition of algorithms for solving LPN by Esser, Kübler and May [EKM17], all time complexities are exponential in the number of unknowns k, with the base of the exponent close to 2^{τ} for small errors (excluding BKW [BKW00] with complexity $2^{\frac{k}{\log k - \log \tau}}$).

Since the number of unknowns $k = \binom{|\mathcal{W}|}{\leq d}$ in the algebraic attack grows much faster than $\tau^{-1} \geq 2^{2d}$, the error bound close to 2^{-2d} provide a sound protection with roughly estimated attack complexity $2^{\tau k} \approx 2^{(|\mathcal{W}|/4)^d}$ or $2^{\frac{|\mathcal{W}|^d}{(d+1)! \log |\mathcal{W}|}}$ using the BKW algorithm. More precise analysis of the complexity of solving LPN instances with such errors is beyond the scope of this work.

3 Shuffling definitions

We first briefly survey the literature on the shuffling countermeasure with a stress on the white-box model in Subsection 3.1 and then proceed with our new definitions. High-level definition of dummy shuffling is given in Subsection 3.2 and its variants in the white-box setting are discussed in Subsection 3.3. Finally, we describe our formal model of dummy shuffling in the algebraic security framework in Subsection 3.4.

3.1 Related work

Shuffling is a side-channel countermeasure that often complements masking. The idea is to randomize the order of the operations to desynchronize sensitive leakage points. A comprehensive study from the side-channel point of view is given by Veyrat-Charvillon, Medwed, Kerckhof and Standaert [VMKS12]. More recently, two works analyzed shuffling in the white-box setting and described two classifications.

In [BRVW19], the authors distinguished two *dimensions* of shuffling in white-box implementations: time and memory. *Time shuffle* randomizes the order of the computations. This is precisely what matters from the classic side-channel point of view, as it desynchronizes the leakage channel. In the white-box setting however, such shuffling can be defeated by synchronizing computational traces by memory addresses, rather than

by time. Therefore, it is necessary to augment time shuffle with *memory shuffle*, which randomizes the addresses of stored intermediate values.

In [GRW20], the authors distinguished *horizontal* and *vertical* shuffling. In horizontal shuffling, the computations are performed at the same time, while the data being processed is shuffled. In vertical shuffling, slots are processed sequentially, and the data is shuffled. Thus, both time and memory shuffle are performed. The authors further allowed dummy slots, which could be based on pseudorandom input or on an irrelevant dummy key.

3.2 Dummy shuffling

In order to distinguish the time/memory and vertical/horizontal separation from the presence of dummy computations, we propose a definition that specifically focuses on the "dummy" part, while being independent of being serial/parallel. The main idea is to hide the real computation among several redundant but similarly looking computations. We start by defining a computational *slot*, which is the target of shuffling: an operation that is computed multiple times independently.

We remark that the definitions in this and the next subsection are informal and introduce only the terminology and broad implementation and hiding strategies.

Definition 8 (Slot (*informal*))**.** A *slot* is a part of the implementation computing a particular sensitive function. In the context of shuffling, it is expected that the implementation contains multiple *slots* for each (sub)function being protected.

Example 4. In a Boolean or arithmetic circuit, an example of a slot is a sub-circuit reproduced multiple times, possibly with modifications or alternative circuit representations. In a program, an example of a slot is a function or a piece of code that is called multiple times, or simply multiple pieces of code each computing the same sensitive function.

We are now ready to provide informal definition of our main protection tool - *dummy shuffling*.

Definition 9 (Dummy shuffling (*informal*))**.** *Dummy shuffling* is an implementation strategy, in which a sensitive function is computed in multiple *slots*, such that during an execution:

1. at least one of the slots *(main slot(s))* computes the function on the correct (*main*) input(s);
2. at least one of the slots *(dummy slot(s))* computes the function on a (pseudo)randomly generated input(s);
3. the locations of the main slots are (pseudo)randomly generated on each execution or on each distinct input.

Dummy shuffling is performed in three phases (see Figure 1):

1. in the *input-shuffling* phase, the dummy inputs are generated and shuffled together with the main inputs;
2. in the *evaluation* phase, the sensitive function is evaluated on each of the inputs, using slots;
3. in the *output-selection* phase, the main outputs are extracted and passed into further computations (by unshuffling or by any other means).

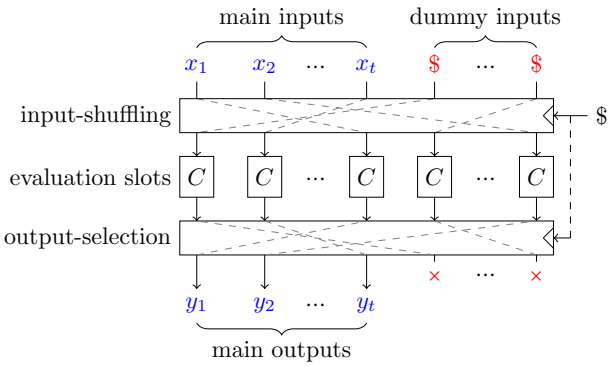


Figure 1: Dummy shuffling. The symbol $ denotes a uniform and independent source of randomness. Implementation of each application of C can be different or, for example, can be one shared procedure in software implementations.

Multiple main slots can be used for two reasons. First, multiple main slots may be running on the *same* main input, with the goal of error detection and/or correction. Second, multiple main slots may be running on *different* main inputs, when in the reference implementation the sensitive function is computed multiple times. The second case corresponds to the standard shuffling, for example, the 16 identical S-boxes (or 4 identical MixColumns operations) in the AES may constitute main slots.

3.3 Hidden and public dummy shuffling

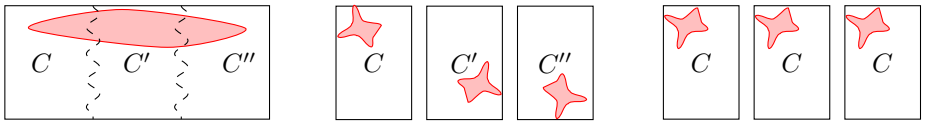
We now introduce a further classification of dummy shuffling techniques with respect to whether the slots are clearly isolated in the implementation or are intertwined with each other to hide the shuffling structure. Furthermore, another important factor is whether all slots have an identical implementation.

Definition 10 (informal)**.** *Hidden dummy shuffling* is an implementation of dummy shuffling for which it must be difficult for an adversary to isolate any single slot or a group of slots, no matter main or dummy.

Public dummy shuffling is an implementation of dummy shuffling in which all slots are clearly separated in the implementation and are easy to isolate. However, the locations of the main/dummy slots must still be difficult to predict for an adversary in any evaluation. Furthermore, if all slots' implementations are fully identical and an adversary is able to interchange them freely, then we say that the dummies are *uniform*.

This definition captures the level at which an obfuscation is performed. In hidden dummy shuffling, the whole implementation is obfuscated and the slots are hard to locate and isolate. In public dummy shuffling, each slot may be obfuscated but is still easy to locate and isolate in the implementation.

In this work we analyze dummy shuffling as a countermeasure against the algebraic attack. In this context, the difference between hidden and public dummy shuffling mainly affects the size of the window that contains all nodes of the circuit used in the attack. Typically, two configurations of attacked nodes arise in the attacks: (1) all attacked nodes are contained in a single slot; (2) attacked nodes contain the same group of nodes in multiple/all slots. Case 2 is illustrated in Figure 2, where the adversary tries to blindly select a window in the full implementation such that it contains the same target sensitive function computed in *each* of the slots; the red areas highlight the uncertainty for selecting such a window.



(a) hidden (b) non-uniform public (c) uniform public

Figure 2: Variants of dummy shuffling and window selection uncertainty. Red areas illustrate possible positions of a window relatively to the slots.

1. In *hidden dummy shuffling*, the slots are not clearly separated and thus a window has to be selected from the *entire implementation* including all slots. Furthermore, in the case (2) the size of the window has to be much larger to be able to cover multiple slots.

2. In *non-uniform public shuffling* (for example, if each slot is obfuscated independently), the slots are easy to isolate. Therefore, a window in a single slot is selected from *that slot only*, reducing the combinatorial complexity and the required window size. A window covering the same group of nodes in multiple slots is still similar to the hidden dummy shuffling case, since it should be hard to find the parts of obfuscated circuits related to the target attacked group.

3. In *uniform public shuffling*, the slots are clearly isolated and are identical. Therefore, in both cases (1) and (2), the window can be selected *inside a single slot*, and extended to *the same area* in the other slots in the case (2). This case allows minimal combinatorial complexity of the attacks. However, from the designer's viewpoint, it removes the high-level obfuscation requirement and leads to a cleaner solution.

Since our work is not focusing on the attack details, we do not analyze the window aspect deeper, as it would require more concrete implementation details.

In Section 6, we describe a proof-of-concept construction for uniform public dummy shuffling. It shows that it is possible to implement dummy shuffling in a way that, even given a black-box access to the slot function, it is hard to distinguish main slots from dummy slots for any particular input. Therefore, a white-box designer aiming to use dummy shuffling does not have to obfuscate the whole implementation including the shuffling procedure and all slot evaluations; obfuscating a single slot function is sufficient.

3.4 Modeling algebraic security of dummy shuffling

In this work, we analyze security of the slot evaluation phase, which is the core of dummy shuffling. It is the most critical part where all the computations of the original implementation take place. This subsection defines a formal model for analyzing security of dummy shuffling in the framework of [BU18].

In the following, let s_{main} denote the number of main inputs/slots, s_{dummy} the number of dummy inputs/slots, and $s := s_{\text{main}} + s_{\text{dummy}}$. For simplicity, we assume that there are no always-duplicate main inputs and all main inputs are independent, i.e. an adversary can set each main input to any value independently.

We analyze the security of the evaluation phase by considering the input-shuffling phase as the "encoding" part of a scheme (S.enc), the slot evaluation phase as the main "implementation" (S.comp), and the output-selection phase as the "decoding" part (S.dec). Finally, the goal is to determine the algebraic security of the resulting scheme S. This gray-box setting is formally described in the following definition.

Definition 11 (Evaluation-Phase Model)**.** Let $C(x) : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation. Let $s_{\text{main}}, s_{\text{dummy}}$ be positive integers, $s := s_{\text{main}} + s_{\text{dummy}}$. In the *evaluation-phase*

model, we analyze the algebraic security (in the sense of Definition 7) of the scheme $\mathsf{EPM}(C, s_{\text{main}}, s_{\text{dummy}}) := \mathsf{S}$, constructed as follows:

Func.
$\mathsf{S.enc}(x, r_e) : (\mathbb{F}_2^n)^{s_{\text{main}}} \times \mathbb{F}_2^{|r_e|} \to (\mathbb{F}_2^n)^s$
  let $v \in (\mathbb{F}_2^n)^s$
  for $i \in [1 \dots s_{\text{main}}]$ **do**
    $v_i \leftarrow x_i$
  $(r'_e, r''_e) \leftarrow r_e$
  for $i \in [(s_{\text{main}} + 1) \dots s]$ **do**
    $v_i \xleftarrow{r'_e} \mathbb{F}_2^n$
  return $x' \xleftarrow{r''_e} \mathsf{Shuffle}(v_1, \dots, v_s)$

Impl. $\mathsf{S.comp}(x') : (\mathbb{F}_2^n)^s \to (\mathbb{F}_2^m)^s$
  let $y' \in (\mathbb{F}_2^m)^s$
  for $i \in [1 \dots s]$ **do**
    $y'_i \leftarrow C(x'_i)$
  return $y' \leftarrow (y'_1, \dots, y'_s)$

Func. $\mathsf{S.dec}(y', r''_e) : (\mathbb{F}_2^m)^s \to (\mathbb{F}_2^m)^{s_{\text{main}}}$
  $y \xleftarrow{r''_e} \mathsf{Unshuffle}(y'_1, \dots, y'_s)$
  return $(y_1, \dots, y_{s_{\text{main}}})$

Here, by $\xleftarrow{r'_e}$ ($\xleftarrow{r''_e}$) we mean that r'_e (r''_e) is used as randomness to generate the value (sample uniformly from $\mathbb{F}_2^n$ shuffle almost-uniformly).

Remark 10. In principle, dummy inputs can be sampled from any chosen distribution, which could be dependent on the sensitive function and even on the main inputs. Why could it be useful? Assume that we want to protect a point function using dummy shuffling. Uniformly sampled inputs will make the function equal to 0 with overwhelming probability. Thus, the linear sum of the function's output over the main input and the dummy inputs will be equal to the function itself and this proves the algebraic insecurity[3] of the shuffling protection in this case. On the other hand, we could sample the dummy inputs by choosing the preimage of 1 with probability 1/2. Then, such attack would not be possible.

While non-uniform sampling of dummy inputs may provide more flexibility in the protection, it increases the difficulty of the analysis and is not a generic solution. In this work, we consider the simplest and the most generic case when the dummy inputs are sampled from the uniform distribution, as is described in Definition 11.

Remark 11. The EPM scheme does not use randomness in the implementation part, so the argument r_c in S.comp is omitted.

Remark 12. We define the decoding function by unshuffling the computed state y using saved randomness r''_e which was used to shuffle in S.enc. Formally, we could include r''_e in S.comp by encrypting it in S.enc so that it does not introduce algebraic leakage, and decrypting in S.dec. This just an example method of implementing the output-selection. As we focus on the evaluation phase, this process is out of scope of this model.

Remark 13. The shuffling permutation does not have to be perfectly uniform. In fact, it is not possible for $s \geq 3$ (because $3! = 6$ does not divide any power of 2), but with a sufficient amount of random bits it can be made computationally indistinguishable from uniform shuffling. In addition, it is easy to show that it is enough to choose uniformly locations of s_{main} *main* slots and shuffle them; shuffling dummy slots does not change the output distribution of S.enc.

4 Algebraic attacks on dummy(less) shuffling

In this section, we describe weaknesses in the algebraic security of dummy(less) shuffling. We start by exhibiting leakage of classic *dummyless* shuffling in the model in Subsection 4.1, where we also sketch a standard linear algebraic attack to highlight the practical relevance. In Subsection 4.2, we develop a *differential algebraic attack* which exploits the leakage more effectively. We show however in Subsection 4.3 that the security model of [BU18] is strong enough to provide security against the differential attack technique out-of-the-box.

[3] On practice, the adversary would need to know the preimage of 1 in order to mount an attack.

We continue by generalizing the attack to a *higher-degree* algebraic attack against shuffling *with dummy slots* in Subsection 4.4. This attack gives an upper-bound on the degree of algebraic security of dummy shuffling depending on the number of dummy slots.

4.1 Standard algebraic attack against dummyless shuffling

Shuffling without dummy slots requires the implementation to have multiple main slots and thus is quite limited in its applications. Nonetheless, a typical application is a block cipher utilizing the Substitution-Permutation Network (SPN) structure and almost all such ciphers use the same S-box in each round, clearly exposing multiple main slots for the substitution layer. The linear layers however have a large variety of structures and the applicability of classic dummyless shuffling depends on each case. Since white-box implementations of SPN ciphers is a typical goal, we analyze this case.

We start by exhibiting a critical weakness of dummyless shuffling. Briefly speaking, shuffling leaks any symmetric function of the permuted values. For a degree one attack, the only such function is the sum of the value over all slots. For higher degrees, there are more possibilities.

Proposition 1. *Let $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation and let $\mathsf{S} := \mathsf{EPM}(C, s, 0)$ for an integer $s \geq 1$. Then, for any $f \in \mathcal{F}(C)$ and any symmetric function $g : \mathbb{F}_2^s \to \mathbb{F}_2$ the following function h is leaked, i.e. there exists $h' \in \mathcal{F}^{(\deg g)}(\mathsf{S.comp})$, such that $h'(\mathsf{S.enc}(x, r_e)) = h(x)$, where*

$$h \;:\; (\mathbb{F}_2^n)^s \to \mathbb{F}_2 \;:\; (x_1, \ldots, x_s) \mapsto g(f(x_1), \ldots, f(x_s)).$$

Proof. Since $f(x_i)$ is computed in clear in each slot, a degree-d symmetric combination h' of these functions belongs to $\mathcal{F}^{(\deg g)}(\mathsf{S.comp})$. The effect of $\mathsf{S.enc}$ only permutes the inputs $x_1, \ldots, x_n$, which does not have an effect on h' since it is symmetric: $h(x) = h'(\mathsf{S.enc}(x, r_e)) = h'(x)$. □

Example 5. The most trivial example is the sum of a sensitive function f over all slots being vulnerable to the algebraic attack. Note that a related technique called *integration attack* was applied to differential power analysis (DPA) of randomized implementations in [CCD00] in order to reduce the introduced noise and lower the required number of traces.

The proposition shows that classic dummyless shuffling does not achieve security in the evaluation-phase model. We now show a concrete practical attack on the example of the AES.

Consider an AES implementation where the 16 S-boxes are shuffled and possibly protected by a linear masking scheme. We target any single bit output of the S-box after the first round. However, as observed above, only the sum of these bits of all 16 S-boxes is leaked. Let $S_1 : \mathbb{F}_2^8 \to \mathbb{F}_2$ denote the first output bit of the AES S-box and define a function f as follows:

$$f \;:\; (\mathbb{F}_2^8)^{16} \to \mathbb{F}_2 \;:\; (x_1, \ldots, x_{16}) \mapsto S_1(x_1 \oplus k_1) \oplus \ldots \oplus S_1(x_{16} \oplus k_{16}),$$

where $k_1, \ldots, k_{16}$ is the first round subkey. Clearly, f can be computed via a linear combination of some intermediate variables in the analyzed implementation. The standard approach of guessing a portion of the key to compute f does not work, since it depends on the full key. We show that in the *chosen-plaintext* (CPA) setting an efficient attack is possible. Note that the algebraic security model assumes CPA and so such attack is covered by the model. The idea is to fix $x_2, \ldots, x_{16}$ to arbitrary constants and guess one bit

$$c := S_1(x_2 \oplus k_2) \oplus \ldots \oplus S_1(x_{16} \oplus k_{16}).$$

Then, after guessing k_1 the value of f can be computed for all 256 values of x_1, i.e. on inputs of the form $(\mathbb{F}_2^8, x_2, \ldots, x_{16})$. The limited number of inputs upper bounds the window size that can be used for the attack which can become a limitation for an attacker. While this is already a proof-of-concept attack, we can further overcome the limitation. Let us guess another bit, which is now a bit of difference

$$S_1(x_2' \oplus k_2) \oplus S_1(x_2 \oplus k_2)$$

for some $x_2' \neq x_2$. This allows to compute the value of f on 256 more inputs of the form $(\mathbb{F}_2^8, x_2', x_3, \ldots, x_{16})$. More generally, we can guess $t \leq 15$ bits of difference (in addition to the 8 bits of k_1) to be able to compute f on $256 \cdot 2^t$ different inputs, which already allows to attack a huge window (although, at the cost of increased base complexity). Further, more difference bits per each byte can be guessed to cover more inputs at a little cost. Of course, this direction leads to more theoretical scenarios with enormous implementations and less practical attacks.

We conclude that dummyless shuffling provides little security even against standard algebraic attack (with modified key guessing method) in the chosen plaintext setting.

4.2 Differential algebraic attack against dummyless shuffling

In this section, we describe a generalization of the algebraic attack called *differential algebraic attack*. The idea follows rather naturally from the previously described attack, where bits of differences were guessed. Let us attack the difference of f on pairs of inputs (i.e. $f(x) \oplus f(x')$), instead of the function f itself (i.e. $f(x)$). Indeed, the difference is at least not harder to compute and, in particular cases, may be much easier.

This modification works very well for the dummyless shuffling setting described above. In fact, it works out-of-the-box with a standard key guessing procedure. First, an attacker chooses pairs (x, x') such that $(x_2, \ldots, x_{16}) = (x_2', \ldots, x_{16}')$ and $x_1 \neq x_1'$. Then, she records computational traces $\mathcal{W}(x), \mathcal{W}(x')$ and computes a new *differential* trace

$$v(x) := (\mathcal{W}_i(x) \oplus \mathcal{W}_i(x') \;\mid\; 1 \leq i \leq |\mathcal{W}|)\,,$$

which is used further as in the standard algebraic attack. Similarly, instead of computing $f(x)$ for a given key guess, the attacker computes $f(x) \oplus f(x')$. In the AES example, it requires only one key byte guess as

$$f(x) \oplus f(x') = S_1(x_1 \oplus k_1) \oplus S_1(x_1' \oplus k_1),$$

while computing $f(x)$ requires 16 key bytes:

$$f(x) = S_1(x_1 \oplus k_1) \oplus \ldots \oplus S_1(x_{16} \oplus k_{16}).$$

The attack can be viewed as a standard algebraic attack with an extra preprocessing step of the collected traces and of the predicted sensitive function.

We now give a formal definition of a general degree-d differential attack, similar to Definition 5 (Algebraic attack).

Definition 12 (Differential Algebraic attack)**.** A *degree-d differential algebraic attack* against an implementation $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ targeting a sensitive function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ consists of the following steps:

1. choose a *window* $\mathcal{W} \subseteq \mathcal{F}(C)$;
2. choose a vector of *pairs* of inputs $x := ((x_1, x_1'), \ldots, (x_t, x_t')) \in (\mathbb{F}_2^n \times \mathbb{F}_2^n)^t$, where $t := \binom{|\mathcal{W}|}{\leq d} + \epsilon$ for some small integer ϵ;

3. compute the t pairs of traces $(\mathcal{W}(x_i), \mathcal{W}(x'_i))$ on these inputs and their degree-d expansion;
4. compute the t pairs of values $(f(x_i), f(x'_i))$ on these inputs;
5. solve the following linear system in z:

$$\begin{pmatrix} \pi_d(\mathcal{W}(x_1)) \oplus \pi_d(\mathcal{W}(x'_1)) \\ \vdots \\ \pi_d(\mathcal{W}(x_t)) \oplus \pi_d(\mathcal{W}(x'_t)) \end{pmatrix} \times z = \begin{pmatrix} f(x_1) \oplus f(x'_1) \\ \vdots \\ f(x_t) \oplus f(x'_t) \end{pmatrix}.$$

The attack succeeds if a solution is found. It is further required that the right part of the equation is nonzero.

Remark 14. For $d \geq 2$, this definition applies degree-d expansion *before* combining the traces in pairs. This is useful for example if each shuffled slot is in addition protected by a nonlinear masking scheme. Then the differential is applied to expanded traces, i.e. to vector spaces containing decoded sensitive values. Whether applying the degree-d expansion *after* combining the traces in pairs has a useful application is unclear.

4.3 Security against differential algebraic attack

We will show that the differential algebraic attack does not provide any advantage against algebraically secure schemes (τ-error-d-AS), in particular, against secure variants of dummy shuffling which we will identify later. To state it formally, we define an analogue of the security notion τ-error-d-AS and show that the new notion is implied by τ-error-d-AS.

Definition 13. Let S be a scheme and let $d \geq 1$ be an integer. Let τ be defined as follows[4]:

$$\tau := \min \Big\{ \mathsf{err}\left(f\big(\mathsf{S.enc}(x,\cdot),\cdot\big) \oplus f\big(\mathsf{S.enc}(x',\cdot),\cdot\big) \right) \\ \Big| \; f \in \mathcal{F}^{(d)}(\mathsf{S.comp}) \setminus \{\mathbf{0},\mathbf{1}\}, x, x' \in \mathbb{F}_2^n \Big\}.$$

If $\tau > 0$, the scheme S is said to be *degree-d differentially algebraically secure with error τ (τ-error-d-DAS).*

We now show that standard algebraic security implies differential algebraic security.

Proposition 2. *Let S be a scheme. If it is τ-error-d-AS for some τ, d, then it is τ'-error-d-DAS with $\tau' = 2\tau(1-\tau) \geq \tau$.*

Proof. Let $f \in \mathcal{F}^{(d)}(\mathsf{S.comp}) \setminus \{\mathbf{0},\mathbf{1}\}$ and $x, x' \in \mathbb{F}_2^n$. Define

$$e := \mathsf{err}\left(f(\mathsf{S.enc}(x,\cdot),\cdot)\right) \geq \tau, \quad e' := \mathsf{err}\left(f(\mathsf{S.enc}(x',\cdot),\cdot)\right) \geq \tau,$$
$$e'' := \mathsf{err}\left(f(\mathsf{S.enc}(x,\cdot),\cdot) \oplus f(\mathsf{S.enc}(x',\cdot),\cdot)\right).$$

Since $f(\mathsf{S.enc}(x,\cdot),\cdot)$ and $f(\mathsf{S.enc}(x',\cdot),\cdot)$ each use independent inputs r_c, r_e, it follows that

$$e'' = e(1-e') + (1-e)e' = e + e' - 2ee',$$

which is minimized when both e and e' are minimized, that is $e'' \geq 2\tau - 2\tau^2 = 2\tau(1-\tau)$. This is always not less than τ, since $\tau \leq 1/2$ and so $2(1-\tau) \geq 1$. □

[4]The randomness variables r_e, r_c are independent in each application of f and $\mathsf{S.enc}$.

The proof shows that, in fact, the error only increases when multiple traces are combined. It is trivial to prove a similar statement for the case of higher-order differentials or general integrals (i.e. adding values of f in more than 2 inputs). Therefore, the differential algebraic attack is not useful against algebraically secure schemes. Note that this was not a problem in the dummyless shuffling setting, because the attack targeted a function with error 0. We conclude that τ-error-d-AS is a strong security notion and automatically covers some extensions of the algebraic attack.

4.4 Generic higher-degree attack

After (crypt)analyzing dummyless shuffling, we switch to dummy shuffling with at least one dummy slot. We consider higher-degree attacks in order to establish an upper bound on the *degree* of the algebraic security of dummy shuffling. We describe a generic degree-$(s_{\text{dummy}}+1)$ attack in the evaluation-phase model (meaning that the attack is very generic), and further sketch how an actual attack would look like in practice. In a way, this attack generalizes the attack from Subsection 4.1. Indeed, the former attack described a degree-1 attack on shuffling with $s_{\text{dummy}} = 0$.

Proposition 3. *Let C be an implementation, and let $s_{main} \geq 1, s_{dummy} \geq 0$. The evaluation-phase model scheme $\mathsf{EPM}(C, s_{main}, s_{dummy})$ is not τ-error-$(s_{dummy}+1)$-AS for any $\tau > 0$.*

Proof. Let $d = s_{\text{dummy}} + 1$. The idea is to select the same sensitive variable $z \in \mathcal{F}^{(1)}(C)$ in arbitrary d slots (for the sake of the proof, any input bit function of $\mathsf{S.comp}$ suffices), and to multiply these linear functions. The resulting function, denoted $\mathbf{z} \in \mathcal{F}^{(d)}(\mathsf{S.comp})$, is always a product of some bits computed on dummy inputs and of the sensitive variable at one (or more) of the main slots.

Let p denote the probability of $z = 1$ when the input is sampled uniformly at random, i.e. $p = \Pr_{x \in \mathbb{F}_2^n}[z(x) = 1] > 0$. Let us consider all main inputs set to the same value, namely x_0 or x_1, such that $z(x_0) = 0, z(x_1) = 1$.

In the first case, the sensitive variable z is equal to 0 in at least one of the considered slots and the product is always equal to zero:

$$\Pr_{r_e}\Big[\mathbf{z}(\mathsf{S.enc}(x_0, r_e)) = 0\Big] = 1.$$

In the second case, the probability of the product being equal to 1 is p^t where t denotes the number of dummy slots among the chosen d slots. It is minimal when all $d-1$ dummy slots are selected. We conclude that the whole product is equal to 1 with probability at least p^{d-1}:

$$\Pr_{r_e}\Big[\mathbf{z}(\mathsf{S.enc}(x_1, r_e)) = 1\Big] \geq p^{d-1}.$$

This concludes the proof, since for the described non-constant function $\mathbf{z} \in \mathcal{F}^{(d)}(\mathsf{S.comp}) \setminus \{\mathbf{0}, \mathbf{1}\}$, the function $\mathbf{z}(\mathsf{S.enc}(x_0, \cdot), \cdot)$ is constant and thus has the error equal to 0. □

The proposition shows that dummy shuffling does not achieve τ-error-d-AS, but it does not prove that it is in fact insecure against the algebraic attack. We go further and sketch a concrete attack that is applicable to an implementation protected with dummy shuffling. Let $\mathcal{W} \subseteq \mathcal{F}(\mathsf{S.comp})$ denote the attacked window and let $w := |\mathcal{W}|$ denote its size, e.g. $w = |\mathsf{S.comp}| = s\,|C|$ for the whole circuit. We assume that there is a sensitive variable $z \in \mathcal{W}^{(1)}$ that defines a balanced or a close to balanced Boolean function.

Let X_0 (resp. X_1) denote the set of inputs for which the sensitive variable is equal to 0 (resp. 1). The adversary chooses $t := w^d/d! + \epsilon$ inputs from X_0 for which the sensitive variable is equal to 0 and computes traces on these inputs. Then, she chooses an input from X_1 for which the sensitive variable is equal to 1 and computes a single trace on it.

She applies the degree-d algebraic attack to the $t+1$ traces together, searching for the vector $(0,\dots,0,1)$ in the space $\mathcal{W}^{(d)}$ restricted to the traced inputs, which has size at most $\binom{w}{\le d} < w^d/d!$. The sensitive function $\mathbf{z}$ constructed as in the proof above would match the first t zeroes with probability 1 and match the last one with probability at least $1/2^{d-1}$. We assume that the probability of other vectors matching (i.e. a false positive) is negligible since t is larger then the dimension of the vector space. With probability $1/2^{d-1}$ an attack trial succeeds. Therefore, $\mathcal{O}(2^d)$ traces with inputs from X_1 are enough to find the desired degree-d combination with high probability. The complexity of the attack is thus $\mathcal{O}(2^d \cdot (w^d/d!)^{2.8})$ (using the Strassen algorithm).

Example 6. Consider an AES implementation protected with dummy shuffling, $s_{\text{main}} = 1$ and $s_{\text{dummy}} \ge 1$, i.e. a slot computes the full cipher. The sensitive variable z is as usual the output of a first-round S-box, and we target $\mathbf{z}$: the product of z taken over all $s = s_{\text{dummy}} + 1$ slots. A guess of the respective subkey byte allows to split the input space into X_0 and X_1. A standard assumption is that the wrong subkey guess results in an incorrect split and leads to an unsuccessful attack. It follows that the correct subkey can be identified by running the attack 256 times.

5 Provable algebraic security of dummy shuffling

After establishing the *limits* of the algebraic security of dummy shuffling in the previous section, we switch to quantifying and *proving* security of dummy shuffling. In Subsection 5.1, we analyze the security of basic dummy shuffling against the linear attack. Next, we develop a refreshing technique which allows to achieve provable security in Subsection 5.2. Finally, we use the same technique to prove security against *higher-degree* algebraic attack in the case of a single main slot in Subsection 5.3.

5.1 Security analysis (linear case)

After showing an upper-bound on the algebraic security degree provided by dummy shuffling, we now study the case of degree-1 attack, and analyze when dummy shuffling indeed provides a protection and evaluate the security parameter τ. We show that algebraic security of the EPM scheme depends on a particular property of the original circuit, which is defined formally in the following definition.

Definition 14. Let C be an implementation. For an integer $d \ge 1$, denote by $\mathsf{err}_d(C)$ the minimum error of a nontrivial function from $\mathcal{F}^{(d)}(C)$:

$$\mathsf{err}_d(C) := \min_{f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0},\mathbf{1}\}} \mathsf{err}(f).$$

Remark 15. Note that err_d has a different role than τ in the τ-error-d-AS definition. The new definition quantifies bias on a uniform distribution of the inputs of an implementation C (which at this stage does not use randomness, since it is an unprotected implementation, not a scheme). In the former definition, τ quantifies bias on a *fixed* main input and uniform distribution of random bits r_e, r_c in the scheme.

We now give a bound on the 1-AS security of the EPM scheme, parameterized by the value err_1 and the number of main and dummy slots.

Theorem 1. *Let C be an implementation and let $s_{main} \ge 1, s_{dummy} \ge 0$ be integers, $s = s_{main} + s_{dummy}$. Then the evaluation-phase model scheme $\mathsf{S} := \mathsf{EPM}(C, s_{main}, s_{dummy})$ is τ-error-1-AS, where*

$$\tau \ge \frac{s_{dummy}}{s} \cdot \mathsf{err}_1(C).$$

Proof. Consider a function $f \in \mathcal{F}^{(1)}(\mathsf{S.comp}) \setminus \{\mathbf{0}, \mathbf{1}\}$ and an arbitrary input x. Since f is nontrivial, it can be expressed without loss of generality as $f(x') = g(x'_1) + h(x'_2, \cdots, x'_s)$, where $g \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ is a function computed in one of the slots, and h is a function computed in the other slots. The slot of g is a dummy slot with probability $\frac{s_{\text{dummy}}}{s}$. In this case, g takes as input an independent uniformly random input (derived from r'_e in $\mathsf{S.enc}$), and its error is lower-bounded by $\mathsf{err}_1(C)$. In the case it is a main slot, the value of g is constant and the error is equal to 0. It follows that

$$\mathsf{err}\left(g(\mathsf{S.enc}(x, \cdot)\right) \geq \frac{s_{\text{dummy}}}{s} \cdot \mathsf{err}_1(C) + \frac{s_{\text{main}}}{s} \cdot 0.$$

For any fixed shuffling order outcome (decided by r''_e in $\mathsf{S.enc}$), g and h are independent, and so the error $\mathsf{err}\left(f(\mathsf{S.enc}(x, \cdot))\right)$ satisfies the same bound. □

Simply stating, the error bound is proportional to $\mathsf{err}_1(C)$ with coefficient equal to the fraction of dummy slots: when all slots are dummy slots, the bound is equal to $\mathsf{err}_1(C)$; when all slots are main slots, the bound is equal to 0.

According to this theorem, dummy shuffling provides security against the linear algebraic attack as soon as at least one dummy slot is used. However, the security parameter τ depends on the original circuit C and thus is not generally a constant. Furthermore, even determining or approximating the bound $\mathsf{err}_1(C)$ for an arbitrary implementation C is not an easy problem. We consider one special case when the bias can be upper bounded.

Corollary 1. *Let $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation and let $r \coloneqq \max_{f \in \mathcal{F}(C)} \deg f$. Then the scheme $\mathsf{EPM}(C, s_{main}, s_{dummy})$ is τ-error-1-AS with*

$$\tau \geq \frac{1}{2^r} \cdot \frac{s_{dummy}}{s}.$$

Proof. We use the well-known facts that the minimum weight of a nonzero Boolean function of degree r is 2^{n-r}, i.e. the minimum error satisfies $\mathsf{err}_1(C) \geq 1/2^r$, and that a linear combination of such functions can not increase the degree. □

In the following subsection, we propose a solution to obtain concrete security guarantees for arbitrary circuits.

5.2 Provable security via refreshing (linear case)

In this solution, we first transform the original implementation C before applying the shuffling countermeasure. For simplicity, we assume that the implementation is based on a Boolean circuit.

First, we add extra inputs to the circuit. After embedding the extended circuit in the EPM scheme, the extra bits would be set to zero on main inputs, while on dummy inputs they would be uniformly random (by the definition of EPM). Then, we use these extra inputs to "refresh" each non-linear gate by an extra XOR. In a main slot, this will have no effect on the computation, since the extra bits are equal to zero. In a dummy slot, this will randomize all computations and maximize the value $\mathsf{err}_1(\tilde{C})$ of the new implementation $\tilde{C}$.

Definition 15 (Refreshed Circuit)**.** Let $C(x) : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a Boolean circuit implementation with l AND gates and an arbitrary amount of XOR and NOT gates. Define the *refreshed circuit* $\tilde{C}(x, r) : \mathbb{F}_2^n \times \mathbb{F}_2^l \to \mathbb{F}_2^m$ as follows. Replace each AND gate $a_k = z_i \wedge z_j$ in C, $1 \leq k \leq l$ by the circuit $a'_k = r_k \oplus a_k = r_k \oplus (z_i \wedge z_j)$, where r_k is the k-th extra bit; each wire using a_k is rewired to use a'_k(see Figure 3).

Refreshing has a useful effect on the computed functions: up to a bijective modification of the input, a refreshed circuit computes only quadratic functions of the input. This immediately implies $\mathsf{err}_1(\tilde{C}) \geq 1/4$ for any circuit C and will also be useful for proving higher-degree security in Subsection 5.3.

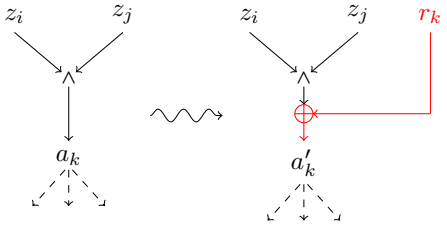


Figure 3: Refreshing an AND gate.

Lemma 1. *Let $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation in a form of a Boolean circuit in the {AND,XOR,NOT} basis using l AND gates and let $\tilde{C}$ be its refreshed version. Then, there exists a bijection h mapping $\mathbb{F}_2^n \times \mathbb{F}_2^l$ to itself, such that $\deg f \circ h^{-1} \leq 2$ for all $f(x,r) \in \mathcal{F}(\tilde{C})$.*

Proof. We use the notation from Definition 15. For all $1 \leq k \leq l$, let

$$g_k \ : \ \mathbb{F}_2^n \times \mathbb{F}_2^l \to \mathbb{F}_2^n \times \mathbb{F}_2^l \ : \ (x,r) \mapsto (x,r'), \text{ where}$$

$$r'_i = \begin{cases} r_i \oplus a_k(x,(r_1,\ldots,r_{k-1})), & \text{if } i = k \\ r_i & \text{if } i \neq k. \end{cases}$$

That is, g_k replaces r_k by $r_k + a_k = a'_k$ in the full state (x,r). Note that a_k is a function of x and $r_1,\ldots,r_{k-1}$ and so g_k is a bijection.

Define $h := g_l \circ \ldots \circ g_1$ and let $(x,r') := h(x,r)$. Then, we have $r'_k = a'_k(x,r)$ for all k. Let $f \in \mathcal{F}(\tilde{C})$ be the function computed in an arbitrary AND gate of $\tilde{C}$. Note that outputs of AND gates are used only to compute a'_k in $\tilde{C}$ and the inputs of AND gates can only be affine functions of x and all refreshed AND gates a'_k. That is,

$$f(x,r) = p(x,a'(x,r))q(x,a'(x,r))$$

for some affine functions p, q. Since $(x,a') = (x,r')$ is the output of $h(x,r)$, it follows that

$$f(x,r) = p(h(x,r))q(h(x,r)).$$

The right-hand side defines (at most) quadratic function $o(z) := p(z)q(z)$ such that $f = o \circ h$. We conclude that $f \circ h^{-1} = o$ has degree at most 2. □

Remark 16. From the proof it can be observed that the last topologically independent AND gates (i.e. those, output of which does not affect any other AND) do not have to be refreshed for the lemma to hold.

The linear algebraic security of dummy shuffling with refreshing follows naturally from the lemma and Corollary 1.

Theorem 2. *Let $C(x) : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation in a form of a Boolean circuit in the {AND,XOR,NOT} basis. Then, $\mathsf{EPM}(\tilde{C}, s_{main}, s_{dummy})$ is τ-error-1-AS, where*

$$\tau \ \geq \ \frac{1}{4} \cdot \frac{s_{dummy}}{s}.$$

In particular, $\mathsf{EPM}(\tilde{C}, 1, 1)$ is a 1/8-error-1-AS scheme.

Proof. The weight/error of any function $f \in \mathcal{F}^{(1)}(\tilde{C}) \setminus \{\mathbf{0}, \mathbf{1}\}$ is unchanged when the function is composed with a bijection (in this case, the bijection h^{-1} from Lemma 1): $\mathsf{err}(f) = \mathsf{err}\left(f \circ h^{-1}\right) \geq 1/4$. Therefore, any considered function f is weight-equivalent to a (non-zero) quadratic function, which has error at least 1/4, and so $\mathsf{err}_1(\tilde{C}) \geq 1/4$. The result follows from Theorem 1. □

5.3 Provable security via refreshing (higher-degree)

We now switch to higher-degree algebraic security. In this subsection we show that the refreshing technique allows to achieve algebraic security of degree matching the upper-bound given by the generic attack given in Subsection 4.4, namely the degree equal to the number of dummy slots.

We will use the following lemma. Intuitively, consider s parallel applications of an implementation $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ and assume $f : (\mathbb{F}_2^n)^s \to \mathbb{F}_2$ be a non-constant function of the s inputs obtained by applying a degree-d function to intermediate functions across all copies of C. Assume that we can set one of the inputs to any constant $c \in \mathbb{F}_2^n$, making all intermediate computations in that C constant as well. However, which one out of s copies is set to the constant is chosen uniformly at random. The lemma says that f can be constant in at most d such choices out of s.

The motivation for the lemma comes from a simple choice of such f and c (inspired by the generic attack from Subsection 4.4) set $c = 0$ and f be (for example) a product of the first input bit of the first d copies of C: $f(x_1, \ldots, x_s) = x_{1,1}x_{2,1}\ldots x_{d,1}$. Clearly, $f = 0$ when $x_1 = c = 0$, or $x_2 = c = 0$, or …, or $x_d = c = 0$. However, it is non-constant in all other $s - d$ choices, namely $x_{d+1} = c = 0$, or …, or $x_s = c = 0$. The lemma thus states that such a choice of f, c is the best an adversary (aiming to find f that is constant as often as possible) can achieve.

Lemma 2. *Let $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an implementation. For an integer $s \geq 1$ denote s parallel applications of C by $C^{\otimes s}$ (as an implementation):*

$$C^{\otimes s} : (\mathbb{F}_2^n)^s \to (\mathbb{F}_2^m)^s : (x_1, \ldots, x_s) \mapsto (C(x_1), \ldots, C(x_s)).$$

Let $f \in \mathcal{F}^{(d)}(C^{\otimes s}) \setminus \{\mathbf{0}, \mathbf{1}\}$ for an integer d, $1 \leq d \leq s$. Then, for any $c \in \mathbb{F}_2^n$ the number of positions i, $1 \leq i \leq s$ such that $f|_{x_i=c}$ is constant is at most d:

$$\left|\left\{f|_{x_i=c} \in \{\mathbf{0}, \mathbf{1}\} \;\middle|\; i \in [1 \ldots s]\right\}\right| \leq d.$$

Proof. The proof is by contradiction. Let g denote the degree-d function associated to f, that is the function applied to $(\mathcal{F}(C))^s$ to obtain f:

$$g : \left(\mathbb{F}_2^{|\mathcal{F}(C)|}\right)^s \to \mathbb{F}_2, \text{ such that}$$
$$g(\mathcal{F}(C)(x_1), \ldots, \mathcal{F}(C)(x_s)) = f(x_1, \ldots, x_s) \text{ for all } x_1, \ldots, x_s \in \mathbb{F}_2^n.$$

Here $\mathcal{F}(C)(x_i)$ is the computational trace of C on input x_i (the bit-vector of all intermediate values computed in C on input x_i).

Assume that there exist (at least) $d + 1$ positions $j_1, \ldots, j_{d+1}$ such that for all $j \in \{j_1, \ldots, j_{d+1}\}$, $f|_{x_j=c}$ is constant. Note that the value of the function is the same constant for all such positions, since these restrictions intersect at $x_{j_1} = c, \ldots, x_{j_{d+1}} = c$. We can assume without loss of generality that the constant is 0. Since f is not constant in general, there exist $a = (a_1, \ldots, a_s) \in (\mathbb{F}_2^n)^s$ such that $f(a) = 1$. Note that $a_{j_k} \neq c$ for any $k \in [1 \ldots d + 1]$. Consider the affine subspace

$$V = V_1 \times \ldots \times V_s, \quad V \in \left(\mathbb{F}_2^{|\mathcal{F}(C)|}\right)^s, \quad \text{where}$$
$$V_i = \begin{cases} \{\mathcal{F}(C)(a_i), \mathcal{F}(C)(c)\}, & \text{if } i \in \{j_1, \ldots, j_{d+1}\}, \\ \{\mathcal{F}(C)(a_i)\}, & \text{otherwise.} \end{cases}$$

Observe that $\bigoplus_{v \in V} g(v) = 1$. Indeed, $g(v) = 0$ for all $v \in V$ except $v = (\mathcal{F}(C)(a_1), \ldots, \mathcal{F}(C)(a_s))$. Since V is a $(d + 1)$-dimensional affine subspace, it follows that $\deg g \geq d + 1$, which is a contradiction. □

We can now prove our main result. At its core, it relies on the above lemma to bound the number of (bad) shuffling outcomes when f is constant, and on Lemma 1 (stating that a refreshed circuit is equivalent to a quadratic circuit) to lower-bound the error in good shuffling outcomes.

Theorem 3 (Main)**.** *Let C be an implementation and $s \geq 2$ an integer. The evaluation-phase model scheme $\mathsf{S} := \mathsf{EPM}(\tilde{C}, 1, s-1)$ is τ-error-d-AS for any $1 \leq d \leq s-1$, with*

$$\tau \geq \frac{1}{2^{2d}} \cdot \frac{s-d}{s}.$$

Proof. Consider an arbitrary $f \in \mathcal{F}^{(d)}(\mathsf{S.comp}) \setminus \{\mathbf{0}, \mathbf{1}\}$. We need to prove that when the input x of $\mathsf{S.enc}(x, r_e)$ is fixed, the error of $f(\mathsf{S.enc}(x, \cdot))$ is at least τ. Recall that $\mathsf{S.enc}$ uses r''_e (part of r_e) to shuffle the sequence $(x, r'_{e,1}, \ldots, r'_{e,s-1})$ (r'_e being another part of r_e), which is then passed to the input to f. By Lemma 2, in at most d/s fraction of the shuffling outcomes (i.e. positions i with $x'_i = x$) the function $f(\mathsf{S.enc}(x, \cdot)) = f|_{x_i = x}$ can be constant. Consider the remaining $(s-d)/s$ fraction of the outcomes. By Lemma 1, we can see $\mathcal{F}^{(1)}(\mathsf{S.comp})$ as spanned by at most quadratic functions of the input (it has the structure of a refreshed circuit), and so $\mathcal{F}^{(d)}(\mathsf{S.comp}) = (\mathcal{F}^{(1)}(\mathsf{S.comp}))^{(d)}$ spanned by functions of degree at most $2d$ (when composed with h^{-1} from Lemma 1). Since in the considered case f is non-constant, we can use the bound $\mathsf{err}(f) \geq 1/2^{2d}$. By combining the two different shuffling outcomes we obtain

$$\mathsf{err}\left(f(\mathsf{S.enc}(x, \cdot))\right) \geq \frac{d}{s} \cdot 0 + \frac{s-d}{s} \cdot \frac{1}{2^{2d}} = \frac{1}{2^{2d}} \cdot \frac{s-d}{s}. \qquad \square$$

This result shows that dummy shuffling together with the refreshing technique provides algebraic security for degrees up to the number of dummy slots. Furthermore, the error bound τ can be seen as close to the maximum possible $1/2^{2d}$ in e.g. Boolean circuit implementations, as was discussed in Subsection 2.3. We conclude that dummy shuffling with refreshing solves the problem of algebraic security, at least in the gray-box model of [BU18].

5.4 Implementation cost estimation

Dummy shuffling with refreshing allows cheap provably secure protection against algebraic attacks of any predetermined degree $d \geq 1$ using a single main slot and d dummy slots ($s_{\text{dummy}} = d$). We estimate roughly the number of gates required for implementing dummy shuffling.

Let l_{A} (resp. l_{X}) denote the number of AND gates (resp. the number of XOR gates) in the original implementation. In the input-shuffling phase, the cost is to generate $s_{\text{dummy}}(|x| + l_{\mathsf{A}})$ bits of randomness and shuffle s vectors of size $|x| + l_{\mathsf{A}}$ bits. For typical complex circuits C, the number of AND gates is much larger than the input size: $l_{\mathsf{A}} \gg |x|$, so we ignore the latter for our estimation. We utilize the controlled swap construction, which can be implemented in Boolean circuits using 4 gates as

$$(x_i, y_i) \mapsto \big((c \wedge (x_i \oplus y_i)) \oplus x_i,\ (c \wedge (x_i \oplus y_i)) \oplus y_i\big)$$

for each index i, where c is the control (random) bit. For $d = 1$, one controlled swap of l_{A}-bit state is sufficient for perfectly uniform shuffling. For $d > 1$, we only have to place the single main slot in a random position. This can be implemented in circuits using s_{dummy} conditional swaps of l_{A}-bit states, assuming a random bitstring of length s with a single one is generated, which would be negligible for the final cost. The total cost of such implementation of input-shuffling is

$$T_{IS} \approx 4 s_{\text{dummy}} \cdot l_{\mathsf{A}} = 4d \cdot l_{\mathsf{A}}$$

gates for swaps and generation of $s_{\text{dummy}} \cdot l_{\mathsf{A}} = d \cdot l_{\mathsf{A}}$ random bits for dummy slots. The output-selection phase can for example be implemented as the inverse of the input-shuffling, excluding randomness generation. Its cost is negligible since it only has to unshuffle the outputs, which we assume are much shorter than the nonlinear part of the circuit shuffled in the input-shuffling phase:

$$T_{OS} \approx 0.$$

The cost of the evaluation phase is

$$T_{EP} \approx s(|C| + l_{\mathsf{A}}) = 2s \cdot l_{\mathsf{A}} + s \cdot l_{\mathsf{X}} = (2d+2)l_{\mathsf{A}} + (d+1)l_{\mathsf{X}}$$

gates. We conclude with the total cost estimation of

$$T_{IS} + T_{EP} + T_{OS} \approx (6d+2)l_{\mathsf{A}} + (d+1)l_{\mathsf{X}}$$

gates and $d \cdot l_{\mathsf{A}}$ random bits.

6 Public dummy shuffling construction

In this section, we describe a construction of *public* dummy shuffling. This proof-of-concept shows that a white-box designer willing to implement dummy shuffling does not have to obfuscate the whole implementation but rather a single slot function.

The goal of the construction is to have a clear slot separation without any interaction between slots except the final merging step, which in our case is simply XOR of outputs of all the slots. The input-shuffling phase is also implicit and is performed inside the slot, using an extra *index* input, specifying the slot index. The high-level description of the scheme is as follows:

$$output = \bigoplus_{0 \le index < s} \text{slot}(input, index).$$

The construction implements dummy shuffling with a single main slot and multiple dummy slots. The location of main slot depends pseudorandomly on the input. More precisely, for any fixed input there exists a unique value of the index i that corresponds to the main slot computation, and this value should be hard to predict for an adversary, even after observing the outputs of slots. For this purpose, the output of each slot is "masked" by a pseudorandom mask, with the property that all masks XOR to zero. Note that the output of the main slot is masked too, since otherwise it would match the final output and thus would be trivial to locate.

When the slot function is implemented as a Boolean circuit, the construction can be implemented in a bit-slice style by performing bitwise operations on 32- or 64-bit registers. This allows to compute up to 32 or 64 slots in parallel without any significant overhead, leading to very efficient implementations.

The construction requires two standard pseudorandom functions (PRFs) and a special primitive called *tweakable zero-sum PRF*, which we formally define in the following.

Definition 16 (TZS-PRF)**.** A function with the signature $F_k[t](x) : \mathbb{F}_2^{|k|} \times \mathbb{F}_2^{|t|} \times \mathbb{F}_2^n \to \mathbb{F}_2^m$ is called a *tweakable zero-sum PRF* if

1. for all $k \in \mathbb{F}_2^{|k|}, t \in \mathbb{F}_2^{|t|}$ the function $F_k[t]$ sums to zero over $\mathbb{F}_2^n$:

$$\bigoplus_{x \in \mathbb{F}_2^n} F_k[t](x) = 0;$$

2. for a uniformly sampled $k \in \mathbb{F}_2^{|k|}$, the family F_k is computationally indistinguishable from a uniformly sampled function family $(f_t : \mathbb{F}_2^n \to \mathbb{F}_2^m)_{t \in \mathbb{F}_2^{|t|}}$ with the constraint $\bigoplus_{x \in \mathbb{F}_2^n} f_t(x) = 0$ for all $t \in \mathbb{F}_2^{|t|}$.

We describe a simple short-input TZS-PRF construction from a PRF in Appendix A, with the TZS-PRF security reduced to the PRF security. It is based on the following simple observation: the zero-sum property is equivalent to requiring each $F_k[t]$ have algebraic degree at most $n-1$. For short inputs, we can sample such functions uniformly by selecting each monomial of degree at most $n-1$ randomly with probability $1/2$. The general idea follows: multiply each monomial of degree at most $n-1$ by a pseudorandom bit derived from the tweak using another PRF, and sum all monomials to get one output coordinate. This construction is tailored to our application, where the TZS-PRF input has size logarithmic in the number of slots and so the number of considered monomials is linear in the number of slots.

Algorithm 1 Public dummy shuffling construction

Input: an implementation $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with l AND gates;
an integer $h \geq 1$;
$G_{k_1}(x) : \mathbb{F}_2^n \to \mathbb{F}_2^h$: a PRF instance (impl.);
$H_{k_2}(x) : \mathbb{F}_2^{n+h} \to \mathbb{F}_2^{n+l}$: a PRF instance (impl.);
$F_{k_3}[t](x) : \mathbb{F}_2^n \times \mathbb{F}_2^h \to \mathbb{F}_2^m$: a tweakable zero-sum PRF instance (impl.);
Output: slot implementation $S(x, i) : \mathbb{F}_2^n \times \mathbb{F}_2^h \to \mathbb{F}_2^m$, such that $\bigoplus_{i \in \mathbb{F}_2^h} S(x, i) = C(x)$.

Input-Shuffling:
1: **if** $G_{k_1}(x) = i$ **then** ▷ $G_{k_1}(x)$ determines the main slot index
2: $x' \leftarrow (x \mid\mid 0^l)$
3: **else**
4: $x' \leftarrow H_{k_2}(x \mid\mid i)$

Slot Evaluation:
5: $y' \leftarrow \tilde{C}(x')$ ▷ x, i are passed through

Output-Selection:
6: $mask \leftarrow F_{k_3}[x](i)$
7: **if** $G_{k_1}(x) = i$ **then** ▷ determine the main output
8: **return** $y \oplus mask$
9: **else**
10: **return** $mask$

We are now ready to describe our proof-of-concept public dummy shuffling construction. The high-level pseudocode is given in Algorithm 1. We now describe each step of the algorithm in details.

Line 1-4 First, the input x is used to determine the index $i \in \mathbb{F}_2^h$ of the main slot. For this purpose, the PRF G_{k_1} (with a hardcoded key) is used. If $G_{k_1}(x)$ is not equal to the value of i passed into the current slot, then the dummy input is generated by applying the PRF H_{k_2} to the full input (x, i). Otherwise, the original input is used and padded with zeroes.

Line 5 Main computation is done by using the refreshed circuit (as in Definition 15). By Line 1-4 of the algorithm, the input in the main slot is the original input x padded with zeroes, and the input in a dummy slot is fully pseudorandom. Note that x, i are passed through the slot evaluation phase. This does not introduce algebraic leakage, since otherwise an algebraic attack would serve as a distinguisher for the PRF G_{k_1} or H_{k_2}.

Line 6 The output mask is generated using the tweakable zero-sum PRF F_{k_3} tweaked by x. The necessary property is that the generated masks XOR to zero for any fixed input x.

Lines 7-10 The PRF G_{k_1} is again used to identify the main slot. In the main slot, the generated mask is XOR-ed with the output y' (which is equal to the main output) and returned. In dummy slots, the generated mask is returned unmodified. As a result, the output of the main slot is the correct output XOR-ed with an output mask, and the output of a dummy slot is simply an output mask. Since all output masks sum to zero, the sum of all slots outputs results in the desired output $C(x)$.

The slot evaluation phase can be proven to provide algebraic security, under the assumption of the pseudorandomness of H. More precisely, by Theorem 3, the scheme S with $\mathsf{S.enc}$ defined by Lines 1-4, $\mathsf{S.comp}$ defined by Line 5, and $\mathsf{S.dec}$ defined by lines 6-10, is τ-error-d-AS for any $1 \leq d \leq s-1$, with

$$\tau \geq \frac{1}{2^{2d}} \cdot \frac{s-d}{s}.$$

This proves that algebraically secure *computations* are possible for any fixed degree and any target circuit. However, the whole construction can be still susceptible to algebraic attacks of degree 2, if the sensitive terms are computed in clear, namely $[G_{k_1}(x) = i]$, which identifies the main slot. Provably secure implementation of these functions is left as future work: it would first require a meaningful extension of the algebraic security model to include encoding and decoding phases[5] .

Note that the output masks used in the construction are used not for achieving the algebraic security, but to prevent black-box slot identification attacks. Indeed, without the masks, all the dummy slots will have the all-zero output and thus, the main slot at each execution would be trivially identifiable. Any obfuscation of the slot procedure would not prevent the attack, since only outputs of the slots are used. Therefore, the outputs should not reveal the location of the main slot. In particular, the output of the main slot should be indistinguishable from an output of any dummy slot, even with the knowledge of the main output. This is naturally guaranteed by the tweakable zero-sum PRF security. Indeed, in our scheme the adversary is given access to the TZS-PRF modified by XORing a constant (the main output of the scheme) to a single output of the TZS-PRF per each tweak. Note that for an ideal TZS-PRF this modification produces the same distribution of random function families independently of which output is modified (and of the constant, which can be chosen adversarially). Therefore, the adversary can not gain any advantage in guessing which output is modified, or, equivalently, what is the index of the main slot.

7 Conclusions

In this work, we analyzed algebraic security of dummyless and dummy shuffling in the gray-box model of [BU18]. Dummy shuffling allows to achieve close to optimal security for arbitrary degrees of the attack with reasonable overhead. This is a rather surprising development, since the minimalist quadratic masking scheme of [BU18] was already rather heavy. We conclude that this work solves the open problem of higher-order algebraic security and provides useful tools for white-box implementations. Nonetheless, there are still many open questions around the topic.

Towards white-box model. The current BU-model covers only the main computation part. A natural question is how to extend this model to cover both encoding and decoding steps, including pseudorandomness generation. Steps were made towards such a solution in the context of probing security [IKL+13, CGZ20]. Finally, dummy shuffling requires to generate a lot of random bits *in the encoding* step. This leads to large intermediate state and may incur a large overhead for further obfuscation. Therefore, a masking-style solution to higher-degree algebraic security is still a desirable tool.

[5] Direct extension is not possible, since input and output are sensitive functions by definition and will be leaked in the encoding/decoding phases.

Public dummy shuffling. We proposed a proof-of-concept construction of public dummy shuffling. An interesting task is to develop an efficient instantiation using existing PRFs or develop new white-box-friendly PRFs. In this work, we defined public dummy shuffling and proposed a proof-of-concept construction. It reduces potential obfuscation overhead in white-box implementations. It relies on a new primitive called *tweakable zero-sum pseudorandom-function*, for which we proposed a simple construction with a security proof. An open question is to optimize its parameters, or develop a new more efficient construction.

Fault attacks. Fault attacks pose a dangerous threat to dummy shuffling. Most importantly, faults can be used to distinguish main slots from dummy slots in public dummy shuffling (as was done in [GRW20]), and aid algebraic attacks in hidden dummy shuffling. For example, the attacker can filter the inputs for which chosen intermediate values lead to a difference in the output when faulted. In a basic dummy shuffling, this would identify the inputs for which those intermediate values belong to a main slot.

We conclude that the topic of algebraic security and, in general, provable countermeasures for white-box implementations still has many interesting open problems and research directions.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. The work of Aleksei Udovenko was partly supported by the French FUI-AAP25 IDECYS+ project and by the French ANR-AAPG2019 SWITECH project; part of his work was performed while he was at the University of Luxembourg and supported by the Luxembourg National Research Fund (FNR) project FinCrypt (C17/IS/11684537).

References

[BGEC04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 227–240. Springer, Heidelberg, 2004. 3

[BGK+19] Andrey Bogdanov, Louis Goubin, Stefan Kölbl, Pascal Paillier, Matthieu Rivain, Elmar Tischhauser, and Junwei Wang. CHES 2019 Capture The Flag Challenge. The WhibOx Contest, 2nd Edition, 2019. https://whibox-contest.github.io/2019/. 3, 4

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 215–236. Springer, Heidelberg, 2016. 3, 5

[BKMS18] Cees-Bart Breunesse, Ilya Kizhvatov, Ruben Muijrers, and Albert Spruyt. Towards fully automated analysis of whiteboxes: Perfect dimensionality reduction for perfect leakage. Cryptology ePrint Archive, Report 2018/095, 2018. 5

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, pages 435–440. ACM Press, 2000. 10

[BRVW19] Andrey Bogdanov, Matthieu Rivain, Philip S. Vejre, and Junwei Wang. Higher-order DCA against standard side-channel countermeasures. In Ilia Polian

and Marc Stöttinger, editors, *COSADE 2019*, volume 11421 of *LNCS*, pages 118–141. Springer, Heidelberg, 2019. 3, 10

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 373–402. Springer, Heidelberg, 2018. 3, 4, 5, 6, 7, 8, 9, 13, 14, 23, 26

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin Kaya Koç and Christof Paar, editors, *CHES 2000*, volume 1965 of *LNCS*, pages 252–263. Springer, Heidelberg, 2000. 15

[CEJv02] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In *Digital Rights Management Workshop*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002. 3

[CEJv03] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, Heidelberg, 2003. 3

[CGZ20] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 342–375. Springer, Heidelberg, 2020. 26

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 486–514. Springer, Heidelberg, 2017. 10

[GPRW20] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *Journal of Cryptographic Engineering*, 10(1):49–66, 2020. 3, 6, 7

[GRW20] Louis Goubin, Matthieu Rivain, and Junwei Wang. Defeating state-of-the-art white-box countermeasures. *IACR TCHES*, 2020(3):454–482, 2020. 3, 4, 5, 6, 11, 27

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06*, volume 3989 of *LNCS*, pages 239–252. Springer, Heidelberg, 2006. 3

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust pseudorandom generators. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP 2013, Part I*, volume 7965 of *LNCS*, pages 576–588. Springer, Heidelberg, 2013. 26

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, 2003. 7

[PCY+17] Emmanuel Prouff, Chen-Mou Cheng, Bo-Yin Yang, Thomas Baignères, Matthieu Finiasz, Pascal Paillier, and Matthieu Rivain. CHES 2017 Capture The Flag Challenge. The WhibOx Contest, 2017. https://whibox-contest.github.io/2017/. 3

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 171–188. Springer, Heidelberg, 2009. 3

[RW19] Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR TCHES*, 2019(2):225–255, 2019. 3

[SEL21] Okan Seker, Thomas Eisenbarth, and Maciej Liskiewicz. A white-box masking scheme resisting computational and algebraic attacks. *IACR TCHES*, 2021(2):61–105, 2021. 3, 5

[THM07] Stefan Tillich, Christoph Herbst, and Stefan Mangard. Protecting AES software implementations on 32-bit processors against power analysis. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 141–157. Springer, Heidelberg, 2007. 3

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 740–757. Springer, Heidelberg, 2012. 3, 10

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *SAC 2007*, volume 4876 of *LNCS*, pages 264–277. Springer, Heidelberg, 2007. 3

A Tweakable zero-sum PRFs

We describe a simple short-input TZS-PRF construction. The general idea is based on the simple observation: the zero-sum requirement over $\mathbb{F}_2^n$ is equivalent to all output coordinates of $F_k[t]$ having degree at most $n-1$. We construct a tweakable degree-$(n-1)$ function by conditionally selecting each possible monomial of the input, with the condition bits derived from the tweak via another PRF. Intuitively, an instance of $F_k[t]$ only allows to derive the condition bits, and so the security reduces to distinguishing the condition bits from fully random, which is guaranteed by the PRF security.

We now prove that the TZS-PRF security of this construction directly reduces to the PRF security of H.

Proposition 4. *For any adversary A, let $\mathsf{Adv}_F^{TZS\text{-}PRF}(A)$ denote the advantage of winning the TZS-PRF security game, i.e., distinguishing F_k for a uniformly sampled $k \in \mathbb{F}_2^{|k|}$ from a uniformly sampled function family $(f_t : \mathbb{F}_2^n \to \mathbb{F}_2^m)_{t \in \mathbb{F}_2^{|t|}}$ with the constraint $\bigoplus_{x \in \mathbb{F}_2^n} f_t(x) = 0$ for all $t \in \mathbb{F}_2^{|t|}$.*

Assume A that does at most Q queries and runs in time T. Then, there exist an adversary B against the PRF H_k doing at most Q queries and running in time $T + \mathcal{O}(m(2^n-1)) \cdot Q$, such that

$$\mathsf{Adv}_F^{TZS\text{-}PRF}(A) \le \mathsf{Adv}_H^{PRF}(B).$$

Algorithm 2 Tweakable zero-sum PRF construction

Require: $H_k(x) : \mathbb{F}_2^{|k|} \times \mathbb{F}_2^{|t|} \to \mathbb{F}_2^{m(2^n-1)}$: a PRF
Input: $k \in \mathbb{F}_2^{|k|},\ t \in \mathbb{F}_2^{|t|},\ x \in \mathbb{F}_2^n$;
Output: $F_k[t](x) \in \mathbb{F}_2^m$.

1: $y \leftarrow 0 \in \mathbb{F}_2^m$
2: $t' \leftarrow H_k(t)$
3: **for all** $u \in \mathbb{F}_2^n, u \neq (1, \ldots, 1)$ **do**
4: $\quad i \leftarrow$ integer representation of u
5: $\quad x^u \leftarrow x_1^{u_1} \ldots x_n^{u_n}$
6: $\quad$ **for all** $j \in [1 \ldots m]$ **do**
7: $\quad\quad y_j \leftarrow y_j \oplus t'_{mi+j} \cdot x^u$
8: **return** y

Proof. Construct an adversary B in the PRF game as follows. Simulate A, and on a query to $F_k[t](x)$ (resp. $f_t(x)$), query $H_k(t)$ (resp. the random oracle) and compute $F_k[t](x)$ as in Algorithm 2, replacing $t' = H_k(t)$ with the query output.

Observe that the transformation from the ANF of y_j to the truth table of y_j is a linear *bijection*. This is also true when the monomial $x_1 x_2 \ldots x_n$ is excluded. More precisely, the $2^n - 1$ ANF coefficients (excluding the coefficient of $x_1 \ldots x_n$) are in bijection with the set of all Boolean functions $\mathbb{F}_2^n \to \mathbb{F}_2$ summing to zero. Since each output coordinate y_j is computed using independent parts of t', all the $m(2^n - 1)$ bits of t' are in bijection with all vectorial Boolean functions $\mathbb{F}_2^n \to \mathbb{F}_2^m$ summing to zero. In the case B accesses the random oracle, it follows from the ANF property that A will access a uniformly sampled function family with the zero-sum constraint. In the case B accesses the real PRF $H_k(\cdot)$, the adversary A will have access to the real TZS-PRF $F_k[\cdot](\cdot)$. If A succeeds in the TZS-PRF game, then B succeeds in the PRF game. Note that each query of A incurs the overhead of computing Algorithm 2 for B, which is $\mathcal{O}(m(2^n - 1))$ bit operations. □