Uniform First-Order Threshold Implementations

Tim Beyne(✉) and Begül Bilgin

ESAT/COSIC, KU Leuven and iMinds, Leuven, Belgium
tim.beyne@student.kuleuven.be, begul.bilgin@esat.kuleuven.be

Abstract. Most masking schemes used as a countermeasure against side-channel analysis attacks require an extensive amount of fresh random bits on the fly. This is burdensome especially for lightweight cryptosystems. Threshold implementations (TIs) that are secure against first-order attacks have the advantage that fresh randomness is not required if the sharing of the underlying function is uniform. However, finding uniform realizations of nonlinear functions that also satisfy other TI properties can be a challenging task. In this paper, we discuss several methods that advance the search for uniformly shared functions for TIs. We focus especially on three-share implementations of quadratic functions due to their low area footprint. Our methods have low computational complexity even for 8-bit Boolean functions.

Keywords: Boolean functions · Correction terms · Masking · Randomness · The threshold implementations · Uniformity

1 Introduction

Side channel attacks (SCA), which are shown to be a great threat to today's cryptosystems [1,14,15], derive sensitive information (e.g. secret key) by correlating various characteristics of the device such as timing, power consumption and electromagnetic emanation leakages with intermediate values of the cryptographic algorithm during execution [15,16]. In this paper, we consider adversaries that can only use first-order SCA, i.e. can use only first-order statistical moments of the side-channel information or equivalently can use information from a single wire [13]. The threshold implementation (TI) method is a countermeasure that is proven to be secure with minimal adversarial and implementation assumptions [4,19,21] and is used for symmetric-key algorithms. Being a masking scheme, its essence lies in splitting the sensitive data into s uniformly distributed shares and adopting the (round) functions to operate on these shares in a way that the correct output is calculated. Unlike other masking schemes, first-order TI additionally requires each output share of a function to be independent of at least one of its input shares. This enables security on demanding non-ideal (such as glitchy) circuits and is called the non-completeness property. The uniformly shared input combined with non-completeness randomizes the calculation, and hence breaks the linear relation between the side-channel information and the sensitive data for each function.

In the most generic case, the non-completeness property implies the bound $s \geq td + 1$ on the number of shares where t is the algebraic degree of the function [4,19] and d is the attack order. Hence, in our setting s increases only with the degree of the underlying function to be calculated. Fortunately, any high degree permutation can be represented by a combination of quadratic functions, by means of sequential combination alone [2] or parallel and sequential combinations together [10,17]. Since more shares typically imply an increase in required resources such as area, it is desired to keep s as low as possible. Therefore, we mainly target implementations with three shares while keeping the discussions generic.

Related Work. When the round-based nature of symmetric-key algorithms and the uniformly shared input requirement are considered, it is useful to construct the sharing of nonlinear functions in each round such that their output, which is the input of the following round, is also uniform. A sharing of a function (realization) satisfying this property is called a uniform sharing (realization).

So far, the strategy for finding uniform realizations has been to exhaustively check uniformity for all possible non-complete realizations. For some Sboxes [20], this strategy yields positive results rather quickly. However, even for small, low-degree Sboxes with few shares the search space of possible realizations is very large [7]. Therefore, proving the (non-)existence of a non-complete uniform sharing for a particular nonlinear function is a difficult task [2,3,18].

Alternative to finding a uniform realization, fresh randomness can be added to the output shares of a nonuniform realization. This operation, which makes the sharing uniform, is commonly referred to as *remasking*. The increased cost of high throughput fresh random number generation is undesired and sometimes even unaffordable for a lightweight system.

Contribution. Even though there may not exist a known uniform realization of a given vectorial Boolean function, it is beneficial to find a subset of outputs for which the realization can be made jointly uniform since this reduces the randomness cost significantly. Starting from this partially uniform realization idea, which is described in Sect. 2.5, we focus on finding uniform realizations for Boolean functions, then combine them appropriately. Finding uniform realizations has two main challenges. First, no efficient method to check the uniformity of a realization has been presented so far. Second, if the realization under test is not uniform, another realization needs to be checked and a systematic way to reduce the search space has not been presented yet.

In this paper, we tackle both of these challenges. In Sect. 3, we introduce an efficient method to check uniformity. In Sects. 4 and 5, we respectively discuss adding linear and quadratic terms to output shares in order to make the realization uniform and provide examples. We prove that any realization which uses a bent function as an output share can not be made uniform by adding only linear terms. We also re-prove that there exists no uniform realization of a nonlinear function with two inputs and one output with three shares. This result was previously shown by exhaustive search [19].

2 Threshold Implementations

2.1 Notation

We denote the vector space of dimension n over the Galois field of order 2 by $\mathbb{F}^n$. We use lower case characters for elements of $\mathbb{F}^n$ and vectorial Boolean functions from $\mathbb{F}^n$ to $\mathbb{F}^m$. Superscripts refer to each bit and each coordinate function, i.e. $x = (x^1, \ldots, x^n)$ where $x^i \in \mathbb{F}$ and $f = (f^1, \ldots, f^m)$ where f^i is a Boolean function. We omit the superscript when $n = 1$ for elements and $m = 1$ for functions. The ring of $n \times m$ matrices over $\mathbb{F}$ is written as $\mathbb{F}^{n \times m}$. The dot-product and the field addition of x, y are denoted by $x \cdot y$ and $x + y$ respectively. $\overline{x}$ represents the bitwise complement of x. $|\mathcal{S}|$ denotes the cardinality of the set $\mathcal{S}$.

The notation used for TIs is similar to [2,4,5,19,21]. A correct s share vector $\boldsymbol{x}^i = (x^i_1, \ldots, x^i_s)$ of x^i has the property that $x^i = \sum_{j=1}^{s} x^i_j$. In particular, $\mathrm{Sh}(x^i)$ is the set of correct sharings for the variable x^i. This notation can be readily extended to elements of $\mathbb{F}^n$ and (vectorial) Boolean functions. The sharing $\boldsymbol{f} = f^1_1, f^1_2, \ldots, f^m_{s_{\text{out}}}$ defined from $\mathbb{F}^{ns_{\text{in}}}$ to $\mathbb{F}^{ms_{\text{out}}}$ with s_{in} input and s_{out} output shares is called a *realization*. The realization is correct if $f^i = \sum_{j=1}^{s_{\text{out}}} f^i_j$ for all i. Each share f^i_j of a coordinate function f^i is called a component function. Constructing a uniform and non-complete realization for a linear function is trivial [19]. Therefore, we focus only on nonlinear functions.

2.2 Non-completeness

Non-completeness is the key property that makes TI secure even on glitchy circuits. Without loss of generality, a first-order non-complete realization has the property that its i^{th} output share is independent of its i^{th} input share [19]. This independence implies that leakage of a single share is independent of the unmasked input, proving the security [19]. As described in Sect. 1, $s_{\text{in}}, s_{\text{out}} \geq t+1$ due to this property [19]. A non-complete three-share realization $\boldsymbol{y} = (y_1, y_2, y_3)$ of an AND gate ($y = f(x) = x^1 x^2$) is provided in Eq. (1) as an example.

$$\begin{aligned}
y_1 &= f_1(x^1_2, x^1_3, x^2_2, x^2_3) = x^1_2 x^2_2 + x^1_2 x^2_3 + x^1_3 x^2_2 \\
y_2 &= f_2(x^1_1, x^1_3, x^2_1, x^2_3) = x^1_3 x^2_3 + x^1_1 x^2_3 + x^1_3 x^2_1 \\
y_3 &= f_3(x^1_1, x^1_2, x^2_1, x^2_2) = x^1_1 x^2_1 + x^1_1 x^2_2 + x^1_2 x^2_1
\end{aligned} \tag{1}$$

2.3 Uniformity

A sharing of a variable is uniform if, for each unshared value $x \in \mathbb{F}^n$, every $\boldsymbol{x} \in \mathrm{Sh}(x)$ occurs with equal probability. A realization $\boldsymbol{f}$ is called uniform if, for uniformly generated input, its output is also uniformly generated. Namely, $\boldsymbol{f}$ is uniform if and only if

$$\mathcal{N}_{\mathcal{U}} = |\{\boldsymbol{x} \in \mathrm{Sh}(x) | \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}\}| = \frac{2^{n(s_{\text{in}}-1)}}{2^{m(s_{\text{out}}-1)}}$$

for each input $x \in \mathbb{F}^n$ and $y = f(x) \in \mathbb{F}^m$ [5].

Definition 1 (Uniformity table $\mathcal{U}$). *Let $\boldsymbol{f}$ be a shared realization from $\mathbb{F}^{ns_{in}}$ to $\mathbb{F}^{ms_{out}}$ and $\mathcal{U}_{x,\boldsymbol{y}}$ be the cardinality of $\{\boldsymbol{x} \in Sh(x) | \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}\}$ for the unshared input x and shared output $\boldsymbol{y}$. The $2^n \times 2^{ms_{out}}$ table which has $\mathcal{U}_{x,\boldsymbol{y}}$ as its $(x, \boldsymbol{y})^{th}$ element is called the uniformity table $\mathcal{U}$ of $\boldsymbol{f}$.*

Here, we assume that the rows and columns of $\mathcal{U}$ are ordered lexicographically by each unshared then shared output. If $\boldsymbol{f}$ is uniform, the elements of $\mathcal{U}$ are equal to either 0 or $\mathcal{N}_{\mathcal{U}}$. We provide the uniformity table of Eq. (1) in Table 1 for completeness [5].

Table 1. The uniformity table of Eq. (1).

(x^1, x^2)	(y_1, y_2, y_3)							
	000	011	101	110	001	010	100	111
(0,0)	7	3	3	3	0	0	0	0
(0,1)	7	3	3	3	0	0	0	0
(1,0)	7	3	3	3	0	0	0	0
(1,1)	0	0	0	0	5	5	5	1

Note that Table 1 shows that the aforementioned realization is not uniform since the table contains elements different from 0 and $\mathcal{N}_{\mathcal{U}} = 4$. Since we want to limit the randomness requirement to minimize resources, we focus on methods to find partially or completely uniform sharings directly. We also keep the number of shares as small as possible for performance considerations.

2.4 Correction Terms

When a realization is not uniform, it is nevertheless possible that a different construction of the component functions yields a uniform realization. One possible way of generating alternative realizations is adding *correction terms* (CTs) to an even number of component functions without breaking the non-completeness property [19]. We assume CTs are generated using only the input shares of the realization.

Consider a realization of a quadratic Boolean function with n variables with three output and input shares. The number of linear and quadratic CTs is $3(n + \binom{n}{2})$. Therefore, there exist $2^{3(n+\binom{n}{2})}$ possible non-complete three share realizations for this function. If we consider such realizations for 3- and 4-bit (quadratic) Sboxes, we get 2^{18} and 2^{30} possibilities for each coordinate function and $(2^{18})^3$ and $(2^{30})^4$ possible realizations for the Sbox [6].

2.5 Partial Uniformity

Definition 2 (Partial Uniformity). *Consider the function f with m coordinate functions f^i. A realization that is uniform in at least one l-combination of its coordinate functions, i.e. without loss of generality with uniform $f_1^1, f_2^1, \dots f_{s_{out}}^l$, is called a partially uniform realization of f.*

The case where $l = m$ implies that f has a uniform realization. The motivation to find a partially uniform function is that, if l output variables are jointly uniform, they do not need to be remasked to make the joint distribution of output shares uniform [3]. Hence, the required randomness for remasking can be reduced from $m \cdot (s_{\text{out}} - 1)$ bits to $(m - l) \cdot (s_{\text{out}} - 1)$ bits. Note that by using this method alone, the authors of [3] gained 60% efficiency on fresh randomness.

A straightforward way to find partially uniform realizations, which we apply, starts by finding uniform realizations for each coordinate function of $\boldsymbol{f}$. These realizations are then combined iteratively until it is not possible to combine any more component functions uniformly[1]. Therefore, we mainly focus on finding uniform realizations of a Boolean function efficiently in the rest of the paper and use their combinations for partial uniformity only on examples.

There are two main obstacles in this approach:

1. It is relatively expensive to check whether a given realization is uniform. So far, the only proposed way to check uniformity is generating the uniformity table $\mathcal{U}$ of the realization completely and checking if its nonzero elements are equal to $\mathcal{N}_{\mathcal{U}}$. This requires $2^{ns_{\text{in}}}$ evaluations of the realization (for all possible s_{in} shares of each of the n input variables) in the worst case.
2. Going through all possible realizations, i.e. trying all possible CTs, can be extremely expensive due to the large amount of CTs as discussed at the end of Sect. 2.3. Even if we focus only on the linear CTs, there exist $2^{ns_{\text{in}}}$ different realizations implying $O(2^{ns_{\text{in}}}2^{ns_{\text{in}}}) = O(2^{2ns_{\text{in}}})$ complexity to check uniformity for all of them.

Therefore, both decreasing the search space of possible realizations and reducing the complexity of checking uniformity for each realization would have significant impact on the overall complexity of finding uniform realizations for Sboxes.

3 Fast Uniformity Check for Boolean Functions

This section aims to reduce the complexity O_U of checking whether a given realization of a Boolean function is uniform. In order to do that, we first analyze the dependencies between the elements $\mathcal{U}_{x,\boldsymbol{y}}$ of the uniformity table. We observe that if the realization has three output shares, it is sufficient to calculate only one row of $\mathcal{U}$ due to the dependency between $\mathcal{U}_{x,\boldsymbol{y}}$, reducing O_U. For this reason, we consider the case $s_{\text{out}} = 3$ in the second half of this section. Note that using three output shares limits the degree of the Boolean function to two. However, any high-degree function can be decomposed into quadratic Boolean functions and using a small number of shares typically reduces the implementation cost [5,7,17].

[1] One possible algorithm to find a (partial) uniform realization is provided in Appendix A for completeness. Note that this algorithm returns a uniform realization if it exists.

3.1 Observations on the Rows of $\mathcal{U}$

Consider a non-complete realization $\boldsymbol{f}$ with s_{in} input and s_{out} output shares, represented by $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively. For each unshared input x, let $N_{x,i}$ denote the number of inputs of $\boldsymbol{f}$ for which the component y_i has a fixed value $b \in \{0,1\}$. That is

$$N_{x,i} = \begin{cases} \sum_{\boldsymbol{y}} y_i \cdot \mathcal{U}_{x,\boldsymbol{y}} & \text{if } b = 1 \\ \sum_{\boldsymbol{y}} \overline{y_i} \cdot \mathcal{U}_{x,\boldsymbol{y}} & \text{if } b = 0. \end{cases} \tag{2}$$

Lemma 1. *$N_{x,i}$ is independent of x.*

Proof. Due to non-completeness of $\boldsymbol{f}$, y_i is (without loss of generality) independent of $(x_i^1, x_i^2, \ldots, x_i^n)$. Hence, $N_{x,i}$ is also independent of $(x_i^1, x_i^2, \ldots, x_i^n)$. Since the input sharing $\boldsymbol{x}$ is uniform, $N_{x,i}$ is independent of x. □

Hence, we will write N_i rather than $N_{x,i}$. The lemma implies that the entries of any row of $\mathcal{U}$ corresponding to some unshared input x are related by the same set of equations for a constant binary value b:

- For $1 \leq i \leq s_{\text{out}}$, N_i satisfies Eq. (2)
- The sum of the values must be equal to $2^{n(s_{\text{in}}-1)}$

$$\sum_{\boldsymbol{y}} \mathcal{U}_{x,\boldsymbol{y}} = 2^{n(s_{\text{in}}-1)}. \tag{3}$$

If $\boldsymbol{y} \in \mathrm{Sh}(\overline{f(x)})$ with $\overline{f(x)}$ the complement of $f(x)$, then $\mathcal{U}_{x,\boldsymbol{y}} = 0$. Hence, we will say that the system of Eqs. (2) and (3) has only $2^{s_{\text{out}}-1}$ unknowns.

Lemma 2. *Given Eq. (3), the equations given in Eq. (2) for $b = 1$ and $b = 0$ are linearly dependent.*

Proof. Form the coefficient matrix of the system of Eqs. (2) and (3) such that each row of the matrix represents an equation for which the columns are the coefficients of $\mathcal{U}_{x,\boldsymbol{y}}$. Clearly, for each i, the rows p and r of this matrix corresponding to N_i when $b = 0$ and $b = 1$ are binary complements. Let j be the row of ones corresponding to Eq. (3). Then $p = j - r$ describes the linear dependence among these rows for each i. □

Lemma 2 implies that there are at most $s_{\text{out}} + 1$ linearly independent equations describing the unknowns $\mathcal{U}_{x,\boldsymbol{y}}$. Since there are $2^{s_{\text{out}}-1}$ unknowns, the values N_i completely determine each row of $\mathcal{U}$ only if $2^{s_{\text{out}}-1} \leq s_{\text{out}} + 1$. This inequality holds only for $s_{\text{out}} \leq 3$. Since s_{out} must be greater than the degree of the function and we focus on nonlinear operations, $s_{\text{out}} = 3$. Note that fixing the number of output shares to three has no implication on the number of input shares, nor on the amount of input variables. In what follows, we investigate the case $s_{\text{out}} = 3$ further. For this case, Appendix B lists the four linearly independent equations for each unshared input x that describe the relation between elements in a single row of $\mathcal{U}$.

3.2 Observations on $\mathcal{U}$ when $s_{\mathbf{out}} = 3$

Theorem 1. *Let $\boldsymbol{f}$ be a realization of a Boolean function with $s_{out} = 3$. Then any row of its uniformity table $\mathcal{U}$ uniquely determines all elements of $\mathcal{U}$.*

Proof. Recall that the rows of $\mathcal{U}$ correspond to the unshared inputs. For any two inputs x, x', the systems of Eqs. (2) and (3) will be identical provided that we choose the same constant value for b. If the elements of some row are known, one can easily deduce the values N_i and hence the system of equations. The proof is completed by the fact that for $s_{\text{out}} = 3$ the system of equations completely determines the elements of any row. □

Note that the theorem does not imply that all rows are equal, since the unknowns in the system of equations for x and x' are different if $f(x) \neq f(x')$. Namely, they are $\mathcal{U}_{x,\boldsymbol{y}}$ with $\boldsymbol{y} \in \text{Sh}(f(x))$ and $\mathcal{U}_{x',\boldsymbol{y}}$ with $\boldsymbol{y} \in \text{Sh}(f(x'))$ respectively. Hence, the rows can in general take only two different values.

Corollary 1. *If the realization $\boldsymbol{f}$ of a Boolean function with $s_{out} = 3$ has a uniform distribution for one unshared input value (one row of $\mathcal{U}$), then it has a uniform distribution for all unshared input values (all rows).*

Proof. If the distribution of output shares is uniform for input x, then all nonzero elements in that row are equal to $\mathcal{N}_{\mathcal{U}}$. Hence, by Theorem 1, all values of $\mathcal{U}$ are fixed. Since the uniformity table of a uniform realization is a possible solution for $\mathcal{U}$, and the solution must be unique, it follows that $\boldsymbol{f}$ is a uniform realization. □

Using Corollary 1, the computational complexity of the uniformity check (O_U) when $s_{\text{out}} = 3$ is reduced to $O(2^{n(s_{\text{in}}-1)})$, i.e. computing a single row of $\mathcal{U}$. To be able to compare with the results of the following section, we note that at most $O(2^{n(s_{\text{in}}-1)}2^{ns_{\text{in}}}) = O(2^{2ns_{\text{in}}-n})$ evaluations are required if checking all linear CTs is desired. We conclude this section with the following theorem from which we will benefit in the remainder of the paper.

Theorem 2. *A realization with one output variable and three output shares is uniform if and only if each of its component functions is a balanced Boolean function.*

Proof. According to Theorem 1, it is enough to solve the system of Eqs. (2) and (3) for a single row of $\mathcal{U}$ to determine all the elements $\mathcal{U}_{x,\boldsymbol{y}}$. By Lemma 2, the equations for either $b = 0$ or $b = 1$ suffice. Hence, we consider the system of equations for $b = 0$ which is provided in Eq. (10) in Appendix B and simplified as the following extended coefficient matrix with columns ordered lexicographically by each shared, then unshared output:

$$\left(\begin{array}{cccccccc|c} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & N_1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & N_2 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & N_3 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2^{(s_{\text{in}}-1)n} \end{array}\right)$$

Depending on whether the output ($y = \sum_i y_i$) is 0 or 1, the elements $\mathcal{U}_{x,\boldsymbol{y}}$ corresponding to either the first or the second four coefficients of the matrix (equivalently either the first or the second line of each equation in Eq. (10)) are non-zero. Here, we only provide the solution for the system $y = 0$ given in Eq. (4). The solution for $y = 1$ is similar.

$$\begin{aligned}
\mathcal{U}_{x,(0,0,0)} &= -2^{(s_{\text{in}}-1)n-1} + \frac{1}{2}(N_1 + N_2 + N_3), \quad \mathcal{U}_{x,(0,1,1)} = 2^{(s_{\text{in}}-1)n-1} + \frac{1}{2}(N_1 - N_2 - N_3),\\
\mathcal{U}_{x,(1,0,1)} &= 2^{(s_{\text{in}}-1)n-1} + \frac{1}{2}(-N_1 + N_2 - N_3), \quad \mathcal{U}_{x,(1,1,0)} = 2^{(s_{\text{in}}-1)n-1} + \frac{1}{2}(-N_1 - N_2 + N_3)
\end{aligned} \tag{4}$$

$$\mathcal{U}_{x,(0,0,1)} = \mathcal{U}_{x,(0,1,0)} = \mathcal{U}_{x,(1,0,0)} = \mathcal{U}_{x,(1,1,1)} = 0$$

($\Rightarrow$): For a uniform realization, all non-zero expressions in Eq. (4) must be equal to each other and have the value $\mathcal{N_U} = 2^{n(s_{\text{in}}-1)-m(s_{\text{out}}-1)} = 2^{n(s_{\text{in}}-1)-2}$. This uniquely determines N_1, N_2 and N_3 for a uniform realization. In particular, we have

$$N_1 = N_2 = N_3 = 2^{n(s_{\text{in}}-1)-1}, \tag{5}$$

implying that each component function is balanced.
($\Leftarrow$): If each output bit is uniform satisfying Eq. (5), then each $\mathcal{U}_{x,\boldsymbol{y}}$ is either 0 or $\mathcal{N_U}$. This implies the uniformity of the realization. □

Note that one side of the proof stating that if the realization is uniform, each of the component functions must be balanced has already been proven in [7] and is independent of the number of shares or the degree of the function.

4 Using Linear Correction Terms Efficiently to Satisfy Uniformity

In this section, we show how to avoid trying all the linear correction terms one by one in order to find uniform realizations of Boolean functions. We benefit from the Walsh-Hadamard transformation (WHT) to directly see which linear correction terms can lead to uniform realizations and eliminate a significant portion of the search space. Even though we describe our method for $s_{\text{out}} = 3$ to benefit from Theorem 2, the idea can be used for efficient uniformity checks of component functions with $s_{\text{out}} > 3$.

Definition 3. *The Walsh-Hadamard transformation of f is denoted by W_f. For $\boldsymbol{\omega} \in \mathbb{F}^n$, it is given by*

$$W_f(\boldsymbol{\omega}) = \sum_{\boldsymbol{x} \in \mathbb{F}^n} (-1)^{f(\boldsymbol{x}) + \boldsymbol{\omega} \cdot \boldsymbol{x}},$$

i.e. the discrete Fourier transform of $(-1)^{f(\boldsymbol{x})}$.

Here, the addition in the exponent is in $\mathbb{F}$, and the summation is in the integers. This transformation can be efficiently calculated with $O(n2^n)$ computational complexity using fast WHT.

Definition 4. *A Boolean function f is called bent if and only if for all vectors $\boldsymbol{\omega} \in \mathbb{F}^n$, $W_f(\boldsymbol{\omega}) = 2^{n/2}$.*

Bent functions only exist for even n [22]. In our study, we will treat bent and non-bent component functions separately for reasons that will be clarified later in this section. Moreover, we will make use of the following well-known result.

Fact. f is balanced if and only if $W_f(\boldsymbol{0}) = 0$. Moreover, $f(\boldsymbol{x}) + \boldsymbol{a} \cdot \boldsymbol{x}$ is balanced if and only if $W_f(\boldsymbol{a}) = 0$.

4.1 Realizations Without Bent Component Functions

Adding linear correction terms to a nonuniform realization $\boldsymbol{f} = (f_1, f_2, f_3)$ is described by the following equations, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are binary correction vectors.

$$f_1' = f_1 + \boldsymbol{a}_{\hat{1}} \cdot \boldsymbol{x} \quad f_2' = f_2 + \boldsymbol{b}_{\hat{2}} \cdot \boldsymbol{x} \quad f_3' = f_3 + (\boldsymbol{a}_{\hat{1}} + \boldsymbol{b}_{\hat{2}})_{\hat{3}} \cdot \boldsymbol{x}.$$

The notation $\boldsymbol{a}_{\hat{i}}$ indicates that the bits corresponding to every i^{th} share are zero. Due to the restrictions implied by this notation, the new sharing is non-complete.

By Theorem 2, (f_1', f_2', f_3') is uniform if and only if f_i's are balanced. Therefore, $W_{f_1}(\boldsymbol{a}_{\hat{1}})$, $W_{f_2}(\boldsymbol{b}_{\hat{2}})$ and $W_{f_3}((\boldsymbol{a}_{\hat{1}} + \boldsymbol{b}_{\hat{2}})_{\hat{3}})$ must be zero which can easily be checked by using fast WHT. We use Algorithm 1.

Algorithm 1. Find linear correction terms for the realization $\boldsymbol{f} = (f_1, f_2, f_3)$.

```
 1: Compute $W_{f_1}$, $W_{f_2}$ and $W_{f_3}$ using WHT.
 2: for $\boldsymbol{a}_{\hat{1}} \in \mathbb{F}^{n s_{\text{in}}}$ do
 3:     if $W_{f_1}(\boldsymbol{a}_{\hat{1}}) \neq 0$ then
 4:         continue
 5:     end if
 6:     for $\boldsymbol{b}_{\hat{2}} \in \mathbb{F}^{n s_{\text{in}}}$, $(\boldsymbol{a}_{\hat{1}} + \boldsymbol{b}_{\hat{2}})_{\hat{3}}$ do
 7:         if $W_{f_2}(\boldsymbol{b}_{\hat{2}}) = 0$ and $W_{f_3}((\boldsymbol{a}_{\hat{1}} + \boldsymbol{b}_{\hat{2}})_3) = 0$ then
 8:             yield $(\boldsymbol{a}_{\hat{1}}, \boldsymbol{b}_{\hat{2}})$
 9:         end if
10:     end for
11: end for
```

The computational complexity of the three Walsh-Hadamard transformations in this algorithm is $O(n(s_{\text{in}} - 1) \cdot 2^{n(s_{\text{in}}-1)})$. The outer for-loop iterates over $2^{n(s_{\text{in}}-1)}$ values, the inner over $2^{n(s_{\text{in}}-2)}$ values. Hence, the for loop has complexity $O(2^{n(2s_{\text{in}}-3)})$. It follows that the total worst-case complexity is

$$O\left(n(s_{\text{in}} - 1) \cdot 2^{n(s_{\text{in}}-1)} + 2^{n(2s_{\text{in}}-3)}\right) = O\left(2^{n(2s_{\text{in}}-3)}\right).$$

To find a single solution the best-case complexity is $O\left(n(s_{\text{in}} - 1) \cdot 2^{n(s_{\text{in}}-1)}\right)$.

The table below summarizes the complexities of each uniformity-check method presented so far when only linear correction terms are considered. The efficiency of using the WHT is clear as the input size of the Boolean function increases. We emphasize that to find uniform realizations of vectorial Boolean functions all the aforementioned methods should be repeated for each coordinate function. Hence the complexity gain observed for a single Boolean function in the following table gains in significance for Sboxes.

Method	Worst-case complexity	$s_{\text{in}} = 3$	$s_{\text{in}} = 4$
Naive	$O(2^{2ns_{\text{in}}})$	$O(2^{6n})$	$O(2^{8n})$
Fast uniformity check	$O(2^{2ns_{\text{in}}-n})$	$O(2^{5n})$	$O(2^{7n})$
Using WHT	$O(2^{2ns_{\text{in}}-3n})$	$O(2^{3n})$	$O(2^{5n})$

Application to $\mathcal{Q}^4_{300}$. It has been shown in [7,10,17] that 4-bit permutations can be decomposed into quadratic functions in order to enable three-share realization of cryptographic algorithms. There exist six 4-bit quadratic permutation classes [7] up to affine equivalence that can be used for decomposition. For all quadratic permutation classes except one (namely $\mathcal{Q}^4_{300}$ as denoted by [7]) a uniform realization with three-shares has been found. However, for class $\mathcal{Q}^4_{300}$ the (non-)existence result was inconclusive so far since the search space is too big for practical verification. By using Algorithm 1 together with Algorithm 3 on the representative of $\mathcal{Q}^4_{300}$, we found that two out of four coordinate functions have jointly uniform realizations as described in Appendix C. This implies a 50% reduction when a permutation from $\mathcal{Q}^4_{300}$ is instantiated, which shows the relevance of this section. We note that no further improvements are possible for this permutation using only linear correction terms.

4.2 Realizations with Bent Component Functions

Theorem 3. *If any component function of a realization—seen as a function on $\mathbb{F}^{n(s_{in}-1)}$—is bent, then this realization is not uniform and it can not be made uniform by using only linear correction terms.*

Proof. Take one of the component functions f_i of the realization of f, viewed as a function on $\mathbb{F}^{n(s_{\text{in}}-1)}$. Further assume f_i is bent. Since the Walsh spectrum of f_i does not contain any zeros, it is clear that neither f_i is a balanced function nor any linear correction term makes f_i balanced. Hence, the realization cannot be made uniform with only linear CTs. Note that for $s_{\text{out}} \geq 4$, balancedness is still a necessary condition. Thus, the theorem also holds for more than three output shares. $\square$

We discuss two ad-hoc solutions to remedy this problem for any nonlinear function in Appendix D. More generally, it is easier to find linear correction terms if the number of zeros of the Walsh-Hadamard transform of each of the components is high. Section 5 provides further insight into this matter.

5 Finding Uniform Realizations of Quadratic Functions

So far we only focused on using linear CTs to find uniform realizations. In this section we benefit from quadratic forms to find quadratic CTs to enable uniform sharing on quadratic Boolean functions even if they have bent component functions.

5.1 Quadratic Forms

Any function $f : \mathbb{F}^n \to \mathbb{F}$ composed of only quadratic terms can be described by its *quadratic form* as $f(x) = x\,S\,x^T$ with S an upper triangular coefficient matrix.

Similarly, its *bilinear form* $B_f(x,y) = f(x+y) + f(x) + f(y)$ can be described by the equation $B_f(x,y) = y\,(S + S^T)\,x^T$. This bilinear map defines a subspace of $\mathbb{F}^n$, given by $\mathrm{rad}(f) = \{x \in \mathbb{F}^n \mid \forall y \in \mathbb{F}^n : B_f(x,y) = 0\}$, i.e. the radical or kernel of f. It follows from the rank-nullity theorem that $\dim(\mathrm{rad}(f)) = n - \mathrm{rank}\,(S + S^T)$.

Proposition 1. [22] *f is bent if and only if* $\dim(\mathit{rad}(f)) = 0$.

Let L be an $n \times n$ invertible matrix. Then $(S + S^T)$ and $L^T(S + S^T)L$ are called *cogredient*. The cogredience relation divides the set of $n \times n$ matrices into mutually disjoint classes of cogredient matrices with the same rank.

It is well known that any symmetric matrix over $\mathbb{F}$ has the following normal form [23]:

$$N = \begin{pmatrix} \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix} & & & \\ & \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix} & & \\ & & \ddots & \\ & & & 0 \end{pmatrix}. \tag{6}$$

That is, there exists an invertible matrix T such that $S + S^T = T\,N\,T^T$. For more information on quadratic forms over fields of characteristic two, see [11,23].

5.2 Quadratic Forms in TI Context

Let $f(x)$ where $x \in \mathbb{F}^n$ be a quadratic Boolean function to be shared with the realization $\boldsymbol{f} = (f_1, \ldots, f_{s_{\text{out}}})$. In addition, let M_i be the matrices associated with the bilinear form of f_i, that is, $B_{f_i}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} M_i \boldsymbol{y}^T$ where

$$M_i = \begin{pmatrix} 0 & X_i^{12} & X_i^{13} & \cdots & X_i^{1\,n} \\ X_i^{12} & 0 & X_i^{23} & \cdots & X_i^{2\,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X_i^{1\,n} & X_i^{2\,n} & X_i^{3\,n} & \cdots & 0 \end{pmatrix}. \tag{7}$$

Each of the X_i^{kj} are $s_{\text{in}} \times s_{\text{in}}$ matrices, with zeros in the i^{th} row and column to satisfy non-completeness. These matrices are zero when $x^k x^j$ does not appear in the unshared function[2].

Similarly, let M be a block-matrix constructed from the matrix $S + S^T$ of the bilinear form B_f. Each block is of size $s_{\text{in}} \times s_{\text{in}}$ and its values equal the value

[2] We assume that there are no *superfluous terms* that appear in an even number of M_i and hence can be canceled out from the realization.

of the corresponding element of $S + S^T$. Hence, the correctness requirement can be stated as $\sum_{i=1}^{s_{\text{out}}} M_i = M$.

Corollary 2. *If rank*$(M_i) = n(s_{in} - 1)$ *for any* i*, then using linear CTs on the realization* $\boldsymbol{f}$ *does not make it uniform.*

Proof. Proof follows from Theorem 3 and Proposition 1. □

The proof of the following theorem clarifies the quadratic form notation for TI.

Theorem 4. *No nonlinear Boolean function with two inputs and one output can be uniformly shared using three shares.*

Proof. Consider the following direct sharing for the product x^1x^2 (AND gate):

$$f_1 = \underline{x_2^1x_3^2 + x_3^1x_2^2} + x_2^1x_2^2, \quad f_2 = \underline{x_1^1x_3^2 + x_3^1x_1^2} + x_3^1x_3^2, \quad f_3 = \underline{x_1^1x_2^2 + x_2^1x_1^2} + x_1^1x_1^2$$

The underlined terms cannot be moved due to the non-completeness property of TIs, hence their corresponding coefficients are fixed to be 1 in M_i whereas the other terms are flexible. It will be shown that any realization of x^1x^2 contains at least one bent Boolean function. Equivalently, by Corollary 2, there exists at least one M_i that is of full rank $n(s_{\text{in}} - 1)$.

We have the following matrices M_i associated with $f_i : \mathbb{F}^{n(s_{\text{in}}-1)} \to \mathbb{F}$ ($i \in \{1, 2, 3\}$):

$$\begin{pmatrix} 0 & 0 & A_i & 1 \\ 0 & 0 & 1 & B_i \\ A_i & 1 & 0 & 0 \\ 1 & B_i & 0 & 0 \end{pmatrix}$$

Note that the zero rows and columns of M_i—corresponding to the unused share due to non-completeness—have been removed in the above notation, since they have no influence, leaving $n(s_{\text{in}} - 1) \times n(s_{\text{in}} - 1)$ matrices.

Due to the orthogonality of the columns and rows, the above matrix is always of rank four, unless $A_i = B_i = 1$. It follows from Proposition 1 that every condition other than $A_i = B_i = 1$ implies that f_i is a bent function. However, the only remaining configuration is not possible since it corresponds to the sharing

$$\begin{aligned} f_1' &= x_2^1x_3^2 + x_3^1x_2^2 + x_2^1x_2^2 + x_3^1x_3^2 \\ f_2' &= x_1^1x_3^2 + x_3^1x_1^2 + x_3^1x_3^2 + x_1^1x_1^2 \\ f_3' &= x_1^1x_2^2 + x_2^1x_1^2 + x_1^1x_1^2 + x_2^1x_2^2, \end{aligned}$$

which is not correct. Note also that any correction must have degree less than three to preserve non-completeness. By Theorem 3, there exist no linear correction terms that makes the realization uniform. Hence, an AND gate has no uniform sharing with three input and output shares. Further, since linear terms have no influence on B_{f_i}, it follows that no nonlinear Boolean function with two inputs can be uniformly shared using three shares. □

The correctness of the above theorem has been shown in [19] by enumeration of all possible correction terms. Our proof indicates that the matrix representation of quadratic forms is a useful tool to study the uniformity of quadratic realizations.

5.3 Using Quadratic Forms to Find Uniform Realizations

Proposition 2. *A uniform realization of a quadratic function can be found only if there exist s_{out} matrices M_i formed as in Eq. (7), satisfying the following properties:*

1. $\sum_{i=1}^{s_{out}} M_i = M$.
2. $\forall i \in \{1, \ldots, s_{out}\} : \mathit{rank}(M_i) < n(s_{in} - 1)$.
3. *Linear correction terms can be found (e.g. by using Algorithm 1)*[3]

Moreover, by Theorem 2, the above requirements are also sufficient if $s_{out} = 3$.

Note that the proposition applies not only to quadratic forms but also quadratic functions in general, since linear terms do not influence the block matrix M of the bilinear form. In what follows, we discuss how the second requirement of Proposition 2 can be met, which is non-trivial. Moreover, we mainly focus on bent functions.

Recall that the matrix $S+S^T$ of the bilinear form of a bent Boolean function f is cogredient to its normal form N, given by Eq. (6). Note that if $S+S^T$ is cogredient to N, then there also is a transformation T such that $M = T\,N'\,T^T$. The matrix T is obtained by replacing ones in the original cogredience transformation matrix with identity matrices, and zeros with zero-blocks of the appropriate size. N' is the following block matrix:

$$N' = \begin{pmatrix} \begin{smallmatrix} 0 & J \\ J & 0 \end{smallmatrix} & & & \\ & \begin{smallmatrix} 0 & J \\ J & 0 \end{smallmatrix} & & \\ & & \ddots & \\ & & & 0 \end{pmatrix}. \tag{8}$$

The matrix J is an $s_{\text{in}} \times s_{\text{in}}$ matrix of ones. It now suffices to select matrices N'_i such that $N' = \sum_{i=1}^{s_{\text{out}}} N'_i$ with rank (N'_i) as low as possible for each i. In particular, since the transformation T preserves the rank and it does not act on individual shares, if one can choose N'_i such that rank $(N'_i) < n(s_{\text{out}}-1)$, then for $M_i = T\,N'_i\,T^T$, the first and second requirements from Proposition 2 are satisfied. One possible way of constructing the matrices N'_i, is by decomposing each of the block matrices J occurring in the normal form of Eq. (8). The decomposition of each J must be chosen such that it induces a linear dependence relation among the rows of at least one of the matrices N'_i, and hence reduces the rank of one of the matrices M_i. Eq. (9) provides three such decompositions of J for $s_{\text{in}} = s_{\text{out}} = 3$:

$$\begin{aligned} J &= \begin{pmatrix} 0&0&0 \\ 0&1&1 \\ 0&1&1 \end{pmatrix} + \begin{pmatrix} 1&0&1 \\ 0&0&0 \\ 1&0&0 \end{pmatrix} + \begin{pmatrix} 0&1&0 \\ 1&0&0 \\ 0&0&0 \end{pmatrix} \\ &= \begin{pmatrix} 0&0&0 \\ 0&0&1 \\ 0&1&0 \end{pmatrix} + \begin{pmatrix} 1&0&1 \\ 0&0&0 \\ 1&0&1 \end{pmatrix} + \begin{pmatrix} 0&1&0 \\ 1&1&0 \\ 0&0&0 \end{pmatrix} \\ &= \begin{pmatrix} 0&0&0 \\ 0&0&1 \\ 0&1&0 \end{pmatrix} + \begin{pmatrix} 0&0&1 \\ 0&0&0 \\ 1&0&1 \end{pmatrix} + \begin{pmatrix} 1&1&0 \\ 1&1&0 \\ 0&0&0 \end{pmatrix}. \end{aligned} \tag{9}$$

[3] The possibility of finding linear CTs increases as the rank of the matrix decreases since a low-rank matrix typically has more zeros in the Walsh spectrum.

Notice that the ith decomposition (from top to bottom) has identical rows in the ith term of the decomposition. The choice of the ith decomposition reduces the rank of M_i by two[4], since J is the only nonzero block in a row in N' and N' is symmetric. Hence, to ensure that each $\operatorname{rank}(M_i) < n(s_{\text{in}} - 1)$, each decomposition from Eq. (9) must be used at least once. This implies that this method can be used effectively only when $n \geq 6$. The method to generate M_i's using the decomposition of J for $s_{\text{out}} = 3$ is formalized in Algorithm 2. Note that lines 2–9 are merely intended to construct the block matrix T from the corresponding matrix L. The computational complexity of the algorithm is as low as $O(n^3)$ since finding the normal form can be done using a particular type of simultaneous row and column reduction, see for example Wan [23] for a description. Since the search space for $n \leq 5$ is feasible, we opted for a generic search algorithm to generate the matrices N_i' for these cases. Specifically, we focused on $n = 4$ since there exist no odd-sized bent functions and Theorem 4 completes the work for $n = 2$. The following theorem formalizes our findings.

Algorithm 2. Low-rank decomposition of the matrix M for $s_{\text{out}} = s_{\text{in}} = 3$.

```
Input: $S + S^T \in \mathbb{F}^{n \times n}$        ▷ The matrix representation of $B_f$.
Output: $M_1, M_2, M_3$        ▷ Matrices of the bilinear forms of the output shares.

 1: Find $T$ such that $S + S^T = TNT^T$ with $N$ the normal form as in Eqn. (6).
 2: Let $L \in \mathbb{F}^{ns_{\text{in}} \times ns_{\text{in}}}$.
 3: for $1 \leq i, j \leq n$ do
 4:     if $T[i, j] = 1$ then
 5:         $L[i : i + s_{\text{in}} - 1, j : j + s_{\text{in}} - 1] \leftarrow I_{s_{\text{in}}}$
 6:     else
 7:         $L[i : i + s_{\text{in}} - 1, j : j + s_{\text{in}} - 1] \leftarrow 0$
 8:     end if
 9: end for
10: Let $M_1, M_2, M_3 \in \mathbb{F}^{ns_{\text{in}} \times ns_{\text{in}}}$.
11: for $1 \leq i, j \leq n$ do
12:     if $N[i, j] = 1$ then
13:         ▷ Choose any decomposition from Eqn. (9).
14:         ▷ Use each decomposition at least once (only possible if $n \geq 6$).
15:         Let $J = J_1 + J_2 + J_3$.
16:         $M_l[i : i + s_{\text{in}} - 1, j : j + s_{\text{in}} - 1] \leftarrow J_l$ for $l = 1, 2, 3$.
17:     end if
18: end for
19: return $L(M_1 + M_1^T)L^T, L(M_2 + M_2^T)L^T, L(M_3 + M_3^T)L^T$
```

Theorem 5. *Let f be any quadratic Boolean function on $\mathbb{F}^n, n \geq 4$. Then there is a sharing $\boldsymbol{f}$ with $s_{in} = s_{out} = 3$ shares, such that none of the output shares of $\boldsymbol{f}$ are bent functions.*

[4] We consider the matrix derived from f_i of size $n(s_{\text{in}} - 1) \times n(s_{\text{in}} - 1)$ without the zero rows and columns.

Furthermore, we conjecture that if the first two requirements of Property 2 hold, then a quadratic Boolean function f can always be made uniform using three shares with linear correction terms. We conclude this section with an example.

Application to an $\mathbb{F}_4$ Multiplier. The AES S-box can be decomposed into several multiplications in $\mathbb{F}_4$, additions and rotations [8]. No three share uniform realization of $\mathbb{F}_4$ has been found so far, which can be explained with the fact that both coordinate functions of this multiplication which are given in Eq. (12) in Appendix E are bent. Since $n = 4$, we used a generic algorithm to find the matrices M_i leading to a realization with non-bent coordinate functions which is provided in Eq. (13) in Appendix E. We then performed a search on linear correction terms as described in Algorithm 1 to make the realization uniform. We found several uniform realizations for both coordinate functions. Details of this investigation leading to an implementation with 50% lower randomness requirements are given in Appendix E.

6 Conclusion

In this paper, we provided methods to find uniform realizations of nonlinear (vectorial) Boolean functions efficiently. We limit ourselves to first-order TIs because the uniformity property is insufficient to provide theoretical security against higher-order attacks. We started by discussing how the uniformity check of especially three output share realizations of Boolean functions can be performed efficiently. We then described how the Walsh-Hadamard transformation can be used to find all linear correction terms that lead to uniform realizations without the need for an exhaustive search. This method can be applied to any n-bit Boolean function with worst-case complexity $O(2^{2ns_{\text{in}}-3n})$ where s_{in} is the number of input shares of the threshold implementation. We proved that if the shared realization has a bent component function, this share can not be made uniform by using only linear correction terms. On the other hand, we showed that we can use the theory of quadratic forms to find uniform realizations for many quadratic functions. We demonstrated the applicability of the theory by providing partially uniform three-share realizations for a representative of the problematic quadratic 4-bit permutation class $\mathcal{Q}^4_{300}$ and a $\mathbb{F}_4$ multiplier that requires 50% less randomness compared to their naive implementations.

Acknowledgement. The authors are especially grateful to Vincent Rijmen for his contributions to this work. Additionally, we would like to thank the anonymous reviewers for providing constructive and valuable comments and Faruk Gologlu for fruitful discussions. This work was supported in part by NIST with the research grant 60NANB15D346, in part by the Research Council KU Leuven (OT/13/071 and C16/15/058) and in part by the Flemish Government through FWO Thresholds G0842.13. Begül Bilgin is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

A Algorithm to Find Partial Uniform Realizations

Algorithm 3. Find (partially) uniform realizations.

Input: $\boldsymbol{f} = (\boldsymbol{f}^1, \dots, \boldsymbol{f}^m)$ s.t. $\boldsymbol{f}^i$ is the realization of the coordinate function f^i of f; The initial set S^0 of all possible correction functions $\boldsymbol{c}^i$;

Output: The set Σ_l of all sets $S^{t_1,\cdots,t_l}$ with elements $(\boldsymbol{c}^{t_1}, \dots, \boldsymbol{c}^{t_l})$ s.t. $\boldsymbol{f}' = \boldsymbol{f}^{t_1,\cdots,t_i} + \boldsymbol{c}^{t_1,\cdots,t_l} = (\boldsymbol{f}^{t_1} + \boldsymbol{c}^{t_1}, \dots, \boldsymbol{f}^{t_l} + \boldsymbol{c}^{t_l})$ is a uniform realization.

```
1: function IsUniform($\boldsymbol{f}$)
2:     return true if $\boldsymbol{f}$ is uniform, false otherwise
3: end function

4: function GenerateCorrectionFunctions($\boldsymbol{f}^i$, $S^0$)
5:     $S^i \leftarrow \emptyset$
6:     for $\boldsymbol{c}^i \in S^0$ do
7:         if IsUniform($\boldsymbol{f}^i + \boldsymbol{c}^i$) then
8:             $S^i \leftarrow S^i \cup \{\boldsymbol{c}^i\}$
9:         end if
10:    end for
11:    return $S^i$
12: end function

13: function CombineCorrectionFunctions($\boldsymbol{f}$, $\Sigma_{l-1}$)
14:    $\Sigma_l \leftarrow \emptyset$
15:    ▷ Denote the set of $l$-combinations from $\{1, \dots, m\}$ by $\mathcal{S}$.
16:    for $\{t_1, \dots, t_l\} \in \mathcal{S}$ do
17:        $S^{t_1,\cdots,t_l} \leftarrow \emptyset$
18:        for $\boldsymbol{c}^{t_2,\cdots,t_l} \in S^{t_2,\cdots,t_l}, \boldsymbol{c}^{t_1} \in \{\boldsymbol{c}^{t_1} | \boldsymbol{c}^{t_1,t_3,\cdots,t_l} \in S^{t_1,t_3,\cdots,t_l}\}$ do
19:            $\boldsymbol{c}^{t_1,\cdots,t_l} \leftarrow \boldsymbol{c}^{t_2,\cdots,t_{l-1}} + \boldsymbol{c}^{t_1}$
20:            if $\forall 3 \leq i \leq l : \boldsymbol{c}^{t_1,\cdots,t_{i-1},t_{i+1},\cdots,t_l} \in S^{t_1,\cdots,t_{i-1},t_{i+1},\cdots,t_l}$
                  and IsUniform($\boldsymbol{f}^{t_1,\cdots,t_l} + \boldsymbol{c}^{t_1,\cdots,t_l}$) then
21:                $S^{t_1,\cdots,t_l} \leftarrow S^{t_1,\cdots,t_l} \cup \{\boldsymbol{c}^{t_1,\cdots,t_l}\}$
22:            end if
23:        end for
24:        $\Sigma_l \leftarrow \Sigma_l \cup \{S^{t_1,\cdots,t_l}\}$
25:    end for
26:    return $\Sigma_l$
27: end function

28: function FindPartiallyUniformRealization($S^0$, $\boldsymbol{g}$)
29:    for $1 \leq i \leq m$ do
30:        $S^i$ = GenerateCorrectionFunctions($\boldsymbol{f}^i$, $S^0$)
31:    end for
32:    $\Sigma_1 \leftarrow \{S^1, \dots, S^m\}$
33:    $l \leftarrow 2$
34:    while $l \leq m$ and $\exists S^{t_1,\cdots,t_{l-1}} \in \Sigma_{l-1} : S^{t_1,\cdots,t_{l-1}} \neq \emptyset$ do
35:        $\Sigma_l \leftarrow$ CombineCorrectionFunctions($\boldsymbol{f}$, $\Sigma_{l-1}$)
36:        $l \leftarrow l + 1$
37:    end while
38:    return $\Sigma_{l-1}$
39: end function
```

B Fast Uniformity Check for $s_{\text{out}} = 3$

For each unshared input x, the four linearly independent equations describing each row of the uniformity table $\mathcal{U}$ of the Boolean function f with $s_{\text{out}} = 3$ are as follows:

$$\begin{aligned}
&1 \cdot \mathcal{U}_{x,(0,0,0)} + 1 \cdot \mathcal{U}_{x,(0,1,1)} + 0 \cdot \mathcal{U}_{x,(1,0,1)} + 0 \cdot \mathcal{U}_{x,(1,1,0)} + \\
&1 \cdot \mathcal{U}_{x,(0,0,1)} + 1 \cdot \mathcal{U}_{x,(0,1,0)} + 0 \cdot \mathcal{U}_{x,(1,0,0)} + 0 \cdot \mathcal{U}_{x,(1,1,1)} = N_1 \\
&1 \cdot \mathcal{U}_{x,(0,0,0)} + 0 \cdot \mathcal{U}_{x,(0,1,1)} + 1 \cdot \mathcal{U}_{x,(1,0,1)} + 0 \cdot \mathcal{U}_{x,(1,1,0)} + \\
&1 \cdot \mathcal{U}_{x,(0,0,1)} + 0 \cdot \mathcal{U}_{x,(0,1,0)} + 1 \cdot \mathcal{U}_{x,(1,0,0)} + 0 \cdot \mathcal{U}_{x,(1,1,1)} = N_2 \\
&1 \cdot \mathcal{U}_{x,(0,0,0)} + 0 \cdot \mathcal{U}_{x,(0,1,1)} + 0 \cdot \mathcal{U}_{x,(1,0,1)} + 1 \cdot \mathcal{U}_{x,(1,1,0)} + \\
&0 \cdot \mathcal{U}_{x,(0,0,1)} + 1 \cdot \mathcal{U}_{x,(0,1,0)} + 1 \cdot \mathcal{U}_{x,(1,0,0)} + 0 \cdot \mathcal{U}_{x,(1,1,1)} = N_3 \\
&1 \cdot \mathcal{U}_{x,(0,0,0)} + 1 \cdot \mathcal{U}_{x,(0,1,1)} + 1 \cdot \mathcal{U}_{x,(1,0,1)} + 1 \cdot \mathcal{U}_{x,(1,1,0)} + \\
&1 \cdot \mathcal{U}_{x,(0,0,1)} + 1 \cdot \mathcal{U}_{x,(0,1,0)} + 1 \cdot \mathcal{U}_{x,(1,0,0)} + 1 \cdot \mathcal{U}_{x,(1,1,1)} = 2^{n(s_{\text{in}}-1)}
\end{aligned} \tag{10}$$

Depending on whether the output ($y = \sum_i y_i$) is 0 or 1, either the first or the second line of each equation in Eq. (10) will have non-zero terms $\mathcal{U}_{x,\boldsymbol{y}}$.

C Finding Uniform Realizations Using Fast WHT

Partial uniform realization for $\mathcal{Q}^4_{300}$. Here, we describe definitive results regarding the use of linear correction terms on the representative permutation of $\mathcal{Q}^4_{300}$ with truth table $[0, 1, 2, 3, 4, 5, 8, 9, 6, 7, 12, 13, 14, 15, 10, 11]$. Namely, it is possible to find multiple uniform realizations for each coordinate function of the permutation using the contribution from this section. However, this does not imply that the realization for the permutation is also uniform. Our algorithm revealed that we can make two out of four coordinate functions jointly uniform. We provide the algebraic description of one such realization where the unshared permutation is described as $(y^1, y^2, y^3, y^4) = f(x^1, x^2, x^3, x^4)$ in Eq. (11).

$$\begin{aligned}
y^1_1 &= x^2_2x^3_2 + x^2_2x^3_3 + x^2_2x^4_2 + x^2_2x^4_3 + x^2_3x^3_2 + x^2_3x^3_3 + x^2_3x^4_2 + x^2_3x^4_3 + x^3_2x^4_2 + x^3_2x^4_3 + x^3_3x^4_2 + x^3_3x^4_3 + x^2_1 \\
y^1_2 &= x^2_1x^3_1 + x^2_1x^3_3 + x^2_1x^4_1 + x^2_1x^4_3 + x^2_3x^3_1 + x^2_3x^4_1 + x^3_1x^4_1 + x^3_1x^4_3 + x^3_3x^4_1 \\
y^1_3 &= x^2_1x^3_2 + x^2_1x^4_2 + x^2_2x^3_1 + x^2_2x^4_1 + x^3_1x^4_2 + x^3_2x^4_1 + x^2_1 \\
y^2_1 &= x^2_2x^3_2 + x^2_2x^3_3 + x^2_3x^3_2 + x^2_3x^3_3 + x^3_2x^4_2 + x^3_2x^4_3 + x^3_2 + x^3_3x^4_2 + x^3_3x^4_3 + x^3_3 + x^4_2 + x^4_3 \\
y^2_2 &= x^2_1x^3_1 + x^2_1x^3_3 + x^2_3x^3_1 + x^3_1x^4_1 + x^3_1x^4_3 + x^3_1 + x^3_3x^4_1 + x^4_1 \\
y^2_3 &= x^2_1x^3_2 + x^2_2x^3_1 + x^3_1x^4_2 + x^3_2x^4_1 \\
y^3_1 &= x^2_2x^3_2 + x^2_2x^3_3 + x^2_2 + x^2_3x^3_2 + x^2_3x^3_3 + x^2_3 + x^4_2 + x^4_3 \\
y^3_2 &= x^2_1x^3_1 + x^2_1x^3_3 + x^2_1 + x^2_3x^3_1 + x^4_1 \\
y^3_3 &= x^2_1x^3_2 + x^2_2x^3_1 \\
y^4_1 &= x^1_3, \quad y^4_2 = x^1_1, \quad y^4_3 = x^1_2.
\end{aligned} \tag{11}$$

This particular realization makes the joint realization of the pair (y_1, y_4) uniform. The component functions corresponding to the coordinate functions (y_2, y_3)

should be remasked for uniformity of the permutation's realization. Hence, the required randomness is reduced by 50% compared to remasking every bit. No further improvements are impossible using only linear correction terms.

D Constructions to Avoid Bent Component Functions

Two generic constructions for avoiding bent functions are listed below:

1. Add a term of degree higher than $n(s_{\text{in}}-1)/2$ which is the maximum degree of a bent function [9]. If $n(s_{\text{in}}-1)/2 < n(s_{\text{in}}-2)$, we must add an additional share due to non-completeness. Hence, this is mainly useful for $s_{\text{in}} \geq 4$.
2. It can be shown that the derivative $D_{\boldsymbol{\omega}} f(\boldsymbol{x}) = f(\boldsymbol{x}) + f(\boldsymbol{x}+\boldsymbol{\omega})$ is a balanced Boolean function if f is bent [9]. Hence, adding $f_i(\boldsymbol{x}+\boldsymbol{\omega})$ to both share i and a new share makes share i balanced if f is bent. The new share can be avoided if some component f is independent of two input shares.

E Using Quadratic Correction Terms For Uniformity

Partial uniform realization for $\mathbb{F}_4$ *multiplier.* It has been shown in [8] that the AES S-box can be decomposed into several multiplications in $\mathbb{F}_4$, additions and rotations. This decomposition has been used for TIs of AES in [12,18]. Since, no uniform realization of $\mathbb{F}_4$ has been found so far, these TIs relied heavily on adding fresh randomness. This can be explained with the fact that both coordinate functions of this multiplication which are given in Eq. (12) are bent.

$$f^1(x) = x^1x^4 + x^2x^3 + x^2x^4 \qquad f^2(x) = x^1x^3 + x^1x^4 + x^2x^3. \tag{12}$$

Since $n = 4$, we used a generic algorithm to find the matrices M_i leading to a realization with non-bent coordinate functions which is provided in Eq. (13). Note that this realization is not uniform. Hence, we performed a search on linear correction terms as described in Algorithm 1. This gave several uniform realizations for both coordinate functions such as Eq. (15) corresponding to y_1.

$$\begin{aligned}
y_1^1 &= x_2^1x_2^2 + x_2^1x_2^3 + x_2^1x_3^4 + x_3^1x_2^4 + x_3^1x_3^4 + x_2^2x_3^3 + x_2^2x_3^4 + x_3^2x_2^3 + x_3^2x_2^4 \\
y_2^1 &= x_1^1x_3^4 + x_3^1x_1^4 + x_1^2x_1^3 + x_1^2x_3^3 + x_1^2x_3^4 + x_3^2x_1^3 + x_3^2x_3^3 + x_3^2x_1^4 + x_3^2x_3^4 \\
y_3^1 &= x_1^1x_1^4 + x_1^1x_2^4 + x_2^1x_2^2 + x_2^1x_2^3 + x_2^1x_1^4 + x_2^1x_2^4 + x_1^2x_2^3 + x_1^2x_1^4 + x_1^2x_2^4 \\
&\quad + x_2^2x_1^3 + x_2^2x_2^3 + x_2^2x_1^4 + x_2^2x_2^4
\end{aligned} \tag{13}$$

$$\begin{aligned}
y_1^2 &= x_2^1x_3^3 + x_2^1x_3^4 + x_3^1x_2^3 + x_3^1x_3^3 + x_3^1x_2^4 + x_2^2x_3^3 + x_2^2x_2^4 + x_3^2x_2^3 \\
y_2^2 &= x_1^1x_3^3 + x_1^1x_1^4 + x_1^1x_3^4 + x_3^1x_1^3 + x_3^1x_1^4 + x_3^1x_3^4 + x_1^2x_3^3 + x_3^2x_1^3 + x_3^2x_3^3 \\
y_3^2 &= x_1^1x_1^3 + x_1^1x_2^3 + x_1^1x_2^4 + x_2^1x_1^3 + x_2^1x_2^3 + x_2^1x_1^4 + x_2^1x_2^4 + x_1^2x_1^3 + x_1^2x_2^3 \\
&\quad + x_2^2x_1^3 + x_2^2x_2^3 + x_2^2x_2^4
\end{aligned} \tag{14}$$

Since no combination of possible uniform realizations for coordinate functions yielded a uniform result, we conclude that the sharing of either one of the coordinate functions should still be remasked. This requires two bits of randomness.

$$f_1^1 = y_1^1 + x_2^1 \qquad f_2^1 = y_2^1 + x_1^2 + x_1^3 \qquad f_3^1 = y_3^1 + x_1^2 + x_1^3 \tag{15}$$

References

1. Balasch, J., Gierlichs, B., Verdult, R., Batina, L., Verbauwhede, I.: Power analysis of atmel cryptomemory – recovering keys from secure EEPROMs. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 19–34. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27954-6_2
2. Bilgin, B.: Threshold Implementations as Countermeasure Against Higher-Order Differential Power Analysis. PhD thesis, KU Leuven and University of Twente (2015)
3. Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., Van Assche, G.: Efficient and first-order dpa resistant implementations of Keccak. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 187–199. Springer, Cham (2014). doi:10.1007/978-3-319-08302-5_13
4. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45608-8_18
5. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-offs for threshold implementations illustrated on AES. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **34**, 1–13 (2015)
6. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Stütz, G.: Threshold implementations of all 3 × 3 and 4 × 4 S-boxes. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 76–91. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33027-8_5
7. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N., Vitkup, V.: Threshold Implementations of Small S-boxes. Crypt. Commun. **7**(1), 3–33 (2015)
8. Canright, D.: A very compact S-box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005). doi:10.1007/11545262_32
9. Carlet, C.: Boolean Functions for Cryptography and Error Correcting Codes (2006)
10. Carlet, C., Prouff, E., Rivain, M., Roche, T.: Algebraic decomposition for probing security. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 742–763. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47989-6_36
11. Carlet, C., Yucas, J.L.: Piecewise constructions of bent and almost optimal Boolean functions. Des. Codes Crypt. **37**, 449–464 (2005)
12. De Cnudde, T., Bilgin, B., Reparaz, O., Nikov, V., Nikova, S.: Higher-order threshold implementation of the AES S-box. In: Homma, N., Medwed, M. (eds.) CARDIS 2015. LNCS, vol. 9514, pp. 259–272. Springer, Cham (2016). doi:10.1007/978-3-319-31271-2_16
13. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). doi:10.1007/978-3-642-55220-5_24
14. Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., Shalmani, M.T.M.: Physical cryptanalysis of KeeLoq code hopping applications. Cryptology ePrint Archive, Report 2008/058 (2008). http://eprint.iacr.org/
15. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_25
16. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). doi:10.1007/3-540-68697-5_9

17. Kutzner, S., Nguyen, P.H., Poschmann, A.: Enabling 3-share threshold implementations for any 4-bit S-box. Cryptology ePrint Archive, Report 2012/510 (2012). http://eprint.iacr.org/
18. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: a very compact and a threshold implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20465-4_6
19. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementations of nonlinear functions in the presence of glitches. J. Cryptology **24**, 292–321 (2010)
20. Poschmann, A., Moradi, A., Khoo, K., Lim, C.-W., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2,300 GE. J. Cryptology **24**(2), 322–345 (2011)
21. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47989-6_37
22. Rothaus, O.: On bent functions. J. Comb. Theory Ser. A **20**(3), 300–305 (1976)
23. Wan, Z.-X.: Lectures on Finite Fields and Galois Rings. World Scientific, Singapore (2003)