Classification of Balanced Quadratic Functions

Lauren De Meyer and Begül Bilgin

KU Leuven, imec - COSIC, Belgium
firstname.lastname@esat.kuleuven.be

Abstract. S-boxes, typically the only nonlinear part of a block cipher, are the heart of symmetric cryptographic primitives. They significantly impact the cryptographic strength and the implementation characteristics of an algorithm. Due to their simplicity, quadratic vectorial Boolean functions are preferred when efficient implementations for a variety of applications are of concern. Many characteristics of a function stay invariant under affine equivalence. So far, all 6-bit Boolean functions, 3- and 4-bit permutations have been classified up to affine equivalence. At FSE 2017, Bozoliv *et al.* presented the first classification of 5-bit quadratic permutations. In this work, we propose an adaptation of their work resulting in a highly efficient algorithm to classify $n \times m$ functions for $n \geq m$. Our algorithm enables for the first time a complete classification of 6-bit quadratic permutations as well as all balanced quadratic functions for $n \leq 6$. These functions can be valuable for new cryptographic algorithm designs with efficient multi-party computation or side-channel analysis resistance as goal. In addition, we provide a second tool for finding decompositions of length two. We demonstrate its use by decomposing existing higher degree S-boxes and constructing new S-boxes with good cryptographic and implementation properties.

Keywords: Affine Equivalence · S-box · Boolean functions · Classification · Decomposition

1 Introduction

For a variety of applications, such as multi-party computation, homomorphic encryption and zero-knowledge proofs, linear operations are considered to have minimal cost. Nonlinear operations on the other hand cause a rapid growth of implementation requirements. Therefore, it becomes important to create cryptographically strong algorithms with minimal nonlinear components. A recent study in this direction called MiMC [AGR+16], which is based on some relatively old observations [NK95], uses the simple quadratic function x^3 in different fields as the only nonlinear block of the algorithm. Another work that minimizes the number of multiplications is the LowMC design [ARS+15], where a quadratic 3-bit permutation is used as the only nonlinear component of a Substitution-Permutation-Network (SPN).

We also see the importance of minimizing the nonlinear components in the field of secure implementations against side-channel analysis. Efforts to decompose the S-boxes of existing algorithms, such as the DES and AES S-boxes, into a minimum number of lower degree nonlinear components (AND-gates, field multiplications or other quadratic or cubic functions), have produced more than a handful of papers. Some of these decomposition tools are generic and work heuristically [CGP+12, RV13, CRV14, CPRR15, GR16, PV16] whereas others focus on enumerating decompositions of all permutations for a certain size [BNN+12, KNP13]. In general, they all make it clear that there is a significant advantage in considering side-channel security during the design process and hence using low degree nonlinear components. As a reaction to this line of research, a variety of novel symmetric-key designs use simply a quadratic permutation [ABB+14, BDP+14, BDP+15, DEMS15]. Examples include Keccak [BDPA13], one instance of which is the new hash function standard, and several candidates of the CAESAR competition. Generating strong, higher degree S-boxes using quadratic functions has also been shown useful in [BGG+16]. These works demonstrate the relevance of our research, which focuses on enumerating quadratic $n \times m$ functions for $n < 7$.

A valuable tool for the analysis of vectorial Boolean functions, which are typically used as S-boxes, is the concept of affine equivalence (AE). AE allows the entire space of $n \times m$ functions to be classified into groups with the same cryptographic properties. These properties include the algebraic degree, the differential uniformity and the linearity of both the function and its possible inverse in addition to multiplicative complexity. Moreover, the randomness cost of a first-order masked implementation is also invariant within a class if countermeasures such as threshold implementations are used [Bil15]. With similar concerns in mind, our research relies on this affine equivalence classification.

1.1 Classification of (Vectorial) Boolean Functions

The classification of Boolean functions dates back to the fifties [Gol59]. The equivalence classes for functions of up to five inputs were identified by 1972 [BW72] and Maiorana [Mai91] was the first to classify all 6-bit Boolean functions in 1991. Fuller [Ful03] confirmed in 2003 that this classification was complete.

For vectorial Boolean functions, only n-bit permutations for $n \leq 4$ have been completely classified so far [Can07, Saa11, BNN+15]. Most of these classifications use the affine equivalence (AE) tool introduced by Biryukov *et al.* in [BCBP03]. This algorithm computes a representative of the affine equivalence class for any n-bit permutation. In [Can07], De Cannière classifies all 4-bit permutations by transversing a graph of permutations connected by single transpositions and reducing them to their affine equivalence class representative. As this method is unpractical for larger dimensions ($n > 4$), no classification of the complete space of 5-bit bijective permutations exists. A classification of the APN classes (which have the best cryptographic properties) does exist by Brinkmann *et al.* [BL08]. The authors build a tree of LUTs of 5-bit permutations, in which each level of the tree specifies one more output of the function. The tree is pruned using on the one hand an APN filter function and on the other hand an affine equivalence filter, which is also based on the algorithm of [BCBP03]. The quadratic 5-bit permutations have been classified by Bozilov *et al.* [BBS17]. Their approach consists of two stages: First, they generate an exhaustive list of 5-bit permutations from quadratic ANF's. Then, they use the affine equivalence algorithm of Biryukov *et al.* [BCBP03] to find the affine representatives of all the candidates in this list. Eliminating the doubles results in 75 quadratic classes. This approach uses the AE algorithm $\approx 2^{23}$ times, resulting in a runtime of a couple of hours, using 16 threads. Again, extending this approach to higher dimensions is not feasible.

Vectorial Boolean functions from n to $m < n$ bits have been used as S-boxes as well (*e.g.* the 6×4 DES S-boxes), yet their classification has been largely ignored. They are also used in the construction of larger 8-bit S-boxes by Boss *et al.* [BGG+16].

1.2 Decompositions of Higher-Degree Functions

The authors of [BNN+12, KNP13] decompose all 4-bit permutations in order to provide efficient implementations against side-channel analysis. The decompositions in both works benefit from the affine equivalence classification of permutations. The main difference between them is that [BNN+12] only focuses on decompositions using quadratic and cubic components. It is shown that not all cubic 4-bit permutations can be composed from quadratics. This work has been extended in [KNP13], in which decomposition of all permutations is enabled by including additions and compositions with non-bijective quadratic functions. The decompositions provided in both these papers have been proven to have the smallest length with the given structure. A possible decomposition for all 6×4 DES S-boxes jointly using 4-bit permutations is also provided as an output of the aforementioned research [BKNN15].

A complementary work which decomposes a function into other quadratic and cubic functions is [CPRR15]. This work starts from a randomly chosen low-degree function. They iteratively enlarge their set of functions using addition and composition. Finally, the generated set of functions is used to get a decomposition for a target function. This approach is not unlike the logic minimization technique of [BP10]. The tool is heuristic and the decompositions provided do not necessarily have the smallest length. The theoretical lower bounds are not necessarily achieved for a randomly selected function decomposition for bigger sizes. However, it performs well for small functions.

1.3 Our Contribution

In this work, we explore the extension of Biryukov's AE algorithm to non-bijective $n \times m$ functions with $m < n$ and analyse its performance. We propose an algorithm that does not only classify all n-bit permutations, but also all balanced $n \times m$-bit functions for $m \leq n$. Our complexity is significantly lower than that of previous algorithms known to date. This allows us to generate all quadratic vectorial Boolean functions with five inputs in merely six minutes, which makes the search for even 6-bit quadratic functions feasible. We also provide the cryptographic properties of these functions and their inverses if possible.

Our work focuses on quadratic functions, since they tend to have low area requirements in hardware, especially for masked implementations. We also introduce a tool for finding length-two quadratic decompositions of higher degree permutations and we use it to decompose the 5-bit AB and APN permutations. Furthermore, we find a set of high quality 5-bit permutations of degree 4 with small decomposition length that can be efficiently implemented.

Our list of quadratic 6-bit permutations is an important step towards decomposing the only known 6-bit APN permutation class as an alternative to [PUB16].

2 Preliminaries

We consider an $n \times m$ (vectorial) Boolean function $F(x) = y$ from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. The bits of x and the coordinate functions of F are denoted by small letter subscripts, i.e. $x = (x_0, \ldots, x_{n-1})$ where $x_i \in \mathbb{F}_2$ and $F(x) = (f_0(x), \ldots, f_{m-1}(x))$ where $f_i(x)$ is from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. We use '$\circ$' to denote the composition of two or more functions, *e.g.* $F_1 \circ F_2(x) = F_1(F_2(x))$ where $F_1 : \mathbb{F}_2^m \to \mathbb{F}_2^l$ and $F_2 : \mathbb{F}_2^n \to \mathbb{F}_2^m$. We use $|.|$ and $\cdot$ for absolute value and inner product respectively.

2.1 (Vectorial) Boolean Function Properties

In this paper, we focus on *balanced* vectorial Boolean functions $F(x) = y$, *i.e.* each output $y \in \mathbb{F}_2^m$ is equiprobable for all inputs $x \in \mathbb{F}_2^n$. When $n = m$, F is thus bijective and typically called an n-bit *permutation*.

A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be uniquely represented by its *algebraic normal form (ANF)*

$$f(x) = \bigoplus_{j \in \mathbb{F}_2^n} \alpha_j x^j \text{ where } x^j = \prod_{i=0}^{n-1} x_i^{j_i}.$$

The *algebraic degree* of f is

$$\mathrm{Degr}(f) = \max_{j \in \mathbb{F}_2^n, \alpha_j \neq 0} \mathrm{HW}(j) \text{ with } \mathrm{HW}(j) = \sum_{i=0}^{n-1} j_i.$$

The algebraic degree of a function $F = (f_0, f_1, \ldots, f_{m-1})$ is simply the largest degree of its coordinate functions, i.e. $\text{Degr}(F) = \max_{0 \le i < m} \text{Degr}(f_i)$.

Definition 1 (Component [NK95])**.** The components of a vectorial Boolean function F are the nonzero linear combinations $\beta \cdot F$ of the coordinate functions of F, with $\beta \in \mathbb{F}_2^m \setminus \{0\}$.

Definition 2 (DDT [BS90, Nyb93])**.** We define the Difference Distribution Table (DDT) δ_F of F with its entries

$$\delta_F(\alpha, \beta) = \#\{x \in \mathbb{F}_2^n : F(x \oplus \alpha) = F(x) \oplus \beta\}$$

for $\alpha \in \mathbb{F}_2^n$ and $\beta \in \mathbb{F}_2^m$. The differential uniformity $\text{Diff}(F)$ is the largest value in the DDT for $\alpha \neq 0, \beta$:

$$\text{Diff}(F) = \max_{\alpha \neq 0, \beta} \delta_F(\alpha, \beta)$$

An n-bit permutation F is said to be *almost perfect nonlinear (APN)* if $\forall \alpha \neq 0, \beta \in \mathbb{F}_2^n$, the DDT element $\delta_F(\alpha, \beta)$ is equal to either 0 or 2. The *DDT frequency distribution* or *differential spectrum* Δ_F of F is a histogram of the elements occuring in the DDT:

$$\Delta_F(\delta) = \#\{(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^m : \delta_F(\alpha, \beta) = \delta\}$$

Definition 3 (LAT and Walsh Spectrum [O'C94, CV94])**.** We define the Linear Approximation Table (LAT) λ_F of F with its entries

$$\lambda_F(\alpha, \beta) = \#\{x \in \mathbb{F}_2^n : \alpha \cdot x = \beta \cdot F(x)\} - 2^{n-1}$$

for $\alpha \in \mathbb{F}_2^n$ and $\beta \in \mathbb{F}_2^m$. The Walsh spectrum of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as

$$\hat{f}(\omega) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} \cdot (-1)^{\omega \cdot x}.$$

A function's LAT is directly related to its two-dimensional Walsh transform $\hat{F}(\alpha, \beta) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\alpha \cdot x} \cdot (-1)^{\beta \cdot F(x)}$ as follows:

$$\lambda_F(\alpha, \beta) = \frac{\hat{F}(\alpha, \beta)}{2}$$

Any column in a function's LAT ($\lambda_F(\alpha, \bar{\beta})$ for $\bar{\beta}$ fixed) is thus the scaled Walsh spectrum of a component of F. The linearity $\text{Lin}(F)$ is the largest value in the LAT for $\beta \neq 0, \alpha$:

$$\text{Lin}(F) = \max_{\beta \neq 0, \alpha} |\lambda_F(\alpha, \beta)|$$

An n-bit permutation F is said to be *almost bent (AB)* if $\forall \beta \neq 0, \alpha \in \mathbb{F}_2^n$, the LAT element $\lambda_F(\alpha, \beta)$ is equal to either 0 or $\pm 2^{(n-1)/2}$. It is known that all AB permutations are also APN. The *LAT frequency distribution* Λ_F of F is a histogram of the absolute values occuring in the LAT:

$$\Lambda_F(\lambda) = \#\{(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^m : |\lambda_F(\alpha, \beta)| = \lambda\}$$

Remark 1. In some works, the linearity is expressed in terms of the Walsh spectrum instead of the LAT table as $\mathcal{L}(F) = \max_{\beta \neq 0, \alpha} |\hat{F}(\alpha, \beta)|$. The two definitions differ by a factor of two, *i.e.* $\mathcal{L}(F) = 2 \cdot \text{Lin}(F)$.

2.2 Affine Equivalence

Functions with algebraic degree 1 are called *affine*. We use them to define affine equivalence relations that classify the space of all $n \times m$ functions.

Definition 4 (Extended Affine Equivalence [CCZ98])**.** Two $n \times m$ functions $F_1(x)$ and $F_2(x)$ are extended affine equivalent if and only if there exists a pair of n-bit and m-bit invertible affine permutations A and B and an $n \times m$ linear mapping L such that $F_1 = B \circ F_2 \circ A \oplus L$.

The algebraic degree and DDT and LAT frequency distributions are invariant over extended affine equivalence.

Definition 5 (Affine Equivalence [CCZ98])**.** Two $n \times m$ functions $F_1(x)$ and $F_2(x)$ are affine equivalent ($F_1 \sim F_2$) if and only if there exists a pair of n-bit and m-bit invertible affine permutations A and B such that $F_1 = B \circ F_2 \circ A$.

Clearly, affine equivalent functions are always extended affine equivalent but not vice versa. Note that the affine equivalence relation also covers linear equivalence, where A and B are linear permutations (i.e. $A(0) = B(0) = 0$). Moreover, also affine equivalence preserves algebraic degree and DDT and LAT frequency distributions. In the case of Boolean functions ($m = 1$), affine equivalence and extended affine equivalence are the same.

It is common practice to take the lexicographically smallest function in an affine equivalence class as the representative, which we denote by R. An efficient algorithm for finding the affine equivalent (AE) representative of any n-bit permutation S was proposed by Biryukov *et al.* in [BCBP03]. In short, it computes the linear representatives of $S(x \oplus a) \oplus b$ for all $a, b \in \mathbb{F}_2^n$ and chooses the lexicographically smallest among them as affine equivalent representative. Since, we rely on this algorithm and modify it according to our needs, we provide a detailed description below.

2.3 Finding the Representative of a Permutation

This recursive algorithm described in [BCBP03] finds for a given permutation S the smallest affine equivalent $R = B^{-1} \circ S \circ A$ by *guessing* some of the output values of the affine permutations A and B and determining the others using the linearity property. Throughout the algorithm, the numbers n_A and n_B record logarithmically for how many input values the outputs of A and B have been defined. For example, $A(x)$ is defined for all $x < 2^{n_A - 1}$. It is possible to fix $A(0)$ (resp. $B(0)$) at the beginning of the algorithm which implies n_A (resp. n_B) being initialized to 1. The number of defined values for $R(x)$ is N_R, i.e. $R(x)$ will be defined for all $x < N_R$.

The computation starts with $x = y = 0$ from the ForwardSweep described in Algorithm 1, which serves as the outer loop of the algorithm. The ForwardSweep enumerates all inputs x for which affine transformation $A(x)$ has already been defined and determines the representative output $y = R(x)$. Either there already exists an output y such that $S \circ A(x) = B(y)$ or we choose y as the next smallest unused power of 2. When the ForwardSweep is complete, we continue with the BackwardSweep in Algorithm 2. Note that when $n_A = 0$ (the very first iteration), there are no inputs to enumerate yet and the computation actually starts with a BackwardSweep.

At the start of Algorithm 2, x is typically a power of 2 which means $A(x)$ cannot be determined from linear combinations and can be chosen freely. If the BackwardSweep is successful (*i.e.* it finds a suitable $A(x)$ such that $S \circ A(x) = B \circ R(x)$), we recurse on the ForwardSweep. If the BackwardSweep fails, we need to guess $A(x)$. This is for example the case in the very first iteration when $n_B = 0$.

Algorithm 1: FORWARDSWEEP(x, y, n_A, n_B)

```
while $x < 2^{n_A - 1}$ do
    Determine $y'$ s.t. $B(y') = S \circ A(x)$;
    if $y'$ not yet defined then
        Pick $y' = 2^{n_B - 1}$;
        Set $B(y') = S \circ A(x)$;
        $n_B = n_B + 1$;
    end
    if SETR($x, y'$) then
        $x = x + 1$;
    else
        Dead end: Stop forward sweep;
    end
end
if $x < 2^n$ then
    BACKWARDSWEEP($x, y, n_A, n_B$);
end
```

Algorithm 2: BACKWARDSWEEP(x, y, n_A, n_B) for invertible S

```
while $y < 2^{n_B - 1}$ do
    Determine $x'$ s.t. $A(x') = S^{-1} \circ B(y)$;
    if $x' < x$ then
        $y = y + 1$;
    else
        if SETR($x, y$) then
            Set $A(x) = S^{-1} \circ B(y)$;
            FORWARDSWEEP($x, y + 1, n_A + 1, n_B$);
            Return;
        end
    end
end
GUESS($x, y, n_A, n_B$);
```

Algorithm 3: GUESS(x, y, n_A, n_B)

```
SETR($x, y$);
for all guesses $g$ for $A(x)$ do
    Set $A(x) = g$;
    Set $B(y) = S \circ A(x)$;
    FORWARDSWEEP($x, y, n_A +$
    $1, n_B + 1$);
end
```

Algorithm 4: SETR(x, y)

```
if $R(x)$ already defined (i.e. $x < N_R$)
then
    if $y > R(x)$ then
        Return False;
    end
    if $y = R(x)$ then
        Return True;
    end
end
Set $R(x) = y$ and $N_R = x + 1$;
Return True;
```

The GUESS function is described by Algorithm 3. It fixes $R(x)$ using Algorithm 4 to the smallest unused y and then loops over all available assignments of $A(x)$. For each guess, we try recursion on the ForwardSweep. We need to try all because any guess can result in a lexicographically smaller representative R.

Algorithm 4 builds the representative R and only changes previously determined outputs if they are smaller than the current one.

a	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(a)$	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0

	$x \to A(x)$	$\overset{S}{\to}$	$B(y) \leftarrow y$			$x \to A(x)$	$\overset{S}{\to}$	$B(y) \leftarrow y$
Guess	$0 \to 0$	$\to$	$1 \leftarrow 0$					
Guess	$1 \to 1$	$\to$	$B \leftarrow 1$	or	Guess	$1 \to 5$	$\to$	$6 \leftarrow 1$
Guess	$2 \to 2$	$\to$	$9 \leftarrow 2$		Guess	$2 \to A$	$\to$	$7 \leftarrow 2$
Fwd	$3 \to 3$	$\to$	$C \leftarrow 4$		Fwd	$3 \to F$	$\to$	$0 \leftarrow 3$
Bwd	$4 \to 7$	$\leftarrow$	$3 \leftarrow 3$		Guess	$4 \to 4$	$\to$	$D \leftarrow 4$
Fwd	$5 \to 6$	$\to$	$F \leftarrow 8$		Fwd	$5 \to 1$	$\to$	$B \leftarrow 6$
Fwd	$6 \to 5$	$\to$	$6 \leftarrow 5$		Fwd	$6 \to E$	$\to$	$5 \leftarrow 8$

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$R(x)$	0	1	2	3	4	6	8									

Figure 1: Example of finding the linear representative for a 4-bit bijective S

This whole procedure of finding the representative of an n-bit permutation is exemplified in Figure 1 for clarification. Note that even though the S-box we use and the one in [BCBP03] are the same, the representative we obtain is different since we focus on the lexicographically smallest one by assigning, for example, $R(0) = 0$. Moreover, for the same reason, the representative on the right side of Figure 1 is favored over the left side.

3 Finding the Representative of a Non-invertible Function

It has been suggested in [BCBP03] that the algorithm in Section 2.3 can be extended to find representatives for non-bijective functions $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$, but that this is only efficient when $n - m$ is small. When S is not invertible (but still balanced), instead of one single solution to the equation $S(x) = y$, there are 2^{n-m} possible x candidates for each y. The additional complexity of enumerating these candidates during BackwardSweep grows larger as m decreases. Therefore, in [BCBP03] the total complexity of finding the affine representative for an $n \times m$ function where $n > m$ is estimated as:

$$n^3 \cdot 2^n \cdot (2^{n-m}!)^{\frac{n}{2^{n-m}}}$$

Figure 2 depicts how the predicted complexity (for fixed $n = 5$) increases monotonously as m decreases.

In what follows, we describe an extension of the algorithm in Section 2.3 which has a non-monotonous complexity behavior as m decreases as can be observed in Figure 3. Note that Figure 3 depicts *experimental* runtimes whereas Figure 2 depicts an *asymptotic* complexity estimation. Their scales are thus very different and should not be compared in magnitude. Instead, we are considering only the difference in *trends*. For $m = n$, the algorithm is identical to [BCBP03]. The runtimes are calculated using a random selection of 500 $5 \times m$ functions for each m. Note that since no pseudo-code is provided in [BCBP03]

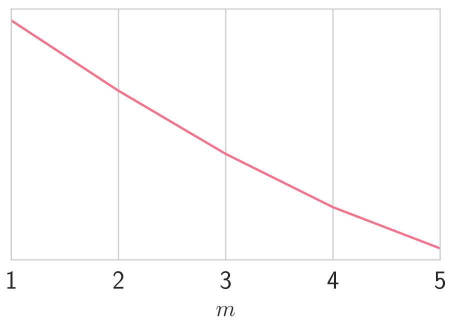

Figure 2: Asymptotic Complexity from [BCBP03] for $5 \times m$ functions.

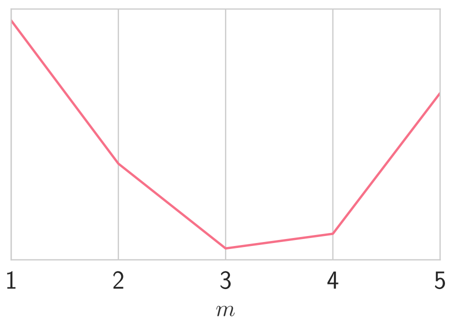

Figure 3: Our experimental runtimes for random $5 \times m$ functions.

and the description is very brief, we can not conclude whether this is due to a complexity estimation error or having a slightly different algorithm. Moreover, the real runtimes might approximate the asymptotic complexity better for $n \to \infty$.

One of the changes caused by the non-invertability of S is that we can no longer compute the inverse S^{-1} and thus we cannot obtain x' in Algorithm 2. We propose Algorithm 5 as an alternative in which we loop over all possible x' for which $S \circ A(x') = B(y)$.

Algorithm 5: BackwardSweep(x, y, n_A, n_B) for non-invertible S

```
while $y < 2^{n_B - 1}$ do
    for all $x'$ s.t. $S \circ A(x') = B(y)$ do
        if $x' < x$ then
            Try next $x'$;
        else
            if Set $R(x, y)$ then
                Set $A(x) = S^{-1} \circ B(y)$;
                ForwardSweep$(x, y, n_A + 1, n_B)$ ;
            end
        end
    end
    if no $x'$ found then
        $y = y + 1$;
    else
        Return;
    end
end
Guess$(x, y, n_A, n_B)$;
```

Another difference is in the assignment of y which is the smallest element in $\mathbb{F}_2^m$ that does not yet have a corresponding input x such that $R(x) = y$. Note that y decides the representative output $R(x)$ in the BackwardSweep and Guess runs. The representative R of a balanced function S has the same output distribution as S, which implies each $y = R(x)$ can only occur once in a bijective permutation. This is why Algorithm 2 immediately increments y after using it. In a non-bijective function on the other hand, y can be reused 2^{n-m} times. Algorithm 5 therefore does not immediately increase y after each BackwardSweep but only when it runs out of candidates x' for which $S \circ A(x') = B(y)$. The complete procedure for finding the representative of a balanced non-injective function is illustrated in Figure 4.

This second feature actually makes the new algorithm very efficient in finding the smallest representative when $n-m$ is not too large. Instead of guessing $A(x)$, which implies a loop over approximately 2^n guesses, now the list of 2^{n-m} candidates x' immediately gives us the guesses $A(x')$ that result in the smallest output value $R(x)$. The more often we can reuse an output value y, the less often we need to guess. This can also be observed by comparing the examples in Figure 1 and 4. As a result, the algorithm to find a representative becomes more efficient for $n \times m$ functions with $m < n$. If m becomes very small, the complexity increases again since the enumeration of 2^{n-m} candidates, which is used also in [BCBP03], becomes the dominant factor. That the complexity first decreases and then increases with m corresponds to our initial observation in Figure 3.

a	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(a)$	1	3	1	0	1	2	3	3	2	0	3	0	2	2	1	0

	$x \to A(x)$	$\xrightarrow{S}$	$B(y) \leftarrow y$
Guess	$0 \to 0$	$\to$	$1 \leftarrow 0$
Bwd	$1 \to 2$	$\leftarrow$	$1 \leftarrow 0$
Bwd	$2 \to 4$	$\leftarrow$	$1 \leftarrow 0$
Fwd	$3 \to 6$	$\to$	$3 \leftarrow 1$
Bwd	$4 \to E$	$\leftarrow$	$1 \leftarrow 0$
Fwd	$5 \to C$	$\to$	$2 \leftarrow 2$
Fwd	$6 \to A$	$\to$	$3 \leftarrow 1$

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$R(x)$	0	0	0	1	0	2	1									

Figure 4: Example for 4-bit non-bijective S

4 Classifying Balanced $5 \times m$ Quadratic Functions

In this section, we first describe how all $5 \times m$ balanced quadratic functions can be classified iteratively using our algorithm. Even though all 5-bit quadratic Boolean functions and permutations have already been classified in [BW72] and [BBS17] respectively, this is the first time such an analysis is performed for $m \notin \{1, 5\}$. Moreover, we introduce novel optimizations using the (non-)linearity of the components to perform this classification much faster. We then compare the performances of finding all quadratic permutations using the method in [BBS17] with ours.

4.1 Naive Iteration

There exist 2^{15} different 5-bit quadratic Boolean functions. Since we target balanced functions, we consider only the balanced 18 259 out of 2^{15} as candidate coordinate functions $f_i : \mathbb{F}_2^5 \to \mathbb{F}_2$. In iterative stages for $m = 1$ to 5, we systematically augment all balanced $5 \times (m-1)$ functions with these 18 259 candidates to form a set of $5 \times m$ functions. We then use the adapted AE algorithm to reduce these functions to their affine equivalent representative. This reduction step is the key feature of the classification algorithm, since it not only provides us with all $5 \times m$ representatives, but also significantly lowers the workload of the next stage. The search procedure is described by Algorithm 6.

Algorithm 6: Generate Quadratic Functions

```
Initialize $\mathcal{R} = \{0\}$, $\mathcal{S} = \varnothing$ and $m = 1$;
Let $\mathcal{F}$ contain all balanced quadratic Boolean functions;
while $m < 5$ do
    for all $S = (S_1, \ldots, S_{m-1}) \in \mathcal{R}$ do
        for all candidates $f \in \mathcal{F}$ do
            if $S' = (S, f)$ is balanced then
                $\mathcal{S} \leftarrow \mathcal{S} \cup \{S'\}$;
            end
        end
    end
    $\mathcal{R} \leftarrow \varnothing$;
    for all $S \in \mathcal{S}$ do
        Find affine equivalent representative $R$ of $S$;
        $\mathcal{R} \leftarrow \mathcal{R} \cup R$;
    end
    Sort and eliminate doubles from $\mathcal{R}$.
    $\mathcal{S} \leftarrow \varnothing$;
    $m \leftarrow m + 1$;
end
```

Table 1 shows the number of representatives we obtain for $m = 1, \ldots, 5$[1]. Our results for $m \in \{1, 5\}$ align with those from previous works and require 50 minutes of computation time, using 4 threads on a Linux machine with an Intel Core i5-6500 processor at 3.20GHz. The comparison of this timing alone with a couple of hours, using 16 threads given in [BBS17] shows the impact of using an iterative approach, made possible by the new AE algorithm of Section 3. Nevertheless, in this section we will describe two ways to further optimize the complexity.

Table 1: Number of affine equivalence classes for $5 \times m$ functions for $m = 1, \ldots, 5$

	5×1	5×2	5×3	5×4	5×5
# classes	3	12	80	166	76

4.2 Impact of Linear Components on Efficiency

The runtime for finding the affine representative of a function depends on the accuracy of *guesses*. That is, the algorithm searches the smallest representative for each guess of $A(x)$. As a result, we notice that, the more nonlinear components the function has, the more dead ends the algorithm encounters and the more quickly it finishes. On the other hand, the more linear components the function exhibits, the more valid solutions for the affine transforms and thus the longer the algorithm needs to search through them. Therefore, the algorithm for finding the linear representative becomes less efficient as the number of linear components of the function increases.

[1]The exact listing of the representatives and their cryptographic properties can be found on `http://homes.esat.kuleuven.be/~ldemeyer/` > Miscellaneous.

In order to illustrate this significant difference, we choose five 5-bit representatives with a different number of linear components. We use the same class enumeration as in [BBS17] and represent the i^{th} quadratic permutation with $Q_i^{(5,5)}$. From each class, we randomly choose 100 permutations and observe the average runtime of the AE algorithm. The results of this experiment are shown in Table 2.

Table 2: Average runtimes of the AE algorithm [BCBP03] for some 5-bit representatives

Class	# Linear Components	Av. Runtime (s.)
$Q_1^{(5,5)}$	15	1.36
$Q_2^{(5,5)}$	7	0.39
$Q_{37}^{(5,5)}$	3	0.017
$Q_{49}^{(5,5)}$	1	0.0083
$Q_{75}^{(5,5)}$	0	0.0053

Moreover, we further analyze the runtime of the AE algorithm by removing coordinates to derive $n \times m$ functions with less linear components. The result is illustrated in Figure 5.

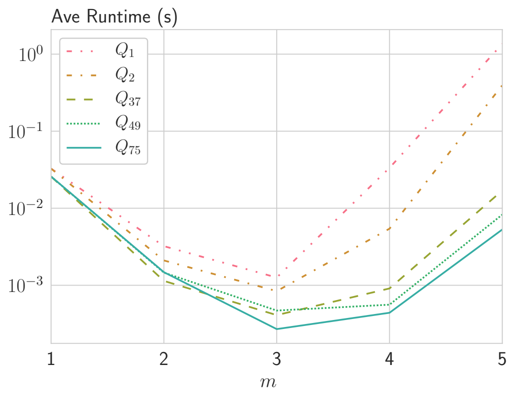

Figure 5: Actual runtimes observed for some 5-bit functions

We introduce the following definition of a *linear extension* in order to define our optimization for the classification.

Definition 6 (Linear Extension)**.** An n-bit permutation $F = (f_0, \ldots, f_{n-1})$ is called the linear extension of an $n \times m$ function $G = (f_0, \ldots, f_{m-1})$ if $\forall m \leq i < n$, f_i is linear.

Any balanced $n \times m$ function can be linearly extended with $n - m$ linear components into a balanced n-bit permutation. Correspondingly, each balanced n-bit permutation with $2^{n-m} - 1$ linear components can be generated as a linear extension of some balanced $n \times m$ function with zero linear components. We therefore initially eliminate all linear coordinate functions from our search, generating $5 \times m$ functions with only nonlinear coordinates in each step. In the very last stage, we obtain a list of 5-bit bijections without linear components. Finally, we add to this list all the linear extensions of the $5 \times m$ representatives found so far (for $m = 1, \ldots, 4$) to also obtain the 5-bit bijections with $2^{n-m} - 1$ linear components. This optimization increases the efficiency of the search in three ways. Firstly, it reduces the number of f_i candidates inserted in each stage ($|\mathcal{F}| \searrow$). Secondly, it discards functions for which finding the AE representative is slow. Finally, it reduces the number of $n \times m$ representatives that each stage starts from ($|\mathcal{R}| \searrow$).

4.3 Impact of the Order of Coordinate Functions on Efficiency

Optimizing our classification algorithm comes down to reducing as much as possible the number of functions to which the AE algorithm must be applied. Ideally, given two intermediate functions F_1 and F_2 which are affine equivalent ($F_1 \sim F_2$), we only want to find the representative of one of them. Affine equivalence is not so easily detected in all cases. However, we can focus on a simpler case. If F_1 and F_2 have the same coordinate functions, but in a different order, then they are naturally affine equivalent. Hence, we will try to fix the order of the coordinate functions of each intermediate function F and in that way eliminate all functions that are equal to F up to a reordering of the coordinates. To do this, we need a property to base the ordering on. Consider the three Boolean quadratic function classes $Q_0^{(5,1)}, Q_1^{(5,1)}$ and $Q_2^{(5,1)}$ for which representative ANF's and nonzero Walsh coefficient distributions are provided in Table 3.

Table 3: 5-bit Boolean functions

Class	Representative	$\#\|\omega : \hat{f}(\omega) = \xi\|$		
		$\xi = 32$	$\xi = 16$	$\xi = 8$
$Q_0^{(5,1)}$	x_0	1	0	0
$Q_1^{(5,1)}$	$x_0 \oplus x_1x_2$	0	4	0
$Q_2^{(5,1)}$	$x_0 \oplus x_1x_2 \oplus x_3x_4$	0	0	16

Lemma 1. *Every $n \times m$ function $F = (f_0, f_1, \ldots, f_{m-1})$ is affine equivalent to an $n \times m$ function $G(x) = (g_0, g_1, \ldots, g_{m-1})$ with $\max \hat{g}_0(\omega) \leq \max \hat{g}_1(\omega) \leq \ldots \leq \max \hat{g}_{m-1}(\omega)$, where $\hat{g}_i(\omega)$ is the Walsh spectrum of $g_i(x)$.*

Since this property is unique for each class of Boolean functions, we will use it to fix the order of coordinate functions during the classification algorithm. Using Lemma 1, in each intermediate step m, we will only allow new coordinate functions f_m for which the linearity is not smaller than the linearity of f_{m-1}.

4.4 Performance Comparison

Table 4 summarizes the results of the optimized search that takes a mere *6 minutes*, using 4 threads. This significant increase of performance enables us to classify for the first time also all 6-bit functions, which is described in Section 6. Note that the first column of Table 4 ($m = 0$) corresponds to the classes of affine $5 \times i$ functions. The last column holds the total number of $5 \times i$ functions, which is the sum of each row and corresponds to Table 1.

Each column starts with the number of "purely nonlinear" $5 \times m$ representatives (only nonlinear coordinates). The rows below the diagonal hold the number of classes that result from linearly extending the classes in previous rows. The last row shows the number of linear components in the corresponding 5×5 bijections and is equal to $2^{5-m} - 1$. We find 22 quadratic 5-bit equivalence classes without any linear components. Adding to this the linear extensions of smaller functions, we obtain all the 75 quadratic and the one affine 5-bit representatives.

Note that the number of classes obtained from linearly extending all $5 \times m$ functions can be much smaller than the number of $5 \times m$ classes itself (for example $23 \ll 93$ for $m = 4$). This can be explained by the fact that linearly extending two extended affine but not affine equivalent functions can result in affine equivalent permutations (*i.e.* a collision in the linear extension). Consider for example the following two 5×3 functions that are

Table 4: Number of affine equivalence classes for $5 \times i$ functions for $i = 1, \ldots, 5$ with $2^{i-m} - 1$ linear components.

# $5 \times i$ representatives	m = 0	1	2	3	4	5	Tot. #
# 5×1	1	2	-	-	-	-	3
# 5×2	1	3	8	-	-	-	12
# 5×3	1	5	19	55	-	-	80
# 5×4	1	3	17	52	93	-	166
# 5×5	1	2	6	22	23	22	76
# Linear Components:	31	15	7	3	1	0	

extended affine equivalent but not affine equivalent:

$$\begin{cases} x_0 \oplus x_1x_2 \\ x_1 \oplus x_2x_3 \\ x_4 \oplus x_0x_1 \end{cases} \not\sim \begin{cases} x_0 \oplus x_3 \oplus x_1x_2 \\ x_1 \oplus x_3 \oplus x_2x_3 \\ x_4 \oplus x_0x_1 \end{cases}$$

It is straightforward to verify that linearly extending both functions with coordinate functions x_2 and x_3 results in two affine equivalent 5-bit permutations.

$$\begin{cases} x_0 \oplus x_1x_2 \\ x_1 \oplus x_2x_3 \\ x_4 \oplus x_0x_1 \\ x_2 \\ x_3 \end{cases} \sim \begin{cases} x_0 \oplus x_3 \oplus x_1x_2 \\ x_1 \oplus x_3 \oplus x_2x_3 \\ x_4 \oplus x_0x_1 \\ x_2 \\ x_3 \end{cases}$$

5 Decomposing and Generating Higher Degree Permutations

We now adapt our algorithm to (de)compose higher degree functions into/from quadratics. This leads to area efficient implementations especially in the context of side-channel countermeasures and masking, where the area grows exponentially with the degree of a function. Below we describe length-two decompositions and constructions of cryptographically interesting permutations.

5.1 Length-two Decomposition

We are trying to decompose a higher degree function $H : \mathbb{F}_2^n \to \mathbb{F}_2^n$. If a quadratic decomposition of length two exists, then we can state that $H = B \circ R_1 \circ A \circ R_2 \circ C$ with A, B, C affine permutations and R_1, R_2 representatives of n-bit quadratic classes. Alternatively, we can state that H is affine equivalent to $R_1 \circ A \circ R_2$. Suppose we fix R_2 (to one of the known n-bit representatives) and we want to find the representative R_1 and the affine permutation A such that

$$H \sim R_1 \circ A \circ R_2. \tag{1}$$

As with the classification of Boolean functions, we perform this search iteratively, starting from $n \times 1$ Boolean functions f for which $f \circ R_2 : \mathbb{F}_2^n \to \mathbb{F}_2$ is extended affine equivalent to a component function of H. We thus select the candidates for f using the following criteria:

(C1) f is balanced

(C2) $\exists \beta \in \mathbb{F}_2^m \setminus \{0\}$ s.t. $\mathrm{Degr}(f \circ R_2) = \mathrm{Degr}(\beta \cdot H)$

(C3) $\exists \beta \in \mathbb{F}_2^m \setminus \{0\}$ s.t. $(\Delta_{f \circ R_2}, \Lambda_{f \circ R_2}) = (\Delta_{\beta \cdot H}, \Lambda_{\beta \cdot H})$

Starting from this list of candidates, we proceed in a similar manner as in Algorithm 6. We only slightly have to tweak this algorithm to obtain the decomposition Algorithm 7. Each time we augment an $n \times (m-1)$ function with one of the Boolean function candidates

Algorithm 7: Find decompositions of length two

```
for all quadratic n-bit representatives $R_2$ do
    Initialize $\mathcal{R} = \{0\}$, $\mathcal{S} = \varnothing$ and $m = 1$;
    $\mathcal{F} \leftarrow$ all quadratic Boolean functions $f$ satisfying above criteria (C1-C3);
    while $m < n$ do
        $\mathcal{D} \leftarrow \{(\Delta_{L \circ H}, \Lambda_{L \circ H})\}_{L \in \mathscr{L}(\mathbb{F}_2^n \to \mathbb{F}_2^m)}$;
        for all $S \in \mathcal{R}$ do
            for all candidates $f \in \mathcal{F}$ do
                if $S' = (S, f)$ is balanced and $(\Delta_{S' \circ R_2}, \Lambda_{S' \circ R_2}) \in \mathcal{D}$ then
                    $\mathcal{S} \leftarrow \mathcal{S} \cup \{S'\}$;
                end
            end
        end
        $\mathcal{R} \leftarrow \varnothing$;
        for all $S \in \mathcal{S}$ do
            Find left affine equivalent representative $R^L$ of $S$;
            $\mathcal{R} \leftarrow \mathcal{R} \cup R^L$;
        end
        Sort and eliminate doubles from $\mathcal{R}$.
        $\mathcal{S} \leftarrow \varnothing$;
        $m \leftarrow m + 1$;
    end
    if $\mathcal{R} \neq \varnothing$ then
        Decomposition of length 2 found;
    end
end
```

f to a balanced $n \times m$ function F, we verify that the DDT and LAT of $F \circ R_2$ have the same frequency distributions as those of some function $H' : \mathbb{F}_2^n \to \mathbb{F}_2^m$, that has as its coordinate functions a subset of the components of H. Let $\mathscr{L}(\mathbb{F}_2^n \to \mathbb{F}_2^m)$ be the set of all balanced linear mappings from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Then, we can describe H' as $L \circ H$ for some $L \in \mathscr{L}(\mathbb{F}_2^n \to \mathbb{F}_2^m)$. In order to eliminate false candidates as early as possible, we verify for each intermediate $n \times m$ candidate F if the composition $F \circ R_2$ *can* be affine equivalent to some subfunction H'. In particular, we check that

$$(\Delta_{F \circ R_2}, \Lambda_{F \circ R_2}) \in \{(\Delta_{L \circ H}, \Lambda_{L \circ H})\}_{L \in \mathscr{L}(\mathbb{F}_2^n \to \mathbb{F}_2^m)}$$

The quadratic function classification algorithm (Algorithm 6) is very efficient because it reduces the lists of intermediate functions to their affine equivalent representatives at each step m. However, we cannot do that in this case as this would change the affine transformation A in the decomposition (see Eqn. (1)). Let $S_1 = B \circ R_1 \circ A$ be a candidate for which $S_1 \circ R_2 \sim H$ and let R_1 be its affine representative. Reducing S_1 to R_1 would discard the affine transformation A. In that case, we would only be able to decompose functions that are affine equivalent to the composition of two representatives: $R_1 \circ R_2$. In other words, if there is another candidate S'_1 affine equivalent to S_1, we do not want to discard it as it will not necessarily result in affine equivalent compositions.

$$S'_1 \sim S_1 \not\Rightarrow S'_1 \circ R_2 \sim S_1 \circ R_2$$

However, without any reductions in the intermediate steps of the algorithm, the search becomes very inefficient as the list of candidate functions grows exponentially. There is still a redundancy in our search because of the affine output transformation B that is included in S_1. If S'_1 is only left affine equivalent to S_1, then their compositions are affine equivalent:

$$S'_1 = B' \circ S_1 \Rightarrow S'_1 \circ R_2 \sim S_1 \circ R_2$$

We therefore adapt the AE algorithm to find the lexicographically smallest function R_1^L that is left affine equivalent to S_1: $R_1^L = B^{-1} \circ S_1 = R_1 \circ A$. We call this function R_1^L the left affine representative of S_1. The algorithm to find R_1^L is identical to finding the affine equivalent representative with the input affine transformation constrained to the identity function. This constraint removes the need for guesses and makes the algorithm very efficient. An example is shown in Figure 6. Algorithm 7 summarizes the resulting decomposition method.

a	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(a)$	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0

$$x \to A(x) \quad \overset{S}{\to} \quad B(y) \leftarrow y$$

Fwd	$0 \to 0$	$\to$	$1 \leftarrow 0$
Fwd	$1 \to 1$	$\to$	$B \leftarrow 1$
Fwd	$2 \to 2$	$\to$	$9 \leftarrow 2$
Fwd	$3 \to 3$	$\to$	$C \leftarrow 4$
Fwd	$4 \to 4$	$\to$	$D \leftarrow 8$
Fwd	$5 \to 5$	$\to$	$6 \leftarrow 5$
Fwd	$6 \to 6$	$\to$	$F \leftarrow 7$

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$R^L(x)$	0	1	2	4	8	5	7									

Figure 6: Example for finding the left representative R^L of S. Input transformation A is fixed to the identity function: $A(x) = x$

5.2 Almost Bent Permutations

There are five APN 5×5 permutations up to affine equivalence [BL08], all corresponding to a power map over $\mathbb{F}_{2^5}$. Two of these are quadratic and AB and correspond to classes $\mathcal{Q}_{74}^{(5,5)}(\sim x^3)$ and $\mathcal{Q}_{75}^{(5,5)}(\sim x^5)$. We demonstrate Algorithm 7 by decomposing the inverses of these classes (resp. x^{11} and x^7), which are cubic and (naturally) also AB. The algorithm is very efficient in this case because the properties of AB's are so well defined: Firstly, all components of the cubic AB's have the same algebraic degree (=3). We also know that the DDT of an AB function contains only zeros and twos and its LAT contains only zeros and elements with absolute value 4. It immediately follows that also the Walsh transform of each coordinate function of the AB is equal to either 0 or ± 8.

Moreover, when we look at all $5 \times m$ subfunctions $H' = L \circ H, \forall L \in \mathscr{L}(\mathbb{F}_2^n \to \mathbb{F}_2^m)$, there is only one permitted differential spectrum and LAT frequency distribution for each m. It is indeed known that all coordinate functions of the AB are (extended) affine equivalent.

We enumerate all 75 candidates for R_2 and perform the search for $S_1 = R_1 \circ A$ using Algorithm 7. When R_2 is the representative of classes $Q_1^{(5,5)}$ to $Q_{74}^{(5,5)}$, the algorithm finds no 5-bit bijections that compose with R_2 to a cubic AB. The search only ends with non-empty $\mathcal{R}$ when we perform it with R_2 the representative of $Q_{75}^{(5,5)}$, which is itself the quadratic AB permutation x^5. The resulting R_1 is equal to R_2 and their composition forms the AB class that holds the inverse of $Q_{75}^{(5,5)}$ (corresponding to power map x^7). This decomposition is easily verified using power maps. Indeed, $x^5 \circ x^5 \circ x^5 = x^{125} = x^{1 \mod 31}$.

Without the constraint that the AB needs to be cubic, we also find a decomposition for class $Q_{75}^{(5,5)}$ itself with $R_1 = R_2 = Q_{74}^{(5,5)}$. A length-two decomposition for the odd cubic AB permutations (x^{11}) is not found. Since the algorithm is exhaustive, this means it does not exist. Indeed, it is shown in [NNR19] that the shortest decomposition of x^{11} has length three.

Table 5: Look-up-tables for the even cubic AB function F and its decomposition $F = S_1 \circ R_2$ with $S_1 \sim R_2$

F	0,1,2,8,4,17,30,13,10,18,5,19,6,20,11,26,16,15,9,23,3,7,29,21,14,12,25,31,28,27,22,24
S_1	0,1,2,4,8,10,16,21,17,28,18,24,23,25,14,7,30,6,19,12,20,15,3,31,9,29,5,22,13,26,27,11
R_2	0,1,2,4,3,8,16,28,5,10,26,18,17,20,31,29,6,21,24,12,22,15,25,7,14,19,13,23,9,30,27,11

5.3 The Keccak χ Inverse

The nonlinear transformation χ used in the Keccak [BDPA13] sponge function family χ (Figure 7) is a quadratic 5-bit permutation from class $Q_{68}^{(5,5)}$ with a cubic inverse. For the possibility of implementing an algorithm using χ^{-1}, we decompose this cubic permutation (see Table 6).

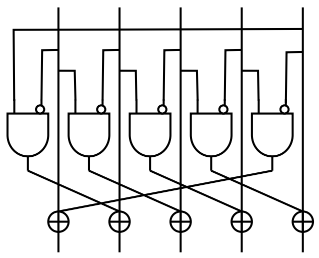

Figure 7: The nonlinear transformation χ from Keccak [BDPA13]

Table 6: Look-up-tables for the Keccak permutation χ and its inverse χ^{-1}

χ	0,9,18,11,5,12,22,15,10,3,24,1,13,4,30,7,20,21,6,23,17,16,2,19,26,27,8,25,29,28,14,31
χ^{-1}	0,11,22,9,13,4,18,15,26,1,8,3,5,12,30,7,21,20,2,23,16,17,6,19,10,27,24,25,29,28,14,31

The Keccak inverse does not have the same strong properties as the AB permutations. Each coordinate function is still cubic but the differential and linear properties are naturally weaker. Firstly, apart from zeros we find both ± 4 and ± 8 in the LAT. For the DDT, there are multiple differential spectra for the intermediate $5 \times m$ sub functions. As explained above, we generate the list of possible DDT and LAT frequency distributions for each $m = 1, \ldots, 4$ and feed this as input to the search algorithm. We filter out all intermediate functions $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ for which the DDT and LAT frequency distributions of $F \circ R_2$ do not occur in this list.

While the search finds many classes with the same cryptographic properties as the Keccak inverse, a decomposition of length 2 for χ^{-1} itself does not appear to exist.

5.4 Towards Higher-Degree Permutations

When it comes to choosing a nonlinear permutation for use in a cryptographic primitive, the designer will sooner go to those with higher degree as they provide more resilience against higher-order differential and algebraic attacks [DEM15]. With masked implementations in mind, we thus want to find strong n-bit permutations with high algebraic degree for which a decomposition into quadratic blocks exists. Our decomposition algorithm can be used for this purpose. If instead of searching for specific functions H, we define a set of more general but strong criteria, we can use the algorithm 7 to generate a list of favorable permutations. In particular, we use the following criteria to perform a search for 5-bit permutations S with optimal algebraic degree and near-optimal cryptographic properties:

- S is balanced
- $\text{Degr}(S) = 4$
- $\text{Lin}(S) \leq 6$
- $\text{Diff}(S) \leq 4$

The first three criteria are easily translated for intermediate $5 \times m$ functions (the bound on the LAT table stays the same). As we are not looking for a known class, this is more difficult for the bound on the DDT. We use the fact that the upperbound on the values in the DDT at most doubles every time we discard one output bit (see Theorem 1). This upperbound is not tight, but can be used to filter some of the unusable intermediate functions F.

Theorem 1 ([Nyb94, Thm. 12])**.** *Let $S = (f_0, f_1, \ldots, f_{n-1}) : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an n-bit bijection with* $\text{Diff}(S)$ *the maximal value in its DDT. Then, for any function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with $m < n$, composed from a subset of the coordinate functions of S, $F = (f_{i_1}, f_{i_2}, \ldots, f_{i_m})$ with $i_1, \ldots, i_m \in \{0, \ldots, n-1\}$, the values in its DDT are upperbounded by* $\text{Diff}(S) \cdot 2^{n-m}$.

Our search delivers 17 quartic affine equivalence classes with very good cryptographic properties, shown in Table 7. One of those is the APN class, which contains the permutation formed by the inversion x^{-1} in $\mathbb{F}_{2^5}$. Indeed, it was shown in [NNR19] that this permutation has decomposition length two. Each of these very strong 5-bit S-boxes have an efficient masked implementation, as they can be decomposed into only two quadratic components. This list is only a sample of the functions that can be found using this method.

6 Classifying $6 \times m$ Quadratic Functions

The efficiency of Algorithm 6 makes it feasible to extend the search for quadratic permutations to $n = 6$ bits for the first time. There are 2^{21} different 6-bit quadratic Boolean functions, of which there are 914 004 nonlinear balanced ones. This is our list $\mathcal{F}$ of candidate coordinate functions $f_i : \mathbb{F}_2^6 \to \mathbb{F}_2$. Generating all classes of $6 \times m$ functions for $m < 6$ without linear components takes 8.5 hours on 24 cores. The total number of classes found for each m is shown in Table 8[2]. Tables 9 to 12 show histograms of the

[2]The exact listing of the representatives and their cryptographic properties can be found on http://homes.esat.kuleuven.be/~ldemeyer/ > Miscellaneous.

Table 7: Strong quartic (degree 4) 5-bit permutations with decomposition length two

Cl.	Representative	Diff	Lin	R_1	R_2
1:	0,1,2,3,4,6,7,8,5,9,16,12,21,26,29,30,10,18,24,13,27,17,20,31,14,11,23,19,22,28,15,25	4	6	$Q_{52}^{(5,5)}$	$Q_{57}^{(5,5)}$
2:	0,1,2,3,4,6,7,8,5,12,16,26,28,18,29,13,9,21,30,25,10,27,20,22,14,19,23,31,17,24,11,15	4	6	$Q_{71}^{(5,5)}$	$Q_{62}^{(5,5)}$
3:	0,1,2,3,4,6,7,8,5,16,18,29,9,15,28,26,10,30,20,19,23,31,24,11,12,22,27,17,13,21,14,25	4	6	$Q_{53}^{(5,5)}$	$Q_{67}^{(5,5)}$
4:	0,1,2,3,4,6,8,11,5,9,16,18,12,17,28,23,7,31,21,19,10,26,14,29,30,25,27,22,24,13,15,20	4	6	$Q_{71}^{(5,5)}$	$Q_{75}^{(5,5)}$
5:	0,1,2,3,4,6,8,11,5,12,16,24,15,21,17,20,7,23,9,18,14,19,25,30,31,10,28,22,13,26,27,29	4	6	$Q_{69}^{(5,5)}$	$Q_{71}^{(5,5)}$
6:	0,1,2,3,4,6,8,12,5,7,16,27,26,15,28,18,9,14,22,17,20,31,24,21,13,29,10,19,25,23,11,30	4	6	$Q_{33}^{(5,5)}$	$Q_{70}^{(5,5)}$
7:	0,1,2,3,4,6,8,12,5,9,16,24,31,10,17,20,7,21,28,22,15,29,14,27,25,19,11,23,13,26,30,18	4	6	$Q_{74}^{(5,5)}$	$Q_{74}^{(5,5)}$
8:	0,1,2,3,4,6,8,12,5,9,16,31,18,17,15,23,7,24,10,29,21,27,11,28,25,30,14,22,26,19,20,13	4	6	$Q_{74}^{(5,5)}$	$Q_{75}^{(5,5)}$
9:	0,1,2,3,4,6,8,12,5,10,16,27,25,19,22,11,7,18,30,13,24,21,28,15,31,9,26,29,23,17,20,14	4	6	$Q_{74}^{(5,5)}$	$Q_{68}^{(5,5)}$
10:	0,1,2,3,4,6,8,12,5,11,16,25,18,10,19,29,7,17,30,21,31,24,13,14,27,22,26,9,20,23,28,15	4	6	$Q_{75}^{(5,5)}$	$Q_{74}^{(5,5)}$
11:	0,1,2,3,4,6,8,12,5,11,16,24,22,26,9,19,7,23,10,13,31,18,20,29,27,30,28,15,14,17,21,25	4	6	$Q_{74}^{(5,5)}$	$Q_{68}^{(5,5)}$
12:	0,1,2,3,4,6,8,12,5,13,16,23,17,18,24,11,7,29,21,27,25,9,22,10,31,14,15,20,19,30,28,26	4	6	$Q_{72}^{(5,5)}$	$Q_{68}^{(5,5)}$
13:	0,1,2,3,4,6,8,12,5,14,16,26,10,27,23,31,7,24,11,28,20,17,9,18,25,21,13,30,15,22,29,19	4	6	$Q_{10}^{(5,5)}$	$Q_{72}^{(5,5)}$
14:	0,1,2,3,4,6,8,12,5,16,13,23,25,21,26,14,7,17,20,28,29,19,11,9,15,10,31,24,27,18,30,22	4	6	$Q_{72}^{(5,5)}$	$Q_{74}^{(5,5)}$
15:	0,1,2,3,4,6,8,12,5,16,21,26,31,22,18,10,7,24,17,13,30,14,19,27,20,9,23,25,11,29,15,28	4	6	$Q_{74}^{(5,5)}$	$Q_{74}^{(5,5)}$
16:	0,1,2,3,4,6,8,16,5,10,20,29,7,31,27,13,9,25,15,18,19,14,22,26,21,17,11,12,30,28,23,24	4	6	$Q_{10}^{(5,5)}$	$Q_{75}^{(5,5)}$
17:	0,1,2,4,3,6,8,16,5,10,15,27,19,29,31,20,7,18,25,21,12,14,24,28,26,11,23,13,30,9,17,22	2	6	$Q_{75}^{(5,5)}$	$Q_{74}^{(5,5)}$

classes' cryptographic properties. It is interesting to note that the two best 6×5 classes in Table 12 correspond to the two AB 5×5 classes $Q_{74}^{(5,5)}$ and $Q_{75}^{(5,5)}$, extended with a sixth unused input bit.

Table 8: Number of affine equivalence classes with/without linear components for quadratic $6 \times m$ functions for $m = 1, \ldots, 5$

	6×1	6×2	6×3	6×4	6×5
# classes without	2	19	604	10 480	7 458
# classes with	3	24	670	11 891	12 647

Table 9: Number of quadratic 6×2 classes with cryptographic properties (Diff, Lin)

	Lin = 8	Lin = 16	Lin = 32
Diff = 32	5	3	0
Diff = 64	2	9	5

Table 10: Number of quadratic 6×3 classes with cryptographic properties (Diff, Lin)

	Lin = 8	Lin = 16	Lin = 32
Diff = 16	57	7	0
Diff = 32	128	252	19
Diff = 64	11	149	47

Table 11: Number of quadratic 6×4 classes with cryptographic properties (Diff, Lin)

	Lin = 8	Lin = 16	Lin = 32
Diff = 8	10	1	0
Diff = 16	1935	845	64
Diff = 32	618	5013	740
Diff = 64	42	2016	607

Table 12: Number of quadratic 6×5 classes with cryptographic properties (Diff, Lin)

	Lin = 8	Lin = 16	Lin = 32
Diff = 4	2	0	0
Diff = 8	111	3	4
Diff = 16	124	1028	424
Diff = 32	0	3343	2993
Diff = 64	4	2843	1768

In order to complete the final stage of the search for all 6-bit quadratic permutations, we generate the list of candidates for the AE algorithm by extending the 6×5 representatives with the Boolean function candidates $f_i \in \mathcal{F}$ and we add the linear extensions of all other $6 \times m$ functions. We split this list into 100 parts and complete the rest of the algorithm on 100 cores. In the end, we find 2 263 classes of 6-bit quadratic permutations (not including the one linear permutation)[3]. Table 13 shows how these classes are distributed among even and odd permutations or how many of them have quadratic/cubic inverses. Table 14 depicts the histogram of cryptographic properties. There are eight classes with Diff = 4 and Lin = 8. These are shown in Table 15. One of those permutations is odd. Finally, Figure 8 shows the total number of affine equivalence classes of quadratic $n \times n$ permutations. While it was already clear that this number grows fast with n, the figure demonstrates how difficult it was before this work to predict just *how* fast.

Table 13: Number of quadratic 6×6 classes with certain properties

Even/Odd	2258	5
Inverse = quadratic/cubic	70	2193

Table 14: Number of quadratic 6×6 classes with cryptographic properties (Diff, Lin)

	Lin = 8	Lin = 16	Lin = 32
Diff = 4	8	0	0
Diff = 8	0	0	12
Diff = 16	0	49	100
Diff = 32	0	49	1067
Diff = 64	0	200	779

[3]The exact listing of the representatives and their cryptographic properties can be found on `http://homes.esat.kuleuven.be/~ldemeyer/` > Miscellaneous.

Table 15: Strong quadratic 6-bit permutations

Cl.	Representative	Diff	Lin	Parity
2256:	0,1,2,3,4,6,7,5,8,12,16,20,32,39,57,62,9,17,21,13,40,51,53,46,50,47,52,41,63,33,56,38,10,45, 27,60,43,15,59,31,58,24,49,19,55,22,61,28,29,35,18,44,25,36,23,42,30,37,11,48,54,14,34,26	4	8	Even
2257:	0,1,2,3,4,6,7,5,8,12,16,20,32,39,57,62,9,17,21,13,41,50,52,47,40,53,46,51,36,58,35,61,10,25, 37,54,33,49,15,31,45,59,24,14,42,63,30,11,29,23,44,38,18,27,34,43,19,28,56,55,48,60,26,22	4	8	Even
2258:	0,1,2,3,4,6,7,5,8,12,16,20,32,39,57,62,9,17,21,13,41,50,52,47,55,42,49,44,59,37,60,34,10,25, 38,53,35,51,14,30,61,43,11,29,56,45,15,26,22,28,36,46,27,18,40,33,23,24,63,48,54,58,31,19	4	8	Even
2259:	0,1,2,3,4,6,8,10,5,11,16,30,32,45,59,54,7,20,41,58,47,63,15,31,48,44,9,21,57,38,14,17,12,25, 24,13,34,52,56,46,40,50,43,49,39,62,42,51,35,36,27,28,33,37,23,19,53,61,26,18,22,29,55,60	4	8	Even
2260:	0,1,2,3,4,6,8,10,5,11,16,30,32,45,59,54,7,24,40,55,48,44,17,13,9,25,49,33,31,12,41,58,14,27, 26,15,57,47,35,53,61,39,62,36,43,50,38,63,46,37,23,28,42,34,29,21,22,18,56,60,51,52,19,20	4	8	Even
2261:	0,1,2,3,4,6,8,10,5,11,16,30,32,45,59,54,7,34,21,48,13,43,17,55,56,18,61,23,19,58,24,49,9,52, 20,41,31,33,12,50,46,28,36,22,25,40,29,44,62,39,51,42,38,60,37,63,35,53,57,47,26,15,14,27	4	8	Even
2262:	0,1,2,3,4,6,8,10,5,12,16,25,32,42,59,49,7,20,14,29,52,36,51,35,53,46,43,48,39,63,55,47,9,58, 44,31,22,38,61,13,19,40,33,26,45,21,17,41,54,23,24,57,30,60,62,28,27,50,34,11,18,56,37,15	4	8	Even
2263:	0,1,2,3,4,8,16,28,5,12,32,41,10,14,57,61,6,62,23,47,33,20,38,19,43,27,29,45,7,58,39,26,9,22, 55,40,11,25,35,49,44,59,53,34,37,63,42,48,21,51,56,30,52,31,15,36,24,54,18,60,50,17,46,13	4	8	Odd

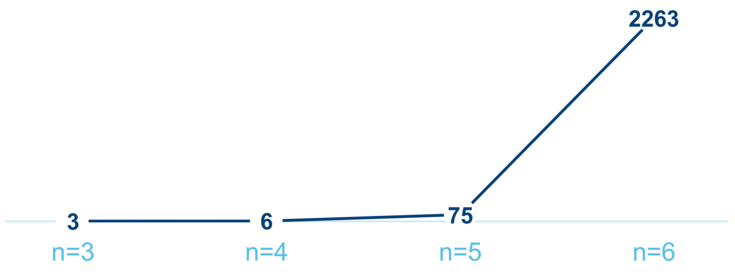

Figure 8: Number of quadratic $n \times n$ classes for growing n

Conclusion

This work studies the classification of quadratic vectorial Boolean functions under affine equivalence. It extends Biryukov's Affine Equivalence algorithm to non-bijective functions for use in a new classification tool that provides us with the complete classification of balanced $n \times m$ quadratic vectorial Boolean functions for $m \leq n$ and $n < 7$. We also introduce a tool for finding length-two quadratic decompositions of higher degree functions.

New cryptographic algorithms should be designed with resistance against side-channel attacks in mind. When it comes to choosing S-boxes, designers can use our classification to pick quadratic components and use our (de)composition tool to create cryptographically strong S-boxes with efficient masked implementations. After the classifications of 4- and 5-bit permutations in previous works, this work expands the knowledge base on both classification and decomposition, bringing us one step closer to classifying 8-bit functions and decomposing the AES S-box using permutations instead of tower field or square-and-multiply approaches.

Acknowledgements

The authors thank Dusan Bozilov for the insights into his algorithm and Prof. Vincent Rijmen for fruitful discussion and helpful comments. This work was supported by the Research Council KU Leuven: C16/15/058. Lauren De Meyer is funded by a PhD fellowship (aspirant) of the Fund for Scientific Research - Flanders (FWO) and Begül Bilgin is a postdoctoral fellow of the FWO.

References

[ABB$^+$14] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. CAESAR submission: PRIMATEs v1.02, March 2014. http://primates.ae/wp-content/uploads/primatesv1.02.pdf.

[AGR$^+$16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.

[ARS$^+$15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.

[BBS17] Dusan Bozilov, Begül Bilgin, and Haci Ali Sahin. A note on 5-bit quadratic permutations' classification. *IACR Trans. Symmetric Cryptol.*, 2017(1):398–404, 2017.

[BCBP03] Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2003.

[BDP$^+$14] Guido Bertoni, Joan Daemon, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Ketje v1, March 2014. https://competitions.cr.yp.to/round1/ketjev1.pdf.

[BDP$^+$15] Guido Bertoni, Joan Daemon, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Keyak v2, August 2015. https://competitions.cr.yp.to/round2/keyakv2.pdf.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *LNCS*, pages 313–314. Springer, 2013.

[BGG$^+$16] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit sboxes with efficient masking in hardware. In Gierlichs and Poschmann [GP16], pages 171–193.

[Bil15] Begül Bilgin. *Threshold Implementations: As Countermeasure Against Higher-Order Differential Power Analysis.* PhD thesis, KU Leuven, Belgium & UTwente, The Netherlands, 2015.

[BKNN15] Begül Bilgin, Miroslav Knezevic, Ventzislav Nikov, and Svetla Nikova. Compact implementations of multi-sbox designs. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 273–285. Springer, 2015.

[BL08] Marcus Brinkmann and Gregor Leander. On the classification of APN functions up to dimension five. *Des. Codes Cryptography*, 49(1-3):273–288, 2008.

[BNN$^+$12] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3x3 and 4x4 s-boxes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2012.

[BNN$^+$15] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Natalia N. Tokareva, and Valeriya Vitkup. Threshold implementations of small s-boxes. *Cryptography and Communications*, 7(1):3–33, 2015.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BW72] Elwyn R. Berlekamp and Lloyd R. Welch. Weight distributions of the cosets of the (32, 6) reed-muller code. *IEEE Trans. Information Theory*, 18(1):203–207, 1972.

[Can07] Christophe De Cannière. *Analysis and Design of Symmetric Encryption Algorithms.* PhD thesis, Katholieke Universiteit Leuven, 2007.

[CCZ98] Claude Carlet, Pascale Charpin, and Victor A. Zinoviev. Codes, bent functions and permutations suitable for des-like cryptosystems. *Des. Codes Cryptography*, 15(2):125–156, 1998.

[CGP$^+$12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 742–763. Springer, 2015.

[CRV14] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 170–187. Springer, 2014.

[CV94] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 356–365. Springer, 1994.

[DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Higher-order cryptanalysis of lowmc. In Soonhak Kwon and Aaram Yun, editors, *Information Security and Cryptology - ICISC 2015 - 18th International Conference, Seoul, South Korea, November 25-27, 2015, Revised Selected Papers*, volume 9558 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2015.

[DEMS15] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. CAESAR submission: ASCON v1.1, August 2015. https://competitions.cr.yp.to/round2/asconv11.pdf.

[Ful03] Joanne Elizabeth Fuller. *Analysis of affine equivalent boolean functions for cryptography*. PhD thesis, Queensland University of Technology, 2003.

[Gol59] Solomon W. Golomb. On the classification of boolean functions. *IRE Trans. Information Theory*, 5(5):176–186, 1959.

[GP16] Benedikt Gierlichs and Axel Y. Poschmann, editors. *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*. Springer, 2016.

[GR16] Dahmun Goudarzi and Matthieu Rivain. On the multiplicative complexity of boolean functions and bitsliced higher-order masking. In Gierlichs and Poschmann [GP16], pages 457–478.

[KNP13] Sebastian Kutzner, Phuong Ha Nguyen, and Axel Poschmann. Enabling 3-share threshold implementations for all 4-bit s-boxes. In Hyang-Sook Lee and Dong-Guk Han, editors, *Information Security and Cryptology - ICISC 2013 - 16th International Conference, Seoul, Korea, November 27-29, 2013, Revised Selected Papers*, volume 8565 of *Lecture Notes in Computer Science*, pages 91–108. Springer, 2013.

[Mai91] James A. Maiorana. A classification of the cosets of the Reed-Muller Code $\mathrm{R}(1, 6)$. *Mathematics of Computation*, 57(195):403–414, 1991.

[NK95] Kaisa Nyberg and Lars R. Knudsen. Provable security against a differential attack. *J. Cryptology*, 8(1):27–37, 1995.

[NNR19] Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Decomposition of permutations in a finite field. *Cryptography and Communications*, 11(3):379–384, 2019.

[Nyb93] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1993.

[Nyb94] Kaisa Nyberg. S-boxes and round functions with controllable linearity and differential uniformity. In Preneel [Pre95], pages 111–130.

[O'C94] Luke O'Connor. Properties of linear approximation tables. In Preneel [Pre95], pages 131–136.

[Pre95] Bart Preneel, editor. *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*. Springer, 1995.

[PUB16] Léo Perrin, Aleksei Udovenko, and Alex Biryukov. Cryptanalysis of a theorem: Decomposing the only known solution to the big APN problem. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 93–122. Springer, 2016.

[PV16] Jürgen Pulkus and Srinivas Vivek. Reducing the number of non-linear multiplications in masking schemes. In Gierlichs and Poschmann [GP16], pages 479–497.

[RV13] Arnab Roy and Srinivas Vivek. Analysis and improvement of the generic higher-order masking scheme of FSE 2012. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 417–434. Springer, 2013.

[Saa11] Markku-Juhani O. Saarinen. Cryptographic analysis of all 4 x 4-bit s-boxes. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2011.