Algorithmic Toolkit for Linearization of S-boxes*

Alex Biryukov[1,2], Philip Tureček[2], and Aleksei Udovenko[2]

DCS, University of Luxembourg, Esch-sur-Alzette, Luxembourg
SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
first-name.last-name@uni.lu

Abstract. Linearization is a cryptanalysis technique in which a nonlinear function (an S-box) is represented by an affine mapping on a certain subset of inputs. Its variants were applied to analyze Keccak, LowMC, RAIN and AIM. In these primitives, the S-boxes are either very small (up to 5 bits) or are very specific monomial functions over a binary field. Linearization of arbitrary S-boxes was never practically explored due to the lack of theoretic, algorithmic, and cryptanalytic understanding.

For the first time, we develop an algorithmic toolkit which allows one to compute strong linearizations of S-boxes, when they exist. For up to $n = 8$ bits, our algorithms are able to find provably the best possible approximations, while for larger S-boxes it is feasible to obtain good approximations together with meaningful upper bounds. We apply our algorithms to a variety of S-boxes from existing primitives, to monomial functions, to so-called APN functions, and to 16-bit Super-Sboxes. We obtain interesting results raising many new open questions and open up new research directions, as well as a foundation for developing cryptanalytic attacks.

To advance the cryptanalytic utility of linearization, we study and solve the problem of covering an S-box with multiple approximations. As an application, we derive a generic linearization approach for the CICO problem (constrained-input-constrained-output) over SPN-based permutations (Substitution-Permutation Networks) with general linear layers. This is the first such general cryptanalysis based on the existence of a strong linearization of the S-box.

Keywords: Boolean functions · Linearization · Vectorial linearity · S-boxes · CICO

1 Introduction

In symmetric-key cryptography, S-boxes are key components of encryption algorithms such as block ciphers or those that involve permutations. S-boxes are nonlinear functions defined over a small domain and can be described by a lookup table. These functions are typically the only source of nonlinearity in those encryption primitives, and thus it is crucial to understand their security properties such as resistance to various attacks.

The first and main security properties of S-boxes that were studied are the differential uniformity and the (non)linearity [36,37], quantifying resistance of the S-box against the differential [8] and linear cryptanalysis [33,31] respectively. Since then, many other attacks and corresponding properties have been studied, such as algebraic degree, correlation/algebraic immunity, boomerang and rectangle uniformity, nonlinear invariants, algebraic degrees of polynomial relations.

A much less studied concept is the *vectorial* linearity of an S-box, which characterizes the size of the best possible linear approximation of the full function (i.e., of all output bits simultaneously), the process called *linearization*. First defined in [16], this notion was only theoretically studied in a few subsequent works [30,13,42,18,34]. More formally, *vectorial nonlinearity* is defined as the Hamming distance from the given vectorial Boolean function (an S-box) to the set of affine mappings with the same signature. Equivalently, *vectorial linearity* is equal to the maximum size of a subset of the domain of the S-box, on which it can be described by an affine map, called *affine approximation*.

In cryptanalysis, linearization is particularly useful in low-data attacks, especially in the setting with only one known plaintext-ciphertext pair, where traditional statistical cryptanalysis techniques such as

* Full version of the paper accepted at EUROCRYPT 2026. Work funded by Luxembourg's FNR projects CryptoFin (C22/IS/17415825) and PQseal (C24/IS/18978392). We thank Luca Bonamino for Rust implementations of the greedy extension algorithm and the greedy reduction variant for the AES Super-Sbox. We also thank the anonymous reviewers for their comments.

differential, linear, and integral cryptanalysis are not applicable. This setting became practically relevant due to post-quantum signature schemes based on symmetric-key primitives, using the so-called multiparty-computation-in-the-head paradigm and its successors. Examples include Picnic/LowMC [46], AIMer [28], Banquet [3]. AIMer was recently selected as the winner of the Korean Post-Quantum Cryptography Competition (KpqC), and the AES-based Banquet is a current candidate of the ongoing NIST Additional Post-Quantum Signatures Competition. The designers of the 64-bit permutation Alzette [5] also studied the linearization of the ARX-based component (Addition-Rotation-Xor) based on solving a system of quadratic equations.

There are several cryptanalysis results using linearization, in particular, low-data algebraic meet-in-the-middle attacks on LowMC [1], RAIN and AIM/AIMer [47]. Note that LowMC has a 3-bit S-box, RAIN employs the binary finite field inversion, and AIM uses monomials with exponents of the form $2^e - 1$ for some e. The authors of [47] use a decomposition of the inverse function as $x^{2^n-2} = (x^{2^{n/2}+1})^{2^{n/2}-2} \cdot x^{2^{n/2}}$ which is only possible for an even field dimension n, and they suggest that using an odd n would prevent their attack. It is thus a very natural question of what is the best linearization of the inverse function on odd numbers of bits. More generally, understanding linearization possibilities for monomial functions is crucially relevant for arithmetization-oriented (AO) primitives. As a further example, the authors of [47] linearized monomial functions of AIM-III with only $45/2^{192}$ elements, which seems very low for such a large field (although here a linearization must hold for two monomial functions at the same time). Linearization is also a key component in the cryptanalysis of hash functions such as Keccak [41,25] or AES-based hashing [17], where sometimes an additional constraint requires that the restricted domain is an affine space rather than an arbitrary set. In a nutshell, the attacker can trade some available degrees of freedom in order to propagate through the S-box in a way that the input and output variables are linearly related or constant.

Finally, we note that the constrained-input-constrained-output (CICO) problem for permutations, underlying the security of sponge-based hash functions and other primitives, is very similar in spirit to the single-plaintext setting, and linearization is also a very useful tool, as we will show in Section 7. The CICO setting is also the main one used for cryptanalysis of AO hash functions.

In this work, our aim is to advance the study of vectorial linearity by developing algorithms for finding vectorial affine approximations of S-boxes in practice, as well as a theoretical framework to study such approximations in terms of the difference or correlation tables of the S-box.

Our contribution Our main contribution is the development of powerful algorithms for finding strong linearizations of S-boxes in practice. For up to dimension 8, we are able to identify *provably optimal* affine approximations, and we offer high-quality heuristic results for larger sizes.

The algorithms[1] are summarized in Table 1. The "extension" algorithms are closely related to the difference distribution table of the S-box (DDT), while the "reduction" algorithms are closely related to the linear approximation table of the S-box (LAT). We develop natural greedy and exhaustive versions of each flavor for completeness, although in practice, the most useful ones are Greedy-Extension and Exhaustive-Reduction. As a by-product, our technique allows one to slightly improve the general lower bound on vectorial linearity from $n+1$ (due to Carlet [13]) to $n+2$, which was posed as an open problem at the NSUCRYPTO 2021 [20] international cryptographic olympiad.

We apply the algorithms to a variety of S-boxes: monomial functions on $\mathbb{F}_{2^n}$, S-boxes designed for block or stream ciphers, APN functions, and several 16-bit Super-Sboxes. We find that the 8-bit S-box of SKINNY has an affine approximation valid on 47 out of 256 inputs, while for most other S-boxes this value is in the range of 18-30, closer to 20-22 for random 8-bit S-boxes. We also show that the AES S-box has no better approximations than the one based on the decomposition from [47], with 18 points. We observe a wide range of vectorial linearity[2] among 16-bit Super-Sboxes, ranging from 49 (MISTY-FI function [32]), and around 110 (LED [26]) to around 1100 (SKINNY64 [6]) and 4096 (Midori64 [2]). We investigate the two latter cases and uncover the reasons for such a weakness. We also ran a simplified variant of the Greedy-Reduction algorithm on the 32-bit AES Super-Sbox (with the zero key) and obtained an approximation of size 60. We leave it as a challenge to improve this difficult case.

[1] All the implementations and results from the paper will be made publicly available at [10] and https://github.com/cryptolu/sbox_linearization.

[2] For 16-bit S-boxes our results are only heuristic but we believe they are close to optimal.

Finally, we describe new methods for covering an S-box with several approximations, which is necessary to increase the success rate of linearization attacks on primitives. We then apply the method to a generic CICO problem over SPN with general linear layers and derive bounds on the vectorial linearity of S-boxes that allow for linearization attacks for varying numbers of rounds.

Table 1. Summary of our algorithms applied to an S-box $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$. For exhaustive variants, the complexity estimates assume that $g = \mathsf{VecLin}_S \geq 2^{n/2}$ and $k = \lfloor \frac{n}{2} \rfloor + 2$. Optimizations for special cases are not included.

Algorithm	Ref.	Nature	Complexity (Per sample)	Complexity (Total)
Brute-force	Sec. 3.1	Exhaustive	$\mathcal{O}(2^n)$	$\mathcal{O}(\frac{2^{n(n+2)}}{(n+1)!})$
Sampling	Sec. 3.1	Heuristic	$\mathcal{O}(2^n)$	$\mathcal{O}(2^{(n+1)(n+1-\log_2 g)})$
Extension-Exhaustive	App. B	Exhaustive	–	$\mathcal{O}(\frac{2^{nk}}{k!})$
Extension-Greedy	Sec. 4.3	Heuristic	$\mathcal{O}(2^n)$	–
Reduction-Exhaustive	Sec. 5.1	Exhaustive	–	$\mathcal{O}(2^{(n+1)(k+0.5)})$
Reduction-Greedy	Sec. 5.2	Heuristic	$\mathcal{O}(2^{n+m})$	–

2 Preliminaries

The *weight* of a map $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, denoted $\mathrm{wt}(S)$, is defined as the number of inputs $x \in \mathbb{F}_2^n$ for which $S(x) \neq 0$. The inner product of two vectors x, y of length n is denoted by $\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$.

Linear approximations The *Walsh transform* of a Boolean function $s\colon \mathbb{F}_2^n \to \mathbb{F}_2$ is the map $W_s\colon \mathbb{F}_2^n \to \mathbb{Z}$ given by $W_s(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x \rangle + s(x)}$. The *Walsh transform* (also known as the *linear approximation table* – LAT) of a vectorial Boolean function $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ is the map $W_S\colon \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{Z}$ given by

$$W_S(u, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle u,x \rangle + \langle v, S(x) \rangle}.$$

The (Boolean) *linearity* of a (vectorial) function S, denoted Lin_S, is the maximum absolute value of W_S excluding $W_S(0, 0) = 2^n$. The Walsh transform can be computed efficiently in time $\mathcal{O}(n2^n)$ for a Boolean function (implying $\mathcal{O}((n + m)2^{n+m})$ for a vectorial Boolean function) using the so-called Fast Walsh-Hadamard Transform (FWHT), which is essentially the multi-dimensional variant of the Fast Fourier Transform.

Algebraic degree Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$. The *algebraic normal form* (ANF) of the function f is the unique representation of f as a multivariate polynomial in $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$. For a vectorial function $F\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, the same representation may be achieved in each of its m coordinates. The *algebraic degree* $\deg F$ of $F = (f_1, \ldots, f_m)$ is defined as the maximum degree among all degrees $\deg(f_i)$ for $i \in \{1, \ldots, m\}$. For $f = 0$, the degree is set to $-\infty$.

Let $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$. For any fixed $\Delta \in \mathbb{F}_2^n$, define the *derivative* of S in the direction of Δ as $\mathbf{D}_\Delta S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, $x \mapsto S(x + \Delta) + S(x)$. We allow the trivial derivative $\mathbf{D}_0(S)$ which is the zero function. For any nonzero function S, taking a derivative strictly decreases the algebraic degree: $\deg \mathbf{D}_\Delta S \leq \deg S - 1$.

The finite field $\mathbb{F}_{2^n}$ can be represented by the polynomials in $\mathbb{F}_2[x]/\langle Q(x) \rangle$, where $Q(x) \in \mathbb{F}_2[x]$ is an irreducible polynomial of degree n. Every function $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ has a unique equivalent representation as a univariate polynomial $P(x)$ in $\mathbb{F}_{2^n}[x]/\langle x^{2^n} - x \rangle$, i.e., of degree at most $2^n - 1$. The algebraic degree of S is equal to the maximum Hamming weight of exponents present in $P(x)$.

Affine Subspaces An *affine subspace* A of a vector space V is a coset of a linear subspace U of V: $A = U + a$ for $U \subseteq V$ and a constant $a \in V$. The *dimension* of A is defined as the dimension of U. Two affine subspaces are said to be *parallel* if they are cosets of the same linear subspace.

An *affine combination* of vectors $a_0, \ldots, a_n$ is a linear combination whose coefficients sum to 1. The set of all affine combinations of $a_0, \ldots, a_n$ is called the *affine span* of $a_0, \ldots, a_n$, also written as $\langle a_0, \ldots, a_n \rangle_{\mathrm{aff}}$; it

is an affine subspace A. We say that these vectors are *affinely independent* if the vectors $a_1 - a_0, \ldots, a_n - a_0$ are linearly independent, in which case these vectors form an *affine basis* of A. An affine basis of a d-dimensional affine subspace consists of $d+1$ vectors.

The number of ordered affine bases of $\mathbb{F}_2^n$ is $\mathcal{N}_n := 2^n \prod_{i=0}^{n-1}(2^n - 2^i) \approx 0.29 \cdot 2^{n(n+1)}$ whereas the number of unordered affine bases of $\mathbb{F}_2^n$ is $\mathcal{N}_n/(n+1)!$.

An *affine map* F is a map of the form $F(x) = L(x) + a$ for a linear map L and a constant vector a. Affine maps preserve affine combinations:

$$F(\lambda_0 a_0 + \cdots + \lambda_n a_n) = \lambda_0 F(a_0) + \cdots + \lambda_n F(a_n)$$

for all vectors $a_0, \ldots, a_n$ and coefficients $\lambda_0 + \ldots + \lambda_n = 1$.

Let $X = \{x_0, \ldots, x_n\}$ be a set of vectors. We define the *affine rank* of X as the maximum number of affinely independent vectors in it. This coincides with the dimension of the (linear) span of the *extended* vectors $(1, x_0), \ldots, (1, x_n)$.

Zero-sum sets

Definition 1. *Let $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$. The* difference distribution table *$DDT_S\colon \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{Z}_{\geq 0}$ of S is given by*

$$DDT_S(\alpha, \beta) = |\{x \in \mathbb{F}_2^n \mid S(x) + S(x+\alpha) = \beta\}|.$$

The function S is said to be differentially δ_S-uniform where δ_S is the maximum value of DDT_S excluding $DDT_S(0,0) = 2^n$.

In our extension algorithms, we will use a variant of the DDT which records the actual solutions instead of their number.

Definition 2 (Zero-sums). *Let k be a positive even integer and $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ an arbitrary map. We say that k pairwise distinct points $p_1, \ldots, p_k \in \mathbb{F}_2^n$ such that $p_1 + \cdots + p_k = 0$ and $S(p_1) + \cdots + S(p_k) = 0$ form a* k-zero-sum *of S. We call the dimension of the affine span of $\{p_1, \ldots, p_k\}$ its* input dimension *and the dimension of the affine span of $\{(p_1, S(p_1)), \ldots, (p_k, S(p_k))\}$ its* full dimension*.*

The smallest (even) k for which a k-zero-sum may exist is 4, because we consider distinct inputs and two distinct elements in characteristic 2 cannot sum to zero. See also a related notion [14, Def. 1].

Proposition 1. *Let k be a positive even integer and $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$. For every k-zero-sum of S formed by pairwise distinct elements $p_1, \ldots, p_k \in \mathbb{F}_2^n$, we have*

$$\lceil \log_2(k) \rceil \leq \textit{input dimension} \leq \textit{full dimension} \leq k - 2.$$

For $k \in \{4, 6\}$, every k-zero-sum has input dimension $k-2$, and thus, also full dimension $k-2$.

Proof. The second inequality is obvious. Regarding the first inequality, we can have k pairwise distinct points only in an affine subspace of dimension at least $\lceil \log_2(k) \rceil$. Regarding the last inequality, at most $k-1$ of k points that sum to zero are affinely independent (because the sum constitutes an affine dependence), which means that the respective affine span has dimension at most $k-2$.

For $k = 4$, we have $\lceil \log_2(k) \rceil = 2 = k - 2$, and thus every 4-zero sum has both input and full dimension 2.

For $k = 6$, we have $\lceil \log_2(k) \rceil = 3 < 4 = k - 2$. If the dimension of the affine span of $p_1, \ldots, p_6$ were 3, then four of the six points, say $p_1, \ldots, p_4$, would be affinely independent, which would mean that p_5 could be written as an affine combination of three of the points $p_1, \ldots, p_4$, say $p_5 = p_1 + p_2 + p_3$, and thus, we would have $p_1 + p_2 + p_3 + p_5 = p_1 + \cdots + p_6 = 0$, and hence $p_4 + p_6 = 0$, i.e., $p_4 = p_6$, a contradiction to the property that the involved points must be pairwise distinct. Thus, every 6-zero-sum has both input and (thus) full dimension 4. □

Definition 3 ([38]). *A map $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ is called* almost perfect nonlinear *(*APN*) if S has no 4-zero-sums. Equivalently, its differential uniformity δ_S is 2.*

3 Linearization

Definitions We start by defining an approximation of an S-box. Throughout this work, we will only consider affine approximations.

Definition 4 (Intersection set, g-approximation). *For $S, A\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, we define the* intersection set *of S and A to be the set*

$$S \cap A = \{x \in \mathbb{F}_2^n \mid S(x) = A(x)\}.$$

*Furthermore, we say that A is a g-*approximation *of S if $|S \cap A| = g$.*

The next definition is one of the key concepts in our study. We remark that previous works defined and studied vectorial *nonlinearity*, which most of the time takes values of the form $2^n - g$ for a small g (for functions with n-bit inputs). To simplify the expressions, we instead focus on the complementary notion of vectorial *linearity*, which is simply equal to g in the expression above.

Definition 5 (Vectorial linearity). *For $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, we define its* vectorial linearity VecLin_S *as the maximum size of a subset $X \subseteq \mathbb{F}_2^n$ such that the restriction $S|_X$ can be extended to an affine map $\mathbb{F}_2^n \to \mathbb{F}_2^m$ (or, equivalently, as the maximum g such that there exists an affine g-approximation of S):*

$$\begin{aligned}\mathsf{VecLin}_S &= \max\{|X| \mid X \subseteq \mathbb{F}_2^n, S|_X = A|_X, A\colon \mathbb{F}_2^n \to \mathbb{F}_2^m \text{ affine}\}\\ &= \max\{|S \cap A| \mid A\colon \mathbb{F}_2^n \to \mathbb{F}_2^m \text{ affine}\}\\ &= 2^n - \min\{\mathrm{wt}(S + A) \mid A\colon \mathbb{F}_2^n \to \mathbb{F}_2^m \text{ affine}\}.\end{aligned}$$

The key problem we consider in this work is *linearization*, which consists in finding an affine g-approximation with g as large as possible.

Problem 1 (Linearization). Given $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, find an affine g-approximation of S with maximal g.

Bounds The following trivial lower bound was given by Carlet.

Proposition 2 ([13, Prop. 6]). *Let $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$. Then, $\mathsf{VecLin}_S \geq n + 1$.*

Proof. Choose an arbitrary affine basis $\mathcal{B}$ of $\mathbb{F}_2^n$ consisting of $n + 1$ points, on which an affine map is already fully determined by choosing an output value for each basis point. By choosing the output values according to the map S, we obtain an affine map A such that $S \cap A \supseteq \mathcal{B}$ and thus $|S \cap A| \geq n + 1$. □

An interesting nontrivial bound based on the differential uniformity of the S-box was achieved by Nagy [35] and Ryabov [42] independently.

Proposition 3. *Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with differential uniformity δ_S. Then,*

$$\mathsf{VecLin}_S \leq \sqrt{\delta_S} \cdot 2^{n/2} + \frac{1}{2}.$$

A similar (yet much weaker) bound exists for the Boolean linearity (see e.g. [30]).

Proposition 4. *Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with Boolean linearity Lin_S. Then,*

$$\mathsf{VecLin}_S \leq 2^{n-1} + \frac{1}{2}\mathsf{Lin}_S.$$

More specific bounds for APN and Bent functions are derived e.g. in [34].

Properties Solutions to Problem 1 can easily be transferred to extended-affine-equivalent functions.

Proposition 5. *Let $S, S' \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ be such that $S' = B \circ S \circ A + C$ for some affine bijections $A \colon \mathbb{F}_2^n \to \mathbb{F}_2^n, B \colon \mathbb{F}_2^m \to \mathbb{F}_2^m$ and an affine map $C \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$. Then, $\mathsf{VecLin}_S = \mathsf{VecLin}_{S'}$.*

Proof. *Let $g = \mathsf{VecLin}_S$ and $g' = \mathsf{VecLin}_{S'}$, and let D be an affine g-approximation of S. Then the map $D' \coloneqq B \circ D \circ A + B$ is also affine and a $\tilde{g}$-approximation of S' with $\tilde{g} \geq g$ because for all $x \in S \cap D$, we have $S'(x) = B(S(A(x))) + C(x) = B(D(A(x))) + C(x) = D'(x)$ and thus $g' \geq \tilde{g} = |S' \cap D'| \geq g$. Conversely, we can write $S = B^{-1} \circ (S' - C) \circ A^{-1} = B^{-1} \circ S' \circ A^{-1} - B^{-1} \circ C \circ A^{-1}$ and apply the same argument as before by replacing the roles of A, B, C by A^{-1}, B^{-1}, $-B^{-1} \circ C \circ A^{-1}$ (which is affine), respectively, and thus we get $g \geq g'$.* □

Remark 1. Note that if S is bijective, then $\mathsf{VecLin}_S = \mathsf{VecLin}_{S^{-1}}$ is false in general. For example, for $n = 4$ and $S = [3, 7, 4, 8, 6, 11, 0, 5, 13, 10, 9, 1, 12, 2, 14, 15]$, we get $\mathsf{VecLin}_S = 7$ and $\mathsf{VecLin}_{S^{-1}} = 8$. This resolves [34, Problem 2].

The authors of [18] observed that for quadratic S-boxes, any approximation may be translated at the input by any constant, which follows from the fact that the derivatives of a quadratic function are affine. We derive a concrete expression since we will use it in specific optimizations.

Theorem 1 ([18, Thm. 3]). *Let $S \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ have algebraic degree at most 2 and let A be an affine g-approximation of S with $X = S \cap A$. Then, for any $\Delta \in \mathbb{F}_2^n$, the map $A'(x) \coloneqq A(x + \Delta) + \mathbf{D}_\Delta S(x)$ is also an affine g-approximation of S with $S \cap A' = X + \Delta$.*

Proof. Since the algebraic degree of S is at most 2, the map $\mathbf{D}_\Delta S$ has algebraic degree at most 1, i.e., it is affine, and thus the map A' is also affine. For $x \in \mathbb{F}_2^n$,

$$A'(x + \Delta) = A(x) + \mathbf{D}_\Delta S(x + \Delta) = A(x) + S(x) + S(x + \Delta),$$

which is equivalent to $A'(x + \Delta) + S(x + \Delta) = A(x) + S(x)$. This implies $x \in S \cap A \iff x + \Delta \in S \cap A'$, and thus $S \cap A' = S \cap A + \Delta = X + \Delta$. □

3.1 Basic algorithms

Basis brute-force algorithm A simple algorithm that theoretically solves Problem 1 is to interpolate possible affine approximations from the S-box values on each affine basis (unordered). The process is as follows: go through all (unordered) affine bases of $\mathbb{F}_2^n$; define in each step an affine map according to the output values of S on the current affine basis; check in how many points the defined affine map coincides with S. The number of unordered bases is given by $\mathcal{O}\left(\frac{2^{n(n+1)}}{(n+1)!}\right)$, which is cheaper than enumerating all $2^{m(n+1)}$ affine maps for $m \geq n$. It costs $\mathcal{O}(2^n)$ operations to count the number of points in the intersection of the interpolated affine map and the given S-box. This leads to the total complexity of $T_n = \mathcal{O}\left(\frac{2^{n(n+2)}}{(n+1)!}\right)$ operations on vectors. Clearly, for $n \geq 7$, this trivial algorithm becomes infeasible on a normal computer, as it requires more than $2^{45.9}$ operations ($2^{59.7}$ for $n = 8$).

Probabilistic sampling version A more useful variant of the approach consists in *sampling* a random affine basis $B \subseteq \mathbb{F}_2^n$, interpolating the affine map A agreeing with S on B, and computing $S \cap A$. Despite its simplicity, it is a powerful method that is especially efficient in the presence of strong approximations. A pseudocode for an iteration is listed in Algorithm 1. In practice, one may run the algorithm for a fixed number of iterations and keep the best found approximations.

Success probability and complexity Let $g = \mathsf{VecLin}_S$ be achieved by an affine approximation A with $X = S \cap A$. Then, the probability of sampling a basis candidate within X is given by (note that random $n+1$ points form an affine basis with probability close to 29%, for simplicity we assume similar behavior within X)

$$\frac{\binom{g}{n+1}}{\binom{2^n}{n+1}} \approx \left(\frac{g}{2^n}\right)^{n+1} = \left(2^{n+1}\right)^{\log_2 g - n}.$$

Algorithm 1 Sampling algorithm

Inputs: function $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$
Output: affine approximation $A\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ of S

1: $B \leftarrow$ a random affine basis of $\mathbb{F}_2^n$
2: $A \leftarrow$ an affine map defined by $A(x) = S(x)$ for all $x \in B$
3: $g \leftarrow |\{x \in \mathbb{F}_2^n \mid A(x) = S(x)\}|$
4: **return** A is a g-approximation of S

For an n-to-n S-box, a general affine map is defined by $(n+1)n$ bits. Therefore, the sampling method yields the factor $1-\log_2 g/n$ of complexity reduction (in bits) compared to direct brute-force. For example, for $n = 8$ and $g = 32$, the estimated complexity is equal to $2^{(n+1)(n-\log_2 g)} = 2^{27}$, while for $g = 16$ it increases to 2^{36}. For the case of vectorial linearity is close to $2^{n/2}$, we get the complexity estimate of $2^{(n+1)n/2}$ samples. Note however that, for large-output S-boxes, the vectorial linearity may be equal to $n+1$, which results in sampling complexity close to quadratic in the exponent: $2^{(n+1)(n-\log_2(n+1))}$. The total time complexity has to be multiplied by the cost $\mathcal{O}(2^n)$ of testing a sample (Step 2 above).

4 Extension algorithms

We will now describe the first class of our algorithms, based on the *extension* paradigm. The high-level idea is to gradually extend the definition of the affine approximation A on an increasing sequence of subspaces, until it is fully defined. As we will show, this approach has close connections to the DDT.

4.1 Relations to zero-sums

Given $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, we know $\mathsf{VecLin}_S \geq n+1$ by Proposition 2. By carrying out the procedure described in the corresponding proof, be it only mentally or actually in practice, we can hope to catch points outside our chosen affine basis for which A and S coincide as well. Let $\mathcal{B} = \{x_0, \ldots, x_n\}$ be a chosen affine basis of $\mathbb{F}_2^n$ and $p \in \mathbb{F}_2^n \setminus \mathcal{B}$ such that $A(p) = S(p)$. Expressing p as an affine combination of the basis vectors x_i: $p = x_{i_1} + \cdots + x_{i_j}$ with odd j, we obtain

$$S(p) = A(p) = A(x_{i_1} + \cdots + x_{i_j}) = A(x_{i_1}) + \cdots + A(x_{i_j}) = S(x_{i_1}) + \cdots + S(x_{i_j}).$$

It follows that $\{p, x_{i_1}, \ldots, x_{i_j}\}$ is a $(j+1)$-zero-sum of S. If $j \in \{3, 5\}$, then, by Proposition 1, any j of the involved $j+1$ points are affinely independent and may constitute a part of the basis.

We can use this observation in the reverse direction: first find a k-zero-sum with $k \in \{4, 6\}$, choose any $(k-1)$-element affine independent subset, and complete the basis arbitrarily. The equation above shows that the excluded k-th point of the zero-sum must satisfy the final affine approximation interpolated on the basis. The following proposition ensures the existence of at least one 4- or 6-zero-sum and thus of one extra point for the case $n \geq 4$ and $m \leq 2n-3$, based on a simple counting argument showing existence of 6-zero-sums for these parameters. The proof is deferred to Section A.

Proposition 6. *Let* $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, $n \geq 4$ *and* $m \leq 2n-3$. *Then,* $\mathsf{VecLin}_S \geq n+2$.

Since we are interested in collecting more than one extra point for the approximations, we will generalize this idea and encompass it into an iterative algorithm collecting extra points at each step. This replaces the ineffective arbitrary basis completion used above.

4.2 General extension algorithm

Following the discussion above, the idea is to gradually build up an affine basis $\mathcal{B} \subseteq \mathbb{F}_2^n$ of the domain, picking up as many "free" points per each dimension as possible.

Initial setup The idea is to start with a d-dimensional affine subspace $X_d \subset \mathbb{F}_2^n$ and an affine map $A_d\colon X_d \to \mathbb{F}_2^m$. For example, we can take as X_d the affine subspace generated by points (in $\mathbb{F}_2^n$) forming a 4-zero-sum ($d = 2$) or a 6-zero-sum ($d = 4$) of S, or simply a singleton ($d = 0$).

Extending the approximation There are $2^{n-d}-1$ another affine subspaces parallel to X_d and we will merge X_d with one of them (name it $a+X_d$) to obtain a $(d+1)$-dimensional affine subspace $X_{d+1} = X_d \sqcup (a+X_d)$. The next step is to extend the definition of the affine map $A_d : X_d \to \mathbb{F}_2^m$ to $A_{d+1} : X_{d+1} \to \mathbb{F}_2^m$ with $A_{d+1}|_{X_d} = A_d$. In order to gain a new point for the approximation, we need to have $A_{d+1}(x_d) = S(x_d)$ for some point in $X_{d+1} \setminus X_d$, i.e., in the chosen parallel affine subspace. It is easy to see that such x_d can be chosen arbitrarily in $a + X_d$. Furthermore, it completely defines the extension A_{d+1} since $A_{d+1}(x')$ for any $x' \in (a + X_d)$ can be deduced from

$$A_{d+1}(x_d) + A_{d+1}(x') = A_{d+1}(x_d - a) + A_{d+1}(x' - a) = A_d(x_d - a) + A_d(x' - a),$$

which is well-defined since both $x_d - a$ and $x' - a$ belong to X_d.

Compatibility of points Observe that x_d by design satisfies the extended approximation with respect to S. What determines the other such points in $a + X_d$? The explicit condition is

$$S(x') \stackrel{?}{=} A_{d+1}(x') = A_d(x_d - a) + A_d(x' - a) + S(x_d).$$

The key observation is that it is symmetric in x' and x_d, and, furthermore, induces a partition of $a + X_d$ into classes of points simultaneously fitting some extension A_{d+1} of the affine map A_d, meaning that these points are jointly *compatible* for an extension of A_d.

Lemma 1. *Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$, $d < n$ and X_d be a d-dimensional affine subspace $X_d \subseteq \mathbb{F}_2^n$, on which an affine map $A_d : X_d \to \mathbb{F}_2^m$ is defined. Let $a \in \mathbb{F}_2^n$ such that $X_d + a \neq X_d$. Then, the following is an equivalence relation on $a + X_d$:*

$$p \sim q \iff S(p) + S(q) = A_d(p - a) + A_d(q - a).$$

Furthermore, for any affine map $A_{d+1} : X_d \cup (a+X_d) \to \mathbb{F}_2^m$, $A_{d+1}|_{X_d} = A_d$, if $p \sim q$, then $A_{d+1}(p) = S(p)$ if and only if $A_{d+1}(q) = S(q)$.

Proof. The $\sim$ condition is equivalent to $f(p) = f(q)$ for $f(x) = S(x) - A_d(x - a)$, which is clearly an equivalence relation describing level sets of f. Note that f is well defined since $(p-a), (q-a) \in X_d$ where A_d is defined. From $p \sim q$ we obtain

$$S(p) + S(q) = A_d(p - a) + A_d(q - a) = A_{d+1}(p - a) + A_{d+1}(q - a) = A_{d+1}(p) + A_{d+1}(q)$$

due to A_{d+1} being an affine extension of A_d, which implies $S(p) + A_{d+1}(p) = S(q) + A_{d+1}(q)$ and both sides can turn to zero only simultaneously. □

If we imagine the points of $\mathbb{F}_2^n \setminus X_d$ graphically and draw a line between two points p and q of the same coset exactly if $p \sim q$, we see that an equivalence class is nothing else than a *clique* (a full subgraph, meaning that every pair of points in the equivalence class is connected by a line). From now on, we will refer to it also as a clique. In this sense, each coset is partitioned into cliques. This means, according to Lemma 1, that for each clique c, there exists (in fact, a unique) affine map A_{d+1} which extends A_d onto the coset that contains c such that A_{d+1} and S coincide for all points of c, besides the points where A_d and S already coincide.

Choosing the extension As mentioned before, we can choose an arbitrary coset $a + X_d$ and an arbitrary point inside it, meaning an arbitrary point x outside of X_d. Assigning $A_{d+1}(x) = S(x)$ extends affinely the definition of A_d. By the explicit condition above and its expression in the lemma, the full clique (i.e., the equivalence class of x) now belongs to the intersection $S \cap A_{d+1}$. It follows that, in particular, we may greedily choose the largest available clique at each step (this will be explored in Subsection 4.3). Alternatively, we can recursively try a merge with every possible clique (this is explored in Section B).

Completing the extension process Finally, we discuss how to update the cliques information after merging the cosets X_{d+1} and $a + X_{d+1}$ and extending the definition of A_d to A_{d+1}. Observe that, once we have $p \sim q$ in an extension step $X_d \subset X_{d+1}$ and $p, q \notin X_{d+1}$, we also have $p \sim q$ in the extension step $X_{d+1} \subset X_{d+2}$ because A_{d+1} is an extension of A_d. Thus, it is sufficient to check the edge condition according to Lemma 1 only for the new differences that arose during the extension $X_d \subset X_{d+1}$.

A practical and elegant way to achieve this is to precompute the differences $\Delta_x := p + q$ and $\Delta_y := S(p) + S(q)$ for each pair $(p, q) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ with $p \neq q$ and to create a lookup table by placing (p, q) at position (Δ_x, Δ_y). This resulting table extends the usual DDT by storing the solution pairs of each corresponding differential equation, instead of their count. A small memory optimization consists in storing only pairs (p, q) with $p < q$.

Now, assume the table is precomputed, A_d is defined (for some $d > 0$) and we are merging X_d with $a + X_d$, so that A_{d+1} is defined on both of them. Let arbitrary $b \in X_d$ and define $L_d = \{x - b \mid x \in X_d\}$ be the linear subspace associated to X_d. The new equivalence edges may only appear between affine subspaces $b + X_d$ and $b + a + X_d$ for some b, connecting a pair $(b, b + a + \Delta)$ of inputs for some $\Delta \in L_d$. Therefore, we are looking for values b such that

$$\Delta_y = A_{d+1}(b + \epsilon) + A_{d+1}(b + a + \Delta + \epsilon) = S(b) + S(b + a + \Delta).$$

This corresponds to the DDT entry with $\Delta_x = a + \Delta$ and $\Delta_y = A_{d+1}(0) + A_{d+1}(a + \Delta)$ is derived from the expression above and A_{d+1}. Here, $\epsilon \in \mathbb{F}_2^n$ is an arbitrary vector required to reach the domain of A_{d+1}. By enumerating the DDT entry solutions for all 2^d values of such Δ_x, we obtain precisely all the edges $(b, b + \Delta_x)$ to be added to the relational graph.

Exhaustive variant We provide an exhaustive variant of the extension algorithm in Section B. Note that it is not used in our experiments and has similar theoretical complexity to the exhaustive reduction algorithm from Subsection 5.1. In the following subsection, we present the greedy extension algorithm, which is based on maximizing the size of the merged clique locally at each step. It is our fastest method and the most useful: nearly all of our experimental results were obtained using this variant.

4.3 Greedy variant

We now present a greedy extension algorithm. At each of the $n + 1$ steps, we add another point to the approximation by choosing it greedily and randomly among points for which the size of the new approximation is maximized. The algorithm's pseudocode is given in Algorithm 2.

Algorithm 2 Extension algorithm (greedy)

Inputs: function $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, parameter $H \in \mathbb{Z}_{>0}$
Output: affine approximation $A\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ of S

$\mathrm{DDT}_{\mathrm{sol}} \leftarrow \{(\Delta_x, \Delta_y) \mapsto \{x \mid S(x) + S(x + \Delta_x) = \Delta_y\}\}$ ▷ DDT with solutions
$\mathsf{LS}_x = \{0\} \subseteq \mathbb{F}_2^n$ ▷ Linear space of guessed input differences
$\mathsf{LS}_{xy} = \{(0, 0)\} \subseteq \mathbb{F}_2^n \times \mathbb{F}_2^m$ ▷ Linear space of guessed input and output differences
$a \xleftarrow{\$} \mathbb{F}_2^n$ ▷ Main clique representative
$\mathsf{cliques} \leftarrow \{\{x\} : x \in \mathbb{F}_2^n\}$ ▷ DSU data structure
$\mathsf{top} \leftarrow$ min-heap initialized with H random elements of $\{1\} \times \mathbb{F}_2^n \setminus \{a\}$
for $d \in \{0, \ldots, n - 1\}$ **do**
    $a' \leftarrow$ largest element of top (via iteration)
    $(\Delta_x, \Delta_y) \leftarrow (a + a', S(a) + S(a'))$
    $\mathsf{LS}_x \leftarrow \mathsf{LS}_x \cup (\Delta_x + \mathsf{LS}_x)$
    Remove pairs (s, x) from top with $(a + x) \in \mathsf{LS}_x$ (if any)
    for all $(\tilde{\Delta}_x, \tilde{\Delta}_y) \in \big((\Delta_x, \Delta_y) + \mathsf{LS}_{xy}\big)$ **do**
        for all $x \in \mathrm{DDT}_{\mathrm{sol}}[\tilde{\Delta}_x, \tilde{\Delta}_y]$ **do**
            Try to merge cliques containing x and $x + \tilde{\Delta}_x$ in $\mathsf{cliques}$
            if merge occurred and $(a + x) \notin \mathsf{LS}_x$ **then** ▷ No more merges inside the current coset
                Insert the pair (size, representative x) of the new clique into top
                If top has size $H + 1$, remove the smallest element from it
            end if
        end for
    end for
    $\mathsf{LS}_{xy} \leftarrow \mathsf{LS}_{xy} \cup ((\Delta_x, \Delta_y) + \mathsf{LS}_{xy})$
end for
$A \leftarrow$ affine map interpolated from the largest clique (or from its graph LS_{xy} and $S(a) = A(a)$)
return A

Implementation details We use the disjoint set union structure (DSU) to maintain and merge cliques. It has amortized cost close to constant for searches and merges. To keep track of the largest cliques, we maintain a min-heap data structure containing H largest available cliques for a small fixed parameter H, e.g. $H = 8$. The min-heap property allows us to efficiently replace the smallest clique with a bigger one. At the beginning of every iteration, we iterate through the full heap to find the largest clique outside of the main coset, and remove those that lie in the main coset.

We maintain the sets $\mathsf{LS}_x \subseteq \mathbb{F}_2^n$, $\mathsf{LS}_{xy} \subseteq \mathbb{F}_2^n \times \mathbb{F}_2^m$ corresponding to the linear subspaces $X_d - X_d$ and $Y_d - Y_d$, where $Y_d = \{(x, A_d(x)) \mid x \in X_d\}$. The former one is used for testing if two inputs are in the same coset; the latter one is used to enumerate the graph of A_d, which allows inferring all possible pairs (Δ_x, Δ_y) in the discussion from the previous subsection.

For starting with a given 4- or 6-zero-sum, one can force the initial choice of a and the choices of a' in the first 2 / 3 iterations.

Complexity analysis The loop $v \in \mathsf{LS}_{xy}$ dominates the complexity of the algorithm. It iterates over 2^d elements in iteration d. For each element v, the algorithm goes through all solutions to the corresponding difference equation, which are located in $\mathrm{DDT}_{\mathrm{sol}}$ at position $(\tilde{\Delta}_x, \tilde{\Delta}_y)$. We use a simplifying assumption that the involved differences $\tilde{\Delta}_x, \tilde{\Delta}_y$ are random (the one corresponding to $(\tilde{\Delta}_x, \tilde{\Delta}_y) = (\Delta_x, \Delta_y)$ is special due to the fact that it connects the largest cliques, while the others are just shifts of it by a subspace). Then, on average we have 1 solution, for which we query the DSU structure (we assume the constant cost although in theory it is shown to be the inverse Ackermann function, which is less than 5 for any representable number), check an LS_x membership via a lookup and update the small heap in time $\mathcal{O}(\log H)$. We conclude with the complexity $\mathcal{O}(\log H \cdot (2^0 + 2^1 + \ldots + 2^{n-1})) = \mathcal{O}((\log H)2^n) = \mathcal{O}(2^n)$ since H is a small constant. This is complexity per execution of the algorithm (that is, per one sampled affine approximation). This is a very high performance since just reading an S-box would take this amount of computations (although, we require random accesses to the precomputed DDT). Its greedy heuristic makes it produce much stronger results than pure sampling (Subsection 3.1) in practice, at essentially the same cost.

5 Reduction algorithms

In this section, we present algorithms based on iterative *reduction* of the full input set down to an affine approximation set. These algorithms also have a spectral component, namely, they employ the analysis of the Walsh transform of the target function. We describe two variants of the algorithm: an exhaustive reduction and a greedy reduction.

The starting idea for this direction is that the vectorial linearity can be seen as a stronger version of the usual (Boolean) linearity: all the output bits of the function must be approximated simultaneously on the same set of inputs. In particular, a strong vectorial approximation should manifest itself in Boolean approximations. Indeed, if $S(x) = A(x)$ for g different inputs x, then (in particular) the first output bit of S can be approximated by $A_1(x)$ with at least g matching inputs. This creates a bias towards the approximation $S_1 = A_1$, which may become the best approximation for this bit, unless the remaining $2^n - g$ inputs are specifically canceling the bias, or another better approximation exists for the bit. Although this observation does not provide guarantees, it can still be used as a heuristic to guide the search.

5.1 Exhaustive variant

In this variant, we will recursively enumerate candidates for the affine maps $A_1, A_2, \ldots, A_m \colon \mathbb{F}_2^n \to \mathbb{F}_2$ approximating the coordinates $S_1, \ldots, S_m$ of S one by one. Each such candidate A_i is defined by a *mask* $\alpha_i \in \mathbb{F}_2^n$ and a constant $c \in \mathbb{F}_2$ such that $A_i(x) = \langle \alpha_i, x \rangle + c$. During the enumeration, we will maintain the subset $X \subseteq \mathbb{F}_2^n$ of inputs satisfying the current guesses of $A_1, \ldots, A_d$ at the recursion depth d. The subsets X at the final depth m correspond to the desired intersections $S \cap A$. The algorithm together with optimizations described below is presented in Algorithm 3.

Upper-bounding the vectorial linearity The first important optimization that we add to the process is cutting off search subtrees for which the subset X of satisfying inputs is smaller than the lower bound parameter LB given as input. This trick relies on the fact that the subset X only reduces with the

Algorithm 3 Exhaustive reduction algorithm

Inputs: function $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, lower bound $\mathsf{LB} \in \mathbb{Z}_+$, set of affine derivatives $\{\mathbf{D}_{\Delta_i} S\}_i$
Output: affine approximation(s) $A\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, with at least LB matching points

function ExhaustiveReduction(d, L, X)
  if $|X| < \mathsf{LB}$ **then**
    return ▷ Cut search branch
  end if
  if $d = m$ **then**
    Report an affine approximation A with $S \cap A = X$
    return
  end if
  for all $\alpha \in \mathbb{F}_2^n$ **do**
    $L' \leftarrow (L || \langle \alpha, x \rangle)\colon \mathbb{F}_2^n \to \mathbb{F}_2^{d+1}$ ▷ Extend by one output bit
    if L' is lex. smallest among $L' + \{\mathbf{D}_{\Delta_i} S\}_i$ **then** ▷ as matrices,
      ▷ ignoring the constant part
      $X'_0 \leftarrow \{x \in X \mid \langle \alpha, x \rangle + S_{d+1}(x) = 0\}$
      $X'_1 \leftarrow \{x \in X \mid \langle \alpha, x \rangle + S_{d+1}(x) = 1\}$
      ExhaustiveReduction($d+1, L', X'_0$)
      ExhaustiveReduction($d+1, L', X'_1$)
    end if
  end for

end function
ExhaustiveReduction($0, (), \mathbb{F}_2^n$)

recursion depth. As we shall see, for the reasonable bound $\mathsf{LB} = 2^{n/2}$, this amounts to roughly halving the search space in bits. If the algorithm finishes without finding any approximation with at least LB points, it follows that the vectorial linearity of the S-box is at most $\mathsf{LB} - 1$. That is why we say that the algorithm has the *upper-bounding property*, while it receives the *lower bound* as input.

In practice, one would first run fast heuristic sampling algorithms (sampling, greedy extension or greedy reduction) and record the maximal size g of observed approximations. Then, the exhaustive algorithm is run with the lower bound $\mathsf{LB} = g + 1$ in order to verify that the vectorial linearity of the S-box is either exactly g (if the exhaustive algorithm does not find an approximation) or higher (if the exhaustive algorithm finds an approximation). In case the latter execution is not feasible, the LB parameter may be increased so that the exhaustive algorithm at least yields an upper bound on the vectorial linearity.

Optimization for quadratic S-boxes Using Theorem 1, we can optimize the search by a factor up to 2^n. Recall that, if A is a g-approximation of S, then $A'(x) = A(x + \Delta) + \mathbf{D}_\Delta S(x)$ is also a g-approximation, for any nonzero Δ. This means that, up to a constant independent of x (note that $A(x + \Delta) = A(x) + A(\Delta) + A(0)$), it is sufficient to check one representative of all translations

$$\{A(x) + \mathbf{D}_\Delta S(x) \mid \Delta \in \mathbb{F}_2^n\},$$

together with all possible constant additions. We consider only the linear parts of the derivatives (which correspond to quadratic parts of the original function). If the linear space of all (linear parts of) derivatives has full dimension n, we can check one out of 2^n candidates for the linear part from the above set. However, in order to use this property during the exhaustive reduction algorithm, we need to recognize the representatives before going too deep into the recursion. For simplicity, we will assume that the S-box has a small number (typically, 1-3) of components which yield full dimension n of the linear parts of the derivatives. Then, we guess these components first in the recursion, and check if the current guess A' is lexicographically the smallest among the equivalent guesses $\{A' + \mathbf{D}_\Delta S\}_{\Delta \in \mathbb{F}_2^n}$ (only considering the linear part of the map), for depth $d < 3$. This allows to cut off the equivalent search subtrees early and yields a factor 2^n reduction to the overall search cost.

Unless the S-box has an extremely sparse set of quadratic monomials, we expect our assumption to hold (and it was in practice in all our experiments). For example, if the matrix describing the quadratic monomials of a coordinate has full rank n, it is sufficient to check this coordinate first. With two coordinates, this is even more likely to be true.

Complexity analysis Let d be the current recursion depth, i.e., we have guessed 2^d masks and constants ($n+1$ bits per level). If we only fix d masks, the 2^d assignments to the constants partition the input space $\mathbb{F}_2^n$. Therefore, each set X at the current level has an average size 2^{n-d}. As a result, we estimate that most of the search subtrees will be cut shortly after depth d satisfying $2^{n-d} < \mathsf{LB}$, equivalent to $d > n - \log_2 \mathsf{LB}$.

This leads to the complexity estimation $2^{(n+1)(\lceil \log_2 \mathsf{LB} \rceil + \varepsilon)}$ for a small integer ε. Focusing on the case when VecLin is close to $2^{n/2}$, we get the complexity $2^{(n+1)(n/2+\varepsilon)}$. Note that for even n, setting $\varepsilon = 0$ does not lead to sufficient filtering as the depth is right on the boundary, and therefore, we need $\varepsilon \geq 1$ to ensure strong filtering; for odd n, this is not necessary. We conclude with the complexity estimate (denoted T_{nodes}) $2^{(n+1)\lfloor n/2+1 \rfloor}$ recursion calls. This complexity is similar to that of the sampling algorithm (Algorithm 1); however, the new algorithm is guaranteed to exhaust the full search space and thus can be used to compute VecLin_S precisely or at least get an upper bound. Note that the complexity in terms of operations requires an extra factor of $2^{1.5n}$ for enumerating 2^{n+1} candidates masks and filtering the set of inputs of size about $2^{0.5n}$ at depth $d = n/2$, yielding total estimate of $\mathcal{O}(2^{(n+1)\lfloor n/2+2.5 \rfloor})$ operations.

For a more precise analysis, we model the partitioning into 2^d sets as fully random (which is plausible since most of the work is spent on random linear masks dividing the set X randomly). We also analyze each single bucket of the partition independently of others. For each input from the domain $\mathbb{F}_2^n$, the probability for it to land in the chosen bucket at depth d is equal to 2^{-d}. Modeling all the inputs independently, we use the binomial distribution to compute the distribution of the number of inputs in the chosen bucket, denoted $p_{d,i}$:

$$p_{d,i} = \Pr[|X| = i \text{ after depth } = d] = \binom{2^n}{i} (2^{-d})^i (1-2^{-d})^{2^n - i}.$$

The next 2^{n+1} candidates in each recursive call are computed explicitly before filtering them by LB, adding an extra 2^{n+1} factor compared to the number of candidates after filtering. This gives an accumulated complexity T_d at depth d:

$$\begin{aligned} T_d &= 2^{(n+1)(d+1)} \cdot \Pr[|X| \geq \text{LB after depth } = d] \\ &= 2^{(n+1)(d+1)} \cdot \sum_{i=\text{LB}}^{2^n} \binom{2^n}{i} (2^{-d})^i (1-2^{-d})^{2^n - i}. \end{aligned}$$

The total complexity is then estimated as the sum of accumulated per-round complexity. For simplicity, we measure it in the number of visited nodes (denoted T_{nodes}). This ignores the different average X sets sizes at different depths, which we set to $2^{n/2}$ to get an approximation for the complexity in terms of word operations (denoted $T = T_{\text{nodes}} 2^{n/2}$). We conclude with the expression

$$T_{\text{nodes}} = \sum_{d=1}^{n} 2^{(n+1)(d+1)} \cdot \sum_{i=\text{LB}}^{2^n} \binom{2^n}{i} (2^{-d})^i (1-2^{-d})^{2^n - i}.$$

While the earlier estimate $2^{(n+1)\lfloor n/2+1 \rfloor}$ can be quickly used to estimate the feasibility of the search, this more precise expression needs to be computed on a computer (note that it is independent of the S-box). We found that it matches quite well the experimental results.

5.2 Greedy variant

By analogy with the greedy extension algorithm, we can choose output coordinates of A that locally (at the current step) maximize the size of the resulting set X'. Furthermore, we can choose to guess *components* (i.e., linear combinations of coordinates) since this may lead to larger sets X'.

For the purpose of optimizing the resulting implementation, we will present an LAT-based formulation of the method, where we will no longer track the intermediate sets X but rather track their size using LAT coefficients. This representation is based on an observation from [30], rewritten in our terminology.

Theorem 2 ([30, Proof of Thm. 4.3]). *Let $S, A \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, where $A = L + c$, L is linear and $c \in \mathbb{F}_2^m$. Then,*

$$\sum_{\beta \in \mathbb{F}_2^m} (-1)^{\langle \beta, c \rangle} W_S(\langle \beta, L \rangle, \beta) = \sum_{\beta \in \mathbb{F}_2^m} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \beta, S(x) + A(x) \rangle} = 2^m |S \cap A|.$$

The theorem says that we can compute the strength of an approximation A (i.e., the size of $S \cap A$) simply by summing up LAT coefficients, where in each column β, we take one coefficient corresponding to the coordinate of the linear part of the affine map (i.e., $\langle \beta, L \rangle$), negated if the inner product of β and c is equal to 1. We can also consider an m'-dimensional linear subspace $V \subseteq \mathbb{F}_2^m$ in the output, where $m' < m$. Then the corresponding sum will yield the size of the set $X = (\pi \circ S) \cap (\pi \circ A)$, where π is the projection of $\mathbb{F}_2^m$ on V discarding the orthogonal complement $V^\perp$, i.e., a partial output linearization.

The pseudocode for the algorithm is given in Algorithm 4. We start by precomputing the full LAT of the S-box, denoted C_0. During the algorithm, we will "compress" it by summing along m independent directions corresponding to output masks. The process consists of m consequent steps. Intermediate tables corresponding to LAT sums over guessed subspaces are denoted by C_d at depth d. These tables have signature $\mathbb{F}_2^n \times \mathbb{F}_2^{m-d} \to \mathbb{Z} : (\alpha, \beta) \mapsto w$, where β is a representative of a coset of the currently guessed d-dimensional linear subspace.

In each step, we choose a pair $(\tilde{\alpha}, \tilde{\beta})$, $\tilde{\beta} \neq 0$, with the highest absolute value of $w = C_d[\tilde{\alpha}, \tilde{\beta}]$, where the ties are resolved at random. Each such choice corresponds to the guess $\langle \tilde{\beta}, A(x) \rangle = \langle \tilde{\alpha}, x \rangle + [w < 0]$ of one component of the desired affine approximation A. Here, $[w < 0]$ is the Kronecker delta, equal to 1 if w is negative, and 0 otherwise. Then, we compress the table C_d into C_{d+1} by setting

$$C_{d+1}[\alpha, \beta] = C_d[\alpha, \beta] + \operatorname{sgn}(w) \cdot C_d[\alpha + \tilde{\alpha}, \beta + \tilde{\beta}]$$

for all $\beta < \beta + \tilde{\beta}$ (according to an arbitrary predetermined order).

The constraint $\beta \neq 0$ ensures that all the chosen β values are linearly independent and the affine approximation A is well-defined at the end of the algorithm. Furthermore, from Theorem 2, we can deduce that $C_m[0, 0]$ (which stores the sum of LAT over the guessed subspace) is equal to $2^m |S \cap A|$.

Algorithm 4 Greedy reduction algorithm

Inputs: function $S \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, parameters $t_d, t_t \in \mathbb{Z}_{>0}$
Output: affine approximation $A \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$

$C_0 \leftarrow \{(\alpha, \beta) \mapsto W_S(\alpha, \beta)\}$ ▷ LAT can be precomputed
$G \leftarrow \{\}$
for all $d \in \{0, \ldots, m-1\}$ **do**
    if $d < t_d$ **then**
        $t \leftarrow t_t$
    else
        $t \leftarrow 1$
    end if
    $(\tilde{\alpha}, \tilde{\beta}, w) \leftarrow$ randomly chosen pair from the largest t coefficients (by the absolute value) in C_d and the associated sum w, $\tilde{\beta} \neq 0$.
    $G \leftarrow G \cup \left\{(\tilde{\alpha}, \tilde{\beta}, w)\right\}$ ▷ $\langle \tilde{\beta}, A(x) \rangle = \langle \tilde{\alpha}, x \rangle + [w < 0]$
    $C_{d+1} \leftarrow \left\{(\alpha, \beta) \mapsto C_d[\alpha, \beta] + \operatorname{sgn}(w) \cdot C_d[\alpha + \tilde{\alpha}, \beta + \tilde{\beta}] \;\middle|\; \beta < \beta + \tilde{\beta}\right\}$ ▷ merge
end for ▷ Note that $C_m[0, 0] = 2^m |S \cap A|$
return affine approximation A reconstructed from G

Increasing variability As we will show in the experiments, the fully-greedy algorithm often yields low variability in its answers, since often the involved maximal absolute biases are given by a small number of masks, and the algorithm gets stuck in the same approximations being local maximums in the search space. In order to tackle this problem, in the first t_d iterations, we explore top-t_t masks by the absolute correlation *equiprobably*. For example, the parameters $(t_d, t_t) = (4, 5)$ make the algorithm consider top-5 masks equally for the first 4 guesses and top-1 masks for the remaining guesses.

Complexity analysis Although the LAT can be precomputed once, the complexity of the algorithm is dominated by the first compression step, which maps the table C_0 of size 2^{n+m} to a table half of that size. Thus, we get $2^{n+m-1-d}$ additions at iteration d which sums to $\mathcal{O}(2^{n+m})$ complexity of one execution. Note that maintaining the largest t_t coefficients and the corresponding mask pairs can be done efficiently using a small heap data structure, similar to the greedy extension algorithm, adding a constant overhead per each computed element of the table C_{d+1}.

6 Experimental evaluation

In this section, we report results of the application of our algorithms to a wide range of S-boxes from the literature.

Procedure The typical procedure consists of running the greedy extension algorithm to get some first heuristic results, initially starting with random 4-zero-sums (or 6-zero-sums for APNs), and then with random single points. The main reasons why we preferred the extension algorithm over the reduction algorithm are the efficient sampling complexity ($\mathcal{O}(2^n)$ versus $\mathcal{O}(2^{n+m})$) and inherent variability of the greedy extension algorithm; it also yielded better approximations in most cases. Sometimes we ran the greedy reduction algorithm to have better coverage. The two algorithms were initially implemented in SageMath [43], later with a port of the greedy extension algorithm to Rust with Python bindings.

After obtaining a heuristic approximation, if feasible (for most cases of $n \leq 8$), we ran the exhaustive reduction algorithm to investigate whether the initial values resulted from the greedy algorithms indeed already gave us the vectorial linearity. The algorithm was implemented in C++ due to its simplicity and performance (parallelization) requirements.

Experimental setup All heuristic algorithms for S-boxes were run on a single core of a modern laptop, typically from a few minutes to a few hours per S-box. Exhaustive algorithms were executed on a high-performance cluster (a few nodes of 128-core AMD EPYC 7452 processors) in order to process a large number of instances in a short time. Since the algorithms do not require a lot of memory, they can in principle be run on normal PC/laptop as well (however, for 16-bit Super-Sboxes, the precomputed DDT requires about 40 GB of RAM). Example timings of the exhaustive evaluation include: a quadratic APN function with LB 16 - 81 core-hours (20 736 without the optimization); the AES S-box with LB 19 - 870 core-hours; the SNOW-3G S-box with LB 27 - 27 core-hours. These timings should approximately indicate the cost of search depending on the expected VecLin.

6.1 Monomial S-boxes

The seminal work of Nyberg [37] showed that monomial maps over the finite field $\mathbb{F}_{2^n}$ can be used as strong S-boxes. In particular, the cube mapping $x \mapsto x^3$ is always APN although it is only a bijection for odd n. The multiplicative inverse mapping $x \mapsto x^{2^n-2}$ is differentially 4-uniform for even n and APN for odd n, and always a bijection, which made it a popular choice in symmetric-key designs, including the standard AES block cipher (up to affine equivalence). The algebraic degrees of these maps are 2 and $n-1$, respectively. We also consider the quadratic S-box given by $x \mapsto x^5$, which is differentially 4-uniform for even n and APN for odd n as well, but may have a cheaper implementation cost. In addition, we consider the monomial $x^{2^{n/2-1}+1}$ for which Nagy [34] showed an approximation of size $2^{n/2}+2$ and an upper bound $\sqrt{2} \cdot 2^{n/2} + \frac{1}{2}$. The results of applying our procedure to these S-boxes for various values of n is given in Table 2. All of them were obtained within 2 days of runtime per S-box (time to the result gradually increasing with n; for $n \leq 8$ a few minutes are sufficient). We note that some of the results in the APN cases for $n \leq 7$ (and x^9 for $n = 8$) were previously obtained in [34].

Table 2. Vectorial linearity of monomial functions x^e, $e \in \{3, 5, 7, 9, 2^n - 2\}$. Values for $n \leq 8$ are exact, values for $n \geq 9$ are lower bounds.

	n												
S	4	5	6	7	8	9	10	11	12	13	14	15	16
x^3	6	7	9	11	16	19	25	28	36	40	55	51	93
x^5	7	7	10	12	16	18	28	26	46	35	55	44	67
x^7	7	7	13	16	22	28	34	32	46	41	58	55	76
x^9	6	7	15	11	18	22	22	27	43	36	42	78	47
$x^{2^{n/2-1}+1}$	6		10		18		34		66		130		258
x^{2^n-2}	7	7	10	12	18	19	34	30	66	42	130	53	91

For the x^9 map, it is surprising that for $n = 6$ the vectorial linearity is equal to 15 while for $n = 7$ it is lower (equal to 11). Note that these are exact values since we were able to verify them using exhaustive algorithms.

For $n = 10$, the smallest (among maximum ones) approximation we found was 22 points again for the x^9 map (note that this is significantly smaller than the general disproven conjectured bound of $2^{n/2}+2 = 34$, yet we cannot claim that the value is exact). Furthermore, for $n = 9$ we achieved the same value of 22 points. From $n = 12$ to $n = 13$, the size of the approximation for x^9 *decreased* again according to our results.

All these surprising heuristic findings raise the question of further theoretical or computational investigation.

Linearization of the inverse mapping for even n The authors of [47] described an elegant generic way to linearize the multiplicative inverse mapping for even values of n, based on the decomposition $x^{2^n-2} = (x^{2^{n/2}+1})^{2^{n/2}-2} \cdot x^{2^{n/2}}$. Later, in [34], a similar approximation of the same size was described for the monomial $x^{2^{n/2-1}+1}$, which is APN for $n = 4k$. It follows that for even n, $\mathsf{VecLin}_{x^{-1}} \geq 2^{n/2} + 2$. For $n = 8$, (where the inverse monomial is affine-equivalent to the AES S-box), we were able to verify that 18 points is indeed the best possible linearization, using the exhaustive reduction algorithm (it recorded visiting 2^{44} nodes).

The $2^{n/2} + 2$ value is what we observed using our algorithms for $6 \leq n \leq 14$. For $n = 16$ we were unable to recover the known approximations of size 258 within 2 days of computation, although we expect that it should be possible with more effort. In fact, for smaller dimensions n, the inverse monomial was always the slowest for our greedy extension algorithm to converge to a result. A possible explanation is that the inverse monomial has a small number of 4-zero-sums (one per each nontrivial DDT row) and the best known approximation described above does not contain any 4-zero-sums. This contradicts the greedy extension heuristic and increases the number of first guesses needed to obtain any information.

This leaves open the question of how the best approximation of the inverse mapping is structured for odd n. In Section D, we prove that, in this case, the decomposition cannot be used to obtain a linearization.

6.2 S-boxes from ciphers

We also applied our algorithms to S-boxes from the literature, namely from cryptographic designs of block/stream ciphers [40]. A selection of results is presented in Table 3; results for the remaining ones are available in the supporting repository. We note that [18] already reported results for 5-bit S-boxes using SAT-solvers and GPU bruteforce; some heuristic results for 8-bit S-boxes were provided (based on an undisclosed method), yet we improve nearly all of them.

Table 3. Vectorial linearity of S-boxes from the literature.

n	S-box	δ_S	Lin_S	$\deg S$	VecLin_S
5	Fides_5	2	8	2	7
	Ascon	8	16	2	11
6	Fides_6	2	16	4	9
	SC2000_6	4	16	5	10
7	WAGE	8	40	6	17
8	$\mathcal{I}'$/LFT216/Wuhan3	6	32	7	17
	AES	4	32	7	18
	Kalyna π_2, π_3	8	48	7	19
	Kalyna π_0, π_1	8	48	7	20
	BelT	8	52	7	20

n	S-box	δ_S	Lin_S	$\deg S$	VecLin_S
8	Kuznyechik	8	56	7	20
	Anubis	8	68	7	21
	Skipjack	12	56	7	21
	Scream	8	64	6	22
	SNOW-3G SQ	8	64	5	26
	iScream	16	64	6	27
	Safer	128	92	7	30
	Skinny_8	64	128	6	47
	CSS	128	256	4	81
9	DryGASCON_{256}	128	256	2	76[3]

We observe that for 8 bits, the strongest permutation among traditional ones is the AES S-box, based on the inversion in the finite field. However, a recent version of [18] mentions S-boxes LFT216 [4] and Wuhan3 [45], which seem to achieve a lower vectorial linearity.[4] We ran our greedy and exhaustive algorithms on

[3] The published version of the paper claims 77, which happened due to errors in the S-box table provided by SageMath. We also verified that these are the best approximations (both 77 for the corrupted one and 76 for the corrected one) using the exhaustive reduction algorithm.

[4] We thank Nicolas T. Courtois who brought this to our attention.

a related S-box $\mathcal{I}'$, which is simply the field inverse function with the first two entries swapped. We confirmed that all these three S-boxes are affine-equivalent. The swap eliminates 0 from all the 15 best approximations of size 18, and does not introduce new ones of size 18. We confirmed with our exhaustive reduction algorithm that it has $\mathsf{VecLin}_{\mathcal{I}'} = 17$, which is the lowest known (so far) vectorial linearity for an 8-bit permutation. Finding a better S-box or proving that this is the minimum value remains a challenging open problem.

Interestingly, there are quite a few other S-boxes which are built using totally different methodologies and have a close value of vectorial linearity 19-22. For example, Kalyna S-boxes [27] were generated using hill-climbing / gradient-descent; BelT S-box is an exponentiation in the finite field and Kuznyechik's S-box has a related but different structure [39]. Skipjack's design is secret but it was suggested that it was also optimized for a flatter Walsh spectrum distribution [9]. We conclude that there are multiple approaches to obtain S-boxes with reasonable vectorial linearity.

We now discuss a few S-boxes with high vectorial linearity. The CSS (Content Scrambling System) S-box used in the old DVD protection mechanisms has an extremely sparse ANF and multiple linear components, thus it is unsurprising that it can be linearized with probability almost 1/3. The SKINNY 8-bit S-box was optimized to be lightweight and can be implemented using 8 NOR and 8 XOR gates, which made it quite weak overall, leading to strong linearization too (probability about 18% for the full S-box).

6.3 Super-Sboxes from ciphers

We recall that a Super-Sbox consists of a layer of four S-boxes, key addition, a MixColumn linear layer, and another layer of four S-boxes, having 8 S-boxes in total. By an abuse of terminology, we also include the MISTY-FI function which is an unbalanced Feistel Network with 3 S-boxes and a 16-bit key, possessing similar properties as Super-Sboxes. We ran our greedy extension algorithm on several 16-bit Super-Sboxes and the MISTY-FI function, with zero and random keys. The results are given in Table 4. In addition, to test the limits of our methods, we ran a simplified variant of the Greedy-Reduction algorithm on the 32-bit Super-Sbox of AES, which attracted a lot of interest in analysis.

Table 4. Lower bounds on vectorial linearity for the 16-bit Super-Sboxes of Midori, SKINNY, LED, the MISTY-FI keyed function, and the 32-bit Super-Sbox of AES.

Super-Sbox	Key	VecLin $\geq$
Midori64	(0,0,0,0)	4096
Midori64	(C,C,C,C)	4096
Midori64	(E,2,1,9)	576
Midori64	(0,A,0,F)	1280
SKINNY64	(0,0,0,0)	1060
SKINNY64	(9,8,2,0)	1046

Super-Sbox	Key	VecLin $\geq$
LED	(0,0,0,0)	122
LED	(5,A,5,9)	123
MISTY-FI	(0,0,0,0)	50
MISTY-FI	(0,8,0,e)	50
AES	(00,00,00,00)	60

For Midori64, it is noticeable that the less distinct the four individual values are, the higher the vector linearity. In the case of complete equality, we get $\mathsf{VecLin} \geq 4096 = 2^{12} = 8^4$. Surprisingly, this is greater than just four 4-bit Midori S-boxes run in succession yielding $\mathsf{VecLin} = 7^4 = 2401$, since one such individual 4-bit S-box has a vectorial linearity of 7. After investigation, we found a simple reason for this: the approximations come from the involutive property of the Super-Sbox which occurs when the four key nibbles are the same. In this case, the approximation is simply the identity map applied to 4096 fixed points. On the other hand, for a random key, such as (E,2,1,9), the best approximation we found is not based on fixed points since the Super-Sbox does not have any.

For SKINNY64, we found strong approximations independently of the key used, although the approximations themselves do depend on it. Our investigation revealed two weaknesses that led to the distinguisher. First, the sparse linear layer (containing only three 4-bit nibble XORs and a nibble permutation) implies one composition of two S-boxes that has no modification in-between (except the key/constant XOR). Furthermore, fixing one input nibble to a constant creates two more such fixed compositions. Second, compositions of the S-box with itself have VecLin ranging from 7 to 13 (depending on the constant in-between), while the S-box itself has VecLin of 7. In the approximation we found, about half of the inputs have the third nibble fixed, which leads to another S-box composition with VecLin equal to 13.

Interestingly, the MISTY-FI function leads with the smallest approximations we could find. An explanation is that it is built from three monomial S-boxes over two different fields ($\mathbb{F}_{2^7}$ and $\mathbb{F}_{2^9}$). The LED Super-Sbox is also strong, which is explained by its strong MDS linear layer and the optimal PRESENT S-box.

For the 32-bit AES Super-Sbox, we managed to find an approximation of size 60. While it is not strong compared to the domain size, it is a nontrivial result and we are not aware of other techniques that could achieve this. Clearly, we cannot make conclusions on the vectorial linearity of the Super-Sbox in this case, and it remains a challenging open question. We invite other researchers to improve our result.

6.4 APNs

We recall two equivalence notions. Two vectorial Boolean functions $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^m$ are said to be affine equivalent if there exist affine bijections $A : \mathbb{F}_2^n \to \mathbb{F}_2^n, B : \mathbb{F}_2^m \to \mathbb{F}_2^m$ and an affine map $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ such that $G = (B \circ F \circ A) + C$. Furthermore, F, G are said to be CCZ-equivalent [15] if there is an affine bijection $M : \mathbb{F}_2^{n+m} \to \mathbb{F}_2^{n+m}$ such that

$$M \times \{(x, F(x)) | x \in \mathbb{F}_2^n\} = \{(x, G(x)) | x \in \mathbb{F}_2^n\}.$$

We applied our algorithms to several known APN extended affine (EA) equivalence classes. Brinkmann and Leander [12] classified all APN functions up to extended-affine equivalence and CCZ-equivalence for n up to 5 bits. Although they showed that all APN functions in dimensions up to 5 are CCZ-equivalent to a monomial mapping (which we covered in Subsection 6.1), vectorial linearity is not necessarily preserved by CCZ-equivalence, which is why we checked the EA-equivalence classes. However, when $3 \leq n \leq 5$, we can use our lower bound $n+2$ for $n \geq 4$ (and $n+1$ for $n = 3$), which coincides with the Sidon-set upper bound applicable to APN functions (see [34]). Thus, the vectorial linearity of APN functions is forced to be equal to $4, 6, 7$ respectively for $n = 3, 4, 5$ simply by the theoretical bounds. We also verified this using our algorithms.

In the case of $n = 6$, we considered the 534 EA-classes of cubic APNs given by Langevin [29]. With the exhaustive reduction algorithm, we confirmed that all of them have a vectorial linearity equal to 9 (the same as the 6-bit APN permutation).

For $n = 7$, we checked the APNs provided in the `sboxU` package. Most of the functions had vectorial linearity 11 (same as the cube mapping), but some of them reached the maximum Sidon set size bound of 12.

For $8 \leq n \leq 10$, we checked the recently found quadratic APNs from [7]. Interestingly, we found a few 8-bit APN functions with vectorial linearity 15, which is smaller than any of the 8-bit monomial mappings. An example of such a function, denoted S_{15}, is given in Section C. It has Boolean linearity 32.

6.5 Discussion

The authors of [30] conjectured a general bound for arbitrary maps $S \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$:

$$\mathsf{VecLin}_S \geq 2^n - \left(1 - 2^{-m}\right)\left(2^n - 2^{n/2}\right).$$

For $m = n$, the right-hand side becomes $2^{n/2} + 1 - 2^{-n/2}$, which means $\mathsf{VecLin}_S \geq 2^{n/2} + 1$ for even n. For example, for $m = n = 8$, this is equal to 17. Very recently, Nagy [34] disproved the conjecture by using relations between APN functions and Sidon sets and some computations, showing that $\mathsf{VecLin}_{x^3} \leq 16$ for $n = 8$ and $\mathsf{VecLin}_{x^3} \leq 21$ for $n = 9$. However, Nagy was not able to find matching approximations.

Our results confirm that $S(x) = x^3$ has $\mathsf{VecLin}_S = 16$ precisely (with independent upper bound verification), and that the function $S = S_{15}$ has $\mathsf{VecLin}_S = 15$ precisely, further departing from the conjecture by one. We recall that the proven lower bound for $n = 8$ is 10. It is thus a challenging open problem to find 8-bit S-boxes with vectorial linearity lower than 14, or to prove that they do not exist.

7 Applications to cryptanalysis

Linearization of S-boxes is a useful tool that has strong potential to aid other cryptanalysis techniques and key recovery processes. The current state-of-the-art has particularly highlighted its relevance in the

setting of low-data attacks, especially when only one plaintext-cipher pair is available, or in the similar CICO problem for permutations [24]. For simplicity, we focus on the balanced case, when the input and the output have the same number of words fixed.

Problem 2 (CICO). Let $F\colon (\mathbb{F}_2^n)^m \to (\mathbb{F}_2^n)^m$ be a bijection and let c, r be positive integers with $c+r = m$. The $\mathrm{CICO}_{c,r}$ problem for F asks to find an input-output pair $y = F(x)$ such that the first c $\mathbb{F}_2^n$-entries of x and y are equal to zero, that is, $y_0 = \ldots = y_{c-1} = x_0 = \ldots = x_{c-1} = 0 \in \mathbb{F}_2^n$.

In the following, we consider the following simplistic approach, which is different from some approaches used in cryptanalysis of hash functions. The key difference is that we consider arbitrary affine approximations instead of those restricted to affine spaces.

The approach is to simply choose approximations for all the involved S-boxes and hope that they hold. Concretely, in the solution to the problem, all inputs to the S-boxes must lie in the corresponding approximation sets. In this case, the S-boxes can be replaced by affine maps and the problems become linear, and can be easily solved. However, a final verification is needed since we do not have linear constraints ensuring the validity of the approximations.

In the following, we will discuss different ways to choose approximations for the S-boxes.

7.1 Covering domain with approximations

In the single-plaintext setting, aiming for all S-boxes to be successfully linearized with a single best approximation is not the best strategy, since it will quickly lead to a low success probability after trying to linearize a few S-boxes in the attacked primitive. The idea instead is to perform a new attack attempt without requesting new encryptions but with guessing a different set of approximations for S-boxes. In this way, we will ensure that for any existing solution to the problem, we have a guaranteed success probability of our attack which only depends on the randomness sampled during the attack.

We consider two solutions to the problem: a deterministic one and a probabilistic one. For both approaches, we will assume that a relatively large number of approximations is found, with maximum or close to the maximum approximation size. In particular, each input from the domain should be included in at least several approximations. This is usually already achieved as a byproduct of the repetitive execution of our algorithms.

Deterministic cover The natural idea is to cover the full input space by t different approximations. Then, it is sufficient to enumerate all possible covers during an attack, which has a multiplicative cost of t guesses per S-box. If the considered problem (key recovery or the CICO problem) has a solution, then there will be a set of guesses agreeing with the S-box inputs from the solution, and linearization will succeed. In the ideal case, the cover becomes a partition, as is (almost) in the case of the inverse mapping, and this was used for the linearization attack in [47].

In order to optimize the attack, we need to minimize t, the number of approximations in the cover. This is an instance of the NP-hard SetCover problem. This problem can be straightforwardly modeled as an integer linear programming system (MILP), see e.g. [44], yielding good quality or optimal solutions, depending on the S-box size.

However, it is inevitable that the best cover by approximations has many overlaps between the approximations. This reduces the effective approximation size $2^n/t$, even if all considered approximations have the same size VecLin_S.

Probabilistic uniform cover We now propose a new alternative method, which yields a probabilistic attack but often with no or small loss in the effective approximation size. It is in particular relevant to S-boxes where approximations do not form a nice partition. The idea is again to cover the input space with approximations, but instead of minimizing their number we will assign probabilities (weights) to each of the available approximations. The requirement would be to obtain a (close to) uniform distribution on the input space, while maximizing the effective approximation size. Then, during the attack, we will sample an approximation per each S-box according to the chosen probabilities. In case of attack failure, we can simply rerun the attack, because the uniform distribution guarantees that each correct input from the solution always has a fixed chance of being covered by the chosen approximation.

More formally, consider an S-box $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ and let $X_1, \ldots, X_N \subseteq \mathbb{F}_2^n$ be the intersection sets of affine approximations $A_1, \ldots, A_N$ with S. Our goal is to assign probabilities $P = (p_1, \ldots, p_N)$, $\sum_{i=1}^N p_i = 1$ such that for each $x \in \mathbb{F}_2^n$, the probability to be covered by an approximation sampled according to P is close to uniform, that is,

$$\left| \Pr_{X \xleftarrow{p_i} X_i}[x \in X_i] - g2^{-n} \right| \leq \epsilon 2^{-n}, \tag{1}$$

where $g = \sum_{i=1}^N p_i |X_i|$ is the expected approximation size and ϵ is a small error, e.g., $\epsilon = 0.01$. In addition, we aim to maximize g within the above constraints.

This problem can be efficiently solved by linear programming. For each $x \in \mathbb{F}_2^n$, observe that the probability to be selected is equal to $\sum_{i:x\in X_i} p_i$. It follows that the problem can be formulated as a linear programming on real variables $p_i, 0 \leq p_i \leq 1$, with the 2^n constraints of the form (1) and the objective function $g = \sum_{i=1}^N p_i |X_i|$ to maximize.

Having a solution P to the problem, we can now mount an attack on a cryptographic primitive by sampling the approximation for each involved S-box application according to the distribution P. Due to the uniformity, each attack attempt is independent from the other, and every S-box call has success chance $g2^{-n}$ (up to the error $\epsilon 2^{-n}$) to be linearized. In other words, if $S(x) = y$ in the solution to the cryptographic attack, with probability $g2^{-n}$ we have $A(x) = y$, where A is the sampled approximation.

Another slightly different option is maximization of the minimum probability of an input being covered, namely

$$\min_{x \in \mathbb{F}_2^n} \Pr_{X \xleftarrow{p_i} X_i}[x \in X_i].$$

This can be easily achieved by linear programming, by introducing a new variable $g \in \mathbb{R}$ being maximized, and adding constraints $g \leq \Pr_{X \xleftarrow{p_i} X_i}[x \in X_i]$ for all $x \in \mathbb{F}_2^n$. The resulting distribution of this optimization provides a *guaranteed* effective approximation size g that would work for *any* solution to the cryptanalysis problem. The difference from the method in (1) is that there we optimized the *average* approximation size while allowing some small deviation from it.

The advantage of the uniform cover method over the deterministic method is that the problem is linear in its nature in contrast to the combinatorial nature of the previous method. Therefore, with a sufficient number of approximations ($c \cdot 2^n$ for a small factor c), the 2^n-dimensional linear space of probabilities $(\Pr[x \in X_i])_{x \in \mathbb{F}_2^n}$ saturates which allows to achieve arbitrary distributions. As long as a sufficient number of approximations of large size are available, the method should succeed without further reducing the effective approximation size. A good example is given by quadratic S-boxes, where a perfect uniform cover from any single approximation is always easy to construct directly.

Proposition 7. *Let $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ be of algebraic degree 2 and let $A\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ be its affine approximation of size $\hat{g}$. Then, it is possible to construct a uniform cover (as defined above) with $\epsilon = 0$ and expected approximation size $g = \hat{g}$.*

Proof. Using Theorem 1, we can set $\{X_i\}_i = \{(A \cap S) \oplus \Delta \mid \Delta \in \mathbb{F}_2^n\}$, $N = 2^n$, and $p_1 = \ldots = p_N = 2^{-n}$. Clearly, every $x \in \mathbb{F}_2^n$ is hit by exactly $\hat{g}$ different offsets Δ (given by $\Delta = x \oplus y$ for y in $A \cap S$) and the statement follows. □

Remark 2. For monomial S-boxes, one can obtain a similar result by shifting the approximation set multiplicatively by all nonzero finite field elements. However, this will not ensure uniformity at $x = 0$, which we then would need to treat in alternative ways or simply ignore the bias.

Example application to the 8-bit SKINNY S-box We exemplify the two approaches on the 8-bit SKINNY S-box, which has vectorial linearity 47. In our experiment, we ran our greedy extension algorithm 2000 times and collected the approximations. More than 75% of them have size at least 40.

For the deterministic cover (partition-like) approach, we use a MILP solver with a short time limit to obtain a herustic solution to the SetCover problem. It results in a set of 11 approximations taken from the pool of 2000 approximations. These 11 approximations cover the full input space. This results in the effective approximation size $2^n/11 \approx 23.27$. This is twice smaller than the best approximation size, or even most of the approximation sizes in the pool. We conclude that this approach significantly decreases the approximation strength.

For the probabilistic uniform cover, we obtain an effective approximation size 43.82 with the maximization of the average and uniformity bound $\epsilon = 1$ (maximum deviation $\epsilon 2^{-n} = -1/256$). Each input is covered with a probability ranging between 0.168 and 0.175. For the alternative option of maximizing the lower bound, we obtain the effective approximation size 43.50, coinciding with the minimum (over the inputs) average (over the distribution) approximation size. This is slightly lower than the value from the first option, but it guarantees to achieve this probability for any cryptanalytic solution.

This example showcases the strength of our probabilistic method compared to the naive deterministic coverage, essentially providing a twice better success probability per an S-box. We remind, however, that the requirement to have a large number of strong approximations poses an additional applicability constraint.

7.2 Solving the CICO problem over SPN by linearization

We consider a $\mathrm{CICO}_{c,r}$ problem over an SPN structure with R S-box layers ($m = r + c$ S-boxes each) and MDS linear layers, starting and ending with an MDS layer. This is a common structure among arithmetization-oriented primitives such as Poseidon [23,21]. Although many such primitives work over large finite fields, often prime fields, a recent trend considers application of smaller S-boxes in some layers (e.g. Monolith, [22], Skyscraper [11]). In Section E, we recall some basic tricks for skipping some S-boxes in the CICO problem, based on available degrees of freedom. Note that for SPNs starting and ending with an S-box layer, the number of attacked rounds can be directly increased by 2 rounds.

Linearization attack By linearizing all $A = mR-(r-c) = r(R-1)+c(R+1)$ active S-boxes (using one of the techniques described in Subsection 7.1), where $r-c$ correspond to the degrees of freedom used to skip some S-boxes, the problem reduces to an $\mathbb{F}_2$-linear system of cn equations and variables which has one solution on average. However, this system does not constraint correctness of the approximations (i.e., that the actual inputs in the solution fall in the guessed approximation). Therefore, the success probability of one such attempt is given by $(g2^{-n})^A = (g2^{-n})^{mR-(r-c)}$ (where g is the expected approximation size) and the inverse of this probability determines the expected number of attempts to succeed, each costing a linear system solution over cn bits.

In particular, for a single round ($R = 1$), we have the cost of $(g2^{-n})^{-2c} = 2^{2c(n-\log_2 g)}$, while the generic bruteforce attack has cost 2^{nc}. This implies that linearization attack for 1 round is effective whenever $g > 2^{n/2}$.

Remark 3. Although a single-round CICO problem (MDS layer, S-box layer, MDS layer) does not look challenging, we are not aware of any general attacks. An MDS linear layer prevents guessing values: guessing one S-box input or output simply eliminates the S-box and leads to an equivalent problem of smaller size without any gain (even worse, one would need to guess $2c$ values, compared to exhaustive search over c values). Typical algebraic attacks in the recent literature on AO primitives rely on algebraic equations describing the S-box, which can generally be of high degree. Two and more rounds are even more challenging and are susceptible to our generic attack only for S-boxes with quite high approximation sizes.

For a larger number of rounds, the rate size starts to contribute to the cost, since the extra degrees of freedom it provides are insufficient to counter the number of new active S-boxes. We will consider the minimal case of $r = c$ ($m = 2c$), so that the complexity is given by $(2^n/g)^{2cR}$, for example $(2^n/g)^{4c}$ for 2 rounds, $(2^n/g)^{6c}$ for 3 rounds, etc. We conclude with the following theorem.

Theorem 3. *The linearization attack applied to* $\mathrm{CICO}_{c,c}$ *(i.e., $r = c$, excluding $R = 1$ where r can be any) improves over the exhaustive search attack when the average approximation size g for the S-box satisfies*

$$g > 2^{n\left(1-\frac{1}{2R}\right)},$$

assuming the cost of solving the linear system is comparable to one evaluation (see Table 5). The total time complexity is then given by

$$\left(\frac{2^n}{g}\right)^{2Rc} = 2^{2Rc\cdot(n-\log_2 g)}.$$

constructions and solutions of square linear systems of dimension cn.

Remark 4. In practice, at least an extra factor of cn should be considered in the complexity cost to accommodate the construction and solution of the linearization system. Note that the matrix can be constructed by querying the cn unit vectors in the input, and the system solving should take comparable time (assuming that the evaluation cost takes a quadratic number of bit operations).

Table 5. Minimum (average) approximation size of an S-box needed for a linearization attack on $\mathrm{CICO}_{c,c}$ (i.e., $r = c$) over SPN with MDS linear layers and m parallel n-bit S-boxes per layer.

Rounds	$g(n)$	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$
1	$2^{n/2}$	4	16	2^8	2^{16}	2^{32}
2	$2^{3n/4}$	8	64	2^{12}	2^{24}	2^{48}
3	$2^{5n/6}$	10.1	102	$2^{13.3}$	$2^{26.7}$	$2^{53.3}$
4	$2^{7n/8}$	11.3	128	2^{14}	2^{28}	2^{56}
R	$2^{n\left(1-\frac{1}{2R}\right)}$					

We have experimentally verified the attack on the $\mathrm{CICO}_{3,3}$ problem over a 1-round SPN with random linear layers and the 8-bit SKINNY S-box, which has a best affine approximation with 47 inputs. By finding a large number of (around 300) approximations in the size range between 40 and 47, we managed to build an almost-uniform probabilistic cover with average approximation size 43.7, leading to $(2^n/g) = 2^{2.55}$ and complexity $2^{5.1c}$ equal to $2^{15.3}$ linearizations for $c = 3$, compared to 2^{24} evaluations in an exhaustive search. During $2^{18.1}$ linearization attempts, we succeeded $15 = 2^{3.9}$ times, yielding the experimental expected number of attempts $2^{-14.2}$. The improvement over $2^{15.3}$ is explained by the fact that there existed two different solutions to the generated CICO problem, found equally often by our method.

8 Conclusions

In this work, we presented efficient algorithms for the linearization of S-boxes in practice for the first time. Our heuristic algorithms quickly provide strong affine approximations of a given S-box. Furthermore, for n up to 8 bits, our exhaustive algorithms can be used to prove (or, in principle, disprove and improve) the optimality of the approximation. The feasibility of the exhaustive algorithms for larger n is limited, and our heuristic results raise many interesting questions about vectorial linearity. As an application, we derived a methodology and illustrated a linearization attack against a CICO problem, which is of high relevance for permutation-based designs. We believe our powerful toolkit achieves a milestone in understanding linearization and our results lead to a wide range of open problems and research directions, which we outline below.

Better exhaustive algorithms The natural question is to improve the exhaustive algorithms so that larger S-boxes can be processed feasibly.

Proving bounds on the results of heuristic algorithms An alternative approach to obtain precise values (or, at least, bounds) of vectorial linearity is to prove bounds on the quality of our heuristic algorithms. This is well motivated by the fact that our algorithms quickly converge to the best approximations known to us and best overall approximations for $n \leq 8$.

Determining smallest possible vectorial linearity Our theoretical results, in particular if they turn out to be optimal, raise many interesting considerations about vectorial linearity, for example, how the sequence of the smallest possible vectorial linearity grows. These values could inspire new conjectures and results in theoretical studies.

Reverse-engineering of approximations In all the cases of our algorithms, we did not immediately see any structure in the approximating set, despite one would expect it at least in the case of monomial maps. The only exception is the inverse mapping for even n, for which our best results match the known structure of [47]. Therefore, an interesting research avenue would be to study the discovered approximation sets in order to find any structure and generalize this to larger dimensions.

More applications to cryptanalysis So far, vectorial linearization of S-boxes has been used in special cases (linearization on subspaces in cryptanalysis of hash functions [19], linearization on subgroups in [47]). The natural goal of this paper's research direction is to expand on these applications.

References

1. Banik, S., Barooti, K., Durak, F.B., Vaudenay, S.: Cryptanalysis of LowMC instances using single plaintext/ciphertext pair. IACR Trans. Symm. Cryptol. **2020**(4), 130–146 (2020). https://doi.org/10.46586/tosc.v2020.i4.130-146 2
2. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 411–436. Springer Berlin Heidelberg, Germany, Auckland, New Zealand (2015). https://doi.org/10.1007/978-3-662-48800-3_17 2
3. Baum, C., Delpech de Saint Guilhem, C., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: Short and fast signatures from AES. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 266–297. Springer, Cham (May 2021). https://doi.org/10.1007/978-3-030-75245-3_11 2
4. Beg, S., Ahmad, N., Anjum, A., Ahmad, M., Khan, A., Baig, F., Khan, A.: S-box design based on optimize lft parameter selection: a practical approach in recommendation system domain. Multimedia Tools and Applications **79**(17), 11667–11684 (May 2020). https://doi.org/10.1007/s11042-019-08464-6, https://doi.org/10.1007/s11042-019-08464-6 15
5. Beierle, C., Biryukov, A., dos Santos, L.C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q., Biryukov, A.: Schwaemm and Esch: lightweight authenticated encryption and hashing using the Sparkle permutation family. NIST round **2** (2019), version 1.1 2
6. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 123–153. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53008-5_5 2
7. Beierle, C., Leander, G.: New instances of quadratic APN functions. IEEE Transactions on Information Theory **68**(1), 670–678 (2022). https://doi.org/10.1109/TIT.2021.3120698 17, 27
8. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO'90. LNCS, vol. 537, pp. 2–21. Springer, Berlin, Heidelberg (Aug 1991). https://doi.org/10.1007/3-540-38424-3_1 1
9. Biryukov, A., Perrin, L.: On reverse-engineering S-boxes with hidden design criteria or structure. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 116–140. Springer, Berlin, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_6 16
10. Biryukov, A., Tureček, P., Udovenko, A.: Algorithmic toolkit for linearization of S-boxes - Supporting code (2026). https://doi.org/10.5281/zenodo.18715466 2
11. Bouvier, C., Grassi, L., Khovratovich, D., Koschatko, K., Rechberger, C., Schmid, F., Schofnegger, M.: Skyscraper: Fast hashing on big primes. Cryptology ePrint Archive, Report 2025/058 (2025), https://eprint.iacr.org/2025/058 20
12. Brinkmann, M., Leander, G.: On the classification of APN functions up to dimension five. DCC **49**(1-3), 273–288 (2008). https://doi.org/10.1007/s10623-008-9194-6 17
13. Carlet, C.: Bounds on the nonlinearity of differentially uniform functions by means of their image set size, and on their distance to affine functions. IEEE Transactions on Information Theory **67**(12), 8325–8334 (2021). https://doi.org/10.1109/TIT.2021.3114958 1, 2, 5
14. Carlet, C.: Two generalizations of almost perfect nonlinearity. Journal of Cryptology **38**(2), 20 (Feb 2025). https://doi.org/10.1007/s00145-025-09538-5 4
15. Carlet, C., Charpin, P., Zinoviev, V.A.: Codes, bent functions and permutations suitable for DES-like cryptosystems. DCC **15**(2), 125–156 (1998). https://doi.org/10.1023/A:1008344232130 17
16. Chen, L., Fu, F.: On the nonlinearity of multi-output boolean functions. Acta Scientificiarum Naturalium University Nankaiensis **34**(4), 28–33 (2001), https://www.alljournals.cn/view_abstract.aspx?pcid=01BA20E8BA813E1908F3698710BBFEFEE816345F465FEBA5&cid=96E6E851B5104576C2DD9FC1FBCB69EF&jid=2F663905325EBF0C8638CFC155B7BC91&aid=A4F31984B5D18D95&yid=14E7EF987E4155E6 1
17. Chen, S., Guo, J., List, E., Shi, D., Zhang, T.: Diving deep into the preimage security of AES-like hashing. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part I. LNCS, vol. 14651, pp. 398–426. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58716-0_14 2
18. Courtois, N.T., Amiel, F., de Fonvillars, A.B.: On maximum size simultaneous linear approximations in Ascon and Keccak and related translation and differential properties. Cryptology ePrint Archive, Report 2024/802 (2024), https://eprint.iacr.org/2024/802 1, 6, 15
19. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on Keccak-224 and Keccak-256. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 442–461. Springer, Berlin, Heidelberg (Mar 2012). https://doi.org/10.1007/978-3-642-34047-5_25 22

20. Gorodilova, A.A., Tokareva, N.N., Agievich, S.V., Beterov, I.I., Beyne, T., Budaghyan, L., Carlet, C., Dhooghe, S., Idrisova, V.A., Kolomeec, N.A., Kutsenko, A.V., Malygina, E.S., Mouha, N., Pudovkina, M.A., Sica, F., Udovenko, A.N.: An overview of the eight international olympiad in cryptography "Non-Stop University CRYPTO". Sibirskie Èlektronnye Matematicheskie Izvestiya [Siberian Electronic Mathematical Reports] **19**(1), A9–A37 (2022). https://doi.org/10.33048/semi.2022.19.023, http://mi.mathnet.ru/semr1488 2
21. Grassi, L., Khovratovich, D., Koschatko, K., Rechberger, C., Schofnegger, M., Schröppel, V., Wu, Z.: Poseidon(2)b: Binary field versions of Poseidon/Poseidon2. IACR Communications in Cryptology **2**(4) (2026). https://doi.org/10.62056/a66ce0zn4 20
22. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: Monolith: Circuit-friendly hash functions with new nonlinear layers for fast and constant-time implementations. IACR Trans. on Symmetric Cryptology **2024**(3), 44–83 (2024). https://doi.org/10.46586/tosc.v2024.i3.44-83 20
23. Grassi, L., Khovratovich, D., Schofnegger, M.: Poseidon2: A faster version of the poseidon hash function. In: El Mrabet, N., De Feo, L., Duquesne, S. (eds.) AFRICACRYPT 23. LNCS, vol. 14064, pp. 177–203. Springer, Cham (Jul 2023). https://doi.org/10.1007/978-3-031-37679-5_8 20
24. Guido, B., Joan, D., Michaël, P., Gilles, V.: Cryptographic sponge functions. 2011-STMicroelectronics NXP Semiconductors, Version 0.1 January 14 (2011) 18
25. Guo, J., Liao, G., Liu, G., Liu, M., Qiao, K., Song, L.: Practical collision attacks against round-reduced SHA-3. Journal of Cryptology **33**(1), 228–270 (Jan 2020). https://doi.org/10.1007/s00145-019-09313-3 2
26. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Berlin, Heidelberg (Sep / Oct 2011). https://doi.org/10.1007/978-3-642-23951-9_22 2
27. Kazymyrov, O., Kazymyrova, V., Oliynykov, R.: A method for generation of high-nonlinear S-boxes based on gradient descent. Cryptology ePrint Archive, Paper 2013/578 (2013), https://eprint.iacr.org/2013/578 16
28. Kim, S., Ha, J., Son, M., Lee, B., Moon, D., Lee, J., Lee, S., Kwon, J., Cho, J., Yoon, H., Lee, J.: AIM: Symmetric primitive for shorter signatures with stronger security. In: ACM CCS 2023. p. 401–415. ACM, New York, NY, USA (2023). https://doi.org/10.1145/3576915.3616579 2
29. Langevin, P.: Classification of APN cubics in dimension 6 over GF(2). (2012), http://langevin.univ-tln.fr/project/apn-6/apn6.html 17
30. Liu, J., Mesnager, S., Chen, L.: On the nonlinearity of S-boxes and linear codes. Cryptogr. Commun. **9**(3), 345–361 (2017). https://doi.org/10.1007/S12095-015-0176-Z 1, 5, 12, 17
31. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT'93. LNCS, vol. 765, pp. 386–397. Springer, Berlin, Heidelberg (May 1994). https://doi.org/10.1007/3-540-48285-7_33 1
32. Matsui, M.: New block encryption algorithm MISTY. In: Biham, E. (ed.) FSE'97. LNCS, vol. 1267, pp. 54–68. Springer, Berlin, Heidelberg (Jan 1997). https://doi.org/10.1007/BFb0052334 2
33. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In: Rueppel, R.A. (ed.) EUROCRYPT'92. LNCS, vol. 658, pp. 81–91. Springer, Berlin, Heidelberg (May 1993). https://doi.org/10.1007/3-540-47555-9_7 1
34. Nagy, G.P.: On the minimum Hamming distance between vectorial Boolean and affine functions. Cryptography and Communications **17**(6), 1703–1720 (Nov 2025). https://doi.org/10.1007/s12095-025-00808-4 1, 5, 6, 14, 15, 17
35. Nagy, G.P.: Sidon sets, thin sets, and the nonlinearity of vectorial Boolean functions. Journal of Combinatorial Theory, Series A **212**, 106001 (2025). https://doi.org/10.1016/j.jcta.2024.106001 5
36. Nyberg, K.: On the construction of highly nonlinear permutations. In: Rueppel, R.A. (ed.) EUROCRYPT'92. LNCS, vol. 658, pp. 92–98. Springer, Berlin, Heidelberg (May 1993). https://doi.org/10.1007/3-540-47555-9_8 1
37. Nyberg, K.: Differentially uniform mappings for cryptography. In: Helleseth, T. (ed.) EUROCRYPT'93. LNCS, vol. 765, pp. 55–64. Springer, Berlin, Heidelberg (May 1994). https://doi.org/10.1007/3-540-48285-7_6 1, 14
38. Nyberg, K., Knudsen, L.R.: Provable security against differential cryptanalysis. In: Brickell, E.F. (ed.) Advances in Cryptology – CRYPTO'92. LNCS, vol. 740, pp. 566–574. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (1993). https://doi.org/10.1007/3-540-48071-4_41 4
39. Perrin, L., Udovenko, A.: Exponential s-boxes: a link between the s-boxes of BelT and Kuznyechik/Streebog. IACR Trans. Symm. Cryptol. **2016**(2), 99–124 (2016). https://doi.org/10.13154/tosc.v2016.i2.99-124, https://tosc.iacr.org/index.php/ToSC/article/view/567 16
40. Perrin, L., Wiemer, F., Stennes, L.: S-boxes used in cryptographic schemes. (2019), https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/sboxes.html, SageMath module 15
41. Qiao, K., Song, L., Liu, M., Guo, J.: New collision attacks on round-reduced Keccak. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 216–243. Springer, Cham (Apr / May 2017). https://doi.org/10.1007/978-3-319-56617-7_8 2
42. Ryabov, V.: Nonlinearity of APN functions: comparative analysis and estimates. [Prikladnaya Diskretnaya Matematika] (61), 15–27 (2023). https://doi.org/10.17223/20710410/61/2, https://www.mathnet.ru/rus/pdm810 1, 5

43. Sage Developers, T.: SageMath, the Sage Mathematics Software System (Version 10.7) (2025), https://www.sagemath.org 14
44. Sasaki, Y., Todo, Y.: New algorithm for modeling S-box in MILP based differential and division trail search. In: Farshim, P., Simion, E. (eds.) Innovative Security Solutions for Information Technology and Communications. pp. 150–165. Springer International Publishing, Cham (2017) 18
45. Yang, M., Wang, Z., Meng, Q., Han, L.: Evolutionary design of s-box with cryptographic properties. In: 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications Workshops. pp. 12–15 (2011). https://doi.org/10.1109/ISPAW.2011.59 15
46. Zaverucha, G., Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D.: Picnic. Tech. rep., National Institute of Standards and Technology (2017), available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions 2
47. Zhang, K., Wang, Q., Yu, Y., Guo, C., Cui, H.: Algebraic attacks on round-reduced rain and full AIM-III. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part III. LNCS, vol. 14440, pp. 285–310. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8727-6_10 2, 15, 18, 21, 22, 27

A Proof of the improved lower bound

Proposition 6. Let $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, $n \geq 4$ *and* $m \leq 2n-3$. *Then,* $\mathsf{VecLin}_S \geq n+2$.

Proof. For an even integer $k = 2l \geq 4$ and a given map $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, the following method shows how to create k-zero-sums for S, in case they exist at all. The idea is to go through all subsets consisting of l pairwise distinct points $x_1, \ldots, x_l \in \mathbb{F}_2^n$ and store the values

$$\begin{pmatrix} x_1 + \cdots + x_l \\ S(x_1) + \cdots + S(x_l) \end{pmatrix} \in \mathbb{F}_2^{n+m}.$$

This way we can hope to find a collision

$$\begin{pmatrix} x_1 + \cdots + x_l \\ S(x_1) + \cdots + S(x_l) \end{pmatrix} = \begin{pmatrix} x'_1 + \cdots + x'_l \\ S(x'_1) + \cdots + S(x'_l) \end{pmatrix}$$

caused by two different sets $\{x_1, \ldots, x_l\} \neq \{x'_1, \ldots, x'_l\}$, which is nothing else than

$$\begin{pmatrix} x_1 + \cdots + x_l + x'_1 + \cdots + x'_l \\ S(x_1) + \cdots + S(x_l) + S(x'_1) + \cdots + S(x'_l) \end{pmatrix} = 0,$$

i.e., a k-zero-sum if $\tilde{l} \coloneqq |\{x_1, \ldots, x_l\} \cap \{x'_1, \ldots, x'_l\}| = 0$. In general, due to a possible cancellation of common elements of the sets $\{x_1, \ldots, x_l\}$ and $\{x'_1, \ldots, x'_l\}$, the above zero-sum is a $(k - 2\tilde{l})$-zero-sum where $0 \leq \tilde{l} \leq l-2$. The case $\tilde{l} = l-1$ is not possible because otherwise there would be indices i and j remaining after cancellation such that $x_i + x'_j = 0$, that is $x_i = x'_j$, and thus $\{x_1, \ldots, x_l\} = \{x'_1, \ldots, x'_l\}$, a contradiction.

Now, regarding the existence of k-zero-sums, let us consider $l = 3$, i.e., we want to create 6-zero-sums. For this, the input dimension n has to be at least 4, otherwise there are no 6-zero-sums (see Proposition 1). There are $\binom{2^n}{l}$ sets consisting of l pairwise distinct elements of $\mathbb{F}_2^n$. Thus, there will be at least one collision (in the way described above) if

$$\binom{2^n}{3} = \frac{2^n(2^n-1)(2^n-2)}{6} > 2^{n+m},$$

which can be written as

$$(2^n - 1)(2^n - 2) > 6 \cdot 2^m.$$

This inequality is satisfied for $m \leq 2n-3$ since, by substituting 2^n by α, we have

$$6 \cdot 2^m \leq 6 \cdot 2^{2n-3} = \frac{3}{4}\alpha^2 < (\alpha - 1)(\alpha - 2) = (2^n - 1)(2^n - 2)$$

for $\alpha \geq 12$, i.e., for $n \geq 4$. Thus, we get a 6-zero-sum (or a 4-zero-sum).

Given such a zero-sum, we extend the affine basis of the zero-sum (consisting of 5 or 3 points respectively) arbitrarily to an affine basis of the full input space $\mathbb{F}_2^n$ (consisting of $n+1$ points), and assign $A(x) = S(x)$ for all inputs x in the basis. Clearly, this defines an affine map $\mathbb{F}_2^n \to \mathbb{F}_2^m$ and the zero-sum ensures that the omitted extra point x' from the zero-sum (it can not be included in the final basis due to the dependency) also satisfies $A(x') = S(x')$, implying $\mathsf{VecLin}_S \geq n+2$. □

B Exhaustive extension algorithm

Here, we present an exhaustive extension algorithm. Its pseudocode is given in Algorithm 5, where we represent the elements of $\mathbb{F}_2^n$ as integers of $\{0,\dots,2^n-1\}$ and the vector addition $+\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ as $\oplus\colon \{0,\dots,2^n-1\} \times \{0,\dots,2^n-1\} \to \{0,\dots,2^n-1\}$.
Line 6 has the effect that by appending new points to the main clique according to an order – in this case the usual order '$<$' on $\mathbb{Z}$ –, each affine approximation A of S (with $|S \cap A| \geq \mathsf{LB}$) is reached only once, i.e., different walks that could potentially lead to the same affine approximation are avoided. Indeed, every approximation set $X = S \cap A$ can only be achieved by choosing minimal in X element a' outside of the current clique and coset, since choosing another $a'' > a'$ element would update the bound to a'' and a' would never be added to the solution. The upper bound $2^n - 2^{n-1-\text{depth}}$ ensures that a new point which is to be appended to the main clique cannot be too large because otherwise, since the next points which are to be appended afterwards can be only larger, we cannot reach a set of full affine rank (i.e., $n+1$). For the same reason, it is sufficient to start with all a such that $a < 2^{n-1}$, see line 35. Thus, even if we set LB to 0 or to a value smaller than $n+1$, we always have $|S \cap A| \geq n+1$, or, in other words, $|S \cap A| \geq \max(\mathsf{LB}, n+1)$. Although there are affine approximations A of S with $|S \cap A| < n+1$ in general, we don't consider them because we are interested in maximizing $|S \cap A|$, and thus, we want to exceed the bounds stated in Proposition 2 and Proposition 6.
Lines 23–26 in Algorithm 5 establish a break condition according to the sum of the maximal clique lengths per coset plus the length of the main clique. Clearly, this is an upper bound for $|S \cap A|$ because we always merge cliques from different cosets. This condition is an important optimization because it allows the algorithm to cut off subtrees of the computation tree and to run it with different values of LB. If the algorithm takes too much time for a specific LB, one can instead increase LB and try again. If the algorithm then finishes without finding any affine approximation, it follows that the vectorial linearity is smaller than LB.

Complexity analysis For simplicity, we assume that the probability for two points (outside the main coset) to merge is always 2^{-m}. For the following reasoning, we consider the cliques outside the main coset to be equally distributed among the cosets and inside each coset. Let $l_d \in \mathbb{Q}$ denote the average clique length (outside the main coset) at the end of depth d, and let $N_d \in \mathbb{Q}$ denote the average number of cliques (outside the main coset) at the end of depth d. Clearly, we have $l_0 = 1$, $N_0 = 2^n - 1$, and $N_d \cdot l_d = 2^n - 2^d$ for $d \in \{0,\dots,n\}$. Then $M_d :=$ MaxReach from line 23 in Algorithm 5 in depth d is approximately

$$M_d = 1 + \sum_{i=0}^{d-1} l_i + (2^{n-d} - 1) l_d$$

because we start with one point and in each depth, we append to it a clique outside the main coset, which gives us the term $1 + \sum_{i=0}^{d-1} l_i$ for the length of the main clique. The last summand results from the number of remaining cosets times the average clique length there. Since for small depths (i.e., in the beginning of the process), mergers rarely occur, l_d grows slowly. To be more precise, we have $l_d = \frac{2^n - 2^d}{N_d}$, and we can find the following recurrence relation for N_d:

$$N_d = \frac{2^n - 2^d}{2^n - 2^{d-1}} N_{d-1} - (2^n - 2^d) 2^{d-2-m}.$$

Here, the first summand is the average number of cliques outside the main coset before the merging process begins, which can be expressed as the average number of cliques in the previous depth scaled by the factor $\frac{2^n - 2^d}{2^n - 2^{d-1}}$ because the main coset grows in each step, and thus, the number of remaining cliques to consider (i.e., outside the main coset) decreases accordingly. The second summand is the average number of mergers that occur, and since each merger reduces the number of cliques by 1, we subtract it accordingly. This summand results from the number of points outside the main clique (i.e., $2^n - 2^d$) times the probability for each point to merge, which is always 2^{-m} by assumption. The factor 2^{d-2} results from the number of input differences v that we check for each point, which is 2^{d-1}, and finally, we divide by 2 due to symmetry.
The above recurrence relation for N_d can be expressed as the following explicit term, which can be shown by induction (for example):

$$N_d = (2^n - 2^d)(1 + 2^{-1-m} - 2^{d-1-m}).$$

With this, we get $l_d = (1 + 2^{-1-m} - 2^{d-1-m})^{-1}$, which we can approximate by $(1 - 2^{d-1-m})^{-1}$ because $1 + 2^{-1-m} \approx 1$. Using the Taylor expansion $(1-x)^{-1} \approx 1 + x$ up to degree 1, we get $l_d \approx 1 + 2^{d-1-m}$.

Algorithm 5 Extension algorithm (exhaustive)

Inputs: function $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, lower bound $\mathsf{LB} \in \mathbb{Z}_+$
Output: set $\mathcal{A}$ of affine approximations $A\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ of S with $S \cap A \geq \mathsf{LB}$

function RecurseExtension(S, depth, a, cliques, cosets, LS, LS_x, LB)
  if depth $= 1$ **then**
    cliques $\leftarrow \{\{x\} : x \in \{0, \ldots, 2^n - 1\}\}$
    cosets $\leftarrow \{\{x\} : x \in \{0, \ldots, 2^n - 1\}\}$
  end if
  $C \leftarrow \{\min_{\mathbb{Z}}(c) : c \in \text{cliques}\} \cap]a, 2^n - 2^{n-1-\text{depth}}[$ ▷ Open interval
  for all $a' \in C$ **do**
    $\Delta_x \leftarrow a \oplus a'$
    if $\Delta_x \in \mathsf{LS}_x$ **then**
      continue
    end if
    $\Delta_y \leftarrow S(a) \oplus S(a')$
    $\Delta_{xy} \leftarrow \Delta_x \cdot 2^m + \Delta_y$
    cliques$'$ $\leftarrow$ cliques
    cosets$'$ $\leftarrow$ cosets
    $\mathsf{LS}' \leftarrow \mathsf{LS}$
    for all $v \in \mathsf{LS}$ **do**
      $v \leftarrow v \oplus \Delta_{xy}$
      $\mathsf{LS}' \leftarrow \mathsf{LS}' \cup \{v\}$
      Merge cliques$'$ according to v using a precomputed DDT of input pairs
    end for
    Merge cosets$'$ according to Δ_x
    MaxReach $\leftarrow \sum_{X \in \text{cosets}'} \max\{\#[x]_{\text{cliques}'} : x \in X \wedge [\min([x]_{\text{cliques}'}) > a' \vee a \in X]\}$
    if MaxReach $< \mathsf{LB}$ **then**
      continue
    end if
    $\mathsf{LS}'_x \leftarrow \mathsf{LS}_x \cup (\Delta_x \oplus \mathsf{LS}_x)$
    if depth $= n$ **then**
      $A \leftarrow$ affine map built from equivalence class $[a]_{\text{cliques}}$
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{A\}$
    else RecurseExtension(S, depth $+ 1$, a', cliques$'$, cosets$'$, LS', LS'_x, LB)
    end if
  end for
end function

$\mathcal{A} \leftarrow \{\}$
for all $a \in \{0, \ldots, 2^{n-1} - 1\}$ **do**
  RecurseExtension($S, 1, a$, None, None, $\{0\}, \{0\}, \mathsf{LB}$)
end for

Hence, we can write

$$\begin{aligned} M_d &\approx 1 + \sum_{i=0}^{d-1} (1 + 2^{i-1-m}) + (2^{n-d} - 1)(1 + 2^{d-1-m}) \\ &= 1 + d + (2^d - 1)2^{-1-m} + 2^{n-d} + 2^{n-1-m} - 1 - 2^{d-1-m} \\ &= d - 2^{-1-m} + 2^{n-d} + 2^{n-1-m} \\ &= d + 2^{n-d} + \varepsilon, \end{aligned}$$

where $\varepsilon \coloneqq 2^{n-1-m} - 2^{-1-m} = 2^{-1-m}(2^n - 1) \approx 2^{n-1-m}$. We see that the dominant term for smaller d is 2^{n-d}, whereas it is d for bigger d. This shows that the smaller LB, the longer the algorithm takes because in such a case, it has to reach a greater depth d so that MaxReach can unfold its filtering/cutting effect. For $m = n \geq 7$, the heuristics shows that the minimal VecLin_S among all functions $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ is somewhere between $n+2$ and $2^{n/2}$, which means that the algorithm needs to run at least until depth $d = \lfloor n/2 \rfloor + 1 =: d^*$ so that MaxReach becomes smaller than a given $\mathsf{LB} < 2^{n/2}$. For this sort of hard S-boxes, we can estimate the complexity, assuming that the number of leaves in the computation tree is mainly due to the number of paths that are cut at depth d^*, so that only a negligible number of paths pass through depth d^*. In depth d, at most $2^n - 2^{d-1}$ elements are chosen each to be appended to the main clique and thus starting a new branch of the tree (if not filtered out by the MaxReach condition), which means that, with the aforementioned assumption, we can roughly estimate the number of leaves by

$$2^n \prod_{i=1}^{d^*} (2^n - 2^{i-1})/(d^* + 1)! \approx 2^{n(d^*+1)}/(d^* + 1)!,$$

which gives us a rough complexity estimate of $\mathcal{O}(\frac{2^{nk}}{k!})$ with $k = \lfloor \frac{n}{2} \rfloor + 2$.

C Lookup tables

The following APN map $S_{15} : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ was found in [7]. We have shown that its vectorial linearity is equal to 15, the smallest known for $n = 8$ bits. Its lookup table is given by $S_{15} =$

```
[0, 1, 0, 6, 0, 12, 2, 9, 0, 24, 50, 45, 4, 17, 52, 38, 0, 48, 87, 96,
 100, 89, 49, 11, 8, 33, 109, 67, 104, 76, 15, 44, 0, 96, 107, 12, 173,
 192, 196, 174, 200, 177, 145, 239, 97, 21, 58, 73, 16, 65, 44, 122,
 217, 133, 231, 188, 208, 152, 222, 145, 29, 88, 17, 83, 0, 192, 76,
 139, 213, 24, 155, 81, 91, 130, 37, 251, 138, 94, 246, 37, 146, 99,
 137, 127, 35, 223, 58, 193, 193, 41, 232, 7, 116, 145, 95, 189, 32,
 129, 7, 161, 88, 244, 125, 214, 179, 11, 166, 25, 207, 122, 216, 106,
 162, 50, 210, 69, 190, 35, 204, 86, 57, 176, 123, 245, 33, 165, 97,
 226, 0, 130, 128, 5, 152, 23, 26, 146, 171, 48, 25, 133, 55, 161, 135,
 22, 181, 6, 98, 214, 73, 247, 156, 37, 22, 188, 243, 94, 238, 73, 9,
 169, 38, 197, 205, 41, 19, 253, 250, 19, 69, 191, 156, 97, 116, 131,
 175, 95, 131, 81, 63, 234, 210, 13, 108, 180, 232, 35, 102, 170, 189,
 123, 49, 240, 64, 3, 140, 200, 13, 67, 195, 138, 176, 234, 78, 19, 249,
 174, 5, 85, 103, 21, 252, 137, 78, 49, 215, 175, 159, 244, 54, 90, 178,
 212, 25, 120, 70, 100, 225, 196, 166, 137, 3, 43, 126, 69, 235, 215,
 154, 172, 13, 60, 113, 98, 129, 149, 245, 235, 7, 30, 65, 75, 131, 142,
 193, 198, 1, 1].
```

D On the multiplicative inverse in odd dimension

In this section, we explore the possibility of linearizing the multiplicative inverse function in the finite field for odd dimension $n \geq 7$. We show that the generalization of the technique from [47] does not lead to a similar result.

First, we try to split off a linear (monomial) part similar to the proposition:

$$x^{2^n-2} = x^{2^n-2-2^k} x^{2^k}.$$

By choosing $a \in \mathbb{F}_{2^n}^*$, we define a linear function $g_a \colon \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ by

$$g_a(x) = a^{2^n-2-2^k} x^{2^k}.$$

If there is another point $b = a\omega \in \mathbb{F}_{2^n}^*$ such that $g_a(b) = b^{-1}$, we have

$$g_a(b) = g_a(a\omega) = a^{2^n-2-2^k} a^{2^k} \omega^{2^k} = a^{-1}\omega^{2^k} = a^{-1}\omega^{-1} = b^{-1},$$

which results in the condition $\omega^{2^k+1} = 1$. Thus, since we also have $\omega^{2^n-1} = 1$, the order of ω is a divisor of $d := \gcd(2^n - 1, 2^k + 1)$. We will now show $d = 1$, which leads to $\omega = 1$, a contradiction. This means that g_a and the inverse function coincide only for 0 and a, and thus, other methods are necessary in order to determine the vectorial linearity of the inverse function.

We use the following standard lemma.

Lemma 2. *Let $a, b \in \mathbb{Z}_{\geq 0}$. Then,* $\gcd(2^a - 1, 2^b - 1) = 2^{\gcd(a,b)} - 1$.

Proof. Follows from the identity $2^b - 1 = (2^a - 1)2^{b-a} + 2^{b-a} - 1$ allowing to apply the Euclidean algorithm on the exponents. □

Since $d \mid 2^k + 1$, we have $d \mid 2^{2k} - 1$, and thus $d \mid \gcd(2^n - 1, 2^{2k} - 1)$. By applying the previous lemma, we get $\gcd(2^n - 1, 2^{2k} - 1) = 2^{\gcd(n,2k)} - 1$. Since n is odd, we have $\gcd(n, 2k) = \gcd(n, k)$, and thus $2^{\gcd(n,2k)} - 1 = 2^{\gcd(n,k)} - 1 = \gcd(2^n - 1, 2^k - 1)$ by applying the previous lemma again. Hence, we have $d \mid \gcd(2^n - 1, 2^k - 1)$, which means that $d \mid 2^k - 1$. Together with $d \mid 2^k + 1$, it follows that $d \mid 2$. Since d is odd, we have $d = 1$.

E Free S-boxes in CICO problems

We recall standard techniques to bypass some rounds and S-boxes in the CICO problem. Here, we will consider an SPN starting and ending with an S-box layer, showing that the first and the last S-box layers add no security, and that is why in Subsection 7.2 we consider an SPN starting and ending with a linear layer.[5]

In the first round, the first c S-boxes are applied to zero inputs, thus the outputs are constant and there is no need to linearize them. The last r S-boxes correspond to free variables, however we can define variables in the output of the S-boxes, and, after solving the problem, precompute their preimages under the S-boxes. This allows to always skip the first S-box layer.

In the second round, whenever $r > c$, we can use the available degrees of freedom and deactivate $r - c$ S-boxes (fix their inputs/outputs to constants). The idea is that fixing $r - c$ outputs of the first MDS layer corresponds to linear relations on the variables, restricting the input space to a vector space of dimension c. This can also be applied to the second-to-last round, but it will spend the same degrees of freedom, so there is not much difference in the general setting.

In the last round, similarly to the first round, the first c S-boxes output zeroes and thus are inactive (their inputs can be precomputed). The other r S-boxes do not have any constraints on outputs and thus can be ignored. This is equivalent to truncating the last MDS layer to output only the first c elements and constraining them to constants.

Proposition 8. *The* $\mathrm{CICO}_{c,r}$ *problem can be reduced to a CICO problem on $(R-2)$-round SPN where the $r - c$ inputs are fixed and the remaining $2c$ inputs are restricted to a c-dimensional subspace; the outputs are restricted to an r-dimensional subspace. In total, the new problem has $m(R - 3) + 2c$ active (non-constant) S-boxes and is expected to have one solution on average (c degrees of freedom and c constraints over $\mathbb{F}_2^n$).*

[5] This was suggested by an anonymous reviewer.