

What to Guess in Key-Recovery Attacks?

Tim Beyne¹, Gregor Leander², Patrick Neumann³, Yevhen Perekuda²,
and Michiel Verbauwhede¹

¹ KU Leuven, Leuven, Belgium `firstname.lastname@esat.kuleuven.be`

² Ruhr University Bochum, Bochum, Germany `firstname.lastname@rub.de`

³ Inria, Paris, France `patrick.neumann@inria.fr`

Abstract. Determining the precise parts of the key that need to be guessed in a key-recovery attack is fundamental for judging its cost: if the same attack can be executed by guessing less key material, then the cipher’s resistance against this attack is overestimated. Although a multitude of prior works provide upper bounds on the key material required, and although these bounds might be tight in some special cases, a precise evaluation of the required key material and the tightness of these bounds is still missing.

We remedy this by enumerating linear trails to iteratively compute the affine hull of the support of the Fourier transform of the key-recovery map. This leads to a generic and practical algorithm that identifies the smallest subspace of key material to be guessed. This algorithm is ready to be used in many different attacks and for a large variety of cipher structures. We demonstrate its impact by showcasing improvements on several published integral, linear, differential-linear, and zero-correlation attacks on the block ciphers PRESENT, SIMON, SKINNY, and GIFT.

Keywords: Symmetric key · cryptanalysis · key recovery · minimal key material · integral attack · linear attack · differential-linear attack · zero-correlation attack · PRESENT · SIMON · SKINNY · GIFT

1 Introduction

The continuous refinement of cryptanalytic techniques is essential for obtaining a realistic understanding of the security margins of cryptographic primitives. Symmetric cryptography plays a leading role here, with its unmatched variety of attack vectors developed over the last decades.

In a nutshell, attacks on symmetric primitives usually decompose into two parts: a distinguisher and a key-recovery part. The distinguisher is a key-independent property that tells a round-reduced version of the cipher apart from a random permutation. Classical distinguishers range from differential [10] and linear [31] distinguishers and their variants to integral distinguishers [19], among others.

Once such a distinguisher is found, it is usually turned into a key-recovery attack by appending, prepending, or otherwise surrounding it with a small number of rounds. The basic idea is to iterate over all possible keys involved in

these additional rounds and check, for each possible key, whether the distinguishing property holds. In integral cryptanalysis, one partially decrypts a set of ciphertexts and checks a zero-sum property. In linear, zero-correlation (ZC), and differential-linear (DL) cryptanalyses, one evaluates a correlation-type statistic depending on the partial decryption. In meet-in-the-middle (MITM) attacks, one computes partial intermediate state values from two sides and checks whether they match. Keys for which the distinguishing property holds are kept as possible key candidates, while keys for which it does not hold are rejected.

In modern block-cipher cryptanalysis, much effort has been devoted to constructing stronger distinguishers: longer differential and linear trails, more accurate estimates of the correlations of linear approximations, stronger integral and division-property distinguishers, and more efficient meet-in-the-middle separations.

A distinguisher can be more or less suitable for adding key-recovery rounds, and a second line of work is dedicated to making key-recovery evaluation more efficient.

How to Guess? Several families of techniques have been proposed to speed up the evaluation of key recovery. In integral cryptanalysis, the main examples are partial sums [23], FFT-based key recovery [35], and combinations of partial sums and FFT [21]. In linear and zero-correlation cryptanalysis, the corresponding tools include FFT-based evaluation [18], Walsh-spectrum puncturing [25] and related works [24,26,27] and quantum variants of key recovery [32,3]. In differential cryptanalysis, automated key-recovery-strategy [13], that take the key-scheduling into account, and generic differential key recovery [33] are used to optimize the cost of guessing key bits. In differential-linear cryptanalysis, partitioning techniques [7] and semi-unconstrained search with partition trees [17] serve a similar role. In meet-in-the-middle attacks, partial matching, splice-and-cut [20], bicliques [11], and sieve-in-the-middle [15] improve how effectively the matching state is evaluated.

However, most of this work, with the notable exception of Walsh-spectrum puncturing, optimizes the *way* in which the key-recovery statistic is computed, but not what actually has to be guessed.

What to Guess? In all these cases, the object evaluated during key recovery can be viewed as a real-valued (family of) functions

$$\begin{aligned} f_k : \mathbb{F}_2^n &\longrightarrow \mathbb{R} \\ c &\longmapsto f_k(c), \end{aligned}$$

where $k \in \mathbb{F}_2^\kappa$ denotes the involved key and c denotes the public data. As an attacker, one has to iterate over the whole family to filter out wrong key candidates. In many cases, and in all interesting ones, the size of the family, i.e., the number of different functions in the family, is significantly smaller than 2^κ . Thus, instead of iterating over all keys $k \in \mathbb{F}_2^\kappa$, it is crucial for the efficiency of an attack to identify a close-to-minimal set $M \subset \mathbb{F}_2^\kappa$ such that

$$\{f_k : k \in \mathbb{F}_2^\kappa\} = \{f_k : k \in M\}.$$

Once such a set has been found, iterating over all key candidates in M is sufficient for the attack. While, in principle, the minimal set M of guessed keys could be arbitrary, for structural reasons as well as for simplicity, M is always considered to be an affine subspace.

The central problem considered in this paper is very fundamental: we ask *how to find in a generic way the smallest affine subspace M of key candidates*.

The fact that some key information may be irrelevant to a key-recovery test is not new. Equivalent-key phenomena have appeared in several forms throughout symmetric cryptanalysis. However, the first attempts to automate their search process were made only recently. In integral attacks, at EUROCRYPT’26, Beierle et al. [6] recently showed that wrong keys may exhibit strong clustering into equivalence classes. Their affine related-key subspace trails approach detects key differences that preserve the integral property and lead to improved attacks on SPN primitives. However, this approach was used only for integral key recovery and to bit-permutation SPNs with full-state key addition. Moreover, the reported computations become especially expensive starting from four key-recovery rounds. Thus, while this paper addresses the same problem, in a slightly more narrow way, the resulting approach lacks both in efficiency, and as a consequence of that as well as generality.

Another relevant direction is the key-dependency analysis used in recent MITM cryptanalysis of **Chilow** at CRYPTO’26. Technically, our work can be seen as a generalization of the approach given there. In particular, Lambooij et al. [29] propagate key dependencies, represented by masks, round by round in order to identify key information that is not needed in their MITM matching. Nevertheless, the original method was designed for a particular cryptanalytic setting and focuses on quadratic round functions. Moreover, as will be shown later, it can overestimate the key support when key schedules are taken into account.

Another research line that is closely related – even though it has focused mainly on ARX ciphers – is the aforementioned Walsh-spectrum puncturing idea. This technique, which can itself be seen as a modern variant of pseudo-neutral bits introduced by Aumasson et al. [1], also aims to minimize the amount of key information that needs to be guessed. Just like the approach we will develop below, it relies on a Walsh (or Fourier) spectrum perspective and is therefore technically similar. Furthermore, the use of trail enumeration and summation has already been explored in this context [27]. However, that approach goes significantly beyond the scope of our work: not only does it investigate which bits can be left unguessed entirely, but it also aims to identify bits that have only a minor influence on key recovery. This constitutes a much more difficult problem, which consequently proves to be less efficient, less exact, and less generic for the time being.

Finally, there have been isolated cases of finding linear key dependencies on the property bits in linear [34], DL [39], or even multiple attack types [14]. However, these dependencies were derived in a dedicated, often local, manner and, as a result, were error-prone. We show that this led to both types of drawbacks:

overestimating the required key material in the first case and, more interestingly, underestimating it in the second one. These examples reveal that a principled and generic method is useful not only for improving attacks, but also for checking existing key-recovery analyses.

To summarize, several significant steps have been made in the past, and from the perspective of Boolean functions, the problem has already been addressed in depth in terms of linear structures. However, the algorithmic problem of practically computing the minimal space of keys to be guessed in a generic way for a large set of modern ciphers has remained unsolved.

Our contribution

Building on previous work, we develop an automated and generic method for determining the effective key-guessing space in key-recovery attacks based on *linear-trail enumeration*.

Table 1: Number of guessed key bits before and after our optimization.

Cipher	Type	Before ^a	After ^a	Δ ^b	Source and Note
PRESENT	Integral	148(80)	108(67)	40(13)	13-round extension; Sec. 6.2.
SIMON32/64	Integral	87(87)	74(74)	13(13)	[37]; Two MITM branches; Sec. 7.1.
SIMON32/64	ZC	54(54)	45(45)	9(9)	[37]; Sec. 7.1.
SIMON96/96	Linear ^c	74(74)	70(70)	4(4)	[30]; Sec. 7.1.
SIMON96/144	Linear	130(126)	126(124)	4(2)	[30]; Sec. 7.1.
SIMON128/128	Linear	112(106)	108(102)	4(4)	[30]; Sec. 7.1.
SIMON128/192	Linear	180(172)	176(167)	4(5)	[30]; Sec. 7.1.
SIMON128/256	Linear	246(241)	242(237)	4(4)	[30]; Sec. 7.1.
GIFT-64	Diff.-linear	98(74)	101(77)	-3(-3)	[39]; a local linear-effect optimization undercounts the key space: EK_{16} [28, 26, 14] still affect the tested property and must be guessed; Sec. 6.1.
GIFT-128	Linear	71(61)	70(60)	1(1)	[34]; the published linear-effect analysis overestimates the key space by one bit: RK_0 [60] affects linearly the property and can be omitted; Sec. 6.1.
SKINNY-64	Integral	64(64)	60(60)	4(4)	[38]; two bits saved in each branch; Sec. 6.3.
SKINNY-128	Integral	128(128)	114(114)	14(14)	[38]; seven bits saved in each branch; Sec. 6.3.

^a An entry $x(y)$ gives the dimension on independent round-key variables, followed in parentheses by the count after key-schedule post-processing.

^b Here $\Delta = \text{Before} - \text{After}$, computed at both levels. A negative value means that our analysis corrects an undercount in the published key list rather than saving key bits.

^c Leurent et al. explicitly considered the SIMON key schedule and reported schedule-reduced independent counts in Tables 13 and 14. For SIMON128/128 and SIMON128/192, the ciphertext-side schedule-reduced baseline counts correspond to the special $(4, 0) \rightarrow (4, 1)$ approximation. However, their attack complexities in Table 10 use the total round-key count κ_g , corresponding to the values outside parentheses.

Our starting point is the well-known characterization of the smallest subspace of key candidates, or rather its complement, as the dual of the support of the Fourier coefficients of the key-recovery functions. The key idea is to use generic modeling techniques developed in recent years for finding linear trails to iteratively construct the linear span of this support. Taking into account the clustering of trails to identify trail cancellations allows us to avoid overestimating the constructed space.

Table 2: Comparison of optimized key-recovery attacks.

Cipher	Type	Rounds	Data	Time	Note and Source
PRESENT	Integral	12	2^{60}	$2^{28.3}$ Add. ^a	[6]
	Integral	13	2^{63}	2^{75} Enc.	Our 13-round extension, Sec. 6.2
SIMON32/64	Integral	21	2^{31}	$2^{54.59}$ Enc.	[37]
	Integral	21	2^{31}	$2^{52.75}$ Enc.	Our optimized key-recovery; also analyzed in [6], but only manually; Sec. 7.1
	ZC	20	2^{32}	$2^{56.96}$ Enc.	[37]
	ZC	20	2^{32}	$2^{51.21}$ Enc.	Our optimized key-recovery, Sec. 7.1
SIMON96/96	Linear	43	2^{94}	$2^{82.09}$ Enc.	[30]; complexity recomputed by us with $a = 23$
	Linear	43	2^{94}	$2^{79.26}$ Enc.	Our optimized key-recovery, $a = 27$, Sec. 7.1
SIMON96/144	Linear	45	2^{95}	$2^{133.89}$ Enc.	[30]; complexity recomputed by us with $a = 16$
	Linear	45	2^{95}	$2^{128.99}$ Enc.	Our optimized key-recovery, $a = 22$, Sec. 7.1
SIMON128/128	Linear	53	2^{127}	$2^{117.15}$ Enc.	[30]; complexity recomputed by us with $a = 18$
	Linear	53	2^{127}	$2^{109.44}$ Enc.	Our optimized key-recovery, $a = 28$, Sec. 7.1
SIMON128/192	Linear	55	2^{127}	$2^{184.21}$ Enc.	[30]; complexity recomputed by us with $a = 14$
	Linear	55	2^{127}	$2^{174.22}$ Enc.	Our optimized key-recovery, $a = 27$, Sec. 7.1
SIMON128/256	Linear	56	2^{126}	$2^{248.56}$ Enc.	[30]; complexity recomputed by us with $a = 13$
	Linear	56	2^{126}	$2^{240.84}$ Enc.	Our optimized key-recovery, $a = 22$, Sec. 7.1

^a In [6], the reported complexity refers only to the partial key-recovery filtering step and does not include the final brute-force term.

A key feature of the approach is its generality. The method is purposely stated at the level of the key-recovery function $f(k, c)$. It is therefore not tied to one cryptanalytic technique. The same framework applies to integral, linear, zero-correlation, differential-linear, and meet-in-the-middle key recovery, as long as the statistic is computed from bits or linear combinations of an intermediate state. Moreover, the method is not restricted to bit-permutation SPNs, full-state key addition primitives or quadratic round functions. For ciphers such as SIMON,

GIFT, or SKINNY, where only part of the state is affected by key material, the state trails clusters are projected through the actual key addition positions.

We also revisit the round-wise mask-propagation method inspired by [29], generalize it to the case of a non-quadratic function, and showcase its applicability not only to MITM. In our framework, it can be viewed as a faster but coarser approximation to the same projected Fourier support. The reason is that it may overestimate the true key support because it does not fully account for key-dependent clustering across several rounds. While we provide a simple example in which such behavior can happen, notably, for the applications covered in the paper, the results of both approaches coincided.

The method is fully automated in our applications. In particular, for cases where the approach of [6] for four key-recovery rounds was reported to take several days, our trail-enumeration approach computes the same effective key space in seconds or minutes.⁴

Table 1 shows the reductions and corrections obtained by our analysis, while Table 2 gives the resulting attack complexities improvement. In particular, we extend the integral key recovery on PRESENT to 13 rounds, improve integral, linear, and zero-correlation key-recovery complexities on SIMON. Additionally, we identify both overcounts and undercounts in published key lists for attacks on GIFT and SKINNY.

Outline. Section 3 formalizes key-recovery computations through the Fourier support of the real-valued target function $f(k, c)$ and explains the resulting classes of equivalent keys. Section 4 revisits and generalizes round-wise mask propagation introduced by Lambooi et al. [29] and explains why it may overestimate key dependencies. Section 5 introduces our linear-trail enumeration algorithm. The applications to SPN ciphers are given in Section 6, and the application to non-SPN is given in Section 7.

2 Preliminaries

The following two sections introduce the main concepts that our techniques are based on. These are linear algebra over the n -dimensional vector space $\mathbb{F}_2^n$ over the finite field $\mathbb{F}_2$ with two elements, and linear cryptanalysis.

2.1 Linear algebra

The sum of two vectors x and y in $\mathbb{F}_2^n$ is denoted by $x + y$, and corresponds to the bitwise xor operation. This is sometimes denoted by $\oplus$ in the literature, but note that $U \oplus V$ denotes the direct sum of two subspaces $U \subseteq \mathbb{F}_2^n$ and $V \subseteq \mathbb{F}_2^m$. The concatenation of two vectors x in $\mathbb{F}_2^n$ and y in $\mathbb{F}_2^m$ is denoted by $x \| y$, and is a vector in $\mathbb{F}_2^{n+m}$.

⁴ The source code and implementation accompanying this work are publicly available on Zenodo at <https://doi.org/10.5281/zenodo.22847782>.

The linear span of a subset S of $\mathbb{F}_2^n$, *i.e.* the set of all linear combinations of elements of S , is denoted by $\text{span}(S)$. Similarly, the affine hull of S is denoted by $\text{aff}(S)$. This is the smallest affine space that contains S . If $S \neq \emptyset$ and $a \in S$, then the affine span of S can be written as

$$\text{aff}(S) = a + \text{span}\{s + a : s \in S\}.$$

Let $\langle x, y \rangle$ denote the dot product of vectors x and y in $\mathbb{F}_2^n$. The orthogonal complement of a subset U of $\mathbb{F}_2^n$ is the vector space $U^\perp$ with

$$U^\perp = \{x \in \mathbb{F}_2^n : \langle x, u \rangle = 0 \text{ for all } u \in U\} = \text{span}(U)^\perp.$$

If U is a vector space, then $\dim U^\perp = n - \dim U$.

The quotient of $\mathbb{F}_2^n$ by a subspace V is the vector space $\mathbb{F}_2^n/V$ obtained by identifying pairs of elements that differ by a vector in V . For the purposes of this paper, we define the quotient space $\mathbb{F}_2^n/V$ as the set of cosets $x + V$ of V . This is itself a vector space. The dimension of $\mathbb{F}_2^n/V$ is equal to $n - \dim V$.

2.2 Linear cryptanalysis

Let us recall some common concepts from the context of linear cryptanalysis. The Fourier transform of a function $f: \mathbb{F}_2^n \rightarrow \mathbb{R}$ is the function $\hat{f}: \mathbb{F}_2^n \rightarrow \mathbb{R}$ defined by

$$\hat{f}(v) = \sum_{x \in \mathbb{F}_2^n} f(x) (-1)^{\langle v, x \rangle}.$$

If f is the sign function of a component function of a function $G: \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$, *i.e.*, $f(x) = (-1)^{\langle w, G(x) \rangle}$, then f corresponds to the correlation of the linear approximation with input mask v and output mask w of G :

$$C_{w,v}^G = \frac{1}{2^n} \hat{f}(v) = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle w, G(x) \rangle + \langle v, x \rangle}.$$

For the composition of functions, we recall the following well-known lemma.

Lemma 1. *Let $G_i: \mathbb{F}_2^{m_i} \rightarrow \mathbb{F}_2^{m_{i+1}}$ and $G = G_{r-1} \circ \dots \circ G_0$. It holds that*

$$C_{v_r, v_0}^G = \sum_{\substack{v_1, \dots, v_{r-1} \\ v_i \in \mathbb{F}_2^{m_i}}} \prod_{i=0}^{r-1} C_{v_{i+1}, v_i}^{G_i}.$$

Here, the tuple $(v_0, \dots, v_r)$ that contains all intermediate masks is referred to as a *linear trail*, and its correlation is $\prod_{i=0}^{r-1} C_{v_{i+1}, v_i}^{G_i}$.

Our main target will be a (partial) iterative cipher $E: \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$, which we describe as the composition of keyed round functions $R_i: \mathbb{F}_2^\kappa \times \mathbb{F}_2^{m_i} \rightarrow \mathbb{F}_2^{m_{i+1}}$, with $m_0 = n$ and $m_r = m$. This construction is depicted in Figure 1. Here, we consider that the round functions could include the key-scheduling. In our context,

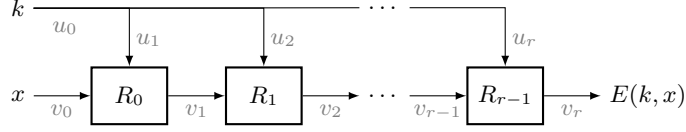


Fig. 1: An iterative cipher with a linear trail $(u_0, \dots, u_r, v_0, \dots, v_r)$ over it.

E will typically be the partial decryption (or encryption) from the ciphertext (or plaintext) side towards a distinguisher. The key space $\mathbb{F}_2^\kappa$ typically corresponds to the master key, but can also be seen as the concatenation of round keys or any other representation of the involved key material. The derivation of the key material used in each round is part of the round functions R_i .

For the iterative cipher E in Figure 1, Lemma 1 implies

$$C_{v_r, (u_0, v_0)}^E = \sum_{\substack{v_1, \dots, v_{r-1}, u_1, \dots, u_r \\ \text{s.t. } \sum_{i=1}^r u_i = u_0}} \prod_{i=0}^{r-1} C_{v_{i+1}, (u_{i+1}, v_i)}^{R_i}. \quad (1)$$

The condition $\sum_{i=1}^r u_i = u_0$ comes from the branching operations in the key path. For additional background about linear cryptanalysis, we refer the reader to other resources such as [8].

3 Theory: Classes of Equivalent Keys

As explained in the introduction, the key-recovery procedure in most attacks can be described in terms of a real-valued function that we call the key-recovery function in Section 3.1. This section formalizes the idea that the Fourier transform of this function reveals which key bits or linear combinations of key bits need to be guessed in the attack. More precisely, it reveals classes of equivalent keys, thereby reducing the number of key guesses.

Although our point of view — and therefore the presentation in this section, which provides the basis for the algorithms we develop in Sections 4 and 5 — is different, many of the ideas have been developed since the late 1980s, starting with the work of Evertse [22] and Lai [28] in terms of linear structures. We refer to [16] for an in-depth treatment of the relevant results.

3.1 Key-recovery function

The central object in the key-recovery phase of many attacks (Figure 2) is the evaluation of a function $f(k, c)$ that is computed by partial encryption or decryption under a guessed key. This function represents some value based on the intermediate state used to check whether a distinguishing property holds. In this paper, we model it as an arbitrary real-valued function

$$\begin{aligned} f: \mathbb{F}_2^\kappa \times \mathbb{F}_2^n &\longrightarrow \mathbb{R} \\ (k, c) &\longmapsto f(k, c). \end{aligned}$$

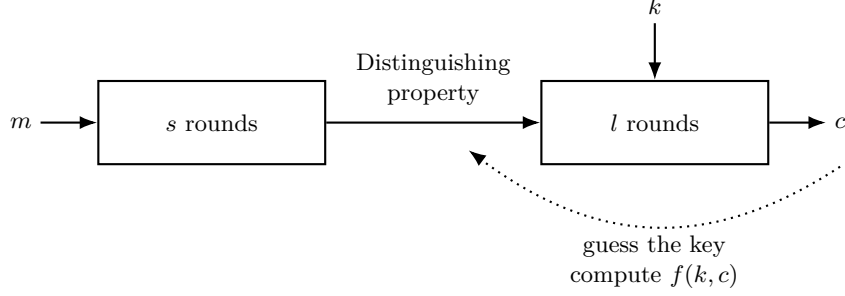


Fig. 2: Key-recovery procedure in block cipher cryptanalysis.

Here, $k \in \mathbb{F}_2^\kappa$ denotes a guess of the key variables that might be required for key recovery. These variables may be subkey bits, linear combinations of subkey bits, or master-key bits restricted to the part used by the partial computation. In applications, c may encode for example a ciphertext, a plaintext-ciphertext pair, or a ciphertext pair. For convenience, we assume without loss of generality that c is a ciphertext.

The precise choice of f depends on the attack. Typically, f is evaluated on N public ciphertext values, producing a key-dependent test statistic

$$t_k = \sum_{i=1}^N f(k, c_i).$$

The candidate key is retained if t_k satisfies the expected distinguishing property. For example, in an integral attack, f can be chosen to be integer-valued and we check whether $t_k \equiv 0 \pmod{2}$.

Example 1. In linear cryptanalysis, the key-recovery function is usually

$$f(k, c) = (-1)^{g(k, c)},$$

where $g: \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is the Boolean function derived from the partial decryption function for the key candidate k and ciphertext c .

As described in Section 2.2, since f is a real-valued function on $\mathbb{F}_2^\kappa \times \mathbb{F}_2^n$, it admits a Fourier expansion:

$$f(k, c) = \frac{1}{2^{\kappa+n}} \sum_{u \in \mathbb{F}_2^\kappa} \sum_{v \in \mathbb{F}_2^n} \hat{f}(u, v) (-1)^{\langle u, k \rangle + \langle v, c \rangle}. \quad (2)$$

Here, the function $\hat{f}: \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \rightarrow \mathbb{R}$ is the Fourier transform of f . Explicitly,

$$\hat{f}(u, v) = \sum_{k \in \mathbb{F}_2^\kappa} \sum_{c \in \mathbb{F}_2^n} f(k, c) (-1)^{\langle u, k \rangle + \langle v, c \rangle}. \quad (3)$$

Throughout this paper, we refer to $u \in \mathbb{F}_2^\kappa$ as the key mask. Similarly, we call $v \in \mathbb{F}_2^n$ the state mask. This terminology, which comes from linear cryptanalysis, will be clarified in Section 3.3.

3.2 Equivalent keys

Looking at Equation (2), we observe that if $\langle u, \delta \rangle = 0$ for all u such that $\widehat{f}(u, v) \neq 0$ for some v , then $f(k + \delta, c) = f(k, c)$ for all $k \in \mathbb{F}_2^\kappa$ and $c \in \mathbb{F}_2^n$. That is, the keys k and $k + \delta$ are equivalent. Likewise, if $\langle u, \delta \rangle = 1$, then $f(k + \delta, c) = f(k, c) + 1$ and the keys are equivalent up to a constant. In both cases we refer to the keys k and $k + \delta$ as *equivalent keys*.

This observation motivates the definition of the following set $\mathcal{A}(f)$ as the affine hull of the support of the Fourier transform of f with respect to the key variable:

$$\mathcal{A}(f) = \text{aff}(\{u \in \mathbb{F}_2^\kappa : \exists v \in \mathbb{F}_2^n \text{ with } \widehat{f}(u, v) \neq 0\}).$$

This can also be interpreted as the smallest affine subspace containing all key masks that have at least one non-zero Fourier coefficient. The reason for using the affine hull, rather than the vector space span, is that in many cases it is sufficient to determine $f(k, c)$ up to a constant factor in k . This leads to the following theorem.

Theorem 1 (Equivalence classes of keys). *Let $f: \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \rightarrow \mathbb{R}$ be a function with $\mathcal{A}(f) = a + U$ for a vector space U . For every c in $\mathbb{F}_2^n$, the map*

$$k \mapsto (-1)^{\langle a, k \rangle} f(k, c),$$

is constant on the cosets of $U^\perp$. In particular, this yields a well-defined function

$$\begin{aligned} \mathbb{F}_2^\kappa / U^\perp &\longrightarrow \mathbb{R} \\ k + U^\perp &\mapsto (-1)^{\langle a, k \rangle} f(k, c). \end{aligned}$$

Proof. Let $\delta \in U^\perp$. For all $u \in a + U$, we have $\langle u, \delta \rangle = \langle a, \delta \rangle$. By the Fourier expansion of f in Equation (2),

$$\begin{aligned} f(k + \delta, c) &= \frac{1}{2^{\kappa+n}} \sum_{u \in \mathcal{A}(f)} \sum_{v \in \mathbb{F}_2^n} \widehat{f}(u, v) (-1)^{\langle u, k+\delta \rangle + \langle v, c \rangle} \\ &= (-1)^{\langle a, \delta \rangle} \frac{1}{2^{\kappa+n}} \sum_{u \in \mathcal{A}(f)} \sum_{v \in \mathbb{F}_2^n} \widehat{f}(u, v) (-1)^{\langle u, k \rangle + \langle v, c \rangle} \\ &= (-1)^{\langle a, \delta \rangle} f(k, c). \end{aligned}$$

Hence, any two keys k and $k + \delta$ are equivalent in the sense that, for all c in $\mathbb{F}_2^n$,

$$(-1)^{\langle a, k+\delta \rangle} f(k + \delta, c) = (-1)^{\langle a, k \rangle} f(k, c).$$

Equivalently, $k \mapsto (-1)^{\langle a, k \rangle} f(k, c)$ is constant on every coset of $U^\perp$, and the quotient map is well defined. $\square$

Example 2. Let $k = (k_0, k_1) \in \mathbb{F}_2^2$, $c \in \mathbb{F}_2$, and key-recovery map $f(k, c) = (-1)^{k_0 + k_1 c}$, which is trivially constant in k_0 up to sign.

Writing $e_0 = (1, 0)$ and $e_1 = (0, 1)$, the non-zero Fourier coefficients are

$$\widehat{f}(e_0, 0) = \widehat{f}(e_0, 1) = \widehat{f}(e_0 + e_1, 0) = 4, \widehat{f}(e_0 + e_1, 1) = -4,$$

while all other coefficients are zero. Fourier inversion therefore gives

$$f(k, c) = \frac{1}{2} \left((-1)^{k_0} + (-1)^{k_0+c} + (-1)^{k_0+k_1} - (-1)^{k_0+k_1+c} \right).$$

Consequently,

$$\mathcal{A}(f) = e_0 + \text{span}\{e_1\} =: a + U.$$

Since $U^\perp = \text{span}\{e_0\}$, we have

$$f(k + e_0, c) = (-1)^{\langle a, e_0 \rangle} f(k, c) = -f(k, c).$$

Thus, changing k_0 only introduces a global sign. Up to this sign, the attacker only needs to guess k_1 . Equivalently, $\{0, e_1\}$ is the set of key representatives for $\mathbb{F}_2^\kappa / U^\perp$.

Theorem 1 is the main reason for considering the affine hull of the key-mask support. The key-recovery function $f(k, c)$ cannot distinguish keys inside the same coset of $U^\perp$ up to a constant. Therefore the relevant key space is not $\mathbb{F}_2^\kappa$, but the quotient $\mathbb{F}_2^\kappa / U^\perp$ of dimension $\kappa - \dim U^\perp = \dim U$.

This perspective gives a more precise way of treating key guesses in the key-recovery phase than the usual round-wise dependency analysis. In many attacks, the guessed key bits are identified by following active wires, active S-boxes, or active words in the partial encryption or decryption. Such a local analysis is safe, but quite often includes redundant key masks. In contrast, the set $\mathcal{A}(f)$ describes exactly the key masks that should be taken into account.

Remark 1. If f is clear from the context, then we also write $\mathcal{A}$ instead of $\mathcal{A}(f)$.

Remark 2. The affine space $\mathcal{A}$ contains the precise dependencies on the key k (in the form of linear masks) for evaluating $f(k, c)$. These are *optimal* with respect to guessing affine equivalence classes of the key input as in Theorem 1.

By definition, $\mathcal{A}$ is the affine hull containing and bounding all key masks so that $\widehat{f}(u, v) \neq 0$ for at least one v . Note, that $u \in \mathcal{A}$ does not necessarily imply that $\widehat{f}(u, v) \neq 0$ for some v . To see the minimality, choose a from the projected Fourier support and suppose that there exist $\delta \notin U^\perp$ and $\varepsilon \in \mathbb{F}_2$ such that

$$f(k + \delta, c) = (-1)^\varepsilon f(k, c)$$

for all k and c . By Equation (3), this implies $\langle u, \delta \rangle = \varepsilon$ for every u in the projected Fourier support. Hence, $\langle u + a, \delta \rangle = 0$ for every such u . Since these vectors $u + a$ span U , it follows that $\delta \in U^\perp$, which is a contradiction.

3.3 Link with linear approximations

In some cases, the Fourier transform of the key-recovery function can be related to the correlations of linear approximations over the partial decryption function in a straightforward way. Let $F: \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ be the partial decryption function, and fix an output mask w in $\mathbb{F}_2^m$ as illustrated in Figure 3. If the key-recovery function f derived from F is

$$f(k, c) = (-1)^{\langle w, F(k, c) \rangle},$$

then the Fourier coefficient of f at (u, v) is

$$\hat{f}(u, v) = \sum_{k \in \mathbb{F}_2^\kappa} \sum_{c \in \mathbb{F}_2^n} (-1)^{\langle w, F(k, c) \rangle + \langle u, k \rangle + \langle v, c \rangle}.$$

Thus, up to multiplication by a factor $2^{\kappa+n}$, the value $\hat{f}(u, v)$ is exactly the correlation of the linear approximation with input masks (u, v) and output mask w over F . That is,

$$C_{w, (u, v)}^F = \frac{1}{2^{\kappa+n}} \hat{f}(u, v).$$

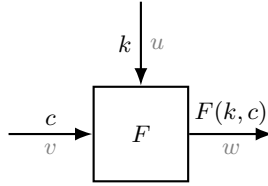


Fig. 3: Partial decryption function F with state masks v and w , and key mask u .

Consequently, determining the affine space $\mathcal{A}(f)$ of key masks that was defined in Section 3.2 is equivalent to finding all linear approximations $((u, v), w)$ of F with fixed output mask w and non-zero correlation. Explicitly,

$$\mathcal{A}(f) = \text{aff} \left(\{u \in \mathbb{F}_2^\kappa : \exists v \in \mathbb{F}_2^n \text{ such that } C_{w, (u, v)}^F \neq 0\} \right).$$

In short, the problem of finding which linear combinations of key bits have to be guessed becomes the problem of enumerating the key masks of all non-zero-correlation linear approximations of F with output mask w .

The remaining task is therefore algorithmic: we need practical ways to find these key masks. In Sections 4 and 5, we discuss two complementary approaches. First, in Section 4, we revisit the mask-propagation method used in [29]. Although originally developed for key-alternating ciphers and applied to MITM attacks on ciphers with quadratic round functions, the method can be interpreted in our framework and can be generalized. In some cases, it is faster than the method

proposed in Section 5, but it may also overestimate $\mathcal{A}(f)$. Next, in Section 5, we describe our generic trail-enumeration method, which follows directly from the discussion above: enumerate non-zero-correlation trails, combine trails with the same key mask, and build the affine hull of the resulting key masks.

4 Propagating Linear Masks

Let us recall a recent technique for deriving key dependencies by Lambooi *et al.* [29]. As we will see, their technique has two shortcomings, both resulting in dependencies that are actually irrelevant for evaluating the key-recovery function f . We remedy one of them by proposing a new version of their approach, which works well for key-alternating ciphers with full-state key addition of independent round keys, but can still fail in other settings.

Although Lambooi *et al.* describe their technique for key-alternating ciphers, generalizing it to our setting of iterative ciphers as depicted in Figure 1 is straightforward. In addition, their goal is to derive dependencies for computing the output difference of the keyed boolean function g , *i.e.*, for the key-recovery function $f'(k, x) = (-1)^{g(k, x) + g(k, x')}$, where x' is a fixed reference value. This is to remove linear key dependencies compared to a direct treatment of $f(k, x) = (-1)^{g(k, x)}$. However, as discussed in the previous section, this only corresponds to recovering U instead of the affine hull $\mathcal{A} = a + U$. Hence, we generalize the method from [29] to analyze f directly.

Essentially, their idea is to compute the Fourier support on a round-by-round basis by propagating linear masks. Say $g(k, c) = \langle w, E(k, c) \rangle$. For ease of notation, let us define $l_w: \mathbb{F}_2^n \rightarrow \mathbb{R}$, $w \in \mathbb{F}_2^n$, as the map $x \mapsto (-1)^{\langle w, x \rangle}$. Similar to the affine hull of the projection onto the key space $\mathcal{A}$, we can introduce the projection onto the data space

$$\mathcal{D}(l_w \circ R_i) = \text{aff} \left(\{v \in \mathbb{F}_2^{m_i} : \exists u \in \mathbb{F}_2^{\kappa} \text{ with } C_{w, (u, v)}^{R_i} \neq 0\} \right).$$

Hence, we can compute the key dependencies for the last round as $\mathcal{A}(l_w \circ R_{r-1})$, but also the input dependencies to that last round as $\mathcal{D}(l_w \circ R_{r-1})$. The idea is then to move one round up and compute the dependencies for all output masks in $\mathcal{D}(l_w \circ R_{r-1})$, and to continue iterating this process.

The linearity of the dot product allows to simplify this process, as it implies that knowledge of $\langle v, y \rangle$ for all $v \in M$ is equivalent to knowledge of $\langle a, y \rangle$ and all $\langle v_i, y \rangle$ for $\text{aff}(M) = a + V$ and a basis $v_0, \dots, v_{d-1}$ of V . It is therefore enough to iterate over a basis of a given set of output masks. Further, to compute the dependencies for output mask a and a basis of masks $v_0, \dots, v_{d-1}$ we can take

$$\mathcal{A}(l_a \circ R_i) + \sum_{j=0}^{d-1} \text{span}(\mathcal{A}(l_{v_j} \circ R_i)) \quad \text{and} \quad \mathcal{D}(l_a \circ R_i) + \sum_{j=0}^{d-1} \text{span}(\mathcal{D}(l_{v_j} \circ R_i))$$

as the affine hull of the key and data masks for that round, respectively.

By taking into account that each round R_i receives the same key as input, we can compute the key dependencies as the affine hull of

$$\left\{ \sum_{i=0}^{r-1} u_i : u_i \text{ is key dependency for round } i \right\}.$$

Pseudocode for this approach is given in Algorithm 1. Note that it is typically easy to compute key and data dependencies for a single round (*e.g.*, because the round function is quadratic as in the application of Lambooij *et al.* [29], or because it is affine equivalent to the parallel application of small-sized S-boxes). This makes the round-by-round approach quite effective, but unfortunately it can produce too many dependencies, as we show next.

Algorithm 1 Propagating Linear Masks (*cf.*, Lambooij *et al.* [29])

Require: Partial cipher E as in Fig. 1 and output mask w

Initialize $a + V \leftarrow \text{aff}(w)$ $\triangleright$ Input masks of prior round, initialized with w

Initialize $U \leftarrow \{0\}$ $\triangleright$ Master key masks

for $i = r - 1, \dots, 0$ **do**

Initialize $b + W \leftarrow a + V$ $\triangleright$ Output masks get input masks of prior round

$a + V \leftarrow \mathcal{D}(l_b \circ R_i)$ $\triangleright$ Input masks for current round

$U' \leftarrow \mathcal{A}(l_b \circ R_i)$ $\triangleright$ Round key masks for current round

for $\tilde{w}$ in basis of W **do**

$V \leftarrow V + \text{span}(\mathcal{D}(l_{\tilde{w}} \circ R_i))$

$U' \leftarrow U' + \text{span}(\mathcal{A}(l_{\tilde{w}} \circ R_i)) = \{u' + v : u' \in U', v \in \text{span}(\mathcal{A}(l_{\tilde{w}} \circ R_i))\}$

$U \leftarrow U + U' = \{u + u' : u \in U, u' \in U'\}$ $\triangleright$ Branching in key path

return U

4.1 Problem 1: key dependencies spanning multiple rounds

A major issue of Algorithm 1 is that it represents the affine hull of all linear trails as a direct sum of per-round mask spaces. As such, it is incapable of representing key-dependencies that span over multiple rounds. As an example, consider a key-recovery map split into two rounds, $f = r_1 \circ R_0$ with $r_1(x) = g(x + k_1)$ and $R_0(x) = x + k_2$. Application of Algorithm 1 would give $\mathcal{A}(g) \oplus \mathcal{A}(g)$, as a superset of the affine span of trails. Let $\mathcal{A}(g) = a + V$. Projecting the superset of trails on the key additions gives that the key-dependencies are V evaluated on k_1 and on k_2 and a evaluated on $k_1 + k_2$. However, in reality, we know that this can be reduced to $\mathcal{A}(g)$ evaluated on $k_1 + k_2$. While this example is relatively trivial, this behavior also arises in practical cryptanalysis, for example, the results on PRESENT of [6, Sec. 5.1] cannot be found with Algorithm 1.

Remark 3. This observation does not directly apply to the original approach, where the state dependency of the propagation of differences is evaluated. In an intermediary step in the original approach, we would want to evaluate the

state dependency, x' , of the function $F(x, x') = R_i(x + x') + R_i(x')$ of a space of masks V . Propagating the basis elements v_i of V would give an affine space of input masks on x of $\mathcal{A}(l_{v_i} \circ R_i) = a_i + U_i$, but only the space U_i on x' . Therefore the state dependency is $\sum_i U_i$, which notably does not contain the a_i , which correspond to dependencies over multiple rounds. Hence, the original version still suffers from not being able to capture key dependencies spanning multiple rounds, but to a lesser degree.

This issue can be resolved by first concatenating the propagated trails and only then taking the affine span. We first do this for the data masks. That is, we derive the affine hull of

$$\text{aff}(\{\text{data masks } w : \exists \text{ key masks } u \in \mathbb{F}_2^{r\kappa} \text{ with non-zero correlation } \}).$$

This affine hull can be seen as a tree, where each node corresponds to a mask on the data path, with w as the root. We therefore enumerate trails as a breadth-first exploration of this tree. We then derive the key dependency from the affine hull of the data masks similar to before. This is given in Algorithm 2.

As we show later, this approach gives the precise key dependencies for key-alternating ciphers with full-state key addition of independent round keys. However, it can still produce too many dependencies in other cases, as we show next.

Algorithm 2 Improved Propagation of Linear Masks

Require: Partial cipher E as in Fig. 1 and output mask w
Initialize $a + V \leftarrow \text{aff}(w)$ $\triangleright$ Input masks of prior round, initialized with w
for $i = r - 1, \dots, 0$ **do**
 Initialize $b + W \leftarrow a + V$
 $a + V \leftarrow \mathcal{D}(l_a \circ R_i) \oplus (a + \{0\})$
 for $\tilde{w}$ in basis of W **do**
 $V \leftarrow V + \text{span}(\mathcal{D}(l_{\tilde{w}} \circ R_i) \oplus (\tilde{w} + \{0\}))$
 $U \leftarrow \mathcal{A}(l_a \circ R_0 \parallel \dots \parallel R_{r-1})$ $\triangleright$ Key-dependencies
 for v in basis of V **do**
 $U \leftarrow U + \text{span}(\mathcal{A}(l_v \circ R_0 \parallel \dots \parallel R_{r-1}))$
return $\{\sum_{i=0}^{r-1} u_i : (u_0, \dots, u_{r-1}) \in U\}$ $\triangleright$ Take branching into account

4.2 Problem 2: trail cancellation

Another problem of Algorithms 1 and 2 is that they are based only on linear trails, and therefore include dependencies even if trails cancel out. We give the precise explanation later, and only give a short example here.

Let us consider the 3-variable chi function $\chi: \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^3$ defined by

$$\chi \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_0 + x_1 x_2 + x_2 \\ x_0 x_2 + x_0 + x_1 \\ x_0 x_1 + x_1 + x_2 \end{pmatrix}.$$

Let $E: \mathbb{F}_2^4 \times \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^3$ be the partial decryption function defined by $(k, x) \mapsto \chi(\chi(x + k_0) + k_1)$ for $k = (h_0, h_1, h_2, h_3)$ and with

$$k_0 = \begin{pmatrix} h_0 \\ h_1 \\ h_2 \end{pmatrix} \quad \text{and} \quad k_1 = \begin{pmatrix} h_3 \\ 0 \\ 0 \end{pmatrix}.$$

In other words, the key addition in the second round is partial. Suppose that $g(k, x)$ is the first output coordinate of E , so that $g(k, x) = \langle w, R_1(k, R_0(k, x)) \rangle$ with $R_i(k, x) = \chi(x + k_i)$ and $w = (1, 0, 0)$.

For this discussion, we focus on Algorithm 2, but the same observation applies to Algorithm 1. Let $e_0, e_1, e_2 \in \mathbb{F}_2^3$ denote the unit basis, so that $w = e_0$. It is easy to verify that

$$\mathcal{D}(l_{e_j} \circ R_i) = e_j + \{e_j\}^\perp$$

for all i, j . As the round keys are added into the data path, the propagation method therefore gives

$$\mathbb{F}_2^3 \oplus (w + \{w\}^\perp)$$

in the six round-key coordinates (k_0, k_1) . Pulling these masks back to the master-key variables (h_0, h_1, h_2, h_3) gives

$$\mathcal{A}_{\text{prop}} = \mathbb{F}_2^3 \times \{1\} \subseteq \mathbb{F}_2^4.$$

It is easy to see that this is an overapproximation. Since χ is an involution, and since $\langle w, R_1 \rangle$ is linear in the only non-zero coordinate of k_1 , we have

$$g(k, x) = \langle w, R_1(k, R_0(k, x)) \rangle = x_0 + h_0 + h_3.$$

Consequently, the exact affine hull of the projected Fourier support in the master-key space is

$$\mathcal{A} = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\},$$

which is strictly smaller than $\mathcal{A}_{\text{prop}} = \mathbb{F}_2^3 \times \{1\}$. This overestimation is caused by trail cancellation, and we provide the explicit cancellations in Section 5.3.

5 Linear-Trail Enumeration

There are drawbacks to the method described in Section 4. In particular, the approach of Lambooi *et al.* might not find the minimal number of classes of equivalent keys. The reason is that an apparent key dependency may disappear after summing several linear trails with the same external masks. This can happen

due to the key schedule, due to partial key additions, or simply because there is no efficient way to describe the exact propagation of masks over an entire round function.

To address this limitation, we should compute the exact correlations of the linear approximations over the key-recovery map, which is natural from the point of view of the theory described in Section 3. This can be done using a trail-based approach, but this is non-trivial from a computational point of view.

5.1 Basic idea

Let $E: \mathbb{F}_2^{\kappa} \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ be an iterative cipher as in Figure 1. In the context of this paper, E corresponds to the partial decryption (or encryption) function. As discussed in Section 2.2, the correlation of a linear approximation with input masks (u_0, v_0) and output mask v_r is given by

$$C_{v_r, (u_0, v_0)}^E = \sum_{\substack{u_1, \dots, u_r \\ v_1, \dots, v_{r-1}}} \prod_{i=0}^{r-1} C_{v_{i+1}, (v_i, u_{i+1})}^{R_i},$$

where the masks on the intermediate keys must satisfy $\sum_{i=1}^r u_i = u_0$.

The basic idea of our trail enumeration approach is to start by finding an arbitrary linear trail with masks $u = (u_0, \dots, u_r)$ and $v = (v_0, \dots, v_r)$ with non-zero correlation, where the output mask v_r is fixed to a given value. That is,

$$\prod_{i=0}^{r-1} C_{v_{i+1}, (v_i, u_{i+1})}^{R_i} \neq 0.$$

However, a single trail with non-zero correlation is not sufficient to conclude that $u_0 \in \mathcal{A}$, because the correlations of all trails with the same external masks u_0 , v_0 and v_r could cancel. Therefore, we subsequently find all trails with the same external masks and conclude that $u_0 \in \mathcal{A}$ only if

$$C_{v_r, (u_0, v_0)}^E = \sum_{\substack{u_1, \dots, u_r \\ v_1, \dots, v_{r-1}}} \prod_{i=0}^{r-1} C_{v_{i+1}, (v_i, u_{i+1})}^{R_i} \neq 0,$$

where the masks on the intermediate keys must again satisfy $\sum_{i=1}^r u_i = u_0$. This process must be repeated until no more trails can be found. This is significantly different from typical uses of linear trails, which only require the enumeration of a limited number of ‘dominant’ high-correlation trails. There are a few applications, such as linear key-recovery attacks based on Walsh-spectrum puncturing (though their objective differs: we keep the recovery map fixed), that require extensive enumeration of trails [26]. This is typically a challenging task, but in Section 5.2 we give an efficient algorithm to compute $\mathcal{A}$ that enumeration across all external masks without losing accuracy.

5.2 Efficient trail enumeration

In many realistic applications of our approach, the number of trails with fixed masks u_0 , v_0 , and v_r is relatively limited. For example, in the simplest case of a key-alternating cipher with different round keys, every choice of the input mask u_0 on the key determines a unique linear trail. Hence, the main challenge is to enumerate the masks u_0 on the key input.

To address this challenge, we propose Algorithm 3. At each iteration of the outer loop, we keep track of a basis set L of masks u_0 . At all times, $\text{aff}(L)$ is a subset of $\mathcal{A}$. Initially, the strategy is as described in Section 5.1: we first search for a single linear trail with masks $u = (u_0, \dots, u_r)$ and $v = (v_0, \dots, v_r)$ and with non-zero correlation. The output mask v_r is given as part of the input. Next, we find all trails with the same masks u_0 , v_0 and v_r , and check whether or not

$$C_{v_r, (u_0, v_0)}^E = \sum_{\substack{u_1, \dots, u_r \\ v_1, \dots, v_{r-1}}} \prod_{i=0}^{r-1} C_{v_{i+1}, (v_i, u_{i+1})}^{R_i} \neq 0,$$

where the masks on the intermediate keys must once again satisfy $\sum_{i=1}^r u_i = u_0$. Note that, although signs are irrelevant for finding effective trails, they are needed to compute this sum. If $C_{v_r, (u_0, v_0)}^E$ is non-zero, then we know that $u_0 \in \mathcal{A}$, so we add u_0 to the basis set L .

However, after each successful iteration, we reduce the search space as follows. Let $\text{aff}(L) = a + V$, with V a vector space. Since we want to find the affine hull $\mathcal{A}$ of key input masks with non-zero correlation, rather than the exact support of the Fourier transform of the key-recovery map, we are only interested in key input masks u_0 that are not already in $\text{aff}(L)$.

In the next step, we search for a new linear trail with masks $v = (v_0, \dots, v_r)$ and $u = (u_0, \dots, u_r)$ as before, but with the additional constraint that $u_0 \notin a + V$. This constraint can be expressed by requiring that at least one of the linear equations defining $a + V$ is not satisfied by u_0 . This process is iterated until no further trail with non-zero correlation contribution can be found.

Since the only trails we are excluding are those whose key masks are already in $\text{aff}(L)$ and those that lead to correlation zero, we obtain $\text{aff}(L) = \mathcal{A}$ at the end of the algorithm.

Note that we are not interested in the precise set of key masks, but only in its affine hull (since affine combinations come for free; we consider the affine hull rather than the span because the offset plays a special role). It is therefore sufficient to obtain an affine basis, which contains at most $\kappa + 1$ masks. The potentially costly part is summing the correlations of all trails with the same external masks (u_0, v_0, v_r) .

An example illustrating the affine-hull search and cancellation test is given in Section A.

As an optional search heuristic, the Hamming weight of the masks can be minimized, since for sparse masks u_0 and v_0 the number of trails is usually smaller. Correctness does not rely on this optimization, and any effective trail outside the affine hull of the current basis set L is sufficient.

Algorithm 3 Linear-trail enumeration to determine $\mathcal{A}$.

Require: A signed trail-search model for E as in Figure 1, an output mask v_r .

Ensure: A basis set L such that $\text{aff}(L) = \mathcal{A}$.

```

1:  $L \leftarrow \emptyset$ 
2:  $Z \leftarrow \emptyset$   $\triangleright$  External masks  $(u_0, v_0, v_r)$  with correlation zero due to cancellation
3: while true do
4:   if  $L \neq \emptyset$  then
5:      $a + V \leftarrow \text{aff}(L)$ 
6:     Find a trail with masks  $u = (u_0, \dots, u_r)$  and  $v = (v_0, \dots, v_r)$  such that
       
$$\prod_{i=0}^{r-1} C_{v_{i+1}, (v_i, u_{i+1})}^{R_i} \neq 0, \quad u_0 \notin a + V \text{ if } L \neq \emptyset, \quad (u_0, v_0, v_r) \notin Z.$$

       Optionally minimize the Hamming weight of the masks.
7:     if no such trail exists then
8:       return  $L$ 
9:     Enumerate all trails with the same external masks  $u_0, v_0$ , and  $v_r$ , and compute
       
$$C_{v_r, (u_0, v_0)}^E = \sum_{\substack{u_1, \dots, u_r \\ v_1, \dots, v_{r-1}}} \prod_{i=0}^{r-1} C_{v_{i+1}, (v_i, u_{i+1})}^{R_i},$$

       where the intermediate key masks satisfy  $\sum_{i=1}^r u_i = u_0$ .
10:    if  $C_{v_r, (u_0, v_0)}^E \neq 0$  then
11:       $L \leftarrow L \cup \{u_0\}$ 
12:    else
13:       $Z \leftarrow Z \cup \{(u_0, v_0, v_r)\}$ 

```

In Algorithm 3, any search model to find linear trails can be used. In our implementation, we used a constraint-programming approach based on Google's ORTools.

Remark 4. For full-state round-key additions without key schedule, the key mask of a round is simply the corresponding state mask. In this case, it is not necessary to compute the sum of the trail correlations. Indeed, there is only one trail for the same external masks and $C_{v_r, (u_0, v_0)}^E = \prod_{i=0}^{r-1} C_{v_{i+1}, (v_i, u_{i+1})}^{R_i}$. This is why Algorithm 2 already produces precise results in this setting.

5.3 Application to the previous example

Let us again look at $E(k, x) = \chi(\chi(x + k_0) + k_1)$ from Section 4.2 with $w = (1, 0, 0)$. All data trails with non-zero correlation contribution are given in Table 3. Since the round keys are added to the state, we can deduce the round key masks from the data trail. Recall that the last two coordinates of k_1 are restricted to zero. This implies that for any data trail (v_0, v_1) with $v_1 = (y_0, y_1, y_2)$ we get the key mask $u = v_0 \parallel y_0$, and we need to cluster over all intermediate masks v_1 that lead

to the same u . Hence, we get that

$$C_{w,(u,v_0)}^E = \sum_{z_1, z_2 \in \mathbb{F}_2} C_{w,v_1+(0,z_1,z_2)}^\chi C_{v_1+(0,z_1,z_2),v_0}^\chi.$$

Example 3 (Explicit trail cancellation). Consider the external key masks

$$u = e_1 \| 1 = (0, 1, 0, 1),$$

which corresponds to $h_1 + h_3$, and implies $v_0 = e_1$ due to the addition of the round key to the state. As shown in Table 3, only the two intermediate masks $v_1 \in \{(1, 1, 0), (1, 1, 1)\}$ give non-zero trails. Hence, we get

$$\begin{aligned} C_{e_0,(e_1\|1,e_1)}^E &= \sum_{\substack{v_1 \in \mathbb{F}_2^3 \\ (v_1)_0 = 1}} C_{v_1,e_1}^\chi C_{e_0,v_1}^\chi \\ &= -\frac{1}{4} + \frac{1}{4} = 0. \end{aligned}$$

Similarly, we see that $C_{w,(u,v_0)}^E$ is only non-zero for $v_0 = (1, 0, 0)$ and $y_0 = 1$. In other words, $A = \{(1, 0, 0, 1)\}$, which is identical to its affine hull and matches the precise key dependencies we identified in Section 4.2.

Table 3: Trails (w_0, w_1) with non-zero correlation contribution for $\langle w, \chi \circ \chi \rangle$.

Masks		Cor.	Masks		Cor.	Masks		Cor.
v_0	v_1		v_0	v_1		v_0	v_1	
(1, 0, 0)	(1, 0, 0)	$\frac{1}{4}$	(1, 1, 0)	(1, 0, 0)	$-\frac{1}{4}$	(0, 1, 1)	(1, 1, 0)	$-\frac{1}{4}$
(1, 0, 0)	(1, 1, 0)	$\frac{1}{4}$	(1, 1, 0)	(1, 0, 1)	$\frac{1}{4}$	(0, 1, 1)	(1, 0, 1)	$\frac{1}{4}$
(1, 0, 0)	(1, 0, 1)	$\frac{1}{4}$	(0, 0, 1)	(1, 0, 1)	$-\frac{1}{4}$	(1, 1, 1)	(1, 0, 0)	$\frac{1}{4}$
(1, 0, 0)	(1, 1, 1)	$\frac{1}{4}$	(0, 0, 1)	(1, 1, 1)	$\frac{1}{4}$	(1, 1, 1)	(1, 1, 1)	$-\frac{1}{4}$
(0, 1, 0)	(1, 1, 0)	$-\frac{1}{4}$	(1, 0, 1)	(1, 0, 0)	$\frac{1}{4}$			
(0, 1, 0)	(1, 1, 1)	$\frac{1}{4}$	(1, 0, 1)	(1, 1, 0)	$-\frac{1}{4}$			

6 Application to SPN ciphers

As a first validation step, we applied our method to the same SPN primitives considered in [6], namely PRESENT, GIFT-64, and RECTANGLE. In all cases, our method returns the same effective key spaces as the ARKST-based analysis of [6]. The difference is in the practical running time. For example, ARKST computation for four key-recovery rounds of RECTANGLE was reported to take several days on a common PC. In contrast, our approach constructs the relevant key-mask space in a matter of seconds or minutes on a standard laptop with an Intel Core Ultra 7 processor and 16GB RAM.

There is also a modelling advantage. In the ARKST-based treatment of [6], the key-recovery part is first analyzed under an independent round key assumption. In our approach, key-schedule relations can be potentially included directly as linear constraints in the trail-search model.

This section illustrates three further uses of our method. First, we revisit published ciphertext-side key-recovery procedures for GIFT. For a differential-linear attack on GIFT-64, we find that the published key list underestimates the number of bits that must be guessed, while for a linear attack on GIFT-128, we find that one key bit was unnecessarily included. In both cases, the published analyses already go beyond a naive active-wire or active-S-box count: the number of key bits considered there shows that there were removed key bits with only a linear or constant effect on the tested property. Our results show that this dedicated dependency analysis is delicate and can lead both to overcounting and to undercounting.

Second, we show that the integral attack on PRESENT can be extended to 13 rounds by combining our linear-trail enumeration method with the PRESENT key-schedule relations. To the best of our knowledge, this gives the best integral key-recovery attack on PRESENT.

Finally, for SKINNY, our approach allows several key bits to be removed from the last partial-decryption steps using integral attacks as an example, although this does not immediately change the dominant complexity of the published attack. However, the reductions also indicate the generality of our linear-trail enumeration approach, since SKINNY is not a bit-permutation cipher and is not feasible for ARKST.

In each case, we keep the underlying distinguisher and the attack structure unchanged. The only modification is the way in which the key-recovery dependencies are computed: instead of following active wires or active S-boxes locally, we compute the affine space $\mathcal{A}(f)$ of key masks that actually occur in the Fourier support of the key-recovery function.

6.1 GIFT

The block cipher GIFT was introduced by Banik et al. [2] at CHES 2017 as a lightweight SPN cipher. It has two variants, GIFT-64 and GIFT-128, both with a 128-bit master key. Its round function consists of a 4-bit S-box layer, a bit permutation, and a partial AddRoundKey layer.

Remark 5. In the ARKST-based treatment in [6], the analysis is first performed as if the key addition were a full-state addition. After the space of admissible key masks is obtained, it is intersected with the subspace of masks that can actually arise from the half-state AddRoundKey operation of GIFT. This gives the correct result, but still, because of partial key addition, key dependencies have to be computed with respect to the actual key-injection map rather than with respect to an idealised full-state key addition. Otherwise, in some cases, we could overlook some bits. In our approach, the partial key addition is modeled from the beginning.

Differential-linear cryptanalysis. We examine the 19-round differential-linear attack on GIFT-64 from [39]. Its ciphertext-side key recovery consists of eight partial-sum steps. In the final step, the accumulated parity is updated as

$$z \leftarrow z + X_{16}[60] + X_{16}[58] + X_{16}[54] \\ + X_{16}[41] + X_{16}[30] + X_{16}[20].$$

The published procedure guesses

$$EK_{16}[31, 30, 21, 20, 11, 10]$$

in this step. These are the two injected key bits associated with the three state terms $X_{16}[60]$, $X_{16}[41]$, and $X_{16}[20]$.

The remaining three terms, $X_{16}[58]$, $X_{16}[54]$, $X_{16}[30]$, which are x_2 -coordinates of S-boxes 14, 13, and 7, respectively. The relevant inverse-S-box relation used in [39] is

$$x_2 = y_0 y_3 + y_0 + y_1 + y_2 + 1.$$

Let k_0 and k_1 denote the two key bits XORed with y_0 and y_1 , respectively. Substitution gives

$$x_2(y_0 + k_0, y_1 + k_1, y_2, y_3) = x_2(y_0, y_1, y_2, y_3) \\ + k_0(y_3 + 1) + k_1.$$

The contribution of k_1 is a data-independent parity flip and therefore only changes the global sign of the signed recovery map. In contrast, the contribution

$$k_0(y_3 + 1)$$

depends on the partially decrypted state. It is consequently not a global sign and cannot be removed by the linear-effect argument.

With the extended-key numbering of [39], this gives the following three missing dependencies:

tested state bit	S-box coordinate	missing key-dependent term
$X_{16}[58]$	x_2 of cell 14	$EK_{16}[28](y_3^{(14)} + 1)$
$X_{16}[54]$	x_2 of cell 13	$EK_{16}[26](y_3^{(13)} + 1)$
$X_{16}[30]$	x_2 of cell 7	$EK_{16}[14](y_3^{(7)} + 1).$

Thus the local S-box equations explain why

$$EK_{16}[28, 26, 14]$$

cannot be discarded as data-independent linear effects.

We verified our observation with Algorithm 3 and confirmed that the number of ciphertext-side guessed bits increases from 74 to 77.

The cost of the final partial-sum step changes from $2^{110.34}$ to $2^{113.34}$. Since this step is not dominant, the total time complexity changes only slightly, from

approximately $2^{119.45}$ to $2^{119.47}$. The small numerical effect is not the main point; rather, the example shows that tracking which key bits are new, which ones affect the property linearly, and which ones are eliminated by the key schedule is error-prone when done ad-hoc.

Our conjecture is that the authors' omission originates from implicitly confusing the key-addition positions of GIFT-128 with GIFT-64. For the former, the two key bits are added to the two middle S-box coordinates, thereby contributing only linear terms. For a recovery test that is invariant under a global sign, neither bit needs to be guessed. For GIFT-64 the bit added to the LSB and cannot be removed using the same linear-effect argument.

Linear cryptanalysis. We next consider the 25-round linear attack on GIFT-128 from [34]. On the plaintext side, the first subkey-guessing step is Step S2. Sun et al. already perform a local optimization in this step: among the key bits associated with the two marked S-boxes in the first key-recovery round for the plaintext side (in their Figure 2), they omit

$$RK_1[54] \quad \text{and} \quad RK_1[62],$$

and guess only $RK_1[55]$ and $RK_1[63]$. The omitted bits affect the evaluated approximation only linearly and therefore need not be enumerated separately. Thus the published attack already goes beyond simply guessing every round-key bit occurring on an active wire.

Our analysis recovers these two optimizations, but also finds that the same reasoning applies to $RK_0[60]$. In our round-key dependency count, this reduces the dimension from 71 to 70. The cost of Step S2 is correspondingly reduced from $2^{96} \cdot 2^{19} = 2^{115}$ to $2^{96} \cdot 2^{18} = 2^{114}$ memory accesses. Since this step is not dominant, the overall attack complexity remains unchanged. This example shows that the manual optimization of could be valid but incomplete: besides $RK_1[54]$ and $RK_1[62]$, the bit $RK_0[60]$ can also be removed.

6.2 PRESENT

PRESENT is a 64-bit lightweight SPN cipher introduced by Bogdanov et al. [12]. It has two key-size variants, 80- and 128-bit versions. In this subsection, we focus on the former one. The round function consists of a 4-bit S-box layer, a bit-permutation layer, and an XOR with the round key. This bit-oriented linear layer makes PRESENT a particularly suitable target for our key-mask trail enumeration method, since the dependency of the final tested bit can be followed very precisely through the last rounds.

Integral cryptanalysis. We further observe that the integral attack on PRESENT can be extended to 13 rounds once the key-equivalence relations detected by our method are combined with the linear dependencies imposed by the PRESENT key schedule.

For key-recovery, we will use a 9-round distinguisher from [9] based on the divisibility properties having data complexity 2^{63} . Previous key-recovery attacks using this distinguisher reached 12 rounds [36,6]. The four bits of the least significant nibble have divisibility exponents $(v_0, v_1, v_2, v_3) = (2, 1, 1, 1)$. Therefore, under the usual independent-filter heuristic, testing the whole nibble gives a filter of $2^{-2}2^{-1}2^{-1}2^{-1} = 2^{-5}$.

A naive local analysis of the four key-recovery rounds identifies 148 round-key bits that may affect the tested nibble. This count follows the locally active wires and treats all corresponding round-key bits as independent.

We apply Algorithm 3 to the four Boolean functions corresponding to the four tested output bits. Under the independent round-key assumption, the resulting joint key-mask space has dimension 108. One coordinate basis for this space consists of all 64 bits of k_{13} and the following 44 bits of k_{12} :

$$\{k_{12}[j] : j \in I_{12}\},$$

where the set I_{12} is given explicitly in Appendix B. Thus, the reduction from 148 to 108 key bits is obtained solely from the linear-trail enumeration: 40 of the dependencies suggested by the active wires analysis are not involved in Fourier support.

We next apply the PRESENT-80 key schedule. We do not treat the complete key schedule as linear, since it contains an S-box update. The relations used here are only the affine bit-copy relations resulting from the 61-bit rotation, together with the known round-counter constants. For every

$$j \in I_{12} \setminus \{16, 17, 18\},$$

the schedule gives

$$k_{13}[j - 19] = k_{12}[j] + \gamma_j,$$

where $\gamma_j \in \mathbb{F}_2$ is a known constant. The 41 positions $j - 19$ are distinct and all belong to the already guessed round key k_{13} . Hence, once k_{13} is known, these 41 bits of k_{12} introduce no additional guessing variables.

The only involved bits of k_{12} that are not covered by these copy relations are

$$k_{12}[16], \quad k_{12}[17], \quad k_{12}[18].$$

Consequently, the final guessing space consists of the 64 bits of k_{13} and these three additional bits of k_{12} . Its dimension is therefore $64 + 3 = 67$, which is the final key-guessing space dimension (Figure 4).

Then we use the same FFT-based evaluation as in the previous 12-round attack. The cost of the transform is

$$2^{63} + 3 \cdot 64 \cdot 2^{67} \approx 2^{74.6}$$

additions. The 2^{-5} divisibility filter leaves about $2^{67} \cdot 2^{-5} = 2^{62}$ key candidates. Since PRESENT has an 80-bit master key, these remaining candidates can

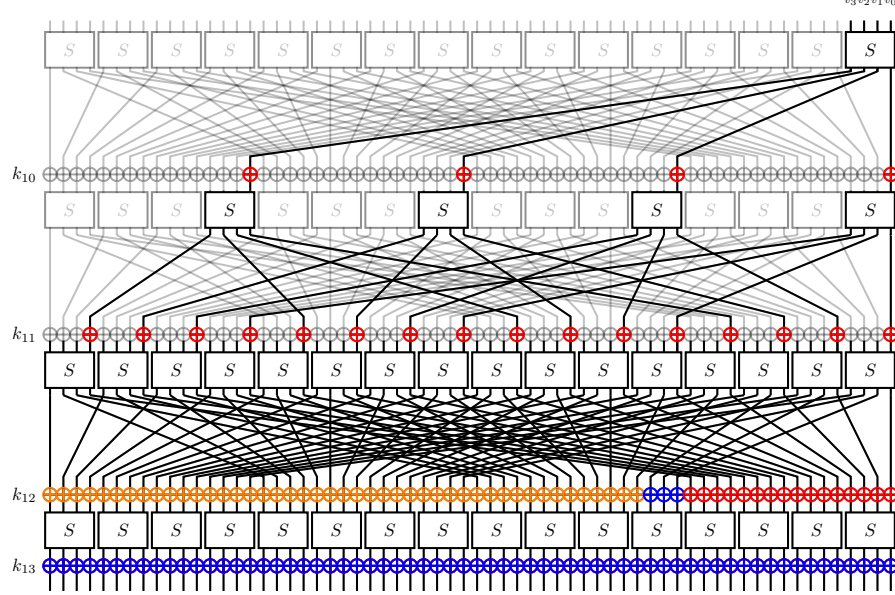


Fig. 4: Key-bit dependencies in the 13-round integral key-recovery attack on PRESENT. Black wires denote the dependencies identified by the initial local analysis. Red denotes the 40 dimensions eliminated by Algorithm 3. Orange denotes the 41 dependencies of k_{12} that are determined by the already guessed bits of k_{13} through the key schedule. Blue denotes the 64 guessed bits of k_{13} and 3 of k_{12} , the effective guessed key bits after our optimization.

be completed by exhaustive search over the remaining 13 key bits. The final exhaustive-search part therefore, costs

$$2^{62} \cdot 2^{13} = 2^{75}$$

encryptions. Consequently, the total complexity of the 13-round attack is dominated by about 2^{75} encryptions, with data complexity 2^{63} .

We emphasize that the time complexities reported in [6] refer only to the key-recovery filtering step. In the 13-round extensions considered here, we also include the final exhaustive search over the remaining PRESENT key bits. This completion step is essential for estimating the full attack complexity (in [6] it was omitted).

6.3 SKINNY

The block cipher SKINNY is a family of tweakable lightweight block ciphers with a 4×4 cell state introduced by Beierle et al. [5]. For SKINNY-64, the cell size is

$c = 4$, while for **SKINNY-128** the cell size is $c = 8$. The round function consists of SubCells, AddConstants, AddRoundTweakey, ShiftRows, and MixColumns, where the tweakey is added only to the first two state rows.

Integral cryptanalysis. Recent integral key-recovery attacks published at EUROCRYPT’26 cover 16 rounds of **SKINNY-n-n**, 18 rounds of **SKINNY-n-2n**, and 20 rounds of **SKINNY-n-3n**, using a 7-round integral distinguisher and a partial-sum key-recovery procedure [38]. The key-recovery phase is divided into two branches corresponding to one zero-sum test and evaluated by a separate partial-sum computations. For example, for **SKINNY-64-64**, one zero-sum test has partial-sum complexity $2^{44.82}$, while the full attack is dominated by four data structures of size 2^{48} , giving total time and data 2^{50} . The corresponding branch-wise partial-sum procedures are given in Tables 8 and 9 of [38].

We analyzed the partial decryption computations appearing in these key-recovery procedures. In **SKINNY-64**, four key bits of guessed key can be omitted, two for each branch. In **SKINNY-128**, the effect is stronger: some components allow up to 14 (7 per branch) key bits to be removed from the guessing space.

However, for the published integral attacks this does not lead to a meaningful improvement of the final time complexity. First, the total complexity is dominated by the data complexity. Second, the saved key bits usually occur only in the last step of the partial-sum procedure, whereas the dominant partial-sum terms come from earlier steps where larger tables and more guessed key material are still present. For instance, in the **SKINNY-64-64** attack, reducing the last guessed nibble by two bits in both branches changes the one-test partial-sum complexity only from $2^{44.82}$ to about $2^{44.80}$. Thus, while our method detects real local reductions of between four and fourteen key bits, exploiting them in a way that improves the attack complexity would require a different organization of the partial-sum procedure. We leave such a redesign as future work.

7 Application to non-SPN ciphers

We now turn to ciphers whose round function is not an SPN. The purpose of this section is to emphasize that our approach is not tied to S-box layers, bit permutations, or full-state key additions. The only requirement is that the key-recovery computation can be described by a trail-search model for the corresponding partial encryption or decryption.

For non-SPN primitives, the local constraints are usually different from the LAT constraints used for S-boxes. For example, in Feistel or AndRX/ARX-like designs, one has to model the propagation of masks through Boolean operations such as AND, rotations, XORs or modular additions. This does not change the theory: these constraints merely define the feasible linear trails of the key-recovery function.

We consider several types of key-recovery procedures: integral, linear, and zero-correlation attacks. In each case, the distinguisher and the attack structure are

left unchanged; only the set of key variables used in the key-recovery computation is recomputed more precisely.

7.1 SIMON

SIMON is a family of lightweight Feistel block ciphers by the NSA [4] whose round function consists of rotations, bitwise AND, and XOR operations. This makes SIMON a natural non-SPN test case for our framework.

Integral cryptanalysis. We first consider the integral key-recovery attack on SIMON32/64 from [37]. The attack uses a 15-round integral distinguisher with 2^{31} chosen plaintexts and a zero-sum property on the least significant bit. To extend the attack to 21 rounds, the original key-recovery procedure uses a meet-in-the-middle split of the tested parity into two branches,

$$\bigoplus (L_{15,\{15\}} \wedge L_{15,\{8\}}) = \bigoplus (L_{15,\{14\}} + L_{16,\{0\}}),$$

and evaluates the two sides separately with partial sums.

Applying our linear-trail enumeration method to the two key-recovery predicates, we find that both branches contain redundant key information. More precisely, before quotienting by the equivalent-key space, each branch depends on 45 key bits. However, the affine key-mask space returned by our method has dimension 37 for each branch. Thus the effective guessed-key space is reduced from $45 + 45$ to $37 + 37$ bits for the two meet-in-the-middle branches.

Using the same partial-sum procedure and the same cost model as in [37], the total key-recovery complexity drops from the published estimate of $2^{54.59}$ to $2^{52.75}$ 21-round SIMON32 encryptions, with the same data complexity 2^{31} .

We note that the same reduction was also discussed in [6]. However, their ARKST algorithm is not applicable to the Feistel/AndRX structure of SIMON. Their SIMON32 analysis was therefore carried out separately through a manual ANF calculation, which does not scale well for more rounds, because the degree of the ANF becomes too high. In contrast, our approach applies uniformly: one only has to encode the SIMON round function constraint in the linear-trail model.

Linear cryptanalysis. We next consider the key-recovery part of the linear attacks on SIMON by Leurent et al. [30]. Their attacks cover several large-state variants of the cipher, namely 43-round SIMON96/96, 45-round SIMON96/144, 53-round SIMON128/128, 55-round SIMON128/192, and 56-round SIMON128/256. In each case, a linear approximation is used for the middle rounds, while several rounds are added on the plaintext and ciphertext sides for key recovery.

The key-recovery procedure has to compute the state bits entering the linear approximation from the outer rounds. This requires guessing some subkey bits, and the original paper gives the corresponding key-bit counts before and after taking the SIMON key schedule into account. In particular, the plaintext-side counts are given in Table 13 of [30].

Applying our analysis, we find that the number of key bits that have to be guessed is smaller than reported there. The difference already appears in the total number of involved key bits, and becomes more visible after reducing with SIMON key schedule (in [30], the authors evaluated complexity with independent round keys), which is linear. Table 4 compares the values obtained by our analysis with those of Table 13 in [30]. The values in parentheses are the previously reported ones in [30].

For a fair comparison, we recompute the attack parameters by optimizing the advantage a . For each row, we choose the integer a minimizing $T(a) = C_1(a)/P_S(a)$, where $C_1(a) = 2^{\kappa_g} + 2^{\kappa-a}$ and κ_g is the number of independent guessed key bits and κ is the master-key size. The success probability $P_S(a)$ is computed using the same single-linear-hull model as in [30].

Remark 6. It should be noted that rotated versions of the attack from [30] can be used to achieve a success rate close to one. However, this repetition is not reflected directly in the reported time complexity, as including it would artificially inflate the complexity. Instead, the authors report the expected time complexity $T(a)$ as the main metric, in addition to the data complexity. We therefore use this metric for our comparison as well.

Table 4: Comparison of plaintext-side key-recovery bits for linear attacks against SIMON. Values in parentheses are those reported in Table 13 of [30].

Rounds	SIMON96			SIMON128			
	total	independent		total	independent		
		$\kappa = 96$	$\kappa = 144$		$\kappa = 128$	$\kappa = 192$	$\kappa = 256$
1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
2	2 (2)	2 (2)	2 (2)	2 (2)	2 (2)	2 (2)	2 (2)
3	7 (7)	7 (7)	7 (7)	7 (7)	7 (7)	7 (7)	7 (7)
4	18 (18)	18 (18)	18 (18)	18 (18)	18 (18)	18 (18)	18 (18)
5	35 (37)	35 (37)	35 (37)	35 (37)	35 (37)	35 (37)	35 (37)
6	63 (65)	58 (61)	63 (65)	63 (65)	58 (61)	63 (65)	63 (65)
7	99 (101)	78 (83)	94 (98)	99 (101)	78 (83)	94 (98)	99 (101)
8	141 (143)	95	120 (126)	143 (145)	98 (104)	122 (129)	138 (142)

We perform the same analysis on the ciphertext side. The results for the approximation $(1, 0) \rightarrow (0, 1)$ and $(4, 0) \rightarrow (4, 1)$ are given in Table 5 and Table 6, for different versions, as different linear approximations were used. These tables show that the same phenomenon occurs on both sides of the linear key-recovery procedure.

The effect of these reductions is summarized in Table 1, where the saved key bits are listed for the whole linear key-recovery procedures, taking the relevant plaintext-side and ciphertext-side contributions together. After recomputing the attack parameters with the reduced guessed-key spaces, the resulting time complexities improve significantly; these optimized complexities are reported in Table 2.

Table 5: Comparison of ciphertext-side key-recovery bits for linear attacks against **SIMON** using the approximation $(1, 0) \rightarrow (0, 1)$. Values in parentheses are those reported in Table 14 of [30].

Rounds	SIMON96			SIMON128	
	total	independent		total	independent
		$\kappa = 96$	$\kappa = 144$		
1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
2	2 (2)	2 (2)	2 (2)	2 (2)	2 (2)
3	7 (7)	7 (7)	7 (7)	7 (7)	7 (7)
4	18 (18)	18 (18)	18 (18)	18 (18)	18 (18)
5	35 (37)	35 (37)	35 (37)	35 (37)	35 (37)
6	63 (65)	57 (61)	61 (61)	63 (65)	63 (65)
7	99 (101)	77 (83)	90 (87)	99 (101)	99 (99)
8	141 (143)	94 (110)	115 (–)	143 (145)	136 (136)

Table 6: Comparison of ciphertext-side key-recovery bits for linear attacks against **SIMON128** using the approximation $(4, 0) \rightarrow (4, 1)$.

Rounds	SIMON128		
	total	independent	
		$\kappa = 128$	$\kappa = 192$
1	0	0	0
2	2	2	2
3	9	9	9
4	21	21	21
5	45	44	45
6	77	64	73
7	117	84	102

Zero-correlation cryptanalysis. Wang et al. [37] give a zero-correlation attack on 20-round **SIMON32/64** using an 11-round zero-correlation distinguisher from round 5 to round 15 and adding five rounds before and four rounds after the distinguisher. The tested relation is

$$L_{5,\{0\}} + R_{16,\{7\}} = 0.$$

The original analysis counts 54 guessed subkey bits. However, the following nine bits from the published 54-bit list do not need to be guessed:

$$k_{0,8}, k_{1,10}, k_{2,12}, k_{3,14}, k_{4,0}, k_{16,7}, k_{17,5}, k_{18,3}, k_{19,1}.$$

The wrong-key survival probability of the zero-correlation test remains $2^{-15.33}$. Hence the expected number of surviving partial-key candidates changes from $2^{54-15.33} = 2^{38.67}$ to $2^{45-15.33} = 2^{29.67}$. The partial-sum phase is reduced to about $2^{50.58}$ 20-round **SIMON32** encryptions. The remaining 45 key relations have full rank; consequently, each surviving candidate leaves $64 - 45 = 19$ master-key bits to enumerate, giving a master-key recovery cost of about $2^{49.72}$. The total

time complexity is therefore

$$2^{50.58} + 2^{49.72} \approx 2^{51.21},$$

with data complexity 2^{32} known plaintexts. This improves the original $2^{56.96}$ estimate while keeping the same zero-correlation distinguisher and the same attack structure.

8 Conclusion

In this work, we provide a generic and versatile algorithm for identifying the minimal key information that must be guessed for a large class of attacks. Owing to its simplicity, optimality, and generality, and because we will make an implementation publicly available, we expect this approach to become a standard method in future analyses.

The focus of this paper was on improving key-recovery for existing distinguishers. A natural next step for future work is to incorporate our approach directly into the search for attacks, by optimizing the distinguisher and the corresponding key-recovery simultaneously. This strategy has proven highly fruitful in the past, and our approach fits well into this framework.

Generative AI disclosure The research ideas and scientific contributions presented in this work are the authors’ own. ChatGPT was used as a supporting tool for preliminary drafts of selected sections and parts of the experimental code. The preliminary text and code were substantially edited and revised by the authors. The authors take full responsibility for the accuracy and correctness of the paper and accompanying code.

Acknowledgments Tim Beyne is supported by a junior postdoctoral fellowship from the Research Foundation — Flanders (FWO) with reference number 1274724N. This work was supported by CyberSecurity Research Flanders with reference number VR20192203. Additionally, this research is funded by the European Union (ERC-2023-COG, SoBaSyC, 101125450). Also, this work was in part supported by the European Research Council (ERC) project 101097056 (SYMTRUST). Part of this work was carried out during Yevhen Perekhuda’s research stay at KU Leuven. Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

References

1. Aumasson, J., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of latin dances: Analysis of salsa, chacha, and rumba. In: Nyberg, K. (ed.) Fast

- Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5086, pp. 470–488. Springer (2008), https://doi.org/10.1007/978-3-540-71039-4_30
2. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017. LNCS, vol. 10529, pp. 321–345. Springer, Cham (2017), https://doi.org/10.1007/978-3-319-66787-4_16
 3. Bashiri, K., Bonnetain, X., Hosoyamada, A., Lang, N., Schrottenloher, A.: Improved quantum linear attacks and application to CAST. IACR Trans. Symmetric Cryptol. 2025(2), 124–165 (2025), <https://doi.org/10.46586/tosc.v2025.i2.124-165>
 4. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Paper 2013/404 (2013), <https://eprint.iacr.org/2013/404>
 5. 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.) Advances in Cryptology - CRYPTO 2016. LNCS, vol. 9815, pp. 123–153. Springer, Berlin, Heidelberg (2016), https://doi.org/10.1007/978-3-662-53008-5_5
 6. Beierle, C., Leander, G., Pehuda, Y.: When the wrong key lives on: The key-recovery procedure in integral attacks. In: Daemen, J., Thomé, E. (eds.) Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10-14, 2026, Proceedings, Part VI. pp. 33–62. Lecture Notes in Computer Science, Springer (2026), https://doi.org/10.1007/978-3-032-25333-0_2
 7. Beierle, C., Leander, G., Todo, Y.: Improved differential-linear attacks with applications to ARX ciphers. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III. pp. 329–358. Lecture Notes in Computer Science, Springer (2020), https://doi.org/10.1007/978-3-030-56877-1_12
 8. Beyne, T., Rijmen, V.: Linear cryptanalysis. Cambridge University Press (November 2025)
 9. Beyne, T., Verbauwhede, M.: Ultrametric integral cryptanalysis. In: Chung, K., Sasaki, Y. (eds.) Advances in Cryptology - ASIACRYPT 2024. LNCS, vol. 15490, pp. 392–423. Springer, Singapore (2024), https://doi.org/10.1007/978-981-96-0941-3_13
 10. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. pp. 2–21. Lecture Notes in Computer Science, Springer (1990), https://doi.org/10.1007/3-540-38424-3_1
 11. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. pp. 344–371. Lecture Notes in Computer Science, Springer (2011), https://doi.org/10.1007/978-3-642-25385-0_19
 12. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsøe, C.: PRESENT: an ultra-lightweight block cipher.

- In: Paillier, P., Verbaauwhede, I. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2007*. LNCS, vol. 4727, pp. 450–466. Springer, Berlin, Heidelberg (2007), https://doi.org/10.1007/978-3-540-74735-2_31
13. Boura, C., David, N., Derbez, P., Boissier, R.H., Naya-Plasencia, M.: A generic algorithm for efficient key recovery in differential attacks - and its associated tool. In: Joye, M., Leander, G. (eds.) *Advances in Cryptology - EUROCRYPT 2024*. LNCS, vol. 14651, pp. 217–248. Springer, Cham (2024), https://doi.org/10.1007/978-3-031-58716-0_8
 14. Broll, M., Canale, F., Flórez-Gutiérrez, A., Leander, G., Naya-Plasencia, M.: Generic framework for key-guessing improvements. In: Tibouchi, M., Wang, H. (eds.) *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security*, Singapore, December 6–10, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13090, pp. 453–483. Springer (2021), https://doi.org/10.1007/978-3-030-92062-3_16
 15. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks. In: Canetti, R., Garay, J.A. (eds.) *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference*, Santa Barbara, CA, USA, August 18–22, 2013. Proceedings, Part I. pp. 222–240. Lecture Notes in Computer Science, Springer (2013), https://doi.org/10.1007/978-3-642-40041-4_13
 16. Carlet, C. (ed.): *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press (2020), <https://doi.org/10.1017/9781108606806>
 17. Chen, Y., Bao, Z., Yu, H.: Differential-linear approximation semi-unconstrained searching and partition tree: Application to LEA and speck. In: Guo, J., Steinfeld, R. (eds.) *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security*, Guangzhou, China, December 4–8, 2023, Proceedings, Part III. pp. 223–255. Lecture Notes in Computer Science, Springer (2023), https://doi.org/10.1007/978-981-99-8727-6_8
 18. Collard, B., Standaert, F., Quisquater, J.: Improving the time complexity of Matsui’s linear cryptanalysis. In: Nam, K., Rhee, G. (eds.) *Information Security and Cryptology - ICISC 2007, 10th International Conference*, Seoul, Korea, November 29–30, 2007, Proceedings. pp. 77–88. Lecture Notes in Computer Science, Springer (2007), https://doi.org/10.1007/978-3-540-76788-6_7
 19. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher square. In: Biham, E. (ed.) *Fast Software Encryption, 4th International Workshop, FSE ’97*, Haifa, Israel, January 20–22, 1997, Proceedings. pp. 149–165. Lecture Notes in Computer Science, Springer (1997), <https://doi.org/10.1007/BFb0052343>
 20. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Improved linear sieving techniques with applications to step-reduced LED-64. In: Cid, C., Rechberger, C. (eds.) *Fast Software Encryption - 21st International Workshop, FSE 2014*, London, UK, March 3–5, 2014. Revised Selected Papers. pp. 390–410. Lecture Notes in Computer Science, Springer (2014), https://doi.org/10.1007/978-3-662-46706-0_20
 21. Dunkelman, O., Ghosh, S., Keller, N., Leurent, G., Marmor, A., Mollimard, V.: Partial sums meet FFT: improved attack on 6-round AES. In: Joye, M., Leander, G. (eds.) *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part I. pp. 128–157. Lecture Notes in Computer Science, Springer (2024), https://doi.org/10.1007/978-3-031-58716-0_5

22. Evertse, J.: Linear structures in blockciphers. In: Chaum, D., Price, W.L. (eds.) *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of Cryptographic Techniques*, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings. pp. 249–266. Lecture Notes in Computer Science, Springer (1987), https://doi.org/10.1007/3-540-39118-5_23
23. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D.A., Whiting, D.: Improved cryptanalysis of rijndael. In: Schneier, B. (ed.) *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*. pp. 213–230. Lecture Notes in Computer Science, Springer (2000), https://doi.org/10.1007/3-540-44706-7_15
24. Flórez-Gutiérrez, A.: Optimising linear key recovery attacks with affine walsh transform pruning. In: Agrawal, S., Lin, D. (eds.) *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV*. Lecture Notes in Computer Science, vol. 13794, pp. 447–476. Springer (2022), https://doi.org/10.1007/978-3-031-22972-5_16
25. Flórez-Gutiérrez, A., Todo, Y.: Improving linear key recovery attacks using walsh spectrum puncturing. In: Joye, M., Leander, G. (eds.) *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I*. Lecture Notes in Computer Science, vol. 14651, pp. 187–216. Springer (2024), https://doi.org/10.1007/978-3-031-58716-0_7
26. Flórez-Gutiérrez, A., Todo, Y.: Divide-and-conquer trail enumeration puncturing: Application to salsa and chacha. In: Hanaoka, G., Yang, B. (eds.) *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part I*. pp. 35–65. Lecture Notes in Computer Science, Springer (2025), https://doi.org/10.1007/978-981-95-5018-0_2
27. Flórez-Gutiérrez, A., Todo, Y.: Improved cryptanalysis of chacha: Beating pnbs with bit puncturing. In: Fehr, S., Fouque, P. (eds.) *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part I*. Lecture Notes in Computer Science, vol. 15601, pp. 427–457. Springer (2025), https://doi.org/10.1007/978-3-031-91107-1_15
28. Lai, X.: Additive and linear structures of cryptographic functions. In: Preneel, B. (ed.) *Fast Software Encryption: Second International Workshop*. Leuven, Belgium, 14-16 December 1994, Proceedings. pp. 75–85. Lecture Notes in Computer Science, Springer (1994), https://doi.org/10.1007/3-540-60590-8_6
29. Lambooi, E., Neumann, P., Verbauwhede, M., Wang, S., Zhang, T.: Meet-in-the-middle attacks on full ChiLow. *Cryptology ePrint Archive*, Paper 2025/1601 (2025), <https://eprint.iacr.org/2025/1601>
30. Leurent, G., Pernot, C., Schrottenloher, A.: Clustering effect in simon and simeck. In: Tibouchi, M., Wang, H. (eds.) *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*. pp. 272–302. Lecture Notes in Computer Science, Springer (2021), https://doi.org/10.1007/978-3-030-92062-3_10
31. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Hellese, T. (ed.) *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993,*

- Proceedings. pp. 386–397. Lecture Notes in Computer Science, Springer (1993), https://doi.org/10.1007/3-540-48285-7_33
32. Schrottenloher, A.: Quantum linear key-recovery attacks using the QFT. In: Handschuh, H., Lysyanskaya, A. (eds.) *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*. pp. 258–291. Lecture Notes in Computer Science, Springer (2023), https://doi.org/10.1007/978-3-031-38554-4_9
 33. Song, L., Liu, H., Yang, Q., Chen, Y., Hu, L., Weng, J.: Generic differential key recovery attacks and beyond. In: Chung, K., Sasaki, Y. (eds.) *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VII*. pp. 361–391. Lecture Notes in Computer Science, Springer (2024), https://doi.org/10.1007/978-981-96-0941-3_12
 34. Sun, L., Wang, W., Wang, M.: Addendum to linear cryptanalyses of three aeads with GIFT-128 as underlying primitives. *IACR Trans. Symmetric Cryptol.* 2022(1), 212–219 (2022), <https://doi.org/10.46586/tosc.v2022.i1.212-219>
 35. Todo, Y., Aoki, K.: FFT key recovery for integral attack. In: Gritzalis, D., Kiayias, A., Askoxylakis, I.G. (eds.) *Cryptology and Network Security - 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*. pp. 64–81. Lecture Notes in Computer Science, Springer (2014), https://doi.org/10.1007/978-3-319-12280-9_5
 36. Todo, Y., Morii, M.: Compact representation for division property. In: Foresti, S., Persiano, G. (eds.) *Cryptology and Network Security. CANS 2016. LNCS*, vol. 10052, pp. 19–35. Springer, Cham (2016), https://doi.org/10.1007/978-3-319-48965-0_2
 37. Wang, Q., Liu, Z., Varici, K., Sasaki, Y., Rijmen, V., Todo, Y.: Cryptanalysis of reduced-round SIMON32 and SIMON48. In: Meier, W., Mukhopadhyay, D. (eds.) *Progress in Cryptology - INDOCRYPT 2014. LNCS*, vol. 8885, pp. 143–160. Springer, Cham (2014), https://doi.org/10.1007/978-3-319-13039-2_9
 38. Zhang, L., Yao, Y., Shi, D., Chai, D., Guo, J., Wang, Z.: Neural-inspired advances in integral cryptanalysis. In: Daemen, J., Thomé, E. (eds.) *Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10-14, 2026, Proceedings, Part VI*. pp. 451–481. Lecture Notes in Computer Science, Springer (2026), https://doi.org/10.1007/978-3-032-25333-0_16
 39. Zhang, Z., Chen, Y., Song, L., Lv, Y.: Improving the differential-linear attack with applications to gift-cofb, GIFT-64 and hyena. In: Cha, S.K., Park, J. (eds.) *Information Security - 28th International Conference, ISC 2025, Seoul, South Korea, October 20-22, 2025, Proceedings*. pp. 3–23. Lecture Notes in Computer Science, Springer (2025), https://doi.org/10.1007/978-3-032-08124-7_1

A Example for Linear-Trail Enumeration

Let $\kappa = 3$, let e_0, e_1, e_2 denote the unit basis of $\mathbb{F}_2^3$, and fix the output mask $v_r = 1$. Suppose that, after two successful iterations of Algorithm 3, we have

$$L = \{e_0, e_0 + e_1\}.$$

The affine space found so far is

$$\text{aff}(L) = e_0 + \text{span}\{e_1\}.$$

Suppose that another key mask in the projected Fourier support is

$$u_0 = e_0 + e_1 + e_2,$$

whereas $e_0 + e_2$ is not in the Fourier support. In particular, $u_0 \notin \text{aff}(L)$.

This also illustrates why the trail search cannot be restricted to an algebraic complement of $V = \text{span}\{e_1\}$. For example, consider

$$V^c = \text{span}\{e_0, e_2\}.$$

The representative of the coset $u_0 + V$ contained in V^c is $e_0 + e_2$. However, this representative need not belong to the Fourier support, even though u_0 does. Restricting the trail search to V^c could therefore miss u_0 . For this reason, Algorithm 3 searches directly under the constraint

$$u_0 \notin \text{aff}(L).$$

Suppose that the first trail found for u_0 has input mask $v_0 = 0$, but that the corresponding cluster cancels:

$$C_{1,(u_0,0)}^E = \frac{1}{4} - \frac{1}{4} = 0.$$

The algorithm therefore adds the external triple

$$(u_0, 0, 1)$$

to Z . This does not exclude the same key mask with another input mask. If a subsequent search gives

$$C_{1,(u_0,1)}^E = \frac{1}{2},$$

then u_0 is added to L . We then obtain

$$\begin{aligned} \text{aff}(L) &= \text{aff}\{e_0, e_0 + e_1, e_0 + e_1 + e_2\} \\ &= e_0 + \text{span}\{e_1, e_2\}. \end{aligned}$$

The algorithm terminates when no trail with a key mask outside $\text{aff}(L)$ remains. This example also shows why Z must contain the full external triple (u_0, v_0, v_r) , rather than only the key mask u_0 .

B Explicit key relations for the 13-round PRESENT-80 attack

We number the bits of a 64-bit round key from 0, for the least significant bit, to 63, for the most significant bit. After applying Algorithm 3 jointly to the four tested output bits, one coordinate basis of the 108-dimensional round-key space consists of all 64 bits of k_{13} and the following 44 bits of k_{12} :

$$I_{12} = \{63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, \\ 47, 46, 45, 44, 43, 41, 39, 38, 37, 36, 35, 33, 31, 30, 29, 28, \\ 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16\}.$$

Let

$$J = I_{12} \setminus \{16, 17, 18\}.$$

Then $|J| = 41$. The corresponding positions in k_{13} are

$$I_{13}^{\text{dep}} = \{j - 19 : j \in J\},$$

that is,

$$I_{13}^{\text{dep}} = \{44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, \\ 28, 27, 26, 25, 24, 22, 20, 19, 18, 17, 16, 14, 12, 11, 10, 9, \\ 8, 7, 6, 5, 4, 3, 2, 1, 0\}.$$

For every $j \in J$, the 61-bit rotation gives

$$k_{13}[j - 19] = k_{12}[j] + \gamma_j,$$

where γ_j is determined by the public round counter. These relations do not pass through the nonlinear S-box update of the key schedule.

The three remaining independent variables from k_{12} are

$$k_{12}[16], \quad k_{12}[17], \quad k_{12}[18].$$

Together with all 64 bits of k_{13} , they form the 67-dimensional guessing space used in the attack.