Efficient Detection of High Probability Statistical Properties of Cryptosystems via Surrogate Differentiation

Itai Dinur[1], Orr Dunkelman[2,⋆], Nathan Keller[3,⋆⋆], Eyal Ronen[4,⋆⋆⋆], and Adi Shamir[5]

[1] Computer Science Department, Ben Gurion University, Israel
`Ben Gurion University`
[2] Computer Science Department, University of Haifa, Israel
`orrd@cs.haifa.ac.il`
[3] Department of Mathematics, Bar-Ilan University, Israel
`Nathan.Keller@biu.ac.il`
[4] Computer Science Department, Tel Aviv University, Israel
`eyal.ronen@cs.tau.ac.il`
[5] Computer Science Department, Weizmann Institute of Science, Israel
`adi.shamir@weizmann.ac.il`

Abstract. A central problem in cryptanalysis is to find all the significant deviations from randomness in a given n-bit cryptographic primitive. When n is small (e.g., an 8-bit S-box), this is easy to do, but for large n, the only practical way to find such statistical properties was to exploit the internal structure of the primitive and to speed up the search with a variety of heuristic rules of thumb. However, such bottom-up techniques can miss many properties, especially in cryptosystems which are designed to have hidden trapdoors.

In this paper we consider the top-down version of the problem in which the cryptographic primitive is given as a structureless black box, and reduce the complexity of the best known techniques for finding all its significant differential and linear properties by a large factor of $2^{n/2}$. Our main new tool is the idea of using *surrogate differentiation*. In the context of finding differential properties, it enables us to simultaneously find information about all the differentials of the form $f(x) \oplus f(x \oplus \alpha)$ in all possible directions α by differentiating f in a single arbitrarily chosen direction γ (which is unrelated to the α's). In the context of

⋆ The second author was supported in part by the Center for Cyber, Law, and Policy in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office and by the Israeli Science Foundation through grants No. 880/18 and 3380/19.

⋆⋆ The third author was supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

⋆⋆⋆ The fourth author is partally supported by Len Blavatnik and the Blavatnik Family foundation, the Blavatnik ICRC, and Robert Bosch Technologies Israel Ltd. He is a member of CPIIS.

finding linear properties, surrogate differentiation can be combined in a highly effective way with the Fast Fourier Transform. For 64-bit cryptographic primitives, this technique makes it possible to automatically find in about 2^{64} time all their differentials with probability $p \geq 2^{-32}$ and all their linear approximations with bias $|p| \geq 2^{-16}$; previous algorithms for these problems required at least 2^{96} time. Similar techniques can be used to significantly improve the best known time complexities of finding related key differentials, second-order differentials, and boomerangs. In addition, we show how to run variants of these algorithms which require no memory, and how to detect such statistical properties even in trapdoored cryptosystems whose designers specifically try to evade our techniques.

1 Introduction

Most cryptanalytic techniques against block ciphers exploit the existence of some statistical property which happens with a higher than expected probability. It is thus essential to find all such anomalies (or to demonstrate that none exists) whenever we are designing a new cryptosystem or attacking an existing cryptosystem developed by others. Note that such a search has to be carried out only once for each cryptosystem, and if it is successful, its results can be used to find an unlimited number of actual keys. Consequently, even a lengthy computational effort to find such properties can be justified.

Due to the centrality of this topic, many papers had been published about it over the last 30 years. Almost all of them had taken a bottom-up approach, in which the attacker first finds the statistical properties of small local elements (such as S-boxes), and then tries to 'glue' them together into a high probability global property. The analysis of a small n-bit S-box (e.g., with $n = 8$) is easy: For example, all its differential properties (which can be grouped together in the form of a difference distribution table, denoted by DDT) and all its linear properties (which can be grouped together in the form of a linear approximation table, denoted by LAT) can be found exhaustively in time 2^{2n}. However, the process of constructing the global properties is usually guided by various heuristics (such as testing only low Hamming weight differences, or using only the highest probability local properties), and thus it can miss many properties. In fact, knowledge of these heuristic restrictions can be exploited by the designers of trapdoored ciphers to evade attacks. For example, it is easy to attach a keyed decorrelation module [43] at the beginning and the end of a cipher in order to force any high probability differential characteristic to have high Hamming weight input and output differences. Other constructions of trapdoored cryptosystems can be found in [40] (with a planted high probability differential characteristic) and [41] (with a planted high probability linear approximation).

Even in standard (non-trapdoored) cryptosystems, such bottom-up techniques can be error-prone: There are many known cases in which the global probability differs significantly from the product of the local probabilities due to subtle correlations (as demonstrated in [12,21,37]), and where a high probability property results from the accumulation of many low probability properties along many differential characteristics or within the linear hull (which was a crucial element in the attacks described in [29,31]). Finally, it is difficult to apply such bottom-up techniques to designs in which the basic operations are block-wide (see, for example, [9]), are defined in terms of large primitives (e.g., 32-bit S-boxes), are available only in the form of a hardware token (with no description of its internal structure), or are provided in an obfuscated form (as done in many whitebox cryptosystems such as [15]).

Developing efficient top-down techniques for finding all the usable statistical properties of functions $f : \{0,1\}^n \to \{0,1\}^n$ with a large n seems to be a hard problem, which had been solved so far only in some special cases. For example, the differential properties of a moderately large cryptographic primitive which uses only additions, rotations, and XOR's (an ARX design) were studied in [8,9,10], and were used to mount differential attacks on Simon, Speck, Ascon, LEA, and other ciphers. The related problem of finding linear biases in the same special case of ARX ciphers was studied in [34,35,47], whose results were used to mount linear attacks on Speck and SM4. Another special case discussed in [1,11,13], is when the adversary uses heuristics to guess the most likely input differences, and wants to simultaneously find all the corresponding output differences in high probability differentials; notice that without such heuristics, any algorithm of this type has a high complexity of $\Omega(2^n)$. A different type of top-down algorithm is described in [17], which deals with general black box functions f, but can find only iterative differential characteristics (in which the input difference is equal to the output difference). Finally, in the quantum setting (which is not the computation model we use in this paper) there are several papers (e.g., [33,46]) which show how to find in polynomial time differential properties in a general f, but only when their probabilities p are extremely close to 1.

When we try to apply a top-down analysis to a large black box function f (e.g., a full cryptosystem with $n = 64$), finding all the 2^{2n} entries in its DDT and LAT becomes both infeasible and unnecessary, since almost all the known attacks use only their highest entries. If we are only interested in differentials $\alpha \to \beta$ which happen with probability that exceeds p, the best previously available technique (described in Sect. 2) is to try all the possible input differences α, and to compute for each α the output differences $f(x) \oplus f(x \oplus \alpha)$ for $O(p^{-1})$ randomly chosen values of x. This reduces the time complexity of finding all the significant entries in the DDT from 2^{2n} to $2^n p^{-1}$. The corresponding algorithm for finding all the significant entries in the LAT requires at least $2^n p^{-2}$ time.

In this paper we introduce a new type of top-down technique which can reduce these two complexities by a major factor of $2^{n/2}$. The main new technique we use is *surrogate differentiation*, in which we obtain information on f by examining its derivative in an arbitrary direction which is not directly related to the statistical properties we want to find. For example, when we search for a differential property in which some input difference α is mapped to some output difference β with high probability p, we want to differentiate the function f in a particular direction α by considering pairs of inputs of the form $(x, x \oplus \alpha)$ and following the evolution of $f(x) \oplus f(x \oplus \alpha)$. This differentiation of f simultaneously achieves two very different purposes: It eliminates certain constants from the expression (for example, an unknown key which was XOR'ed to the input), and it makes it possible to exploit a sequence of high-probability differential events in order to successfully predict the output difference. However, when we try to find all the high probability properties in a new cryptosystem, we do not know a-priori the actual directions α with respect to which we want to differentiate f. Our novel idea is that if we replace the real but unknown α by an unrelated but known *surrogate value*[1] γ, we can still benefit from the elimination of unknown constants, and we can save a lot of time by using the same arbitrarily chosen surrogate value $\gamma \neq 0$ to simultaneously analyze all the possible values of α via a single unified computation.

The new idea of surrogate differentiation yields a plethora of algorithms with significantly improved complexities for detecting a large variety of statistical properties in general black box functions. In the case of differentials with probability p, our new algorithm (described in Section 2) requires $O(2^{n/2}p^{-1})$ time, compared to the best previous time complexity of $O(2^n p^{-1})$. This new complexity is almost optimal, as an information-theoretic argument shows that any algorithm for this problem requires $\Omega(2^{n/2}p^{-1/2})$ evaluations of the black-box function f.

A worst-case variant of this algorithm can deal with backdoored functions: This variant requires $O(2^{n/2}p^{-3/2})$ time, and detects a hidden differential with probability p even in trapdoored cryptosystems in which the locations of the right pairs with respect to the characteristic were chosen adversarially. In addition, we present a memoryless variant of this algorithm whose time complexity is $O(\max(2^{n/2}p^{-2}, p^{-3}))$. At the end of the Section, we describe an experimental verification of our worst-case algorithm which finds all the high-probability 5-round and 6-round differentials of the NSA-designed cryptosystem Speck, and compares our top-down results to the bottom-up analysis presented in [10].

Our next algorithm (described in Section 3) can detect all linear biases of at least p in time $O(2^{n/2}p^{-2})$. Note that in terms of complexity, the results on differential and linear properties are comparable, since to sense a bias of p we need $O(p^{-2})$ data, whereas to sense a differential with probability p we need only $O(p^{-1})$ data.

These improvements make it possible to apply a top-down analysis to full size cryptosystems with $n = 64$, and to find in about 2^{64} time all their differentials with probabilities $p \geq 2^{-32}$ and all their linear biases $|p| \geq 2^{-16}$. Previously, these tasks had required at least 2^{96} time.

In Section 4 and we present improved algorithms for boomerangs, related-key differentials, and second-order differentials. Here, one cannot hope to obtain an algorithm as good as for differential and linear properties, as information-theoretic arguments yield lower bounds of $\Omega(2^{3n/4}p^{-1/4})$ for second-order dif-

[1] According to Wikipedia, a surrogate marker in clinical trials is a known measure which may correlate with the unknown clinical markers we would like to follow, but does not necessarily have a guaranteed relationship.

Property	Time	Data	Memory	Section
Differentials (fundamental alg)	$O(2^{n/2}p^{-1})$	$O(2^{n/2}p^{-1})$	$O(2^{n/2}p^{-1})$	Sect. 2.2
Differentials (memory-efficient)	$\tilde{O}(2^{n/2}p^{-1})$	$\tilde{O}(2^{n/2}p^{-1})$	$\tilde{O}(p^{-2})$	App. A
Differentials (memoryless)	$O(2^{n/2}p^{-2})$	$O(2^{n/2}p^{-2})$	O(1)	Sect. 2.3
Differentials (worst-case)	$O(2^{n/2}p^{-3/2})$	$O(2^{n/2}p^{-3/2})$	$O(2^{n/2}p^{-1/2})$	Sect. 2.4
Linear approximations	$O(2^{n/2}p^{-2})$	$O(2^{n/2}p^{-2})$	$O(2^{n/2}p^{-2})$	Sect. 3
Boomerangs	$O(2^{n}p^{-1})$	$O(2^{n})$	$O(2^{n}p^{-1})$	Sect. 4.1
Second-order differentials	$O(2^{n}p^{-2})$	$O(2^{n})$	$O(2^{n}p^{-2})$	App. B.1
Related-Key differentials	$O(2^{n}p^{-2})$	$O(2^{n}p^{-1})$	$O(2^{n}p^{-1})$	App. B.2

n — block size and key size.

Table 1: Our main results for probabilities $p \geq 2^{-n/2}$ and biases $|p| \geq 2^{-n/4}$

ferentials and boomerangs and $\Omega(2^n p^{-1/2})$ for related-key differentials. Our new algorithms have complexities of at most $O(2^n p^{-2})$ for all three types of properties, thus making it possible to detect such properties for 32-bit and sometimes 48-bit constructions in a practical amount of time.

A summary of our main results can be found in Table 1.

Our research leads to many interesting open problems, some of which are listed in the concluding Section 5. In particular, in spite of the significant improvements over previous results, our upper bounds still do not match the best known lower bounds, and there are additional statistical properties to which we do not know how to apply our new techniques.

2 Efficient Algorithms for Detecting High-Probability Differentials

Differential cryptanalysis [6] is a central cryptanalytic technique, based on tracing the development of differences during the encryption process of a pair of plaintexts. The central notion in differential cryptanalysis is a *differential*. We say that the differential $\alpha \to \beta$ for the function $f : \{0,1\}^n \to \{0,1\}^n$ holds with probability p, if $\Pr[E(x) \oplus E(x \oplus \alpha) = \beta] = p$, where $x \in \{0,1\}^n$ is chosen uniformly at random. The pairs $(x, x \oplus \alpha)$ that satisfy $f(x) \oplus f(x \oplus \alpha) = \beta$ are called *right pairs* with respect to the differential. As differential attacks exploit high-probability differentials, a central goal in differential cryptanalysis is to detect high-probability differentials efficiently.

In this section we present an algorithm that allows detecting all differentials of $f : \{0,1\}^n \to \{0,1\}^n$ that hold with probability $\geq p$, with complexity of $O(2^{n/2}p^{-1})$. This algorithm is almost optimal, as by an information-theoretic lower bound presented below, any generic algorithm for this task has complexity of $\Omega(2^{n/2}p^{-1/2})$. We also present three variants of the algorithm: a worst-case algorithm that allows detecting with a high probability also high-probability differentials that are adversarially hidden, a memoryless algorithm with only a slightly higher time complexity, and a memory-efficient algorithm that allows reducing the memory complexity to $O(p^{-2})$ without increasing the data and time complexities. We then experimentally verify our algorithm, by using it to detect all high-probability differentials of 5-round and 6-round variants of SPECK [2].

Throughout this paper (and especially when we estimate running times and the probability of false alarms) we assume that the black-box function behaves in a sufficiently random way. Any gross deviation (such as the discovery of a huge multicollision in a supposedly random function, which can slow our algorithms) is likely to cast serious doubts about the soundness of the cryptosystem's design, even if no high probability differential or linear properties are actually found. In addition, for the sake of clarity we ignore poly-logarithmic factors (i.e., factors that are polynomial in n) in all our probability and complexity estimates.

For the sake of simplicity, we first analyze the algorithms in the scenario where f has only one differential $\alpha \to \beta$ that holds with probability p, while all other differentials have significantly lower probabilities. We then show that the algorithms can be easily generalized to finding all $\geq p$-probability differentials, with no increase in the complexity. In addition, we first present the algorithms under the natural assumption that $p \geq 2^{-n/2}$, and afterwards explain the adjustments required for smaller values of p.

2.1 Previous algorithms and a lower bound

Previous algorithms. Algorithms for detecting high-probability differentials are abundant in the literature. However, almost all of them operate in a bottom-up fashion, that is, construct a 'long' differential characteristic[2] by concatenating 'short' differential characteristics, and use the probability of the differential characteristic as a lower bound on the probability of the differential. In such algorithms, the short differential characteristics can be found easily and the challenge is to find characteristics that can be 'glued together'.

Top-down algorithms for finding high-probability differentials were considered in several special cases: In [9], Biryukov and Velichkov initiated the study of algorithms detecting all high-probability differentials of the addition operation in ARX ciphers – a problem they coined 'constructing the partial DDT (pDDT)' of the operation. Several follow-up papers (e.g., [8,10]) further studied the pDDT and used it in attacks on the ciphers SIMON, SPECK, Ascon, and LEA. In [1], Albrecht and Leander initiated the study of the case where the adversary had guessed the input difference of a differential using some heuristic, and is interested in finding all output differences to which it leads with a high probability. Essentially the same problem was studied in several other works (e.g., [11,13]), under the name of *multiple differential cryptanalysis*, and its algorithms for solving it were used in attacks on SPECK (see [23,3]). In [17], Dinur et al. studied the problem of finding all high-probability *iterative* differentials of general functions, and used their results in attacks on the cipher Simon and on the iterated Even-Mansour construction. In [33], Li and Yang showed that in the

[2] We remind the reader that a differential characteristic predicts all the intermediate differences, whereas a differential is concerned only with the input difference and the output difference.

quantum setting, differentials with probability very close to 1 can be detected in polynomial time (in n), using the Bernstein-Vazirani algorithm [4]. The follow-up paper [46] further enhanced the technique and used it to attack several block cipher constructions. While these works obtained significant advancements in special cases, neither of them applies in general.

A natural top-down algorithm for detecting all differentials of a function $f : \{0,1\}^n \to \{0,1\}^n$ that hold with probability $\geq p$ is the following adaptation of the classical algorithm for constructing the Difference Distribution Table (DDT):

1. For all $\alpha \in \{0,1\}^n$, do:
   (a) Choose $4/p$ random values $x_1, x_2, \ldots, x_{4/p} \in \{0,1\}^n$.
   (b) For each $1 \leq i \leq 4/p$, compute $f(x_i) \oplus f(x_i \oplus \alpha)$ and insert it into a hash table.
   (c) Output all values β that appear in the table at least 2 times.

The data and time complexity of the algorithm is $O(2^n p^{-1})$ and its memory complexity is $O(p^{-1})$. The probability of a differential with probability $\geq p$ to be detected is more than 90% (according to standard approximation by a Poisson distribution), and the probability of a differential with probability $\ll p$ to be detected by mistake is small.

Lower bound. A simple information-theoretic argument yields a lower bound of $\Omega(2^{n/2}p^{-1/2})$ for the task of generically detecting a differential $\alpha \to \beta$ that holds with probability p. Indeed, in order to detect such a differential, the adversary must observe at least one pair with input difference α and output difference β. Assuming that those pairs are distributed randomly, this means that the adversary must observe $\Omega(1/p)$ pairs with input difference α for all α values, i.e., a total of $O(2^n \cdot p^{-1})$ pairs is needed. This number of pairs cannot be generated (even to cover most values of α) unless the plaintext set is of size $\Omega(2^{n/2}p^{-1/2})$. Thus, the complexity of any algorithm for our problem is $\Omega(2^{n/2}p^{-1/2})$.

2.2 The fundamental algorithm

In this subsection we present a probabilistic algorithm which almost matches the lower bound, and under some randomness assumptions finds any probability-p differential (with overwhelming probability) $\alpha \to \beta$ with data, memory, and time complexity of $\tilde{O}(\max(2^{n/2}p^{-1}, p^{-2}))$. We also note that the memory complexity of the algorithm can be improved to $O(p^{-2})$ without affecting the data and time complexities, as will be shown in the memory-efficient algorithm of Appendix A.

Main idea. The main observation behind the algorithm is that the output difference β can be cancelled by differentiating with a completely unrelated surrogate difference γ, and searching for right pairs $(x, x \oplus \alpha)$ for which $(x \oplus \gamma, x \oplus \gamma \oplus \alpha)$ is also a right pair. The search for two "companion" right pairs instead of a single pair has some price, and is the reason of the complexity being higher than the lower bound by a factor of $p^{-1/2}$.

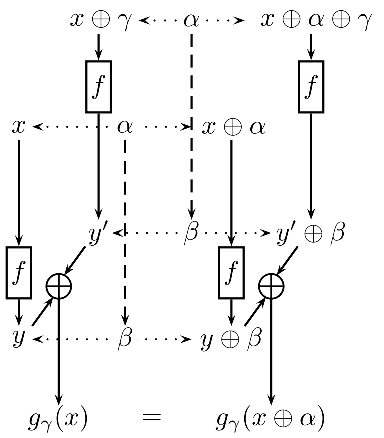


Fig. 1: A Right Quartet for the Fundamental Algorithm

Detailed description. We choose an arbitrary nonzero value $\gamma \in \{0,1\}^n$, and consider the function $g_\gamma : \{0,1\}^n \to \{0,1\}^n$ defined by $g_\gamma(x) = f(x) \oplus f(x \oplus \gamma)$. We examine the *collisions* in the function $g_\gamma(x)$. Observe that if both $(x, x \oplus \alpha)$ and $(x \oplus \gamma, x \oplus \gamma \oplus \alpha)$ are *right pairs* with respect to the differential $\alpha \to \beta$, then

$$\begin{aligned} g_\gamma(x) \oplus g_\gamma(x \oplus \alpha) =& \Big(f(x) \oplus f(x \oplus \gamma)\Big) \oplus \Big(f(x \oplus \alpha) \oplus f(x \oplus \alpha \oplus \gamma)\Big) \\ =& \Big(f(x) \oplus f(x \oplus \alpha)\Big) \oplus \Big(f(x \oplus \gamma) \oplus f(x \oplus \alpha \oplus \gamma)\Big) = \beta \oplus \beta = 0 \end{aligned}$$

and thus, the pair $(x, x \oplus \alpha)$ yields a collision in g_γ, as depicted in Figure 1. We call quartets $(x, x \oplus \alpha, x \oplus \gamma, x \oplus \alpha \oplus \gamma)$ for which this is satisfied *right quartets* for g_γ.

The fundamental algorithm is detailed in Algorithm 1. In the detection phase, we find collisions in g_γ for random inputs. From each collision, we calculate the corresponding (input difference, output difference) pair, denoted by (α, β), and increase its counter by one. Than, in the verification phase, we go over all such (α, β) pairs that were suggested sufficiently many times (i.e., with high counters). For each such (α, β) pair, we verify that indeed the probability of the differential $\alpha \to \beta$ is larger than p. We do that by taking $O(p^{-1})$ random pairs with input difference α, and test that sufficiently many of them have an output difference β.

Randomness Assumptions. The correctness of the fundamental algorithm relies on the following randomness assumptions. We assume that for any γ the event that the pair $(x \oplus \gamma, x \oplus \gamma \oplus \alpha)$ is a right pair is independent of the event that the pair $(x, x \oplus \alpha)$ is a right pair (which is similar to some of the randomness assumptions of the boomerang attack). Under this assumption, the probability of the quartet to be a right quartet for a random $x \in \{0,1\}^n$ is p^2, and thus, the expected number of collisions (each corresponding to a quartet) of this form is $p^2 2^{n-1}$ (the division by 2 is since each pair is counted twice).

In the presence of multiple differentials with probability p (or close to p) we also need to assume (for the claim that the algorithm finds almost all dif-

Algorithm 1: The Fundamental Algorithm

```
Initialize an empty list $L$ of counter tuples $(\alpha, \beta, cnt)$ and an empty hash table $H$.
Choose $M = \sqrt{n} \cdot 2^{n/2} p^{-1}$ distinct random values $x_1, x_2, \dots, x_M \in \{0,1\}^n$.
Pick at random an $n$-bit non-zero value $\gamma$.
for all $1 \leq i \leq M$ do
    Compute $g_\gamma(x_i)$ and insert it into a hash table $H$.
//Detection phase
for all collisions $g_\gamma(x_i) = g_\gamma(x_j)$ in the hash table do
    Compute the suggested (input difference, output difference) pair
      $(\alpha = x_i \oplus x_j,\ \beta = f(x_i) \oplus f(x_j))$.
    if $(\alpha, \beta, *) \notin L$ then
        add $(\alpha, \beta, 1)$ to $L$.
    else
        Increment the counter of the tuple $(\alpha, \beta, cnt)$ to $(\alpha, \beta, cnt + 1)$.
//Verification phase
for each $(\alpha, \beta, cnt) \in L$ s.t. $cnt \geq n/4$ do
    Pick $n/p$ distinct random values $\chi_1, \chi_2, \dots, \chi_{n/p} \in \{0,1\}^n$.
    Count how many times $f(\chi_i) \oplus f(\chi_i \oplus \alpha) = \beta$.
    If the counter is greater than $n/2$, output $(\alpha, \beta)$.
```

ferentials), that the distribution of right quartets for a given differential is not affected by the existence of other quartets (for a different differential).

Both assumptions are reasonable with respect to cryptographic primitives, and were used, in different contexts in previous works on cryptanalysis [7,27]. We show in Section 2.4 a worst-case algorithm which does not rely on these assumptions. Furthermore, the algorithm of Section 2.4 can find such high probability differentials even when the designer constructed the scheme to withstand the fundamental algorithm.

Success Analysis. Assume that the function f has a differential $\alpha \to \beta$ with probability p. Our following analysis suggests that (under the above randomness assumptions) this differential is going to be detected with probability higher than 99%. Furthermore, we show that the probability of a differential with probability much lower than p, e.g., $p/10$, to be proposed by our algorithm is negligible.

The data contains $M = \sqrt{n} \cdot 2^{n/2}/p$ inputs, that can be combined into $n/2 \cdot 2^n/p^2$ pairs (of $g_\gamma()$ outputs). Each such pair (of $g_\gamma()$ outputs) determines an α value, and thus, for each α value we expect about $n/2 \cdot 1/p^2$ pairs $(x, x \oplus \alpha)$ and $(x \oplus \gamma, x \oplus \gamma \oplus \alpha)$. As per our randomness assumption, each of these two pairs is right w.r.t. the differential $\alpha \to \beta$ with probability p. Hence, out of the $n/(2 \cdot p^2)$ pairs (of pairs), we expect $n/2$ cases where both pairs $(x, x \oplus \alpha)$ and $(x \oplus \gamma, x \oplus \gamma \oplus \alpha)$ are right ones. When these pairs are right pairs, they suggest a collision in the output of $g_\gamma()$. Hence, we expect $n/2$ collisions in H for the (input difference, output difference) pair (α, β).

Detection Probability \ Block Size	$n = 32$	$n = 64$	$n = 128$	$n = 256$	$n = 512$
High prob. differential (true positive)	0.99	0.999	0.999996	$1 - 2^{-32.7}$	$1 - 2^{-61.4}$
Low prob. differential (false positive)	$2^{-36.6}$	$2^{-71.0}$	$2^{-139.1}$	$2^{-275.0}$	$2^{-546.2}$

Table 2: Probability that a high (low) probability differential is detected by the fundamental algorithm leading to a true (false) positive result

Assuming that the number of actual collisions follows the Poisson distribution with a mean value $n/2$ (which is the approximation of the binomial distribution in this case) with a very high probability, the counter of (α, β) is advanced at least $n/4$ times. We list this probability for common values of n in Table 2 and will offer the full analysis in the full version of the paper. Thus, the differential $\alpha \to \beta$ is detected with probability of over 99%. This holds for all differentials with probability at least p.

We note that the verification step at the end of the algorithm verifies that the candidate (α, β) offers a differential. The probability of an (α, β) with probability p (or higher) to fail the verification step is negligible (again under the assumption that the number of right pairs follows the Poisson distribution with a mean value of $n/2$).

We now turn our attention to the probability that a "wrong" differential is detected (i.e., a differential with probability less than $p/10$). We start the discussion with the detection phase (what is the chance that such a differential is suggested). Table 2 contains the probability of a low probability differential (i.e., with probability at most $p/10$) to be offered more than $n/4$ times in $n/(2 \cdot p^2)$ quartets. While the full analysis will be given in the full version of the paper, it is easy to see that the probability of such a differential to be detected is lower than 2^{-n}. Hence, as there are at most $10 \cdot 2^n/p$ differentials with probability $p/10$, we expect at most $O(1/p)$ such differentials to be analyzed in the verification step.

We note that the probability of a differential to pass the verification step is equal to the probability of the detection phase. This follows the fact that we picked the number of pairs and the threshold to be the same as in the detection phase. As the number of right pairs following a differential is distributed according to the Poisson distribution is the same, we conclude that these are the same passing probabilities. We note that if one can reduce the complexity of the verification step in exchange for possibly higher number of "wrong" differentials which pass the verification step.

Of course, the probability of those differentials to pass the verification step is negligible.

Complexity Analysis. We first note that if there are no differentials with probability even close to p (e.g., all other differentials happen with probability close to 2^{-n}), the probability of a collision in $g_\gamma()$ is 2^{-n}. Hence, the data is expected to contain $M^2/2 \cdot 2^{-n} = (\sqrt{n}2^{n/2}/p)^2/2 \cdot 2^{-n} = n \cdot p^{-2}$ "random" collisions, for which the proposed (input difference, output difference) values are distributed randomly over the 2^{2n} possible values. Hence, the probability that some random (input difference, output difference) phase is suggested $n/4$ times is negligible.

As discussed above, the probability that a differential $\alpha' \to \beta'$ with probability at most $p/10$ is suggested in the detection phase is less than 2^{-n} (see Table 2). Hence, at most $10/p$ such differentials are expected to pass the detection phase.

The time complexity of the data collection phase is $2 \cdot M = \sqrt{n}2^{-n/2+1}/p$ calls to $f(\cdot)$ (in addition to M XORs and M memory accesses).

If there is a single high probability differential, and not too many "wrong" differentials, we expect besides the collisions related to it to have another $O(n/p^2)$ random collisions. Each such collision is expected to be suggesting a single value[3] for (α', β'). As stated above, we expect to have about $O(n/p^2)$ collisions, and they are expected to be distributed uniformly (even for the high probability differential). Hence, we expect only a few increments to take place.

If there are many "wrong" differentials (i.e., with probability lower than $p/10$, but not negligible), we expect many collisions — we expect about $n/200$ collisions for each such differential. While the chances of any such differential to pass into the verification step is negligible (and there are at most $O(n/p)$ of those), they can still incur a very high computational load — there are at most $10 \cdot 2^n/p$ such differentials, and if each of them leads to $n/200$ collisions, we expect about $O(n \cdot 2^n/p)$ collisions. However, when there are many of those, we can identify that the function which is studied is far from being a "random" function, which would suggest it is not suitable for cryptographic uses.[4]

The verification step takes $O(n/p)$ for each differential that passed the detection phase. When there is a single right differential and only very low probability differentials, then this step costs $2n/p$ calls to $f(\cdot)$. When there are multiple "wrong" differentials, as noted before, we expect to have $2n/p$ calls to $f(\cdot)$ for each of them, i.e., about $20n/p^2$ calls for $f(\cdot)$ in total.

Hence, the time complexity of the verification step is expected to be $O(n/p)$ when there are not many "wrong" differentials. When there are many of those (an event which is detected in the detection step), the complexity is $O(n/p^2)$.

The memory complexity of the algorithm is determined by the hash table size and the size of the list L. The size of the hash table is $O(M) = O(\sqrt{n}2^{n/2}/p)$ words of memory. The size of the list L depends on the number of collisions. As mentioned above, when there are only differentials with negligible probability (besides the right differential), this list is going to contain $O(n/p^2)$ values (each corresponding to a random collision) and about n ones for the real differential. When there are many "wrong" differentials, the size of this list is going to grow (and this would be a good indication that the function $f(\cdot)$ is not a "good" pseudo-random function).

[3] If there are more than two values colliding, then each pair of collisions suggests a value for α' and β'.

[4] In other words, one can easily define a statistical test based on the fundamental algorithm, and reject that function as a random function (or a random permutation) if the number of collisions exceeds $O(n/p^2)$.

To conclude, the time complexity of the algorithm is $O(n \cdot 2^{n/2}/p)$ calls to $f(\cdot)$ and similar memory complexity. This holds as long as there are not too many "wrong" differentials which exist in the scheme (i.e., there are not too many differentials with probability below $p/10$ that are detected). The existence of many of those may suggest that the function is not suitable for cryptographic uses, and while the fundamental algorithm succeeds in finding the high probability differential (and discarding all "wrong" differentials), its complexity may be higher.

The case $p \leq 2^{-n/2}$. The algorithm can be applied in this case as well, however the number of plaintexts pairs of g_γ it examines – $O(2^n p^{-2})$ – is larger than 2^{2n-1}, which is the total number of plaintext pairs of g_γ. In order to obtain more than 2^{2n-1} plaintext pairs, we can consider functions g_{γ_i} for different values of the 'surrogate' difference γ_i. (This trick is similar to the use of 'flavors' in Hellman's classical time/memory tradeoff attack [25]). Note however that collisions are meaningful only within the same function g_γ and not between two functions g_{γ_i} and g_{γ_j}. Thus, in order to obtain $O(2^n p^{-2})$ pairs, we have to consider the entire codebook of 2^n inputs for $O(2^{-n} p^{-2})$ functions g_{γ_i}.

Thus, the data complexity of the algorithm in this case is 2^n (the entire codebook), and the time and memory complexity is $O(p^{-2})$, which is the expected total number of collisions in the functions g_{γ_i}. Recall that the simple algorithm described above allows detecting a probability-p differential in time $O(2^n p^{-1})$. Our algorithm outperforms this algorithm for all values of p.

Unifying the two ranges of p values, the data complexity of the algorithm is $O(\min(2^{n/2}p^{-1}, 2^n))$ and its time and memory complexity is $O(\max(2^{n/2}p^{-1}, p^{-2}))$.

Detecting all high-probability differentials. If there are k differentials with probability p, the algorithm will simply detect all of them, with no additional cost. (The only case in which some additional cost is incurred is when there are *lots* of high-probability differentials: If $k > \max(2^{n/2}p^{-1}, p^{-2})$, then the complexity is $O(k)$, as the function g_γ is expected to have $O(k)$ collisions.)

We note that for the above complexity analysis we assume that there are not too many differentials which are suggested in the detection phase. When this happens, the time complexity of the algorithm may not be correct. Specifically, if there are more than $2^{n/2}$ differentials with probability higher than p, we expect the verification step to be more expensive than the detection phase. We note that the existence of many high probability differentials (which can be detected by observing there are many candidates for the verification step), may suggest that the studied primitive is far from being a secure one.

2.3 A memoryless variant of the algorithm

We now present a memoryless variant of the algorithm, with query complexity[5] of $O(\min(2^{n/2}p^{-2})$ and time complexity of $O(\max(2^{n/2}p^{-2}, p^{-3}))$. In other words, the cost for using only a constant-sized memory is increasing the data and time complexity by a factor of $1/p$, compared to the fundamental algorithm presented above. This variant outperforms all previously known algorithms for this task for all $p \geq 2^{-n/2}$. Moreover, it is trivially parallelizable.

The algorithm. A memoryless variant of the fundamental algorithm presented above is given in Algorithm 2. Note that the collision finding steps for each value of i are completely independent. This allows for a simple parallelization of the algorithm.

Algorithm 2: The Memoryless Algorithm

```
while $(\alpha, \beta)$ were not identified do
    Pick at random an $n$-bit non-zero value $\gamma$.
    //Collision finding phase
    Find a collision $g_\gamma(x) = g_\gamma(y)$ in the function $g_\gamma$ using Pollard Rho's
      algorithm.
    Denote $\alpha = x \oplus y$, $\beta = f(x) \oplus f(y)$.
    Choose $c/p$ random values $y_1, y_2, \ldots, y_{c/p} \in \{0,1\}^n$. ($c$ depends on the
      success rate)
    //Verification phase
    for $j = 1, 2, \ldots, c/p$ do
        Check whether $f(y_i) \oplus f(y_i \oplus \alpha) = \beta$.
        Output $(\alpha, \beta)$ if equality is obtained at least $c/2$ times and break.
```

Analysis. By the analysis presented above, assuming that there are not too many "wrong" differentials, collisions suggested by right pairs form a fraction of about p^2 of the collisions of g_γ. This follows the fact that out of the $O(1/p^2)$ collisions found by the fundemental algorithm and thus, after $1/p^2$ collisions found by the Pollard Rho algorithm, we expect such a special collision. In such a case, the values of (α, β) it suggests will be verified in the last steps with a high probability. On the other hand, input differences suggested by 'random' collisions will not be approved in these steps. We note that the value of c depends on the success rate. If we wish to make sure that there are no false positive left, we need to pick c such that the probability that the $O(1/p^2)$ proposed differentials (each collision suggests a differential) are filtered. Hence, c can be picked accordingly (see for example Table 2).

[5] We alert the reader that as we discuss a memoryless algorithm, the algorithm cannot store previous values. Instead, we discuss "query" complexity to refer to the number of evaluations of the function $f(\cdot)$, which may be higher than 2^n.

The data complexity of the algorithm is $O(2^{n/2}p^{-2})$ queries (as each application of Pollard's Rho requires $O(2^{n/2})$ adaptively chosen inputs). Note that in order to avoid obtaining the same collisions many times, we use different functions g_{γ_i}, which comes at no additional cost as each collision is searched separately. As for the time complexity – if $p \geq 2^{-n/2}$, then the time complexity of the verification phase (which is p^{-3}) is smaller than the time complexity of the Pollard Rho phase, and thus, the overall time complexity is $O(2^{n/2}p^{-2})$. If $p < 2^{-n/2}$ then the verification phase is dominant, and thus, the overall time complexity is $O(p^{-3})$. Therefore, the overall time complexity of the algorithm (for any p, unifying the two regions) is $O(\max(2^{n/2}p^{-2}, p^{-3}))$.

Detecting all high-probability differentials. If there are k differential characteristics with probability p, the algorithm will simply detect all of them, by being called $k \cdot \ln(k)$ more times (due to the Coupon collector's nature of the problem — each collision suggests a different (α, β) pair, but those may repeat). Again, as before, if there is a huge number of such high probability differentials, the complexity of the algorithm may "explode".

Comparison with previous algorithms. The complexity of our algorithm should not be compared against the adaptation of the DDT computation (with complexity $O(2^n p^{-1})$) presented above, as this adaptation does not apply in the memoryless setting. In fact, the natural adaptation of the DDT computation to memoryless detection of probability-p differentials has complexity of $O(2^{2n}p^{-1})$, as one has to check each candidate differential $\alpha \to \beta$ separately, and each such check requires $O(1/p)$ time. Therefore, our algorithm is significantly faster.

However, for values of $p < 2^{-n/2}$ our algorithm is outperformed by an adaptation of the NestedRho algorithm [18]. The NestedRho algorithm considers a function $h : \{0,1\}^n \to \{0,1\}^n$ and detects – in a memoryless manner – all values $y \in \{0,1\}^n$ such that $\Pr[h(x) = y] \geq p$, when $x \in \{0,1\}^n$ is chosen uniformly at random. In our case (i.e., search for differentials), for each fixed input α, one can consider the function $h_\alpha(x) = f(x) \oplus f(x \oplus \alpha)$, and apply to it the NestedRho algorithm to detect all values β such $\Pr[h(x) = \beta] \geq p$, which are exactly all values of β such that the differential $\alpha \to \beta$ holds with probability $\geq p$. This yields an algorithm with time complexity $2^n \cdot T$, where T is the the complexity of the NestedRho algorithm for the corresponding value of p. Substituting the results from [18], one obtains complexity of $2^n p^{-1}$ for $p > 2^{-n/2}$, $2^{n/2}p^{-2}$ for $2^{-3n/4} < p < 2^{-n/2}$, $2^{-n}p^{-4}$ for $2^{-7n/8} < p < 2^{-3n/4}$, etc.

Our algorithm is faster than this variant of NestedRho for $p \geq 2^{-n/2}$, as $2^{n/2}p^{-2} < 2^n p^{-1}$ in this range. For $p < 2^{-n/2}$, the adaptation of NestedRho is faster.

2.4 A worst-case variant of the algorithm

While the fundamental algorithm presented above succeeds with a high probability when the right pairs with respect to the differentials are distributed randomly, it can be easily fooled by a trapdoor designer capable of planting the right pairs adversarially. For example, if the $t = p2^{n-1}$ right pairs with respect to the differential characteristic $\alpha \to \beta$ form a linear subspace, then only for $2t$ values of γ (which reside in this subspace) there exists some x such that both $(x, x \oplus \alpha)$ and $(x \oplus \gamma, x \oplus \gamma \oplus \alpha)$ are right pairs. As for all other values of γ, the fundamental algorithm fails almost surely, its success probability is at most $2t/2^n = p$, which might be very small.

In this section we present a worst-case algorithm which receives a function $f : \{0,1\}^n \to \{0,1\}^n$ that may be designed adversarially, and allows detecting a hidden differential characteristic that holds with probability p or distinguishing f from a random function. The memory complexity of the algorithm is $\tilde{O}(2^{n/2}p^{-1/2})$ and its data and time complexity are $\tilde{O}(2^{n/2}p^{-3/2})$. Note that the time complexity is higher than that of the fundamental algorithm by only a factor of $\tilde{O}(p^{-1/2})$.

The algorithm. The worst-case algorithm is given in Algorithm 3. We note that for $p > 2^{-n/3}$ one can simplify the algorithm (as explained later).

Algorithm 3: Worst-Case Algorithm

```
Initialize an empty list $L$ of counter tuples $(\alpha, \beta, cnt)$.
Choose $S = 200n/p$ random non-zero values $\gamma_1, \gamma_2, \ldots, \gamma_S$.
for all $1 \le i \le S$ do
    Choose $M = 4 \cdot 2^{n/2} p^{-1/2}$ random values $x_1, x_2, \ldots, x_M \in \{0,1\}^n$.
    Initialize an empty list $L_{tmp}$ of differential tuples $(\alpha, \beta)$ and an empty
      hash table $H$.
    for all $1 \le j \le M$ do
        Compute $g_{\gamma_i}(x_j)$ and insert it into a hash table $H$.
    for all collisions $g_{\gamma_i}(x_j) = g_{\gamma_i}(x_{j'})$ in the hash table do
        Compute the suggested (input difference, output difference) pair
          $(\alpha = x_i \oplus x_j,\ \beta = f(x_i) \oplus f(x_j))$.
        if $(\alpha, \beta) \notin L_{tmp}$ then
            add $(\alpha, \beta)$ to $L_{tmp}$.
    for each tuple $(\alpha, \beta) \in L_{tmp}$ do
        if $(\alpha, \beta, *) \notin L$ then
            add $(\alpha, \beta, 1)$ to $L$.
        else
            Increment the counter of the tuple $(\alpha, \beta, cnt)$ to $(\alpha, \beta, cnt + 1)$
For each $(\alpha, \beta, cnt) \in L$ such that $cnt \ge 0.28S \cdot p = 56n$ output the (input
  difference, output difference) pair $(\alpha, \beta)$.
```

Analysis. We first analyze the success probability of the algorithm in finding the differentials with probability at least p.

Lemma 1. *For $1 \le i \le S$, consider iteration i of the algorithm. Then,*

1. *For $0 < \epsilon < 1$, the counter for every differential whose probability is at most $p \cdot \epsilon$ is incremented with probability at most $16 \cdot p \cdot \epsilon^2$.*
2. *Assume that $n > 1$ and $p \cdot 2^n \geq 4$. Then, the counter for every differential whose probability is at least p is incremented with probability at least $\frac{2p}{5}$.*

Proof. Let $0 \leq q \leq 1$ and fix a differential (α, β) whose probability is q.

Consider $x_1, x_2, \ldots, x_M \in \{0,1\}^n$ picked at iteration i. For a value of γ_i, we call a pair $(x_j, x_{j'})$ γ_i-surrogate-right if both $(x_j, x_{j'})$ and $(x_j \oplus \gamma_i, x_{j'} \oplus \gamma_i)$ are right pairs.

Assume that $(x_j, x_{j'})$ is a right pair. Note that $(x_j \oplus \gamma_i) \oplus (x_{j'} \oplus \gamma_i) = x_j \oplus x_{j'} = \alpha$, and since γ_i is uniform, $(x_j \oplus \gamma_i, x_{j'} \oplus \gamma_i)$ is uniformly distributed among all pairs with difference α. Consequently,

$$\Pr_{\gamma_i}[(x_j \oplus \gamma_i, x_{j'} \oplus \gamma_i) \text{ is right} \mid (x_j, x_{j'}) \text{ is right}] = \frac{q \cdot 2^n - 2}{2^n - 2}. \tag{1}$$

Let $\mathcal{E}_{j,j'}$ be an indicator for the event that $(x_j, x_{j'})$ is a right pair. Note that for any $j \neq j'$,

$$\Pr[\mathcal{E}_{j,j'}] = 2^{-n} q,$$

and denote $q' = 2^{-n} q$.

Let $\mathcal{G}$ count the number of unordered γ_i-surrogate-right pairs. Note that the counter for (α, β) is incremented in iteration i if and only if $\mathcal{G} > 0$. Therefore, we analyze $\Pr[\mathcal{G} > 0]$ under the assumptions of each part of the lemma.

Part 1. We prove part 1 of the lemma (assuming $q \leq p\epsilon$)

For any γ_i and $j \neq j'$, let $\mathcal{G}_{j,j'}$ be an indicator for the event that $(x_j, x_{j'})$ is a γ_i-surrogate-right pair. We have

$$\mathrm{E}[\mathcal{G}_{j,j'}] = \Pr[\mathcal{G}_{j,j'} = 1] = \Pr[(x_j, x_{j'}) \text{ is } \gamma_i\text{-surrogate-right}] =$$
$$\Pr[(x_j, x_{j'}) \text{ is right}] \cdot \Pr_{\gamma_i}[(x_j \oplus \gamma_i, x_{j'} \oplus \gamma_i) \text{ is right} \mid (x_j, x_{j'}) \text{ is right}] =$$
$$q' \cdot \tfrac{q \cdot 2^n - 2}{2^n - 2} \leq q' \cdot q.$$

Therefore,

$$\begin{gathered} \mathrm{E}[\mathcal{G}] = \sum_{j<j'} \mathcal{G}_{j,j'} = 1/2 \cdot M(M-1) \cdot q' \cdot q \leq \\ M^2 \cdot 2^{-n} q \cdot q = 16 \cdot 2^n p^{-1} 2^{-n} q^2 \leq 16 \cdot p \cdot \epsilon^2. \end{gathered} \tag{2}$$

By Markov's inequality, $\Pr[\mathcal{G} > 0] = \Pr[\mathcal{G} \geq 1] \leq 16 \cdot p \cdot \epsilon^2$, concluding the first part of the lemma.

Part 2. We prove part 2 of the lemma (assuming $q \geq p$).

Let $\mathcal{E}$ count the number of unordered right pairs in $x_1, x_2, \ldots, x_M$. We begin by lower bounding $\Pr[\mathcal{E} > 0]$. We have

$$\mathrm{E}[\mathcal{E}] = \sum_{j<j'} \mathcal{E}_{j,j'} = 1/2 \cdot M(M-1) q',$$

and

$$\begin{gathered}\mathrm{E}[\mathcal{E}^2] = \mathrm{E}[(\textstyle\sum_{j_1<j_2} \mathcal{E}_{j_1,j_2})^2] \leq \\ 1/2 \cdot M(M-1)\, \mathrm{E}[\mathcal{E}^2_{j_1,j_2}] + \\ M^2(M-1) \cdot \textstyle\sum_{\{j_1,j_2,j_3\} \text{ distinct}} \mathrm{E}[\mathcal{E}_{j_1,j_2}\mathcal{E}_{j_1,j_3}] + \\ 1/4 \cdot M^2(M-1)^2 \textstyle\sum_{\{j_1,j_2,j_3,j_4\} \text{ distinct}} \mathrm{E}[\mathcal{E}_{j_1,j_2}\mathcal{E}_{j_3,j_4}] \leq \\ 1/2 \cdot M(M-1)q' + 0 + 1/4 \cdot M^2(M-1)^2(q')^2 = \\ 1/4 \cdot M(M-1)q' \cdot (2 + M(M-1)q'),\end{gathered}$$

where the last inequality uses the fact that $\Pr[\mathcal{E}_{j_1,j_2}\mathcal{E}_{j_1,j_3} = 1] = 0$, while the random variables $\mathcal{E}_{j_1,j_2}$ and $\mathcal{E}_{j_3,j_4}$ are negatively correlated.

Hence, by the second moment method,

$$\begin{gathered}\Pr[\mathcal{E} > 0] \geq \tfrac{(\mathrm{E}[\mathcal{E}])^2}{\mathrm{E}[\mathcal{E}^2]} \geq \\ \tfrac{1/4 \cdot M^2(M-1)^2(q')^2}{1/4 \cdot M(M-1)q' \cdot (2+M(M-1)q')} = \tfrac{M(M-1)q'}{2+M(M-1)q'}.\end{gathered}$$

Recall that $M = 4 \cdot 2^{n/2}p^{-1/2}$, and since $n > 1$, then $M - 1 \geq M/2$. Therefore, $M(M-1)q' \geq 1/2 \cdot M^2 2^{-n} q \geq M^2 2^{-n} p = 8$, and

$$\Pr[\mathcal{E} > 0] \geq \frac{8}{10} = \frac{4}{5}.$$

Combining this with (1) we obtain

$$\begin{gathered}\Pr[\mathcal{G} > 0] \geq \\ \Pr[\mathcal{E} > 0] \cdot \Pr_{\gamma_i}[(x_j \oplus \gamma_i, x_{j'} \oplus \gamma_i) \text{ is right} \mid (x_j, x_{j'}) \text{ is right}] \geq \tfrac{4}{5} \cdot \tfrac{q \cdot 2^n - 2}{2^n - 2} \geq \tfrac{2p}{5},\end{gathered}$$

where the last inequality uses the assumption that $q \cdot 2^n \geq p \cdot 2^n \geq 4$ (and therefore $\frac{p \cdot 2^n - 2}{2^n - 2} \geq \frac{p}{2}$). This concludes the proof of the second part of the lemma. ■

Lemma 2 (Correctness of Algorithm 3). *Assume that $n > 1$ and $p \cdot 2^n \geq 4$. Then, with probability at least $1 - 2^{-0.4n}$:*

1. *No differential with probability at most $p/10$ is output by the algorithm, and*
2. *All differentials with probability at least p are output by the algorithm.*

Note that the lemma does not guarantee anything about differentials with probability in the range $(p/10, p)$. This has to be taken into account when setting the value of p.

Proof. Fix a differential (α, β) and assume that it is output in an iteration with probability q. Let $\mathcal{C}$ be the value of the counter for this differential at the end of the algorithm. We have

$$\mathrm{E}[\mathcal{C}] = S \cdot q.$$

Since the iterations are independent we will use a standard Chernoff bound, which states that for any $0 < c < 1$,

$$\Pr[|\mathcal{C} - \mathrm{E}[\mathcal{C}]| \geq c \cdot \mathrm{E}[\mathcal{C}]] \leq e^{-\frac{c^2 \cdot \mathrm{E}[\mathcal{C}]}{3}}.$$

Recall that the differential is output if its counter value is at least $0.28S \cdot p$.

Case 1. If the probability of the differential is at most $p/10$, by the first part of Lemma 1 (invoked with $\epsilon = 1/10$), $\mathrm{E}[\mathcal{C}] \leq 1/6 \cdot S \cdot p$. By the Chernoff bound,

$$\begin{gathered}\Pr[\mathcal{C} \geq 0.28 S \cdot p] \leq \Pr[|\mathcal{C} - \mathrm{E}[\mathcal{C}]| \geq 0.1 \cdot S \cdot p] = \\ \Pr[|\mathcal{C} - \mathrm{E}[\mathcal{C}]| \geq \tfrac{0.1 \cdot S \cdot p}{\mathrm{E}[\mathcal{C}]} \cdot \mathrm{E}[\mathcal{C}]] \leq \\ e^{-\frac{(0.1 \cdot S \cdot p)^2}{3 \cdot \mathrm{E}[\mathcal{C}]}} \leq e^{-\frac{S \cdot p}{50}} \leq e^{-4n} < 2^{-2.4n},\end{gathered}$$

as $S = 200n/p$. Taking a union bound over 2^{2n} values of (α, β) values gives the first part of the lemma.

Case 2. By the second part of Lemma 1, if the probability of the differential is at least p, then $\mathrm{E}[\mathcal{C}] \geq 2/5 \cdot S \cdot p$. By the Chernoff bound,

$$\begin{gathered}\Pr[\mathcal{C} \leq 0.28 S \cdot p] \leq \Pr[|\mathcal{C} - \mathrm{E}[\mathcal{C}]| \geq \mathrm{E}[\mathcal{C}] - 0.28 \cdot S \cdot p] \leq \\ \Pr[|\mathcal{C} - \mathrm{E}[\mathcal{C}]| \geq 0.3\,\mathrm{E}[\mathcal{C}]] \leq \\ e^{-\frac{0.09 \cdot \mathrm{E}[\mathcal{C}]}{3}} \leq e^{-\frac{S \cdot p}{100}} \leq e^{-2n} < 2^{-2.4n},\end{gathered}$$

as $S = 200n/p$. Taking a union bound over 2^{2n} (α, β) values gives the second part of the lemma. ■

Lemma 3 (Time Complexity of Algorithm 3). *Let $q_{\alpha,\beta}$ denote the probability of the differential (α, β) in f. Then, the expected time complexity of Algorithm 3 is*

$$\tilde{O}\left(2^{n/2} p^{-3/2} + p^{-2} \cdot \sum_{(\alpha,\beta)|\alpha \neq 0} q^2_{\alpha,\beta}\right).$$

We note that for a random function $q_{\alpha,\beta} = \tilde{O}(2^{-n})$ for all (α, β) with very high probability, implying that $\sum_{(\alpha,\beta)|\alpha \neq 0} q^2_{\alpha,\beta} = \tilde{O}(1)$. In this case, the second term in the complexity formula is $\tilde{O}(p^{-2}) \ll \tilde{O}(2^{n/2} p^{-3/2})$ (assuming $p \gg 2^{-n}$), and therefore can be neglected. For an arbitrary function, the term $\sum_{(\alpha,\beta)|\alpha \neq 0} q^2_{\alpha,\beta}$ may become dominant. This happens when the DDT of f has many unusually large entries. The analysis below implies that we can still detect this property in time complexity $\tilde{O}(2^{n/2} p^{-3/2})$, as it results in an unusually high number of collisions in the hash table H (even though we may not be able to find the largest entry whose probability is p).

Proof. Ignoring collisions in the hash table, the expected time complexity is $\tilde{O}(S \cdot M) = \tilde{O}(2^{n/2} p^{-3/2})$. We show that the expected number of collisions in each one of the S iterations is $\tilde{O}(p^{-1} \sum_{(\alpha,\beta)|\alpha \neq 0} q^2_{\alpha,\beta})$, which completes the proof.

Fix some iteration i and a differential (α, β) with probability $q_{\alpha,\beta}$. Recall from the proof of Lemma 1 that the number of collisions in the hash table resulting from (α, β) is equal to the number γ_i-surrogate-right pairs. By (2), $\mathrm{E}[\mathcal{G}] \leq 16 \cdot p^{-1} q^2_{\alpha,\beta}$. Summing this expression over all (α, β) concludes the proof. ■

The case of $p > 2^{-n/3}$ We note that the analysis suggests that for a given γ_i value we expect $O(1/p)$ collisions, and we can test each of those using the verification procedure of the fundamental algorithm in time $O(1/p)$. Hence, instead of storing L_{tmp} and collecting those, we can just take any (α, β) difference suggested, and test them. Hence, when $1/p^2 < 2^{n/2}p^{-1/2}$ (i.e., which implies $p > 2^{-n/3}$), we do not need the counters (as we essentially wait for the first time (α, β) is suggested). The analysis above is of course still valid (up to the fact that the memory complexity can be reduced).

2.5 Experimental verification

We implemented and experimentally verified the worst-case variant of the algorithm described in Section 2.4 (which was designed to find even planted differential properties whose right pairs were adversarially chosen in order to evade the fundamental algorithm). We used our algorithm to search for all the high-probability 5-round and 6-round differentials of the NSA-designed SPECK [2]. Our top-down algorithm automatically found all the state-of-the-art differential properties which were constructed by the bottom-up analysis presented in [10]. In particular, the best 5-round differential we found was

$$(0x0211, 0x0a04) \rightarrow (0x8000, 0x840a) \quad with \quad p \approx 2^{-9}$$

and the best 6-round differential we found was

$$(0x0211, 0x0a04) \rightarrow (0x850a, 0x9520) \quad with \quad p \approx 2^{-13}.$$

3 Efficient Algorithms for Detecting High-Probability Linear Approximations

Linear cryptanalysis [36] is a central cryptanalytic technique, based on exploiting probabilistic relations between the parities of a subset of the plaintext bits and a subset of the ciphertext bits. The central notion in linear cryptanalysis is a *linear approximation.* We say that the linear approximation $\alpha \rightarrow \beta$ for the function $f : \{0,1\}^n \rightarrow \{0,1\}^n$ holds with bias p, if $\Pr[\beta \cdot f(x) = \alpha \cdot x] = \frac{1}{2} + p$, where $x \in \{0,1\}^n$ is chosen uniformly at random and '$\cdot$' denotes a scalar product modulo 2. The values x that satisfy $\beta \cdot f(x) = \alpha \cdot x$ are called *right values* with respect to the approximation. As linear attacks exploit approximations with a high bias (in absolute value), a central goal in linear cryptanalysis is to detect high-bias approximations efficiently.

In this section we present an algorithm that allows detecting all linear approximations of $f : \{0,1\}^n \rightarrow \{0,1\}^n$ with bias $\geq p$ in absolute value with complexity of $O(2^{n/2}p^{-2})$, provided $p \geq 2^{-n/4}$.

For the sake of simplicity, we omit the words 'in absolute value' in the sequel, but throughout this section, all 'high-bias' approximations detected by the algorithms include those with a strong negative bias.

3.1 Previous algorithms and a lower bound

Previous algorithms. Algorithms for detecting high-bias linear approximations are abundant in the literature. However, as was described in the introduction, almost all of them operate in a bottom-up fashion, that is, construct a 'long' linear approximation by concatenating 'short' linear approximations. In such algorithms, the short approximations can be found easily and the challenge is to find approximations that can be 'glued together'. Top-down algorithms for finding high-bias linear approximations were considered in several papers, under the name *partial linear approximation table* (pLAT), and were applied to attack the ciphers Speck and SM4 [34,35,47]. However, all these papers considered the special case of the addition operation in ARX ciphers, and not the general case.

A linear approximation with bias of $\pm 1/2$ can be found in polynomial time in n, by solving a system of $2n$ linear bit equations in the variables α, β. For somewhat smaller biases, algorithms for the Learning Parity with Noise (LPN) problem (see, e.g., [22] and the references therein) can be used to detect (α, β) in time faster than 2^n. However, the amount of noise increases rapidly as the bias is reduced, so that these algorithms are not effective even for moderately small biases like $1/4$.

A natural top-down algorithm for detecting all linear approximations of a function $f : \{0,1\}^n \to \{0,1\}^n$ that hold with bias $\geq p$ is the following adaptation of the classical algorithm for constructing the Linear Approximation Table (LAT), which also uses the classical Goldreich-Levin algorithm [24]:

1. For all $\beta \in \{0,1\}^n$, do:
   (a) Define an auxiliary Boolean function $f_\beta : \{0,1\}^n \to \{0,1\}$ by $f_\beta(x) = \beta \cdot f(x)$.
   (b) Use the Goldreich-Levin algorithm to find all Fourier coefficients $\hat{f}_\beta(\alpha)$ that are larger than p in absolute value.
   (c) For each such α, output the pair (α, β) as the (input,output) mask of a high-bias linear approximation.

The time complexity of the algorithm is $\tilde{O}(2^n p^{-6})$, as the Goldreich-Levin algorithm (whose complexity is $\tilde{O}(p^{-6})$) is applied 2^n times.[6] By the analysis of the Goldreich-Levin algorithm, with a high probability all linear approximations with bias $\geq p$ are detected, and no linear approximation with bias $\ll p$ is detected by mistake.

Lower bound. Unlike the case of differentials, the information-theoretic lower bound for finding high-bias linear approximations is rather low. Indeed, $O(p^{-2})$ samples are sufficient for detecting any linear approximation that holds with

[6] As in most applications of the Goldreich-Levin algorithm, p is assumed to be a constant independent of n (or at least larger than $\mathrm{poly}(n^{-1})$), the exact dependence of the algorithm's complexity on p was not computed explicitly. We use the value $\tilde{O}(p^{-6})$, obtained by tracing the proof of the Goldreich-Levin theorem presented in [38]. We note that other proofs may lead to a better dependence on p, especially if one may assume that there are only a few 'large' coefficients.

bias $\geq p$ with a high probability. Given this amount of samples, all linear approximations can be detected by an exhaustive search over all possible values of (α, β), reusing the same data set.

3.2 A new efficient algorithm

In this subsection we present an algorithm which detects a 'hidden' linear approximation $\alpha \to \beta$ that holds with a bias of p, with data, memory, and time complexity of $O(2^{n/2}p^{-2})$. In fact, it detects all linear approximations that hold with a bias of $\geq p$ with the same complexity (unless the number of such approximations is extremely large, in which case the complexity is approximately equal to the number of approximations). The algorithm uses surrogate differentiation, as well as a shrinking step and application of the Fast Fourier Transform (or more precisely, the Walsh-Hadamard transform).

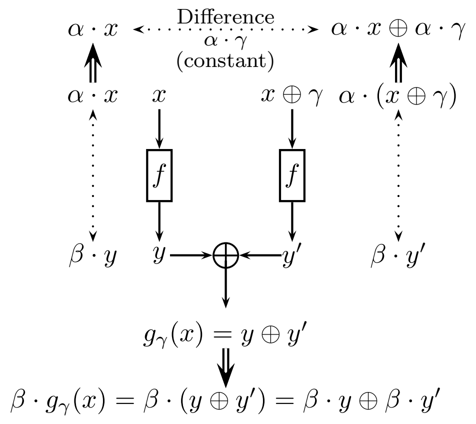
Main idea. The basic observation behind the algorithm is that the input mask α of the linear approximation can be 'cancelled' by using surrogate differentiation – that is, by considering the function $g_\gamma(x) = f(x) \oplus f(x \oplus \gamma)$ for an arbitrary value γ and examining its linear approximations of the form $0 \to \beta$. Indeed, note that for any fixed γ, we have

$$\beta \cdot g_\gamma(x) = \beta \cdot (f(x) \oplus f(x \oplus \gamma)) = (\beta \cdot f(x) \oplus \alpha \cdot x) \oplus (\beta \cdot f(x \oplus \gamma) \oplus \alpha \cdot (x \oplus \gamma)) \oplus \alpha \cdot \gamma.$$

As $\alpha \cdot \gamma$ is a constant that does not depend on x, it affects only the *sign* of the bias of the approximation $0 \to \beta$ via g_γ but not its absolute value. Hence, we can assume that $\alpha \cdot \gamma = 0$ and neglect it, remembering that the sign of the bias may be reversed. After neglecting this term, we see that $\beta \cdot g_\gamma(x) = 0$ if and only if either both $x, x \oplus \gamma$ are 'right values' with respect to the approximation $\alpha \to \beta$ for f, or neither of them is. Therefore, the linear approximation $0 \to \beta$ for g_γ holds with bias of $\pm 2p^2$ (as a concatenation of two linear approximations with bias β). The relation between the approximation $\alpha \to \beta$ for f and the approximation $0 \to \beta$ for g_γ is demonstrated in Figure 2.

While the bias of approximations of this form (i.e., $2p^2$) is significantly lower than the bias of the original approximation of f, they do not contain the parameter α, which will allow us to detect them more efficiently. We note that in this technique we create a linear relation between two outputs of f, using two linear approximations that connect these outputs of f to the corresponding inputs and an unknown but fixed relation between the inputs. Similar ideas were used in differential-linear cryptanalysis [30], e.g., in the differential-linear attack on the cipher COCONUT98 [5], where a decorrelation module applied in the middle of the cipher makes the output difference of the differential (which is the difference between the inputs to the linear approximation) unknown, but leaves it fixed.

Detailed description. As written above, we choose an arbitrary value $\gamma \in \{0,1\}^n$, and consider the function $g_\gamma : \{0,1\}^n \to \{0,1\}^n$ defined by $g_\gamma(x) = f(x) \oplus f(x \oplus \gamma)$. We want to find all linear approximations of g_γ of the form $0 \to \beta$ that hold



Difference
$\alpha \cdot \gamma$
(constant)
$\alpha \cdot x$
$\alpha \cdot x \oplus \alpha \cdot \gamma$
x
$x \oplus \gamma$
$\alpha \cdot (x \oplus \gamma)$
f
$\beta \cdot y$
y
y'
$\beta \cdot y'$
$g_\gamma(x) = y \oplus y'$
$\beta \cdot g_\gamma(x) = \beta \cdot (y \oplus y') = \beta \cdot y \oplus \beta \cdot y'$

Fig. 2: The relation between the linear approximation $\alpha \to \beta$ for f and the linear approximation $0 \to \beta$ for g_γ.

with bias $\geq 2p^2$. In other words, we want to find all high values in the *row* of the LAT of g_γ that corresponds to input mask 0. Note that this task is different from the usual way of computing the LAT, which works column-wise (i.e., by fixing the output mask β, as was described above). Usually, these tasks are equivalent, as the rows in the LAT of a permutation correspond to columns in the LAT of the inverse permutation. However, in our case, we do not have access to the inverse of g_γ (which is not even well defined since g_γ is not a permutation), and so a somewhat more complex procedure is needed.

A standard way to achieve this goal is to define an auxiliary function $h_\gamma : \{0,1\}^n \to \mathbb{Z}_{\geq 0}$ by $h_\gamma(y) = |\{x \in \{0,1\}^n : g_\gamma(x) = y\}|$. Note that for each mask β, the bias of the linear approximation $0 \to \beta$ for g_γ is

$$\frac{1}{2}\left(|\{x : g_\gamma(x) \cdot \beta = 0\}| - |\{x : g_\gamma(x) \cdot \beta = 1\}|\right) = \frac{1}{2}\hat{h}_\gamma(\beta).$$

Hence, the values of β we search for consist of the set $\{\beta \in \{0,1\}^n : |\hat{h}_\gamma(\beta)| \geq 4p^2\}$. Using the Goldreich-Levin algorithm, all these values can be found in time $\tilde{O}(p^{-6})$, once all inputs of h_γ are known. However, computing all these inputs requires 2^n time, and we aim at a significantly faster algorithm.

Instead, we first apply a *shrinking transformation*, in a way that resembles the LF1 algorithm [32] for the LPN problem. Specifically, we shrink the output size of g_γ to $n/2 + \lfloor \log(p^{-2}) \rfloor$ bits by looking only at values x such that the last $\lceil n/2 - \log(p^{-2}) \rceil$ bits of $g_\gamma(x)$ are zeros.[7] (The choice of the range's size is explained below). Note that for any β, the contribution of each of these values of x to the linear approximation $0 \to \beta$ of g_γ is equal to its contribution to the linear approximation $0 \to \bar{\beta}$ of the restriction of g_γ to the first $n/2 + \lfloor \log(p^{-2}) \rfloor$ output bits, where $\bar{\beta}$ is the restriction of β to the same bits. Hence, we can find $\bar{\beta}$ by examining the restricted function $\bar{g}_\gamma$ whose range is $\{0,1\}^{n/2+\lfloor \log(p^{-2}) \rfloor}$, and find the rest of β by repeating the procedure with restriction to the last bits.

[7] We note that one can choose any constant as the "target", as long as it is consistent with the constant used in the second part of algorithm mentioned later.

Once the shrinking is applied, we find all linear approximations of the form $0 \to \bar{\beta}$ of the function $\bar{g}_\gamma$ by defining the corresponding auxiliary function $\bar{h}_\gamma : \{0,1\}^{n/2+\lfloor \log(p^{-2}) \rfloor} \to \mathbb{Z}_{\geq 0}$ and computing its Walsh–Hadamard transform. The values $\bar{\beta}$ such that $|\hat{\bar{h}}_\gamma(\bar{\beta})| \geq 4p^2$ are those which correspond to the high-bias approximations we search for, as was explained above. The fundamental algorithm is detailed in Algorithm 4. In the first part of the algorithm, for a vector $y \in \{0,1\}^m$, we denote by y_{upper} (resp., y_{lower}) the truncation of y to the $n/2 + \lfloor \log(p^{-2}) \rfloor$ upper (resp., lower) bits. In the second part of the algorithm, we denote by $y_{upper'}$ (resp., $y_{lower'}$) the truncation of y to the $n/2 + \lfloor \log(p^{-1}) \rfloor$ upper (resp., lower) bits.

Randomness Assumptions. The correctness of the fundamental algorithm relies on the following randomness assumptions. We assume that for any γ, the event that x satisfies the linear approximation is independent of the event that $x+\gamma$ satisfies the approximation (which is similar to some of the randomness assumptions of differential-linear attacks). Under this assumption, the probability that either both $x, x+\gamma$ or neither of them satisfy the approximation is $1/2 \pm 2p^2$.

In the presence of multiple linear approximations with bias p (or close to p) we also need to assume that the distribution of values which satisfy one linear approximation is not affected by the distribution of values that satisfy the other high-bias approximations.

Success Analysis. Assume that the function f has a linear approximation $\alpha \to \beta$ with bias p. Our following analysis suggests that (under the above randomness assumptions) this approximation is going to be detected with an overwhelming probability. Furthermore, we show that the probability of a linear approximation with bias much lower than p, e.g., $p/10$, to be proposed by our algorithm is negligible.

The data contains $M = n \cdot 2^{n/2}/p^2$ inputs. After the first shrinking phase, the sum of the counters in each of the lists L_1, L_2 is expected to be between $n/2p^4$ and n/p^4 (depending on p, due to the rounding), and with an overwhelming probability is at least $n/4p^4$. (The high probability comes from the multiplication of the amount of data by a factor of n.) This size of the lists guarantees that for any β s.t. there exists a linear approximation $\alpha \to \beta$ with bias $\geq p$, we have $|\hat{\bar{h}}_{\gamma,1}(\beta_{upper})| \geq 2p^2$ and $|\hat{\bar{h}}_{\gamma,2}(\beta_{lower})| \geq 2p^2$ with an overwhelming probability, and hence, β is going to be suggested at the first stage of the algorithm. On the other hand, for any β s.t. for any α, the bias of the linear approximation $\alpha \to \beta$ is less than $p/10$, we have $|\hat{\bar{h}}_{\gamma,1}(\beta_{upper})| < 2p^2$ and $|\hat{\bar{h}}_{\gamma,2}(\beta_{lower})| < 2p^2$ with an overwhelming probability, and hence, β is not going to be suggested at the first stage of the algorithm.

At the second stage of the algorithm (which is performed for any value of β that was suggested at the first stage), the expected size of the lists L_3, L_4 is between $n/2p^2$ and n/p^2 (depending on p, due to the rounding), and with an overwhelming probability is at least $n/4p^2$. (The high probability comes from the multiplication of the amount of data by a factor of n.) This size of the lists

Algorithm 4: Efficient Algorithm for Detecting Linear Approximations

```
Initialize the following empty lists: $L_1, L_2$ of counter tuples $(y', cnt)$, where $y'$
  is $n/2 + \lfloor\log(p^{-2})\rfloor$ bits long; $\bar{L}_1, \bar{L}_2$ of $n/2 + \lfloor\log(p^{-1})\rfloor$-bit values;
  $L_3, L_4, \bar{L}_3, \bar{L}_4$ of $n/2 + \lfloor\log(p^{-1})\rfloor$-bit values; and $L_5$ of $n$-bit values.
Choose $M = n \cdot 2^{n/2}p^{-2}$ random distinct values $x_1, x_2, \ldots, x_M \in \{0,1\}^n$.
Pick at random an $n$-bit non-zero value $\gamma$.
for all $1 \le i \le M$ do
    Compute $g_\gamma(x_i)$.
    if the last $n/2 - \lceil\log(p^{-2})\rceil$ bits of $g_\gamma(x)$ are zeros then
        if $(g_\gamma(x)_{upper}, *) \notin L_1$ then
            add $(g_\gamma(x)_{upper}, 1)$ to $L_1$.
        else
            Increment the counter of $(g_\gamma(x)_{upper}, cnt)$ to $(g_\gamma(x)_{upper}, cnt+1)$
    if the first $n/2 - \lceil\log(p^{-2})\rceil$ bits of $g_\gamma(x)$ are zeros then
        if $(g_\gamma(x)_{lower}, *) \notin L_2$ then
            add $(g_\gamma(x)_{lower}, 1)$ to $L_2$.
        else
            Increment the counter of $(g_\gamma(x)_{lower}, cnt)$ to $(g_\gamma(x)_{lower}, cnt+1)$
//First Walsh-Hadamard Transform (WHT) phase – Finding $\beta$
for i=1,2 do
    Define $\bar{h}_{\gamma,i} : \{0,1\}^{n/2+\lfloor\log(p^{-2})\rfloor} \to \mathbb{Z}_{\ge 0} = cnt$ (for $(y', cnt) \in L_i$).
    Apply the fast WHT to $\bar{h}_{\gamma,i}$ to find all values $\bar{\beta}$ such that $|\hat{\bar{h}}_{\gamma,i}(\bar{\beta})| \ge 2p^2$.
    Store these values in the list $\bar{L}_i$.
Add to $L_5$ all values $\beta \in \{0,1\}^n$ such that $\beta_{upper} \in \bar{L}_1$ and $\beta_{lower} \in \bar{L}_2$.
for all $\beta \in L_5$ do
    Define the function $f_\beta(x) : \{0,1\}^n \to \{0,1\}$ by $f_\beta(x) = (-1)^{\beta \cdot f(x)}$.
    for all $1 \le i \le n \cdot 2^{n/2}p^{-1}$ do
        if the last $n/2 - \lceil\log(p^{-1})\rceil$ bits of $x_i$ are zeros then
            Insert $(x_i)_{upper'}$ to $L_3$.
        if the first $n/2 - \lceil\log(p^{-1})\rceil$ bits of $x_i$ are zeros then
            Insert $(x_i)_{lower'}$ to $L_4$.
    //Second Walsh-Hadamard Transform phase – Finding $\alpha$
    Define the function $\bar{h}_{\beta,3} : \{0,1\}^{n/2+\lfloor\log(p^{-1})\rfloor} \to \{-1,0,1\}$ by
      $\bar{h}_{\beta,3}(x) = f_\beta(x)$ if $x_{upper'} \in L_3$ and $\bar{h}_{\beta,3}(x) = 0$ otherwise.
    Define the function $\bar{h}_{\beta,4} : \{0,1\}^{n/2+\lfloor\log(p^{-1})\rfloor} \to \{-1,0,1\}$ by
      $\bar{h}_{\beta,4}(x) = f_\beta(x)$ if $x_{lower'} \in L_4$ and $\bar{h}_{\beta,4}(x) = 0$ otherwise.
    for i=3,4 do
        Apply the fast WHT to $\bar{h}_{\beta,i}$ to find all values $\bar{\alpha}$ such that $|\hat{\bar{h}}_{\beta,i}(\bar{\alpha})| \ge p$.
        Store these values in the list $\bar{L}_i$.
    Output $(\alpha, \beta)$ for all $\alpha \in \{0,1\}^n$ such that $\alpha_{upper'} \in \bar{L}_3$ and $\alpha_{lower'} \in \bar{L}_4$.
```

guarantees that for any α s.t. $\alpha \to \beta$ with bias $\ge p$, we have $|\hat{\bar{h}}_{\beta,3}(\alpha_{upper'})| \ge p$ and $|\hat{\bar{h}}_{\beta,2}(\alpha_{lower'})| \ge p$ with an overwhelming probability, and hence, α is going to be suggested at the second stage of the algorithm. On the other hand, for any α s.t. the bias of the linear approximation $\alpha \to \beta$ is less than $p/10$, we have $|\hat{\bar{h}}_{\beta,3}(\alpha_{upper'})| < p$ and $|\hat{\bar{h}}_{\beta,4}(\alpha_{lower'})| < p$ with an overwhelming probability, and hence, α is not going to be suggested at the second stage of the algorithm.

We note that unlike the case of differential characteristics, an additional verification step is not needed, since the Walsh-Hadamard steps filter out all linear approximations with bias of $< p/10$ with an overwhelming probability. A full analysis will be presented in the full version of the paper.

Complexity analysis. The first shrinking step of the algorithm has complexity of $O(n2^{n/2}p^{-2})$. As the filtering step checks the equality of $n/2 - \lceil \log(p^{-2}) \rceil$ output bits to zeros, the sum of the counters in each of the lists L_1, L_2 is expected to be $O(np^{-4})$. The functions $\bar{h}_{\gamma,i}$ are on $n/2 + \lfloor \log(p^{-2}) \rfloor$ bits, and hence, applying the Walsh–Hadamard transform to each of them requires about $O(n^2 2^{n/2}p^{-2})$. As explained above, after this step for each value β s.t. there exists a linear approximation $\alpha \to \beta$ with bias $\geq p$, the values β_{upper} and β_{lower} will be suggested with an overwhelming probability. The suggestions for β can be reconstructed from the suggested values of β_{upper} and β_{lower} efficiently, by going over the possible values of the $2\lfloor \log(p^{-2}) \rfloor$ common bits of β_{upper} and β_{lower}, finding collisions and completing the value of β for the colliding values. Thus, the complexity of this stage is negligible w.r.t. the complexity of the previous stages.

The second phase of the algorithm is performed for all values of β suggested in the first part. For each such value of β, the complexity of the shrinking phase is $O(n2^{n/2}p^{-1})$ and the sizes of the lists L_3, L_4 constructed in it is expected to be $O(np^{-2})$. The functions $\bar{h}_{\beta,i}$ are on $n/2 + \lfloor \log(p^{-1}) \rfloor$ bits, and hence, applying the Walsh–Hadamard transform to each of them requires about $O(n^2 2^{n/2}p^{-1})$ steps. As was explained above, after this step for each value α s.t. the bias of the linear approximation $\alpha \to \beta$ is $\geq p$, the values $\alpha_{upper'}$ and $\alpha_{lower'}$ will be suggested with an overwhelming probability. The suggestions for α can be reconstructed from the suggested values of $\alpha_{upper'}$ and $\alpha_{lower'}$ efficiently, like in the first phase of the algorithm.

Hence, the time complexity of the algorithm is $O(n^2 2^{n/2}p^{-2} + tn^2 2^{n/2}p^{-1})$, where t is the number of values of β suggested in the first phase of the algorithm. Therefore, if the number of values of β s.t. there exists α for which the linear approximation $\alpha \to \beta$ holds with a bias of $\geq p$ is $O(p^{-1})$ then the time complexity of the algorithm is $O(n^2 2^{n/2}p^{-2} = \tilde{O}(2^{n/2}p^{-2})$ and the algorithm outputs all linear approximations with a bias of $\geq p$. If the number of such values of β is $\gg p^{-1}$, then the algorithm still outputs all of the high-bias approximations, but its time complexity is increased, proportionally to the number of β values.

The data complexity of the algorithm is $O(n2^{n/2}p^{-2})$ and the memory complexity is $O(n2^{n/2}p^{-2})$ (which is dominated by storing the data and applying the fast Walsh-Hadamard transform in the first phase of the algorithm. We note that the output size of the shrinking was chosen in order to balance the complexities of the first shrinking and the first Walsh-Hadamard transform steps (while the complexity of the following steps is significantly lower, unless the algorithm suggests many values of β, as was explained above).

4 Detecting Other High-Probability Statistical Properties

Finally, we use surrogate differentiation to devise algorithms for detecting three other types of statistical properties commonly used in cryptanalysis: boomerangs, second-order differentials, and related-key differentials. As mentioned in the introduction, here we cannot hope for complexity as low as $O(2^{n/2})$, as in all three cases, the information-theoretic lower bound is at least $\Omega(2^{3n/4}p^{-1/4})$. We present algorithms for all three cases with complexity of at most $O(2^n p^{-2})$, which improves over the previously known results by a factor of at least $2^{n/2}$. Our algorithms allow, for the first time, to detect all high-probability boomerangs, second-order differentials and related-key differentials in 48-bit ciphers. We will now describe the algorithm for Boomerangs. The algorithms for second-order differentials and related-key differentials can be found in Appendix B.

4.1 Boomerangs

The boomerang attack [44] is a widely used cryptanalytic technique, based on viewing a cipher E as a cascade $E = E_1 \circ E_0$ and combining unrelated high-probability differentials $\alpha \to \bar{\alpha}$ of E_0 and $\bar{\beta} \to \beta$ of E_1 into a distinguisher for the entire cipher. In a boomerang distinguisher, the adversary encrypts pairs of plaintexts $(x, y = x \oplus \alpha)$, shifts the corresponding ciphertext pairs $(E(x), E(y))$ by β, and decrypts the resulting values to obtain $(z, w) = (E^{-1}(E(x) \oplus \beta), E^{-1}(E(y) \oplus \beta))$. Then she checks whether $z \oplus w = \alpha$. Analysis based on standard independence assumptions shows that if the differentials $\alpha \to \bar{\alpha}$ of E_0 and $\bar{\beta} \to \beta$ of E_1 hold with probabilities p_0, p_1, respectively, then in the boomerang process described above we have $\Pr[z \oplus w = \alpha] = p_0^2 p_1^2$, and this can be used to distinguish E from a random cipher, provided p_0, p_1 are sufficiently large.

In [37], Murphy showed that the 'naive' analysis of the boomerang attack is highly inaccurate in various practical cases, due to dependence issues in the transition between the subciphers E_0, E_1. As a result, numerous works aimed at formulating frameworks that will allow for estimating the complexity precisely (e.g., [14,16,21,27,42]). A central framework that was studied extensively in recent years is the Boomerang Connectivity Table (BCT), introduced by Cid et al. [16]. Designed to resemble the DDT and the LAT, the BCT is defined as follows. Given a function $f : \{0,1\}^n \to \{0,1\}^n$, we say that the boomerang $\alpha \to \beta$ holds for f with probability p, if $\Pr[f^{-1}(f(x) \oplus \beta) \oplus f^{-1}(f(x \oplus \alpha) \oplus \beta) = \alpha] = p$, where $x \in \{0,1\}^n$ is chosen uniformly at random. The BCT is a table of size 2^{2n}, whose entries are the probabilities of all the boomerangs of f, multiplied by 2^n. In [16], the BCT was constructed for the S-boxes in the transition between the subciphers E_0, E_1, and thus, it could be constructed by simple exhaustive search, due to the small size of the S-boxes. In [45], the BCT was extended to cover several rounds at the middle of the cipher. This increases the value of n that should be considered (as the adversary has to consider 'Super S-boxes' instead of S-boxes), thus making computation of the entire BCT infeasible and forcing the adversary to concentrate on the high-probability boomerangs.

In this subsection we use surrogate differentiation to present a new algorithm for finding all boomerangs of a function $f : \{0,1\}^n \to \{0,1\}^n$ that hold with probability $\geq p$. The algorithm has complexity of $O(2^n p^{-1})$, which makes it possible to practically detect high-probability boomerangs of 32-bit and 48-bit functions, that may represent either entire ciphers (for lightweight ciphers like Simon and Speck), BCTs computed over several rounds in general ciphers, or even single-round BCTs in ARX designs. It should be noted that the complexity of the algorithm is higher than the complexities of the algorithms for finding differentials and linear approximations presented in Sections 2 and 3, which lie in the vicinity of $2^{n/2}$. However, an information-theoretic lower bound shows that any algorithm for finding the high-probability boomerangs has complexity of more than $2^{3n/4}$, and hence, the complexity of our algorithm is not far from the lower bound.

Lower bound. Assume that the function $f : \{0,1\}^n \to \{0,1\}^n$ has a boomerang $\alpha \to \beta$ that holds with probability p. This means that there exist $p2^n$ quartets (x, y, z, w) such that $x \oplus y = z \oplus w = \alpha$ and $f(x) \oplus f(z) = f(y) \oplus f(w) = \beta$. These quartets are called *right quartets* with respect to the boomerang.

In order to detect the boomerang, the adversary must observe at least one right quartet. Assuming that the right quartets are distributed randomly, this implies (by a birthday paradox argument) that in order to observe a right quartet, the adversary must check at least $\Omega(2^{4n}/p2^n) = \Omega(2^{3n}p^{-1})$ plaintext quartets. This amount of quartets is contained in the data set only if the data set contains $\Omega(2^{3n/4}p^{-1/4})$ plaintexts. Hence, any algorithm that detects all probability-p boomerangs of f has complexity of $\Omega(2^{3n/4}p^{-1/4})$.

Previous results. As in the cases of differentials and linear approximations, a natural top-down algorithm for finding the high-probability boomerangs is an adaptation of the algorithms for computing the BCT. In [16], it was stated that the entire BCT can be computed in time $O(2^{3n})$. In [20], Dunkelman showed that the BCT can be computed in time $O(2^{2n})$ by guessing the 'ciphertext shift' β, defining the auxiliary function $\bar{f}_\beta(x) = f^{-1}(f(x) \oplus \beta)$ and finding all differentials of $\bar{f}$ of the form $\alpha \to \alpha$, which correspond to the boomerang $\alpha \to \beta$ of f. As all *iterative* differentials of a function $\bar{f} : \{0,1\}^n \to \{0,1\}^n$ can be found in $O(2^n)$ operations by an algorithm presented in [17], the overall complexity of finding the BCT is $O(2^{2n})$.

The algorithm of [20] can be easily modified to find all probability-p boomerangs of f. Indeed, this amounts to finding all probability-p iterative differential characteristics of $\bar{f}_\beta$, which can be done in time $O(2^{n/2}p^{-1/2})$ by an adaptation of the algorithm of [17]. This reduces the overall complexity of the algorithm to $O(2^{3n/2}p^{-1/2})$. The complexity cannot be reduced further by similar methods, since detection of a single probability-p differential of $\bar{f}_\beta$ requires $\Omega(2^{n/2}p^{-1/2})$ operations, by the information-theoretic lower bound presented in Section 2.

A new algorithm. As in the algorithms presented in the previous sections, we begin with surrogate differentiation, that is, we choose an arbitrary $\gamma \in \{0,1\}^n$

and define the function $g_\gamma(x) = f(x) \oplus f(x \oplus \gamma)$. We observe that collisions in g_γ may be used to detect boomerangs.

Indeed, consider a right quartet (x, y, z, w) with respect to the boomerang $\alpha \to \beta$. If $x \oplus z = \gamma$, then we have

$$\begin{aligned} g_\gamma(x) &= f(x) \oplus f(x \oplus \gamma) = f(x) \oplus f(z) = \beta = f(y) \oplus f(w) \\ &= f(x \oplus \alpha) \oplus f(x \oplus \alpha \oplus \gamma) = g_\gamma(x \oplus \alpha). \end{aligned}$$

Consequently, among the $O(2^n)$ collisions of g_γ, $O(p2^n \cdot 2^{-n}) = O(p)$ collisions on expectation stem from right quartets with respect to the boomerang. These collisions can be used to find the boomerang efficiently. (Note that such a collision $g_\gamma(x_i) = g_\gamma(x_j)$ suggests the boomerang $x_i \oplus x_j \to f(x_i) \oplus f(x_i \oplus \gamma)$.) This gives rise to Algorithm 5.

Analysis. For each value of γ_i, the function g_{γ_i} is expected to have about 2^{n-1} collisions. Hence, the algorithm examines a total of about $16 \cdot 2^n p^{-1}$ collisions, and at least 16 of them (on expectation) stem from each boomerang that holds with probability $\geq p$. As the (input, output) value suggested by all those collisions that stem from the same boomerang is the same pair (α, β) while the (input,output) values suggested by the 'random' collisions are spread among 2^{2n} values, it is expected that only the right (α, β) values will be detected.

The complexity of the algorithm is $O(2^n p^{-1})$, dominated by finding the collisions in the functions g_{γ_i}.

We note that similarly to the algorithm described above, one can easily adapt the variants of the fundamental algorithm presented in Section 2 to finding high-probability boomerangs, with complexity increased by a factor of about $2^{n/2}$.

5 Summary and Open Problems

In this paper we presented major complexity improvements in the best known techniques for detecting a wide variety of statistical properties of cryptographic primitives which deviate from random behavior in a significant way. The new algorithms can be applied to any black-box function, and in particular they are fast enough to be directly used to analyze 64-bit cryptosystems.

Besides the obvious question of whether our techniques can be further improved, here are some of the problems left open by our research:

1. Can we use similar techniques to speed up the search for significant truncated differentials?
2. Can we use similar techniques to speed up the search for significant differential-linear properties?
3. Can we close the small gap of $\sqrt{p^{-1}}$ between the upper and lower bounds on the time needed to find the significant differentials of a function f?
4. Can surrogate differentiation could be used to solve other problems in cryptography and complexity theory?

References

1. Albrecht, M.R., Leander, G.: An All-In-One Approach to Differential Cryptanalysis for Small Block Ciphers. In: Proceedings of SAC 2012. LNCS, vol. 7707, pp. 1–15. Springer (2012)
2. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. IACR Cryptol. ePrint Arch. report 404/2013 (2013)
3. Benamira, A., Gérault, D., Peyrin, T., Tan, Q.Q.: A Deeper Look at Machine Learning-Based Cryptanalysis. In: Advances in Cryptology - Proceedings of EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 805–835. Springer (2021)
4. Bernstein, E., Vazirani, U.V.: Quantum Complexity Theory. SIAM J. Comput. **26**(5), 1411–1473 (1997)
5. Biham, E., Dunkelman, O., Keller, N.: Enhancing Differential-Linear Cryptanalysis. In: Advances in Cryptology - Proceedings of ASIACRYPT 2002. LNCS, vol. 2501, pp. 254–266. Springer (2002)
6. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer (1993)
7. Biryukov, A.: The Boomerang Attack on 5 and 6-Round Reduced AES. In: Proceedings of the 4th International AES Conference. LNCS, vol. 3373, pp. 11–15. Springer (2004)
8. Biryukov, A., Roy, A., Velichkov, V.: Differential Analysis of Block Ciphers SIMON and SPECK. In: Proceedings of FSE 2014. LNCS, vol. 8540, pp. 546–570. Springer (2014)
9. Biryukov, A., Velichkov, V.: Automatic Search for Differential Trails in ARX Ciphers. In: Benaloh, J. (ed.) Proceedings of CT-RSA 2014. LNCS, vol. 8366, pp. 227–250. Springer (2014)
10. Biryukov, A., Velichkov, V., Corre, Y.L.: Automatic Search for the Best Trails in ARX: Application to Block Cipher Speck. In: Proceedings of FSE 2016. LNCS, vol. 9783, pp. 289–310. Springer (2016)
11. Blondeau, C., Gérard, B., Nyberg, K.: Multiple Differential Cryptanalysis Using LLR and χ 2 Statistics. In: Proceedings of SCN 2012. LNCS, vol. 7485, pp. 343–360. Springer (2012)
12. Blondeau, C., Leander, G., Nyberg, K.: Differential-Linear Cryptanalysis Revisited. J. Cryptol. **30**(3), 859–888 (2017)
13. Blondeau, C., Nyberg, K.: New Links between Differential and Linear Cryptanalysis. In: Advances in Cryptology - Proceedings of EUROCRYPT 2013. LNCS, vol. 7881, pp. 388–404. Springer (2013)
14. Boukerrou, H., Huynh, P., Lallemand, V., Mandal, B., Minier, M.: On the Feistel Counterpart of the Boomerang Connectivity Table Introduction and Analysis of the FBCT. IACR Trans. Symmetric Cryptol. **2020**(1), 331–362 (2020)
15. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A White-Box DES Implementation for DRM Applications. In: Proceedings of DRM 2002. LNCS, vol. 2696, pp. 1–15. Springer (2002)
16. Cid, C., Huang, T., Peyrin, T., Sasaki, Y., Song, L.: Boomerang Connectivity Table: A New Cryptanalysis Tool. In: Advances in Cryptology - Proceedings of EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 683–714. Springer (2018)
17. Dinur, I., Dunkelman, O., Gutman, M., Shamir, A.: Improved Top-Down Techniques in Differential Cryptanalysis. In: Proceedings of LATINCRYPT 2015. LNCS, vol. 9230, pp. 139–156. Springer (2015)

18. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Memory-Efficient Algorithms for Finding Needles in Haystacks. In: Advances in Cryptology - Proceedings of CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 185–206. Springer (2016)
19. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient Dissection of Bicomposite Problems with Cryptanalytic Applications. J. Cryptol. **32**(4), 1448–1490 (2019)
20. Dunkelman, O.: Efficient Construction of the Boomerang Connection Table. IACR Cryptol. ePrint Arch. report 631/2018 (2018)
21. Dunkelman, O., Keller, N., Shamir, A.: A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. J. Cryptol. **27**(4), 824–849 (2014)
22. Esser, A., Kübler, R., May, A.: LPN Decoded. In: Advances in Cryptology - Proceedings of CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 486–514. Springer (2017)
23. Gohr, A.: Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning. In: Advances in Cryptology - Proceedings of CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 150–179. Springer (2019)
24. Goldreich, O., Levin, L.A.: A Hard-Core Predicate for all One-Way Functions. In: Proceedings of STOC 1989. pp. 25–32. ACM (1989)
25. Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Trans. Inf. Theory **26**(4), 401–406 (1980)
26. Kelsey, J., Schneier, B., Wagner, D.A.: Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In: Advances in Cryptology - Proceedings of CRYPTO 1996. LNCS, vol. 1109, pp. 237–251. Springer (1996)
27. Kim, J., Hong, S., Preneel, B., Biham, E., Dunkelman, O., Keller, N.: Related-Key Boomerang and Rectangle Attacks: Theory and Experimental Analysis. IEEE Trans. Inf. Theory **58**(7), 4948–4966 (2012)
28. Knudsen, L.R.: Truncated and higher order differentials. In: Proceedings of FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer (1994)
29. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON Block Cipher Family. In: Advances in Cryptology - Proceedings of CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 161–185. Springer (2015)
30. Langford, S.K., Hellman, M.E.: Differential-Linear Cryptanalysis. In: Advances in Cryptology - Proceedings of CRYPTO 1994. LNCS, vol. 839, pp. 17–25. Springer (1994)
31. Leurent, G., Pernot, C., Schrottenloher, A.: Clustering Effect in Simon and Simeck. In: Advances in Cryptology - Proceedings of ASIACRYPT, Part I. LNCS, vol. 13090, pp. 272–302. Springer (2021)
32. Levieil, É., Fouque, P.: An Improved LPN Algorithm. In: Proceedings of Security and Cryptography for Networks SCN 2006. LNCS, vol. 4116, pp. 348–359. Springer (2006)
33. Li, H., Yang, L.: Quantum differential cryptanalysis to the block ciphers. CoRR **abs/1511.08800** (2015)
34. Liu, Y., Fu, K., Wang, W., Sun, L., Wang, M.: Linear cryptanalysis of reduced-round SPECK. Inf. Process. Lett. **116**(3), 259–266 (2016)
35. Liu, Y., Liang, H., Wang, W., Wang, M.: New Linear Cryptanalysis of Chinese Commercial Block Cipher Standard SM4. Secur. Commun. Networks **2017**, 1461520:1–1461520:10 (2017)
36. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Advances in Cryptology - Proceedings of EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer (1993)

Algorithm 5: Algorithm for Detecting High-Probability Boomerangs

```
Initialize an empty array of $2n$-bit counters and an empty hash table $H$.
Choose $S = 32/p$ random values $\gamma_1, \gamma_2, \ldots, \gamma_S$.
Pick at random an $n$-bit non-zero value $\gamma$.
for all $1 \leq i \leq S$ do
    for all $x \in \{0,1\}^n$ do
        Compute $g_{\gamma_i}(x)$ and insert it into a hash table $H$.
    for all collisions $g_{\gamma_i}(x_i) = g_{\gamma_i}(x_j)$ in the hash table do
        Increment the counter that corresponds to the (input,output) pair
          $(x_i \oplus x_j, f(x_i) \oplus f(x_i \oplus \gamma_i))$.
Output each (input, output) pair $(\alpha, \beta)$ whose counter was advanced at least 8
  times.
```

37. Murphy, S.: The Return of the Cryptographic Boomerang. IEEE Trans. Inf. Theory **57**(4), 2517–2521 (2011)
38. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press (2014)
39. van Oorschot, P.C., Wiener, M.J.: Parallel Collision Search with Cryptanalytic Applications. J. Cryptol. **12**(1), 1–28 (1999)
40. Peyrin, T., Wang, H.: The MALICIOUS Framework: Embedding Backdoors into Tweakable Block Ciphers. In: Advances in Cryptology - Proceedings of CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 249–278. Springer (2020)
41. Rijmen, V., Preneel, B.: A Family of Trapdoor Ciphers. In: Proceedings of FSE 1997. LNCS, vol. 1267, pp. 139–148. Springer (1997)
42. Song, L., Qin, X., Hu, L.: Boomerang Connectivity Table Revisited. Application to SKINNY and AES. IACR Trans. Symmetric Cryptol. **2019**(1), 118–141 (2019)
43. Vaudenay, S.: Decorrelation: A Theory for Block Cipher Security. J. Cryptol. **16**(4), 249–286 (2003)
44. Wagner, D.A.: The Boomerang Attack. In: Proceedings of FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer (1999)
45. Wang, H., Peyrin, T.: Boomerang Switch in Multiple Rounds. Application to AES Variants and Deoxys. IACR Trans. Symmetric Cryptol. **2019**(1), 142–169 (2019)
46. Xie, H., Yang, L.: Using Bernstein-Vazirani algorithm to attack block ciphers. Des. Codes Cryptogr. **87**(5), 1161–1182 (2019)
47. Yao, Y., Zhang, B., Wu, W.: Automatic Search for Linear Trails of the SPECK Family. In: Proceedings of Information Security ISC 2015. LNCS, vol. 9290, pp. 158–176. Springer (2015)

A A fixed amount of available memory variant of the high-probability differentials detection algorithm

Recall that the fundamental algorithm has time and memory complexities of $O(\max(2^{n/2}p^{-1}, p^{-2}))$, while the memoryless variant has time complexity of $O(\max(2^{n/2}p^{-2}, p^{-3}))$ (but is suboptimal for $p < 2^{-n/2}$).

We show how to exploit a fixed amount of available memory to obtain tradeoffs between these two algorithms. In fact (assuming $p \geq 2^{-n/2}$), we describe an algorithm with time complexity of $\tilde{O}(2^{n/2}p^{-1})$, similarly to the fundamental algorithm. Yet, this algorithm has reduced memory complexity of $\tilde{O}(p^{-2})$, and can therefore be considered as a strict improvement over the fundamental algorithm.

In particular, we describe two different tradeoff algorithms, where the first is an extension of the memoryless algorithm and is preferable for small values S of memory (compared to $1/p$). The second algorithm is an extension of the fundamental algorithm and performs better for larger values of S.

Both of the tradeoff algorithms use the classical Parallel Collision Search (PCS) algorithm [39], which finds C collision pairs in a random function $f : \{0,1\}^n \to \{0,1\}^n$ with memory of $S = \tilde{O}(C)$ bits in time complexity T such that $T = \tilde{O}(C \cdot 2^{n/2}S^{-1/2})$.

Tradeoff algorithm 1. Recall that we need to find $C = O(p^{-2})$ collisions in the function g_γ (which we assume to behave as a random function for the sake of the analysis). Given memory $S = \tilde{O}(C) = \tilde{O}(p^{-2})$, this is done in time complexity of $T = \tilde{O}(p^{-2}2^{n/2}S^{-1/2})$ using the PCS algorithm.

The first tradeoff algorithm tests all these $\tilde{O}(p^{-2})$ collisions (as in the memoryless algorithm), requiring additional time complexity of $\tilde{O}(p^{-3})$. Consequently, the overall time complexity becomes

$$\tilde{O}(\max(2^{n/2}p^{-2}S^{-1/2}, p^{-3}))$$

(assuming $S = \tilde{O}(p^{-2})$). Thus, the complexity of the testing phase is negligible in case $p \geq S^{1/2} \cdot 2^{-n/2}$.

Tradeoff algorithm 2. For larger values of S (and assuming $S = \tilde{O}(2^n)$), the second tradeoff algorithm eliminates the testing phase by finding more collisions (similarly to the fundamental algorithm). Specifically, an internal loop of the PCS algorithm finds a batch of (about) S collisions in g_γ in time $2^{n/2}S^{1/2}$. We repeat this loop (using different flavors) until we find two collisions that suggest the same input-output difference. The probability of this event is about $(S \cdot p^2)^2 = S^2p^4$, and hence the total time complexity is

$$\tilde{O}(2^{n/2}S^{1/2}S^{-2}p^{-4}) = \tilde{O}(2^{n/2}p^{-4}S^{-3/2}).$$

This complexity is better than that of the first tradeoff algorithm in the case $2^{n/2}p^{-4}S^{-3/2} < p^{-3}$, or $p > 2^{n/2}S^{-3/2}$ (i.e., $S > 2^{n/3}p^{-2/3}$).

In particular, for $S = \tilde{O}(p^{-2})$ (assuming $p \geq 2^{-n/2}$), the time complexity of the algorithm is $\tilde{O}(2^{n/2}p^{-1})$, which is an improvement over the fundamental algorithm as claimed above.

We note that as for the memoryless algorithm, variants of the NestedRho algorithm become faster than the algorithms described above for small values of p and S.

B Detecting Other High-Probability Statistical Properties

B.1 Second-order differentials

Higher-order differential cryptanalysis [28] is a widely used cryptanalytic technique, based on tracing the development of higher-order derivatives during the encryption process of a structured set of plaintexts. The central notion in higher-order differential cryptanalysis is a *higher-order differential.* We say that the d'th-order differential $(\alpha_1, \alpha_2, \ldots, \alpha_d) \to c$ for the function $f : \{0,1\}^n \to \{0,1\}^n$ holds with probability p, if $\Pr[\bigoplus_{v \in V} f(x \oplus v) = c] = p$, where $V \subseteq \{0,1\}^n$ is the linear subspace spanned by $\alpha_1, \ldots, \alpha_d$ and $x \in \{0,1\}^n$ is chosen uniformly at random. The values $x \in \{0,1\}^n$ that satisfy $\bigoplus_{v \in V} f(x \oplus v) = c$ are called *right values* with respect to the differential.

In this subsection we concentrate on second-order differentials, and aim at detecting all second-order differentials of $f : \{0,1\}^n \to \{0,1\}^n$ that hold with probability $\geq p$. A virtue of the lower bound argument presented for boomerangs shows that any algorithm for this task has complexity of $\Omega(2^{3n/4}p^{-1/4})$. We present a new algorithm which achieves the goal with complexity of $O(2^n p^{-2})$. Like all algorithms presented above, our algorithm uses surrogate differentiation. In addition, it makes use of the *dissection* technique presented in [19]. The algorithm can be naturally generalized to d'th order differentials for any $d \geq 2$, however its complexity becomes impractically high already for $d = 3$.

Lower bound. The lower bound argument is similar to the lower bound in the case of boomerangs. Assume that the function $f : \{0,1\}^n \to \{0,1\}^n$ has a second-order differential $(\alpha_1, \alpha_2) \to c$ that holds with probability p. This means that there exist $p2^n$ quartets $(x, x \oplus \alpha_1, x \oplus \alpha_2, x \oplus \alpha_1 \oplus \alpha_2)$ such that $f(x) \oplus f(x \oplus \alpha_1) \oplus f(x \oplus \alpha_2) \oplus f(x \oplus \alpha_1 \oplus \alpha_2) = c$.

In order to detect the second-order differential, the adversary must observe at least one of these $p2^n$ quartets. Assuming that the right values are distributed randomly, this implies (by a birthday paradox argument) that in order to observe such a quartet , the adversary must check at least $O(2^{4n}/p2^n) = O(2^{3n}p^{-1})$ plaintext quartets. This amount of quartets is contained in the data set only if the data set contains $\Omega(2^{3n/4}p^{-1/4})$ plaintexts. Hence, any algorithm that detects all probability-p second-order differentials of f has complexity of $\Omega(2^{3n/4}p^{-1/4})$.

Previous algorithms. The most basic top-down algorithm for detecting all second-order differentials of a function $f : \{0,1\}^n \to \{0,1\}^n$ that hold with probability $\geq p$ is to guess the 'input basis vectors' α_1, α_2, encrypt $O(1/p)$ plaintext quartets of the form $(x, x \oplus \alpha_1, x \oplus \alpha_2, x \oplus \alpha_1 \oplus \alpha_2)$ for random choices of x, and check which value c appears several times as $f(x) \oplus f(x \oplus \alpha_1) \oplus f(x \oplus \alpha_2) \oplus f(x \oplus \alpha_1 \oplus \alpha_2) = c$. The complexity of this algorithm is $O(2^{2n}p^{-1})$.

A more efficient algorithm is to guess α_1, define an auxiliary function $f_{\alpha_1} : \{0,1\}^n \to \{0,1\}^n$ by $f_{\alpha_1}(x) = f(x) \oplus f(x \oplus \alpha_1)$, and observe that a second-order differential $(\alpha_1, \alpha_2) \to c$ of f corresponds to a differential $\alpha_2 \to c$ of f_{α_1} that holds with the same probability. All such differentials that hold with probability $\geq p$ can be found in time $O(2^{n/2}p^{-1})$ using the fundamental algorithm presented above, and thus, all second-order differentials of f with probability $\geq p$ can be detected with overall complexity of $O(2^{3n/2}p^{-1})$.

A new algorithm. Like in all previous algorithms, our first step is using surrogate differentiation to cancel the output parameter c. We choose an arbitrary value $\gamma \in \{0,1\}^n$, and consider the function $g_\gamma : \{0,1\}^n \to \{0,1\}^n$ defined by $g_\gamma(x) = f(x) \oplus f(x \oplus \gamma)$. Note that if both x and $x \oplus \gamma$ are 'right values' with respect to the second-order differential, then $g_\gamma(x) = 0$. Thus, surrogate differentiation allows to cancel the parameter c, at the price of reducing the probability of the second-order differential we search for from p to p^2.

Our second step is to find efficiently quartets of plaintexts that stem from the second-order differential $(\alpha_1, \alpha_2) \to 0$ of g_γ with a non-negligible probability. To this end, we consider quartets (x, y, z, w) such that

$$x \oplus y \oplus z \oplus w = g_\gamma(x) \oplus g_\gamma(y) \oplus g_\gamma(z) \oplus g_\gamma(w) = 0. \tag{3}$$

The expected number of such quartets is $O(2^{4n} \cdot 2^{-2n}) = O(2^{2n})$, and among these quartets, about $p^2 2^n$ are of the form $(x, x \oplus \alpha_1, x \oplus \alpha_2, x \oplus \alpha_1 \oplus \alpha_2)$, where x is a right value with respect to the differential $(\alpha_1, \alpha_2) \to 0$ of g_γ. Hence, if we collect $n \cdot 2^n p^{-2}$ random quartets that satisfy (3), then with a high probability they will contain $n/2$ quartets that stem from the differential $(\alpha_1, \alpha_2) \to 0$. This allows finding α_1, α_2, as from each quartet (x, y, z, w) that satisfies (3) we can derive the suggestion $(x \oplus y, x \oplus z)$ for the two 'input basis vectors', and the right value (α_1, α_2) will be the most common suggestion with a high probability. Then, the value c can be found in time $O(1/p)$ by encrypting $O(1/p)$ quartets of the form $(x, x \oplus \alpha_1, x \oplus \alpha_2, x \oplus \alpha_1 \oplus \alpha_2)$, and checking which value c appears several times as $f(x) \oplus f(x \oplus \alpha_1) \oplus f(x \oplus \alpha_2) \oplus f(x \oplus \alpha_1 \oplus \alpha_2) = c$.

Therefore, it remains to explain how to find $O(2^n p^{-2})$ quartets that satisfy (3) efficiently. This amounts to finding efficiently many solutions to the *4-XOR problem* within a list of $2n$-bit vectors $(x_i, g_\gamma(x_i))$ (as Equation (3) is equivalent to stating that the XOR of four such vectors is zero). The *dissection* technique presented in [19] suggests the following algorithm, which allows detecting $O(2^n)$ solutions in time $O(2^n)$:

1. Choose arbitrarily a subset $S \subset \{1, 2, \ldots, 2n\}$ of size n, and denote by $\bar{x_i} \in \{0,1\}^n$ the restriction of $(x_i, g(x_i))$ to the indices in S.
2. Fix a value $A \in \{0,1\}^n$.
3. Go over the 2^n possible values $x_i \in \{0,1\}^n$ and for each of them, insert $\bar{x_i}$ into a hash table.
4. Find all pairs (x_i, x_j) such that $\bar{x_i} \oplus \bar{x_j} = A$. (The expected number of these pairs in $O(2^n)$.)
5. For all remaining pairs, insert $(x_i \oplus x_j, g_\gamma(x_i) \oplus g_\gamma(x_j))$ into a hash table, and check for collisions.
6. Each of the $O(2^n)$ collisions yields a quartet (x_i, x_j, x'_i, x'_j) that satisfies (3).

It is clear that the complexity of the algorithm is $O(2^n)$. Hence, by repeating it with $O(p^{-2})$ random values of γ, we can obtain $O(2^n p^{-2})$ quartets that satisfy (3) with complexity of $O(2^n p^{-2})$.

At this point, some care should be taken. If we make the natural choice $S = \{1, 2, \ldots, n\}$ – or equivalently, if we fix $\bar{x_i} = x_i$, then our algorithm searches only for quartets (x, y, z, w) that satisfy $x \oplus y = A$. Unless $A = \alpha_1$, it will miss all quartets that stem from the second-order differential $(\alpha_1, \alpha_2) \to 0$ for sure! The same will happen for other choices of S, as long as the bits of A in the coordinates of $S \cap \{1, 2, \ldots, n\}$ do not agree with the corresponding bits of α_1. To overcome this problem, we choose $S = \{n + 1, n + 2, \ldots, 2n\}$, i.e., we define $\bar{x_i} = g(x_i)$. As casting a restriction only on the outputs is not expected to affect input differences, the number of 'good' quartets we expect to find is $O(2^{-n} p^2)$ of the total number of quartets yield by the algorithm. In the following, we call this algorithm (with the choice $S = \{n + 1, \ldots, 2n\}$) *Alg_Dissect*, and use it as a subroutine.

Our new method for detecting all high-probability second-order differentials is described in Algorithm 6.

Algorithm 6: New Algorithm for Detecting High-Probability Second-Order Differentials

```
Initialize an empty array of $2n$-bit counters and an empty hash table $H$.
Choose $M = n/p$ random values $\gamma_1, \gamma_2, \ldots, \gamma_M \in \{0,1\}^n$.
for all $1 \le i \le M$ do
    Use the algorithm Alg_Dissect to find about $2^n$ quartets $(x, y, z, w)$ such
      that $x \oplus y \oplus z \oplus w = g_{\gamma_i}(x) \oplus g_{\gamma_i}(y) \oplus g_{\gamma_i}(z) \oplus g_{\gamma_i}(w) = 0$.
    For each suggested quartet, increment the counter that corresponds to
      $(\alpha_1, \alpha_2) = (x \oplus y, x \oplus z)$.
Output each pair $(\alpha_1, \alpha_2)$ whose counter was advanced at least $n/2$ times.
for each remaining pair $(\alpha_1, \alpha_2)$ do
    Choose $n/p$ values $x_j$, and insert into the table $H$ the values
      $f(x_j) \oplus f(x_j \oplus \alpha_1) \oplus f(x_j \oplus \alpha_2) \oplus f(x_j \oplus \alpha_1 \oplus \alpha_2)$.
    Output all values $c$ whose that appear in the table at least $n/2$ times.
```

Analysis. The first part of the algorithm finds (α_1, α_2). As was described above, application of the subroutine *Alg_Dissect* for $O(p^{-2})$ values of γ allows detecting $O(2^n p^{-2})$ quartets that satisfy (3) in $O(2^n p^{-2})$ time. Each of these quartets suggests a value of $(\alpha_1, \alpha_2) \in \{0,1\}^{2n}$. Among these quartets, we expect several quartets that stem from the differential $(\alpha_1, \alpha_2) \to 0$ of the corresponding function g_{γ_i} and all of them suggest the 'right' value (α_1, α_2). Hence, assuming $p \ge 2^{-n/2}$, only a few values of (α_1, α_2) will be suggested several times, and they will include the right value.

The second part of the algorithm examines the remaining pairs (α_1, α_2) and finds the corresponding output values c. Here, the 'wrong' suggestions for (α_1, α_2)

will be discarded with high probability, as for them, no value of c will appear at least $n/2$ times. The complexity of this step is $O(p^{-1})$. Hence, the algorithm finds all probability-p second order differentials of f with data complexity of 2^n and memory and time complexity of $O(2^n p^{-2})$.

B.2 Related-key differentials

Related-key differential cryptanalysis [26] is a widely used cryptanalytic technique, based on tracing the development of differences during the encryption process of two plaintexts, encrypted under unknown keys whose difference is known to (or chosen by) the attacker. The central notion here is a *related-key differential*. Given a keyed cipher $E : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$, we say that the related-key differential $\alpha \to \beta$ under key difference Δ holds for E with probability p, if $\Pr[E_K(x) \oplus E_{K \oplus \Delta}(x \oplus \alpha) = \beta] = p$, where $x \in \{0,1\}^n, K \in \{0,1\}^k$ are chosen uniformly at random.

Our goal in this subsection is to present an algorithm that allows detecting all related-key differentials of $E : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ that hold with probability $\geq p$. An information-theoretic argument similar to the argument presented in Section 2 yields a lower bound of $\Omega(2^{(n+k)/2} p^{-1/2})$ for this task. In the other direction, an algorithm with complexity $O(2^{n+k} p^{-1})$ can be obtained by guessing the values of α, Δ, encrypting $O(1/p)$ plaintext pairs with plaintext difference α and key difference Δ, and finding the ciphertext differences that are encountered at least several times.

We will show that a simple adaptation of the fundamental algorithm allows achieving the goal with time complexity of $O(\max(2^{(n+k)/2} p^{-1}, 2^k p^{-2}))$, which is not far from the lower bound when k is not much larger than n.

New algorithm. The fundamental algorithm presented in Section 2 extends naturally to our setting, by considering the function $f : \{0,1\}^{n+k} \to \{0,1\}^n$ defined by $f(x, K) = E_K(x)$. Specifically, we choose an arbitrary value $(\alpha', K') \in \{0,1\}^n \times \{0,1\}^k$, define $g_{\alpha',K'}(x, K) = E_K(x) \oplus E_{K \oplus K'}(x \oplus \alpha')$, and examine collisions of $g_{\alpha',K'}$. Each such collision $g_{\alpha',K'}(x_i, K_i) = g_{\alpha',K'}(x_j, K_j)$ suggests the related-key differential $x_i \oplus x_j \to E_{K_i}(x_i) \oplus E_{K_j}(x_j)$ under key difference $K_i \oplus K_j$. We insert the suggested differentials into a $(2n+k)$-bit hash table and output the differentials that were suggested several times.

Analysis. By virtue of the analysis of the fundamental algorithm, we see that the function $g_{\alpha',K'}$ is expected to have $2^{2(n+k)-n-1} = 2^{n+2k-1}$ collisions, and that out of them, about $2^{n+k-1} p^2$ stem from the 'right' related-key differential. Thus, in order to observe several collisions that suggest the right differential, we need $O(2^{(n+k)/2} p^{-1})$ plaintexts, which lead to $O(2^k p^{-2})$ collisions of $g_{\alpha',K'}$. As these collisions are checked using a hash table and the differentials suggested by 'random' collisions are discarded with a high probability, the overall complexity of the algorithm is $O(\max(2^{(n+k)/2} p^{-1}, 2^k p^{-2}))$. In particular, for $k = n$, the complexity of the algorithm is $O(2^n p^{-2})$, while the information-theoretic lower bound is $O(2^n p^{-1/2})$.

Other variants of the algorithm. The variants of the fundamental algorithm presented in Section 2 (i.e., memoryless variant, variant with a fixed amount of memory, and worst-case variant), can be generalized to the related-key differential setting in a similar way. We omit the details.