IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2018, No. 3, pp. 265–289. DOI:10.13154/tosc.v2018.i3.265-289

Generating Graphs Packed with Paths

Estimation of Linear Approximations and Differentials

Mathias Hall-Andersen[1] and Philip S. Vejre[2]

[1] University of Copenhagen, Copenhagen, Denmark
mathias@hall-andersen.dk
[2] Technical University of Denmark, Kongens Lyngby, Denmark
psve@dtu.dk

Abstract.
When designing a new symmetric-key primitive, the designer must show resistance to known attacks. Perhaps most prominent amongst these are linear and differential cryptanalysis. However, it is notoriously difficult to accurately demonstrate e.g. a block cipher's resistance to these attacks, and thus most designers resort to deriving bounds on the linear correlations and differential probabilities of their design. On the other side of the spectrum, the cryptanalyst is interested in accurately assessing the strength of a linear or differential attack.
While several tools have been developed to search for optimal linear and differential trails, e.g. MILP and SAT based methods, only few approaches specifically try to find as many trails of a single approximation or differential as possible. This can result in an overestimate of a cipher's resistance to linear and differential attacks, as was for example the case for PRESENT.
In this work, we present a new algorithm for linear and differential trail search. The algorithm represents the problem of estimating approximations and differentials as the problem of finding many long paths through a multistage graph. We demonstrate that this approach allows us to find a very large number of good trails for each approximation or differential. Moreover, we show how the algorithm can be used to efficiently estimate the key dependent correlation distribution of a linear approximation, facilitating advanced linear attacks. We apply the algorithm to 17 different ciphers, and present new and improved results on several of these.

Keywords: Linear cryptanalysis · Differential cryptanalysis · Linear Approximations · Differentials · Trail search · Correlation distributions · Graph theory

1 Introduction

Whenever a new design for a symmetric-key primitive is proposed, it is usually accompanied by a design rationale. This rationale explains how the specific choice of components ensure resistance to a set of common attack techniques. However, thoroughly checking maybe a dozen different attacks is laborious work for the designer, and it is therefore common to somehow make an estimate of how well a design resists a specific attack.

Two attack techniques that are almost always featured in the security analysis of a new design, due to their long history and many strong results, are linear [Mat93] and differential [BS90] cryptanalysis. However, it is notoriously difficult to make an accurate and complete analysis of a cipher's security against these attacks, and for this reason methods of estimating the strength of these attacks feature prominently in the initial analysis of a new design. For block ciphers, this will often consist of lower-bounding the number of active S-boxes in a linear or differential trail, thus showing how many rounds the cipher needs to resist these attacks.

Received: 2018-06-01, Accepted: 2018-08-01, Published: 2018-09-04

Nevertheless, several examples exist of this approach not giving the full picture, in particular due to the existence of linear approximations or differentials that contain a very large number of good trails. This effect was already recognised for differential cryptanalysis in [LMM91] and subsequently extended to linear cryptanalysis in [Nyb94] where it was dubbed the linear hull effect. As an example of this phenomenon, it was demonstrated in [Ohk09] that for the block cipher PRESENT the difference between a single linear trail and the full linear approximation is quite significant. Thus, it would be extremely helpful for a designer if a simple tool existed that could more accurately find linear approximations and differentials for a given design. This would not only save the designer time, but potentially also allow for exploration of a larger design space as well as enabling a more informed choice of the number of rounds needed to obtain adequate security.

1.1 Previous Work

Several approaches for finding linear and differential trails have been suggested in the literature. Perhaps the most well known technique is Matsui's original branch-and-bound algorithm [Mat94], which can essentially be viewed as a depth-first search with pruning. While this algorithm does guarantee to return the optimal trail for any starting value, one still needs to have some idea what a good starting value might be. Moreover, while the algorithm can be adapted to return multiple trails, this is not very efficient if the number of trails is extremely large.

Several other approaches for finding linear and differential trails have been proposed. Amongst these are MILP based algorithms [MWGP11, SHW+14, FWG+16, YML+17] and SAT based algorithms [KLT15, MP13, AK18], as well as more dedicated approaches [DEM15, FLN07, Ste13]. Both the MILP and SAT based approaches can be extended in order to find multiple trails by removing already known trails from the solution space, but this approach also has the problem of scaling linearly with the number of trails. Additionally, in order to use these algorithms, every design one wishes to analyse has to be formulated as a MILP/SAT model.

A few approaches for finding large numbers of linear or differential trails have been suggested. Matsui's algorithm was generalised in [CMST15a, CMST15b] to search for multiple differential trails of generalised Feistel networks. A more versatile approach was presented in [Abd12], where the idea of using partial, sparse correlation/differential transitions matrices to find multiple trails was proposed. While this approach does scale well with the number of trails found, it potentially has high memory requirements. This problem was acknowledged in [AAA+15] where the matrix method was combined with the MILP method to improve results for ARX designs. Still, these works do not offer a general, design agnostic strategy for choosing the partial matrices.

While the mentioned works focus on estimating expected differential probabilities or expected squared correlations, we note that for linear cryptanalysis especially, there has recently been an increased focus on the key dependent distribution of correlations. Namely, several works developing models for the key dependent behaviour of correlations have been published [BT13, HVLN15, BN17, BN16] as well as some advanced attack techniques exploiting these correlations distributions [BV17, BTV18]. Thus, it is of additional interest to develop algorithms that also allow for efficient estimation of these distributions.

1.2 Contributions

In this work, we propose a new algorithm for linear and differential trail search. The overall concept of the algorithm is to represent all linear/differential trails as paths in a multistage graph, and then find a manageable subgraph which hopefully contains good trails. By performing a breadth first traversal of this subgraph, we can very efficiently consider a larger number of trails when estimating the squared correlation/differential

probability, and even do so for many linear approximations/differentials simultaneously. Moreover, for linear cryptanalysis, the algorithm allows us to very efficiently approximate the correlation distributions over the key space.

While the overall concept of this approach is related to the idea of partial correlation and difference transition matrices, the graph representation allows a designer or cryptanalyst to gain additional insight, e.g. one can extract the actual trails from the graph or visualise the trail structure in order to gain deeper understanding of a cipher's linear and differential behaviour (see e.g. Figure 5). Moreover, we can more easily obtain the key-dependent linear correlations without having the recompute everything for each new key. In more detail, we achieve the following:

- **Efficient graph generation** We present a heuristic approach for selecting a subgraph of the linear/differential trail graph, i.e. we identify good approximations/differentials over a single round. For SPN ciphers, we give a highly efficient algorithm for generating these. Moreover, we show how to remove redundant information from the graph in order to reduce memory costs. As opposed to the strategy for choosing partial correlation/difference matrices in [Abd12], our heuristic is design agnostic.

- **Algorithm optimisations** We present a number of optimisations to the basic algorithm which both reduces the time it takes to generate the trail graph and the amount of memory consumed while generating the graph. The latter is done by removing single round approximations/differentials which are not part of any trail before it is ever added to the graph. While the most effective improvements only apply to SPN ciphers, they allow us to increase the effective size of our search space; as an example, for Midori64 [BBI+15] we were able to include as many as $2^{46.5}$ single round approximations in our search space.

- **Improved estimates of ELP and EDP** Compared to algorithms that find one trail at a time (e.g. MILP and SAT based methods), our graph representation allows us to consider a much larger number of trails when estimating the expected squared correlation or the expected differential probability. As an example, we are able to consider $2^{112.4}$ linear trails for a single approximation of PUFFIN [CHW08]. This ensures a more accurate estimate of these statistics.

- **Extensive application** We use the new algorithm to find linear approximations and differentials for 17 different SPN ciphers. The selection of ciphers have block sizes ranging from 48 to 128 bits, use 4- and 8-bit S-boxes, and apply a variety of different design approaches for choosing the linear layer, e.g. from very lightweight bit permutations to heavy MDS matrices. We present new results on several ciphers, and improve existing results for five ciphers.

- **Correlation distributions** We demonstrate that for SPN ciphers, the graph representation can also be used to efficiently obtain estimates for the key dependent correlation distribution of a linear approximation. In particular, it takes at most a couple of minutes to generate key dependent correlation values for 10 000 randomly chosen keys. We use this fact to investigate the correlation distributions of several ciphers, and show for example that `GIFT-64` [BPP+17] exhibits multiple approximations with asymmetries similar to those observed for DES in [BV17]. In general, this feature of our algorithm facilitates easier application of more advanced linear attacks.

- **Software implementation** Finally, we make our implementation of the algorithm freely available at https://gitlab.com/psve/cryptagraph. This implementation is written in Rust, and is highly optimised and parallelised. At the time of writing,

it supports analysis of SPN ciphers whose substitution layer consists of applying S-boxes to the state, as well as Feistel ciphers with SPN-like F-functions. Additionally, adding new ciphers to the implementation only requires the usual implementation of the S-box and the linear layer, as opposed to MILP and SAT based tools that require modelling of the cipher in the relevant framework. We hope that the availability of this tool, as well as its ease of use, will facilitate more informed design processes and improved cryptanalysis.

The rest of this work is structured as follows: In Section 2 we introduce the basic definitions for linear and differential cryptanalysis. Section 3 introduces the idea of the graph framework, while Section 4 outlines the basic algorithm for trail search. Section 5 contains several improvements to the basic algorithm. In Section 6 and Section 7 we present the results we obtain by using the algorithm on various ciphers. Finally, Section 8 discusses some prospects for future work.

2 Preliminaries

Throughout the paper we consider *block ciphers*, i.e. functions of the type

$$\mathcal{E}(k, m) : \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \to \mathbb{F}_2^n,$$

where $\mathcal{E}$ is a permutation on the plaintext space $\mathbb{F}_2^n$ for each key $k \in \mathbb{F}_2^\kappa$. In particular, we consider *iterative block ciphers* where $\mathcal{E}$ is defined as a composition of several (potentially different) round-functions, i.e.

$$\mathcal{E} = f_r \circ \ldots \circ f_1.$$

We define a *distinguisher* as an algorithm which attempts to distinguish between the function $\mathcal{E}$ and a permutation picked uniformly at random from the space of all permutations on $\mathbb{F}_2^n$. In particular, the cryptanalyst is interested in a distinguisher which succeeds with high probability and uses time less than 2^κ.

In the following, we briefly describe the main ideas of linear and differential distinguishers as well as the problem of finding good properties of these types. While we only describe the techniques from a distinguisher viewpoint, distinguishers of both types can be turned into key-recovery attacks in most cases.

2.1 Linear Cryptanalysis

We define a *linear approximation* of a block cipher as the pair of masks $(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$. Let $\langle \cdot, \cdot \rangle$ denote the canonical inner product on $\mathbb{F}_2^n$. We say that the approximation (α, β) has a *linear correlation* defined by

$$C^k_{(\alpha,\beta)} = 2 \cdot \Pr_{m \in \mathbb{F}_2^n}(\langle \alpha, m \rangle = \langle \beta, \mathcal{E}(k, m) \rangle) - 1.$$

Note that the correlation is key dependent, and thus has some distribution over $\mathbb{F}_2^\kappa$. For a randomly chosen permutation, the correlation is drawn from the distribution $\mathcal{N}(0, 2^{-n})$ [DR07]. Thus, if there exists a linear approximation (α, β) of a block cipher such that $C^k_{(\alpha,\beta)}$ is distributed significantly differently from $\mathcal{N}(0, 2^{-n})$, we can use this approximation to build a distinguisher.

2.2 Differential Cryptanalysis

We define a *differential* of a block cipher as the pair of differences $(\Delta, \nabla) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$. Let $\oplus$ denote the componentwise addition of vectors in $\mathbb{F}_2^n$. Then we say that the differential

(Δ, ∇) has a *differential probability* defined by

$$p^k_{(\Delta,\nabla)} = \Pr_{m \in \mathbb{F}_2^n}(\mathcal{E}(k,m) \oplus \mathcal{E}(k, m \oplus \Delta) = \nabla).$$

For a randomly chosen permutation, we expect the differential probability to be close to 2^{-n}. Thus, if there exists a differential (Δ, ∇) of a block cipher such that $p^k_{(\Delta,\nabla)}$ is significantly bigger than 2^{-n} for almost all keys, we can us this differential to build a distinguisher.

2.3 Finding Approximations and Differentials

Determining either $C^k_{(\alpha,\beta)}$ or $p^k_{(\Delta,\nabla)}$ is not feasible for the values of n and κ used in practice. Therefore, for iterative block ciphers, the problem is usually reduced to that of finding linear and differential *trails*. A linear trail of an approximation (α, β) is defined as a sequence of masks $U = (u_0, \ldots, u_r)$, with $(u_0, u_r) = (\alpha, \beta)$. Then, we define the *correlation contribution* of this trail as

$$C^k_U = \prod_{i=0}^{r} C_{(u_i,u_{i+1})}(i),$$

where $C_{(u_i,u_{i+1})}(i)$ is the correlation of the approximation (u_i, u_{i+1}) for the i'th round function f_i. Since the f_i usually have a simple form, it is easier to determine the correlation of these functions. The set of all trails of an approximation is called the *linear hull* of the approximation. It can be shown that the correlation of (α, β) is the sum of the correlation contributions of all trails in the linear hull [DR02], i.e.

$$C^k_{(\alpha,\beta)} = \sum_{(u_0,u_r)=(\alpha,\beta)} C^k_U.$$

The situation is analogous for differentials. Although the number of trails is extremely large, it often suffices to find a set of trails with high correlation or probability contribution, such that computing the partial sum over these trails is a good approximation of the actual correlation or probability. Thus, finding a good set of trails is essential to both linear and differential cryptanalysis, and it is this problem that we will consider in the following.

A note on ELP and EDP As explained above, the linear correlation and differential probability of a cipher depends on the specific key used. However, for the initial analysis of these attacks, it is often more convenient to consider the *expected linear potential* and the *expected differential probability*.

In the case of differentials, EDP is defined as $\mathrm{E}(p^k_{(\Delta,\nabla)})$ and it is often assumed that $p^k_{(\Delta,\nabla)} \approx \mathrm{E}(p^k_{(\Delta,\nabla)})$ for most keys. For approximations, ELP is defined as $\mathrm{E}((C^k_{(\alpha,\beta)})^2)$, and it can be shown that $\mathrm{E}((C^k_{(\alpha,\beta)})^2) \approx \sum (C^k_U)^2$, and that for key-alternating ciphers (or Feistel ciphers with SPN-like structures) $(C^k_U)^2$ is independent of the key [DR02].

Thus, considering ELP and EDP eliminates the key and therefore greatly simplifies the search, and usually gives a good indicator for the strength of an approximation or differential. We will therefore initially take this approach, and then show in Section 7 how to find the key-dependent linear correlation distributions.

3 Trail Search Viewed as a Graph Problem

Although finding trails of a specific approximation or differential is already a difficult problem, for a newly designed block cipher it might not even be clear what approximations

or differentials we should be considering. In the following, we will view the problem of finding good approximations and differentials more abstractly as a graph problem. This perspective will help us develop a trail search algorithm which does not require any initial understanding of the linear or differential behaviour of the cipher being analysed. We will describe the graph problem and the algorithm in terms of linear cryptanalysis, but all observations are directly applicable to the case of differential cryptanalysis.

We first introduce some graph notation. A *directed graph* G is a set of *vertices* V and a set of *directed edges* E. We associate a value to each vertex. Throughout the paper, we will not differentiate between a vertex and its value, and use the two concepts interchangeably. We denote a directed edge from a vertex $u \in V$ to a vertex $v \in V$ by $u \to v$. For a weighted graph, each edge $u \to v$ has a *length*, denoted by $l(u \to v)$. We furthermore denote a *path* from a vertex u to a vertex v by $u \rightsquigarrow v$. If $v = v_1, \ldots, u = v_k$ are the vertices traversed by this path, then we define the length of the path as:

$$l(u \rightsquigarrow v) = \prod_{i=1}^{k-1} l(v_i \to v_{i+1}).$$

Furthermore, we call the set of all paths $u \rightsquigarrow v$ the *hull of* (u, v). We denote the hull by $u \diamond v$ and associate to it a *weight* defined as:

$$w(u \diamond v) = \sum l(u \rightsquigarrow v),$$

i.e. the sum of the length of all the paths contained in the hull. We will exclusively consider a special type of directed graph, called a *multistage graph*.

Definition 1 (Multistage Graph)**.** Let G be a directed graph with vertices V and edges E. If the vertices in V are partitioned into ℓ subsets $S_0, \ldots, S_{\ell-1}$, called *stages*, such that any edge in E has the form $u \to v$ with $u \in S_i$ and $v \in S_{i+1}$, for some $i \in [0, \ell - 1[$, we call the graph a multistage graph.

By definition, a multistage graph is a directed acyclic graph (DAG). We now define a weighted multistage graph $G_{\mathcal{E}}$ which represents the linear hulls of all approximations of an iterative block cipher $\mathcal{E}$. Assume that $\mathcal{E}$ has r rounds. Then $G_{\mathcal{E}}$ has $r + 1$ stages each with 2^n vertices representing the elements of $\mathbb{F}_2^n$. $G_{\mathcal{E}}$ contains all edges $u \to v$ for $u \in S_i$ and $v \in S_{i+1}$, $i \in [0, r[$. The length of an edge is defined as

$$l(u \to v) = (C_{(u,v)}(i))^2 \quad \text{if } u \in S_i.$$

Note that if $\alpha \in S_0$ and $\beta \in S_r$, then a path $(\alpha \rightsquigarrow \beta)$ is equivalent to a linear trail $U = (\alpha, \ldots, \beta)$ and its length is exactly $(C_U^k)^2$. Moreover, $\alpha \diamond \beta$ corresponds exactly to the linear hull of the approximation (α, β) and its weight is equal to the ELP of (α, β). Thus, the graph $G_{\mathcal{E}}$ represents the linear hulls of all approximations of $\mathcal{E}$. Finding good approximations therefore corresponds to finding pairs of vertices $(\alpha, \beta) \in S_0 \times S_r$ such that $w(\alpha \diamond \beta)$ is as large as possible. In the following section, we describe an algorithm that aims to solve this problem.

4 A New Algorithm for Trail Search

The graph $G_{\mathcal{E}}$ defined above is exceedingly huge; it has $(r + 1) \cdot 2^n$ vertices and $r \cdot 2^{2n}$ edges. Thus, it is completely infeasible to run even a linear time algorithm on the graph[1]. We therefore have to somehow reduce the size of the graph, i.e. we have to reduce the size of the *search space*. Straight away, we can remove any edges $u \to v$ with $l(u \to v) = 0$ as

[1]Note that the longest path problem can be solved in linear time for DAGs.

any path which includes this edge will have length zero and therefore not contribute to the hull. Nevertheless, for most ciphers the set of non-zero edges in $G_{\mathcal{E}}$ is still intractable. Thus, we propose the following approach:

1. Determine an interesting subgraph $\bar{G}_{\mathcal{E}}$ of $G_{\mathcal{E}}$.
2. Calculate $w(\alpha \diamond \beta)$ for all $(\alpha, \beta) \in S_0 \times S_r$ in $\bar{G}_{\mathcal{E}}$.

For this approach to give a good result, would like many of the longest paths of $\alpha \diamond \beta$ to appear in $\bar{G}_{\mathcal{E}}$. How to ensure this is clearly highly dependent on the cipher $\mathcal{E}$ in question. Moreover, at first glance it seems that if we can specify $\bar{G}_{\mathcal{E}}$, then we already know a good collection of trails. However, we note that finding good approximations in some sense corresponds to finding a minimal subgraph. In contrast, in the process of finding the subgraph $\bar{G}_{\mathcal{E}}$ we can start with a larger subgraph that might contain a lot of unnecessary vertices and edges. While this graph might be too large for us to perform Step 2 above, we can then remove any superfluous information and hopefully arrive at a suitable subgraph $\bar{G}_{\mathcal{E}}$.

In Section 4.1, we propose a simple, generic approach to Step 1. Section 4.2 then details how to efficiently perform Step 2 on the resulting subgraph. In Section 5 we propose various improvements to the naive algorithm.

4.1 Choosing a Subgraph

We propose the following general, design agnostic heuristic for generating $\bar{G}_{\mathcal{E}}$.

- *Selection:* Select the k longest edges going out from each stage in $G_{\mathcal{E}}$.
- *Pruning:* Remove any irrelevant edges and vertices from the resulting graph.

It is clear that this way of selecting edges does not guarantee that we find optimal paths. Indeed, it could be the case that the longest paths contain a single very short edge. However, as long as we are able to use fairly large values of k, we should be able to cover a good fraction of the search space. Additionally, if we *do* find paths using this strategy, we can at least be confident that they are quite close to optimal. A similar heuristic was used in [Abd12, AAA+15] for constructing partial correlation matrices – here, single round approximations with low hamming weight were selected. How well this heuristic works is however heavily dependent on the cipher's structure. Indeed, choosing the longest edges seem like an approach that will work well in a more general setting.

We now show how the selection step can be performed efficiently for ciphers with SPN-like round-functions and then detail how the pruning step works.

4.1.1 Edge Selection for SPN Ciphers

For the sake of simplicity, we will initially consider *substitution-permutations networks* (SPN ciphers) with identical round-functions (aside from the key addition), i.e. $\forall i, f_i = f \oplus k_i$, although the approach also applies to the more general case of SPN ciphers with different round-functions. Our goal is then to find a set $\mathcal{A}$ of the k approximations (each representing an edge) with largest squared correlation. Following Section 2.3, we can ignore the key addition in the following, and hence the SPN round-function takes on the form:

$$f = \mathcal{L} \circ \mathcal{S},$$

where $\mathcal{L}$ is a linear transformation of the state and $\mathcal{S}$ is the parallel application of t independent S-boxes to the state. Let $\mathcal{S}_i$ be the i'th S-box, i.e.

$$\mathcal{S} = \mathcal{S}_0 \| \cdots \| \mathcal{S}_{t-1}.$$

Then, in the usual way, the correlation of an approximation (α, β) of f is entirely determined by the approximation $(\alpha, \mathcal{L}^{-1}(\beta))$ of $\mathcal{S}$. This is in turn entirely determined by the component approximations of the individual S-boxes so that

$$(C_{(\alpha,\beta)}(f))^2 = \prod_{i=0}^{t-1} (C_{(\alpha_i, \mathcal{L}^{-1}(\beta)_i)}(\mathcal{S}_i))^2.$$

We can now reduce the problem of finding the k best approximations over f to the problem of finding certain classes of approximations over $\mathcal{S}$. To this end, we introduce the notion of an *S-box pattern*.

Definition 2 (S-box pattern)**.** Let $\mathcal{S} = \mathcal{S}_0 \| \cdots \| \mathcal{S}_{t-1}$ be the parallel application of t independent S-boxes to a cipher state. Then a pattern of $\mathcal{S}$ is a tuple $p \in \mathbb{R}^t$. The pattern represents a set of approximations of $\mathcal{S}$ such that the squared correlation of $\mathcal{S}_i$ is equal to p_i. We associate a value to a pattern p, namely $C(p) = \prod p_i$, i.e. the squared correlation of any approximation represented by p.

We say that a pattern *expands* into a number of approximations, and denote this set of approximations by $\mathrm{Ex}(p)$. As an example, consider an $\mathcal{S}$ function consisting of five copies of a 4-bit S-box which has two approximations with squared correlation 2^{-2}, namely $(\texttt{0x3}, \texttt{0xd})$ and $(\texttt{0x7}, \texttt{0x4})$. Then the pattern

$$p = (1, 2^{-2}, 1, 1, 2^{-2})$$

would have value $C(p) = 2^{-4}$ and expands into the set of four approximations

$$\begin{aligned}\mathrm{Ex}(p) = \{&(\texttt{0x03003}, \texttt{0x0d00d}), (\texttt{0x03007}, \texttt{0x0d004}),\\ &(\texttt{0x07003}, \texttt{0x0400d}), (\texttt{0x07007}, \texttt{0x04004})\}.\end{aligned}$$

We note that this expansion can be done in amortized linear time in the size of $\mathrm{Ex}(p)$, independent of t. Moreover, if we just desire to know the input or output masks of the approximations in $\mathrm{Ex}(p)$, these can also be generated in amortized linear time in the number of inputs/outputs.

Now, if we can determine the set $\mathcal{P}$ of patterns with the k' largest values, then clearly $\mathrm{Ex}(\mathcal{P}) = \mathcal{A}$ contains approximations over f with the $|\mathcal{A}|$ largest correlations. This problem can be efficiently solved using the approach of finding critical paths presented in [YDG89]. We briefly outline the idea of the algorithm here:

1. Compute lists L_i of unique values in the LAT of each $\mathcal{S}_i$ and sort them in descending order.
2. Maintain a max-heap of partially determined patterns sorted by their current value. Add a fully undetermined pattern $p = (?, ?, \ldots, ?)$ to the heap.
3. Create an empty set $\mathcal{P}$. Repeat the following until $\mathcal{P}$ has the desired size:
   (a) Pop the top pattern p off the heap. If it was fully determined, add it to $\mathcal{P}$.
   (b) Find the last determined position of p, say p_i, and generate two new patterns:
      i. Replace p_i with the next value in the list L_i and insert the pattern in the heap.
      ii. Replace the undetermined value p_{i+1} with the first value on the list L_{i+1} and insert the pattern in the heap.

Note that this pattern representation, aside from making it easy to find approximations sorted by their correlation, is a very useful time-memory trade-off: each pattern can represent a large number of approximations, allowing us to select a large number of candidate edges for the graph $\bar{G}_{\mathcal{E}}$ without storing them explicitly. However, we need to spend time expanding each pattern whenever we explicitly need the approximations.

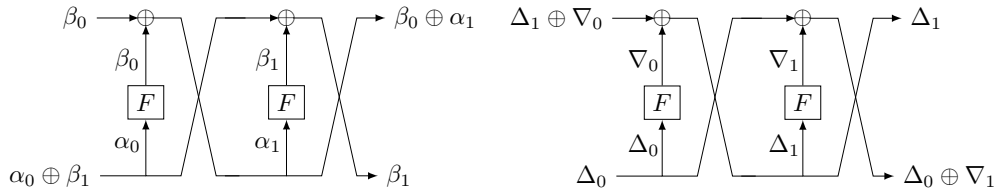


Figure 1: An illustration of how linear approximations/differentials over the F-functions of two consecutive Feistel rounds determine a linear approximation/differential over those rounds.

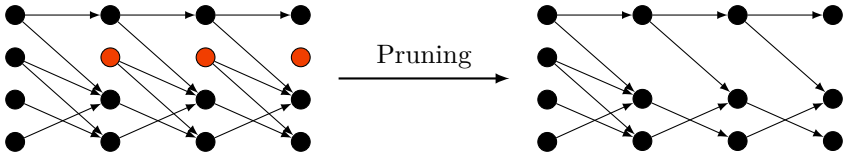


Figure 2: Left: A graph representing parts of linear/differential trails over three rounds of a cipher. Right: The graph after all edges and vertices which are not part of a full trail have been removed.

About Feistel constructions and other designs The process described here for selecting edges is very efficient for SPN designs. However, it is less clear how to perform the edge selection for other types of designs. For Feistel designs with SPN-like F-functions, we can use the same approach with a slight modification: We let an S-box pattern describe approximations over the F-functions of two consecutive rounds and then derive approximations over two rounds from this pattern. The resulting two-round approximation is shown in Figure 1. This concept can be extended to generalised Feistel constructions.

Concerning radically different design approaches, i.e. ARX and AND-RX designs, we note that [BV14] and [KLT15] present formulas for the differential probabilities of SPECK and SIMON-like round-functions, respectively. The latter work also gives a method for determining linear correlations of SIMON-like round-functions. These results could potentially be used to perform efficient edge selection for these types of designs.

4.1.2 Graph Pruning

By using the pattern representation introduced above, we can store a large set of interesting edges in a space efficient way. However, not all edges in $\mathcal{A}$ might be relevant when added to the graph $\bar{G}_{\mathcal{E}}$. Consider Figure 2. On the left we show a graph which was generated from a set of patterns, i.e. each edge represents an approximation over the round-function f. The vertices marked in red cannot be a part of a path from a vertex in the first stage to a vertex in the last stage. Hence, we can remove these vertices and all their edges, leaving us with the second, smaller graph, which only contains the information we are interested in. In other words:

- Remove any vertex in S_0 with no outgoing edges.
- Remove any vertex in S_1 to S_{r-1}, if it does not have at least one incoming and one outgoing edge, remove it.
- Remove any vertex in S_r with no incoming edges.

We potentially have to repeat this process until no more vertices can be removed. There is one major problem with naively generating the graph $\bar{G}_{\mathcal{E}}$ in this way, namely that we

have to store the initial graph before pruning (which takes roughly $r \cdot |\mathcal{A}|$ space), which can be many times larger than the pruned graph. This essentially limits the number of single round approximations we can consider, i.e. it limits the size of the search space we can cover. In Section 5 we present a number of improvements that alleviate this problem.

4.2 Finding Linear Hulls and Differentials

Once the graph $\bar{G}_{\mathcal{E}}$ has been generated, we can quite easily calculate $w(\alpha \diamond \beta)$ for all pairs $(\alpha, \beta) \in S_0 \times S_r$ by essentially performing a breadth first traversal of the graph for each α while doing some bookkeeping. The idea is the following:

1. Let $\mathcal{H}$ be an empty hash table. Choose an $\alpha \in S_0$ and let $\mathcal{H}(\alpha) = 1$.
2. For each stage S_0 to S_{r-1} of $\bar{G}_{\mathcal{E}}$, do the following:
   (a) Create an empty hash table $\mathcal{H}'$.
   (b) For each key of $\mathcal{H}$, let u be the corresponding vertex in $\bar{G}_{\mathcal{E}}$. Let $c = \mathcal{H}(u)$. Then, for each edge $u \to v$, if $\mathcal{H}'(v)$ does not exists, let $\mathcal{H}'(v) = c \cdot l(u \to v)$. Otherwise, let $\mathcal{H}'(v) = \mathcal{H}'(v) + c \cdot l(u \to v)$.
   (c) Let $\mathcal{H} = \mathcal{H}'$.
3. $\mathcal{H}(\beta)$ now contains $w(\alpha \diamond \beta)$.
4. Repeat for a new value of α.

Note that the number of paths in any $\alpha \diamond \beta$ can also be calculated with a bit of extra bookkeeping in step 2.b. The time complexity of this algorithm is $\mathcal{O}(|S_0| \cdot |\bar{G}_{\mathcal{E}}|)$ and the memory complexity is $\mathcal{O}(|S_0| \cdot |S_r|)$. The memory complexity can be reduced to a constant by only storing the hulls with highest weight calculated so far in Step 3. The time complexity can be reduced to $\mathcal{O}(|\bar{G}_{\mathcal{E}}|)$ by considering all $\alpha \in S_1$ simultaneously. However, this will increase the memory complexity, and in practice we find that this slows down the search due to a poorer cache locality. Moreover, the procedure outlined above is trivially parallelisable over different α values.

It is interesting to note that when the paths contained in $\alpha \diamond \beta$ are not completely edge disjoint, the number of paths can be many orders of magnitude larger than the size of $\bar{G}_{\mathcal{E}}$. Thus, this way of computing $w(\alpha \diamond \beta)$ can be much more efficient than explicitly finding each possible path of $\alpha \diamond \beta$ and adding their lengths. This graph representation of linear hulls therefore allows us to compactly express a large number of trails for a linear approximation, and potentially enables us to capture a much larger part of the linear hull than if we had used a more direct trail search.

5 Improvements

The graph generation algorithm presented in Section 4.1 has two main limitations: First, the time we spend generating the graph is proportional to the number of single round approximations we consider, and second, we initially have to store a much larger graph than we ultimately need. In the following, we present some improvements to the algorithm which prevent this, as well as some additional useful techniques.

5.1 Vertex Generation

We first note that we can perform the pruning step of Section 4.1.2 without initially storing all $r \cdot |\mathcal{A}|$ edges. Let us denote by $\mathrm{Ex}_{\mathrm{in}}(\mathcal{P})$ and $\mathrm{Ex}_{\mathrm{out}}(\mathcal{P})$ the expansion of $\mathcal{P}$ into only input masks and output masks, respectively. As noted in Section 4.1.1, for SPN ciphers

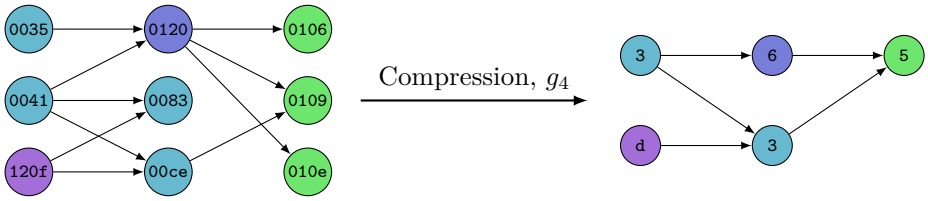


Figure 3: An example of graph compression using the compression function g_4. The values of the vertices are shown in hexadecimal notation. Vertices in the same stage with non-zero nibbles in the same position are identified, and any multiple edges are removed.

we can generate $\mathrm{Ex}_{\mathrm{in}}(\mathcal{P})$ or $\mathrm{Ex}_{\mathrm{out}}(\mathcal{P})$ in linear time in the number of inputs or outputs. Moreover, observe that if a vertex in any of the stages S_1 to S_{r-1} does not correspond to a mask contained in $\mathrm{Ex}_{\mathrm{in}}(\mathcal{P}) \cap \mathrm{Ex}_{\mathrm{out}}(\mathcal{P})$, then it will be removed during the pruning process. Thus, we can initially set

$$S_i = \begin{cases} \mathrm{Ex}_{\mathrm{in}}(\mathcal{P}) & i = 0, \\ \mathrm{Ex}_{\mathrm{in}}(\mathcal{P}) \cap \mathrm{Ex}_{\mathrm{out}}(\mathcal{P}) & 1 \leq i \leq r-1, \\ \mathrm{Ex}_{\mathrm{out}}(\mathcal{P}) & i = r. \end{cases}$$

Then, when adding edges, we generate $\mathcal{A}$ and only add an edge if the corresponding vertices already exists in the graph. Since usually $\mathrm{Ex}_{\mathrm{in}}(\mathcal{P}) \cap \mathrm{Ex}_{\mathrm{out}}(\mathcal{P}) \ll \mathrm{Ex}_{\mathrm{in}}(\mathcal{P}) \cup \mathrm{Ex}_{\mathrm{out}}(\mathcal{P})$, the memory usage is greatly reduced. In practice, the time taken to generate the graph is also reduced, even though we still have to generate the entire set $\mathcal{A}$, as inserting edges and vertices is much more expensive than checking set membership. Finally, we note that we still have to prune the resulting graph to obtain the desired $\bar{G}_{\mathcal{E}}$.

5.2 Graph Compression and Pattern Elimination

While vertex generation somewhat improves memory and running time, it still might be the case that some patterns in $\mathcal{P}$ ultimately did not contribute to $\bar{G}_{\mathcal{E}}$, i.e. all edges expressed by the pattern are removed during pruning. We will call such a pattern a *dead pattern*. Clearly, it would be preferable if we ignored dead patterns completely. We now present a technique for finding dead patterns quickly while using little memory.

We first introduce the notion of a compression function $g_j(x) : \mathbb{F}_2^n \to \mathbb{F}_2^{n/j}$. Let $y = g_j(x)$. Then for $0 \leq i < j$, $y_i = 1$ iff $(x_{j \cdot i}, \ldots, x_{j \cdot (i+1)-1})$ is non-zero. For example,

$$g_4(\texttt{0xf1a0000500705200}) = \texttt{0b1110000100101100} = \texttt{0xe12c},$$

i.e. the value $\texttt{0xf1a0000500705200} \in \mathbb{F}_2^{64}$ is compressed to the value $\texttt{0xe12c} \in \mathbb{F}_2^{16}$. Note that this is similar to the concept of truncated differentials/approximations. With some abuse of notation, for a graph G will say that $g_j(G)$ is the graph obtained by applying g_j to all vertices, identifying vertices in the same stage that have the same compressed value, and then removing multiple edges. An example of this process is shown in Figure 3.

We can use compression to find dead patterns in a space efficient way. Instead of generating $\bar{G}_{\mathcal{E}}$, we first generate an approximation to $g_j(\bar{G}_{\mathcal{E}})$, say $\hat{g}_j(\bar{G}_{\mathcal{E}})$, by applying g_j to all values when generating the graph. Note that this does not yield $g_j(\bar{G}_{\mathcal{E}})$; any path between two vertices in $\bar{G}_{\mathcal{E}}$ is preserved in $\hat{g}_j(\bar{G}_{\mathcal{E}})$, but there might be some additional false paths. As a result, when applying pruning to the compressed graph, some vertices might not be removed, although they would have been removed in the actual graph.

Note now that if a pattern is dead when considering $\hat{g}_j(\bar{G}_{\mathcal{E}})$, it is also dead when considering $\bar{G}_{\mathcal{E}}$ (although the converse does not hold). Thus, we can use $\hat{g}_j(\bar{G}_{\mathcal{E}})$ to remove some dead patterns. We propose the following approach to removing dead patterns while conserving memory:

1. Generate a set of patterns $\mathcal{P}$.
2. Pick a $j > 1$ such that j is a power of two, and do the following:
   (a) Generate the graph $\hat{g}_j(\bar{G}_\mathcal{E})$ as described above.
   (b) Remove any patterns from $\mathcal{P}$ which are dead according to $\hat{g}_j(\bar{G}_\mathcal{E})$.
   (c) If $j = 2$ then stop. Otherwise set $j = j/2$ and repeat.

The main insight here is that initially $\mathrm{Ex}(\mathcal{P})$ can be many times larger than what we can store in memory. By gradually using a finer compression, we decrease the size of $\mathrm{Ex}(\mathcal{P})$, while still keeping the intermediate graphs manageable and without losing any information from the original search space. In practice, for ciphers with a block size of 64 bits and 4-bit S-boxes, we find that starting with $j = 4$ works well. How many dead patterns occur varies between different designs, but we find that in general, if there are few dead patterns, we also rarely need to use a large set $\mathcal{A}$ to get good results.

Speedup for SPN Ciphers While the above processes has the potential to greatly reduce memory usage, we still need to calculate the initial set $\mathcal{A}$ at least once, and potentially multiple times if few patterns are eliminated. For SPN ciphers we can improve this by observing that if j is a multiple of the width of the S-box, we can calculate the compression of an approximation (α, β) simply by calculating $\mathcal{L}^{-1}(\beta)$. This is true, since if the output mask of any S-box is non-zero, then so is the corresponding input mask of that S-box, which is all we need to know to calculate the compressed value of α. In this case we can therefore generate $\hat{g}_j(\bar{G}_\mathcal{E})$ in time $\mathcal{O}(|\mathrm{Ex}_{\mathrm{in}}(\mathcal{P})| + |\mathrm{Ex}_{\mathrm{out}}(\mathcal{P})|)$ (recall that vertex generation has this time complexity for SPN ciphers). This greatly improves the running time of the algorithm for this type of ciphers.

5.3 Vertex Anchoring

One big limitation with the algorithm presented here is that the search space is limited by how big a set $\mathcal{A}$ the available computing power allows us to consider. While the improvements presented so far increase the possible size of $\mathcal{A}$, we wont be able to find paths that locally have very short edges. Note that such a path might still be comparatively long, if all other edges of the path are long.

Without having cipher specific insight, it seems difficult to know when it is beneficial to add a locally bad edge, and especially which edge to add. This problem is exacerbated by the fact that short edges represent approximations which usually activate many S-boxes, and so the number of short edges is usually much larger than the number of long edges.

We propose a partial solution to this problem by introducing a technique called *vertex anchoring*. Consider the example given in Figure 4. Here, the red and black subgraph is the graph we might obtain from a set of approximations $\mathcal{A}$, before pruning. Note that all the red vertices would be removed from this graph during pruning, as they are not part of a path from a vertex in S_0 to a vertex in S_7. Nevertheless, the red paths might be quite long paths and it is therefore potentially wasteful to discard such nearly complete paths. Instead, note that we can freely add a vertex to S_0, as long there exists an edge between this vertex and any vertex of S_1. Such an edge would be outside the set $\mathcal{A}$, and including it will effectively increase our search space. These edges are shown in blue in Figure 4, and they ensure that the red subgraph is not removed during any subsequent pruning. As the result of these observations, we propose the following approach:

1. Generate the graph $\bar{G}_\mathcal{E}$ for $r - 2$ rounds. Denote the stages $S_1, \ldots, S_{r-1}$.
2. Add a stage S_0 at the start (respectively S_r at the end) of $\bar{G}_\mathcal{E}$ consisting of all vertices and edges in $\mathcal{A}$ which are incident to a vertex in S_1 (S_{r-1}).

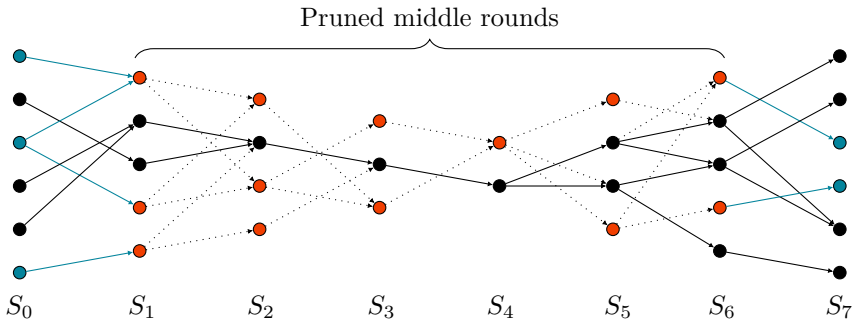


Figure 4: An illustration of vertex anchoring. The black and red graph represents trails built from the set of single round approximations $\mathcal{A}$. The red subgraph would be removed if the red/black graph was pruned. The blue anchor vertices are added to prevent the red subgraph from being removed, increasing the number of trails found.

3. For any vertex in S_1 (respectively S_{r-1}) which does not have an incoming (outgoing) edge, find the longest possible edge going into (out of) this vertex, and add this edge and its start (end) vertex to S_0 (S_r).

For SPN ciphers, Step 3 can be done efficiently simply by finding the output (input) mask to the S-box layer represented by each vertex, and then choosing the best possible input (output) masks for each S-box. In practice, we limit the number of anchor vertices added so as to not increase the search time too much. We find that this method dramatically improves the results for some ciphers.

5.4 Parallelisation

As a practical improvement to the algorithm, we note that most aspects can be parallelised. In particular, whenever we need to calculate $\mathrm{Ex}(\mathcal{P})$, $\mathcal{P}$ can be split into parts and distributed across different threads. Often a main thread will have to collect the results form each of the worker threads, e.g. when calculating $\mathrm{Ex}_{\mathrm{in}}(\mathcal{P}) \cap \mathrm{Ex}_{\mathrm{out}}(\mathcal{P})$ during vertex generation, but this work is quite minimal. Moreover, as mentioned in Section 4.2, the search for hulls can easily be parallelised by distributing different α values across threads. Thus, while the scaling is not perfect, the algorithm benefits quite heavily from increasing the number of threads, especially when $\mathcal{A}$ is large, which is often the case since we want to cover as large a search space as possible.

6 Searching for Linear Approximations and Differentials

We applied the algorithm described here to 17 different SPN ciphers. The types of designs vary from ciphers with very lightweight permutation layers, such as PRESENT, to ciphers with very heavy permutation layers, such as KLEIN. While we did also apply the algorithm to some Feistel designs (i.e. TWINE and MIBS), the main improvements over the basic algorithm presented in Section 5 apply to strict SPN designs, and we were unable to obtain any interesting results for these ciphers due to the increased running time. Moreover, the current implementation of the algorithm only supports ciphers with identical S-boxes, although adding this functionality would not slow down the implementation noticeably.

Note that we investigate three ciphers that use a PRINCE-like design, namely PRINCE, `MANTIS`, and `QARMA`. For these ciphers, we generate a graph for the first half of the rounds, as described above, reverse this graph, and then stitch these two graphs together through the central permutation layer.

Table 1: Result for linear cryptanalysis obtained using the algorithm presented in this work. $\mathcal{A}$ is the set of single round approximations considered, a is the number of anchor vertices used, $\alpha \diamond \beta$ is the set of trails found for the best approximation, ELP is the expected squared correlation for the best approximation, and T_g and T_s is the time in hours to generate and search through the graph, respectively. Entries annotated by † indicates an improvement over a previously published result.

Cipher (Total rounds, block size)	**Rounds**	$\lvert\mathcal{A}\rvert$	a	$\lvert\alpha \diamond \beta\rvert$	**ELP**	T_g	T_s
AES [oST01] (10, 128)	3	$2^{29.9}$	$2^{24.0}$	2^{1}	$2^{-53.36}$	0.0	0.0
	4	$2^{38.8}$	$2^{24.0}$	2^{4}	$2^{-147.88}$	2.5	20.0
EPCBC-48 [YKPH11] (32, 48)	15 † [Bul13]	$2^{26.1}$	–	$2^{31.3}$	$2^{-43.74}$	0.0	0.4
	16 † [Bul13]	$2^{26.1}$	–	$2^{34.0}$	$2^{-46.77}$	0.0	0.4
EPCBC-96 [YKPH11] (32, 96)	31	$2^{27.6}$	–	$2^{63.6}$	$2^{-94.47}$	0.0	0.4
	32	$2^{27.6}$	–	$2^{63.6}$	$2^{-97.59}$	0.0	0.4
Fly [KG16] (20, 64)	8	$2^{32.5}$	–	$2^{6.5}$	$2^{-54.83}$	0.1	6.0
	9	$2^{32.5}$	–	$2^{6.1}$	$2^{-63.00}$	0.2	8.8
GIFT-64 [BPP+17] (28, 64)	11	$2^{31.8}$	–	$2^{5.1}$	$2^{-55.00}$	0.1	8.0
	12	$2^{32.7}$	–	$2^{3.6}$	$2^{-64.00}$	0.2	41.5
Khazad [BR00] (8, 64)	2	$2^{18.3}$	$2^{25.0}$	2^{0}	$2^{-37.97}$	0.0	0.0
	3	$2^{30.1}$	$2^{25.0}$	2^{0}	$2^{-68.01}$	0.2	0.2
KLEIN [GNL11] (12, 64)	5	$2^{30.8}$	$2^{17.0}$	2^{0}	$2^{-46.0}$	0.0	0.0
	6	$2^{39.6}$	$2^{16.9}$	2^{0}	$2^{-66.0}$	0.3	0.0
LED [GPPR11] (32, 64)	4	$2^{24.7}$	2^{25}	2^{2}	$2^{-48.68}$	0.0	0.9
MANTIS$_7$ [BJK+16] ($2 \cdot 8$, 64)	$2 \cdot 4$	$2^{34.3}$	$2^{24.0}$	$2^{15.0}$	$2^{-49.05}$	0.1	0.0
Midori64 [BBI+15] (16, 64)	6	$2^{44.3}$	–	$2^{19.0}$	$2^{-53.02}$	25.9	0.8
	7	$2^{46.5}$	–	$2^{21.9}$	$2^{-62.88}$	53.1	5.5
present [BKL+07] (31, 64)	23 † [Ohk09]	$2^{31.1}$	–	$2^{55.0}$	$2^{-61.00}$	0.1	6.8
	24 † [Ohk09]	$2^{31.1}$	–	$2^{57.9}$	$2^{-63.61}$	0.1	6.9
	25 † [Ohk09]	$2^{31.1}$	–	$2^{60.7}$	$2^{-66.21}$	0.1	6.9
PRIDE [ADK+14] (20, 64)	15	$2^{27.1}$	–	2^{0}	$2^{-58.00}$	0.0	0.0
	16	$2^{37.4}$	–	2^{3}	$2^{-63.99}$	1.8	0.0
PRINCE [BCG+12] ($2 \cdot 6$, 64)	$2 \cdot 3$	$2^{18.1}$	–	$2^{2.0}$	$2^{-54.00}$	0.0	0.0
	$2 \cdot 4$	$2^{38.3}$	–	$2^{6.8}$	$2^{-63.82}$	2.1	0.4
PUFFIN [CHW08] (32, 64)	32	$2^{26.8}$	–	$2^{112.4}$	$2^{-51.90}$	0.0	0.0
QARMA [Ava17] ($2 \cdot 8$, 64)	$2 \cdot 3$	$2^{24.8}$	$2^{24.0}$	$2^{5.0}$	$2^{-53.71}$	0.0	0.0
RECTANGLE [ZBL+14] (25, 64)	12 † [ZBL+14]	$2^{31.1}$	–	$2^{15.0}$	$2^{-52.27}$	0.1	21.1
	13 † [ZBL+14]	$2^{31.1}$	–	$2^{15.9}$	$2^{-58.14}$	0.1	25.9
	14 † [ZBL+14]	$2^{31.1}$	–	$2^{18.3}$	$2^{-62.98}$	0.1	31.1
SKINNY-64 [BJK+16] (32, 64)	8	$2^{41.4}$	$2^{23.7}$	$2^{34.4}$	$2^{-50.46}$	0.7	50.7
	9	$2^{41.4}$	$2^{23.9}$	$2^{31.3}$	$2^{-69.83}$	0.4	8.9

Table 2: Result for differential cryptanalysis obtained using the algorithm presented in this work. $\mathcal{D}$ is the set of single round differentials considered, a is the number of anchor vertices used, $\Delta \diamond \nabla$ is the set of trails found for the best differential, EDP is the expected differential probability for the best differential, and T_g and T_s is the time in hours to generate and search through the graph, respectively. Entries annotated by † indicates an improvement over a previously published result.

Cipher (Total rounds, block size)	Rounds	$\lvert\mathcal{D}\rvert$	a	$\lvert\Delta \diamond \nabla\rvert$	EDP	T_g	T_s
AES [oST01] (10, 128)	3	$2^{18.7}$	$2^{24.0}$	2^{0}	$2^{-54.00}$	0.0	0.0
	4	$2^{36.9}$	$2^{24.0}$	2^{0}	$2^{-150.00}$	0.7	0.3
EPCBC-48 [YKPH11] (32, 48)	13	$2^{28.4}$	–	$2^{21.2}$	$2^{-43.86}$	0.1	13.7
	14	$2^{28.4}$	–	$2^{20.4}$	$2^{-47.65}$	0.1	14.0
EPCBC-96 [YKPH11] (32, 96)	20	$2^{32.8}$	–	$2^{16.9}$	$2^{-92.73}$	1.1	21.6
	21	$2^{32.8}$	–	$2^{19.9}$	$2^{-97.78}$	1.1	22.6
Fly [KG16] (20, 64)	8	$2^{31.6}$	–	$2^{4.9}$	$2^{-55.76}$	0.1	2.6
	9	$2^{33.2}$	–	$2^{7.3}$	$2^{-63.35}$	0.2	17.8
GIFT-64 [BPP+17] (28, 64)	12 † [ZDY18]	$2^{22.4}$	–	$2^{3.3}$	$2^{-56.57}$	0.0	0.0
	13	$2^{22.4}$	–	$2^{3.6}$	$2^{-60.42}$	0.0	0.0
Khazad [BR00] (8, 64)	2	$2^{25.8}$	$2^{24.8}$	2^{0}	$2^{-45.42}$	0.0	0.0
	3	$2^{25.8}$	$2^{25.0}$	2^{0}	$2^{-81.66}$	0.0	0.0
KLEIN [GNL11] (12, 64)	5	$2^{30.8}$	$2^{17.0}$	$2^{1.0}$	$2^{-45.91}$	0.0	0.0
	6	$2^{39.7}$	$2^{24.0}$	$2^{1.0}$	$2^{-69.00}$	0.3	6.4
LED [GPPR11] (32, 64)	4	$2^{37.7}$	$2^{24.0}$	2^{1}	$2^{-49.42}$	0.5	0.1
MANTIS$_7$ [BJK+16] ($2 \cdot 8$, 64)	$2 \cdot 4$	$2^{37.7}$	–	$2^{18.6}$	$2^{-47.98}$	0.9	0.1
Midori64 [BBI+15] (16, 64)	6	$2^{42.2}$	$2^{23.9}$	$2^{19.6}$	$2^{-52.37}$	1.6	1.0
	7	$2^{42.2}$	$2^{23.9}$	$2^{22.8}$	$2^{-61.22}$	1.0	0.9
present [BKL+07] (31, 64)	15	$2^{30.3}$	–	$2^{27.2}$	$2^{-58.00}$	0.1	16.2
	16 † [Abd12]	$2^{30.3}$	–	$2^{28.9}$	$2^{-61.80}$	0.1	18.0
	17	$2^{30.3}$	–	$2^{32.9}$	$2^{-63.52}$	0.1	18.8
PRIDE [ADK+14] (20, 64)	15	$2^{35.9}$	$2^{23.6}$	$2^{5.0}$	$2^{-58.00}$	0.5	36.5
	16	$2^{35.9}$	$2^{23.6}$	$2^{17.4}$	$2^{-63.99}$	0.5	44.1
PRINCE [BCG+12] ($2 \cdot 6$, 64)	$2 \cdot 3$ † [CFG+14]	$2^{14.0}$	2^{19}	2^{1}	$2^{-55.91}$	0.0	0.0
	$2 \cdot 4$	$2^{38.7}$	–	$2^{9.0}$	$2^{-67.32}$	3.0	1.0
PUFFIN [CHW08] (32, 64)	32	$2^{26.0}$	–	$2^{63.7}$	$2^{-59.63}$	0.0	0.0
QARMA [Ava17] ($2 \cdot 8$, 64)	$2 \cdot 3$	$2^{24.8}$	$2^{26.0}$	$2^{7.3}$	$2^{-56.47}$	0.1	0.0
RECTANGLE [ZBL+14] (25, 64)	13 † [ZBL+14]	$2^{31.1}$	–	$2^{15.3}$	$2^{-55.64}$	0.1	32.2
	14 † [ZBL+14]	$2^{31.1}$	–	$2^{15.9}$	$2^{-60.64}$	0.1	41.3
	15 † [ZBL+14]	$2^{31.1}$	–	$2^{18.2}$	$2^{-65.64}$	0.1	50.2
SKINNY-64 [BJK+16] (32, 64)	8	$2^{39.4}$	$2^{24.0}$	$2^{31.0}$	$2^{-50.72}$	0.2	15.0
	9	$2^{41.7}$	$2^{23.8}$	$2^{31.2}$	$2^{-69.64}$	0.4	6.4

6.1 Results for ELP and EDP

We ran the algorithm using an Intel Xeon E5-2650 v4 processor (24 threads at 2.2 GHz) with 256 GB of memory available. The results for linear cryptanalysis are shown in Table 1 and the results for differential cryptanalysis in Table 2. Note that the number of rounds stated here is the number of non-linear layers (i.e. S-box layers) applied.

The number of single round approximations or differentials considered when generating the graph is the smallest that gave the stated result – for most ciphers, we investigated larger search spaces without obtaining any improvements. In general, it is interesting to note that for the majority of ciphers, actually generating the graph is quite fast, while searching through the graph can take considerably longer. If one has an idea of what input/output masks/differences are good, the graph can be restricted to paths between these interesting values, which will greatly reduce the search time. A general strategy for using the algorithm could therefore be to find some preliminary interesting approximations/differentials using a small search space, and then increase the search space while restricting the graph to these approximations/differentials in order to improve the estimates.

Entries annotated with a † indicate improvements over previous best results. Entries that are not annotated are either new or do not improve on known results. For many of the ciphers, the search found multiple approximations/differentials that were equally good. It is therefore possible that multiple linear/differential attacks could be mounted on a larger number of rounds than stated here.

We highlight a few interesting results. For RECTANGLE, the designers did take into account multiple trails in [ZBL+14], and estimated that over 14 rounds the best differential has EDP $2^{-62.83}$. We improve this to $2^{-60.64}$, demonstrating that being able to include a larger number of trails can improve estimates.

For `GIFT-64`, [ZDY18] used a MILP based tool to find a 12 round differential trail with probability 2^{-60}. By taking into account multiple trails, we improve this to $2^{-56.57}$ and find a 13 round differential with probability $2^{-60.42}$. Thus, we can potentially extend their attack by one round.

For PRESENT, we improve some results of [Abd12]. In particular, we improve their result for 16 round differentials from $2^{-62.58}$ to $2^{-61.80}$ and furthermore find a 17 round differential with probability $2^{-63.52}$. For linear cryptanalysis, we match the results of [Abd12], although interestingly we find fewer trails. This shows that the algorithm presented here can match or even improve the results obtained by the partial correlation/difference transition matrix method, all the while being more versatile.

6.2 Visualising Trail Graphs

An interesting side effect of applying our new algorithm is that we can visualise the linear/differential trails in order to get a better understanding of how the cipher's structure influences its resistance to linear and differential cryptanalysis. Figure 5 show the linear hull graphs that we generated for three different ciphers: PRESENT, PRIDE, and KLEIN. The vertices in each stage are ordered by their value as integers.

While the search spaces selected for the three ciphers are comparable in size, the resulting graphs have widely different structures. The graph for PRESENT show that each stage is identical, and that the stages are highly connected. Thus, as observed in [Ohk09], there exists a very large number of trails for many approximations of PRESENT that have similar structure and therefore similar correlation contribution. PRIDE also exhibits identical stages, and we can even observe iterative trails, but there are only very few vertices in each stage, preventing the number of trails from exploding. The graph for KLEIN (which has a very heavy linear layer), shows a very large number of edges in the graph, but the structure of the stages vary, resulting in no clustering of trails. Indeed, Table 1 shows that we only found one trail for the best approximations over 5 and 6 rounds.

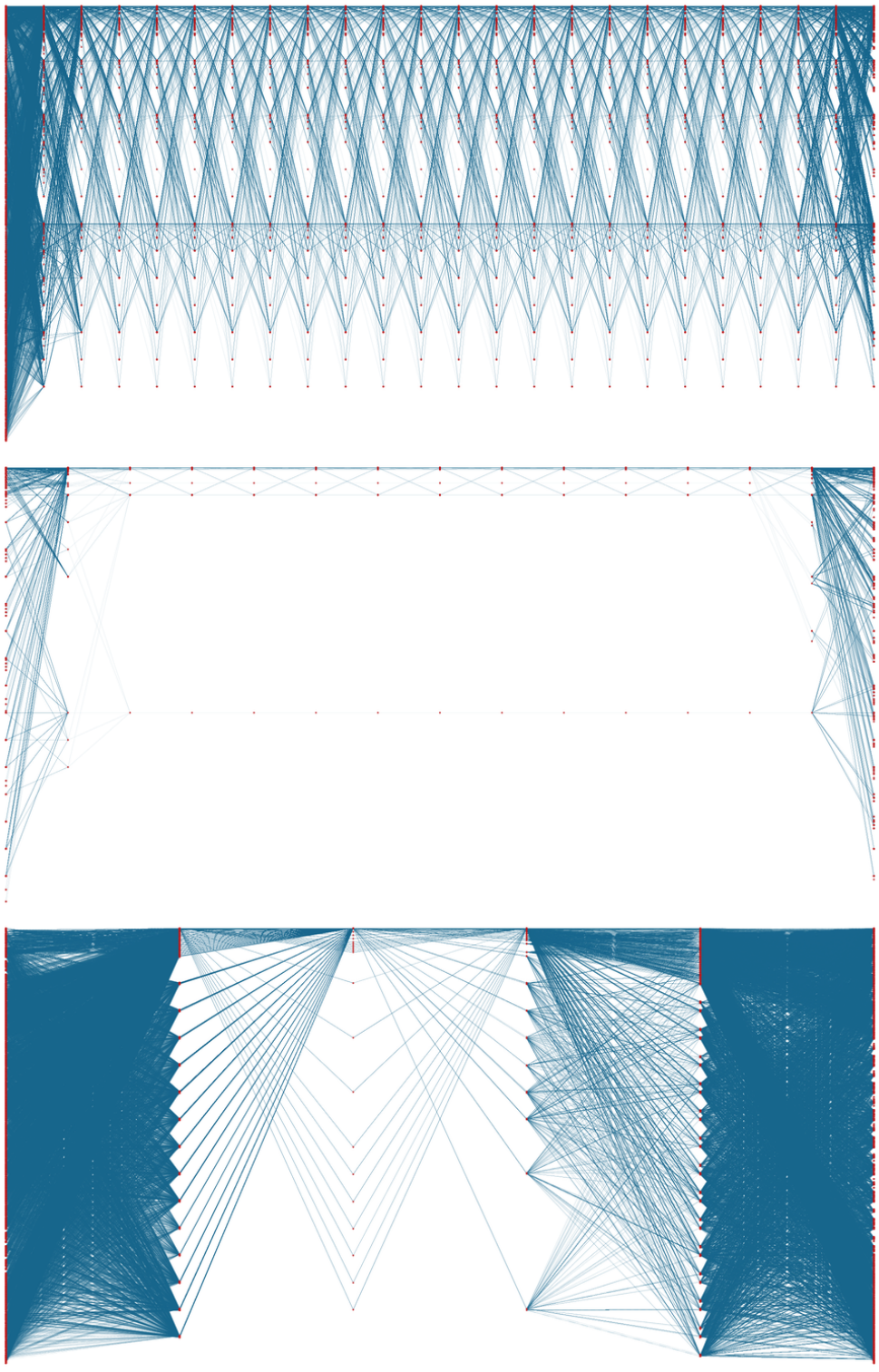


Figure 5: Examples of linear hull graphs generated by our algorithm. Top: 23 rounds of PRESENT using $|\mathcal{A}| = 2^{24.7}$ single round approximations. Middle: 14 rounds of PRIDE, also with $|\mathcal{A}| = 2^{24.7}$. Bottom: 5 rounds of KLEIN, with $|\mathcal{A}| = 2^{26.8}$ and using 2^{17} anchoring vertices.

7 Correlation Distributions

Determining the ELP and EDP of the best linear approximations and differentials is important when assessing the strength of a cipher against these attacks. However, these summary statistics do not paint to full picture: in reality, the linear correlation and differential probability vary over the key space, and more detailed knowledge about the distribution of these values can lead to stronger distinguishers. As an example, [BV17] demonstrated how asymmetries in the joint correlation distribution of multiple linear approximations of DES can be used to improve attacks.

For differentials, not much is known about how the differential probabilities vary as the key changes. For linear cryptanalysis, there has been an increased interest in developing more accurate models for the key dependent behaviour, see e.g. [BT13, HVLN15, BN17, BN16, BTV18]. This line of research is in large part facilitated by the following useful result.

Theorem 1 ([DR02])**.** *Let (α, β) be a linear approximation of an SPN cipher and let $\bar{k}$ denote the concatenation of the cipher's round keys for the encryption key k. Then the linear correlation is given by*

$$C^k_{(\alpha,\beta)} = \sum_U (-1)^{s_U \oplus \langle U, \bar{k} \rangle} |C^k_U|,$$

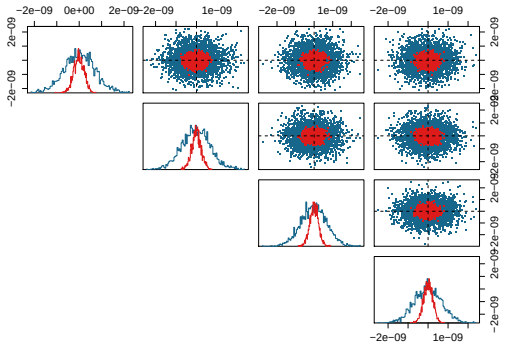
where the sum is over trails $U = (\alpha, \ldots, \beta)$, s_U is the sign bit of U, and $|C^k_U|$ is independent of k.

The above theorem indicates that for an SPN cipher we can determine the key dependent correlation by adjusting the sign of each trail's correlation contribution. Consequently, we can estimate the distribution over the key-space by doing this for a large number of keys. A similar result holds for Feistel ciphers with SPN like F-functions.

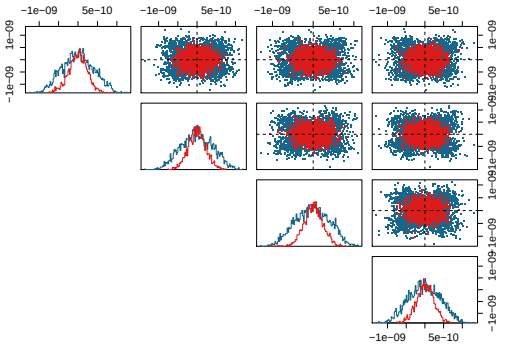
7.1 Finding Key-Dependent Distributions

Our algorithm for estimating ELP can easily be adapted to efficiently calculate key dependent correlations instead. The main idea is simply to construct the graph $\bar{G}_{\mathcal{E}}$, but using the signed correlation values instead of the squared correlation as edge weights, and then adjust the sign of the edges for each different key. Note that we can easily find the signs of each edge after we have generated $\bar{G}_{\mathcal{E}}$, as we know the input and output masks each edge represents. Thus, we can find the signed correlation of an approximation by using a slightly adapted version of the algorithm presented in Section 4.2 (we assume that a pre-whitening key k_0 is used):

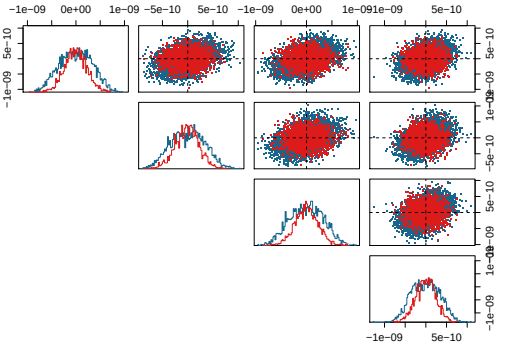
1. Choose an encryption key k.
2. Let $\mathcal{H}$ be an empty hash table. Choose an $\alpha \in S_1$ and let $\mathcal{H}(\alpha) = (-1)^{\langle \alpha, k_0 \rangle}$.
3. For each stage S_0 to S_{r-1} of $\bar{G}_{\mathcal{E}}$, do the following:
   (a) Let k_i be the current round-key.
   (b) Create an empty hash table $\mathcal{H}'$.
   (c) For each key of $\mathcal{H}$, let u be the corresponding vertex in $\bar{G}_{\mathcal{E}}$. Let $c = \mathcal{H}(u)$. Then, for each edge $u \to v$, if $\mathcal{H}'(v)$ does not exists, let $\mathcal{H}'(v) = c \cdot (-1)^{\langle v, k_i \rangle} \cdot l(u \to v)$. Otherwise, let $\mathcal{H}'(v) = \mathcal{H}'(v) + c \cdot (-1)^{\langle v, k_i \rangle} \cdot l(u \to v)$.
   (d) Let $\mathcal{H} = \mathcal{H}'$.
4. $\mathcal{H}(\beta)$ now contains $C^k_{(\alpha,\beta)}$.



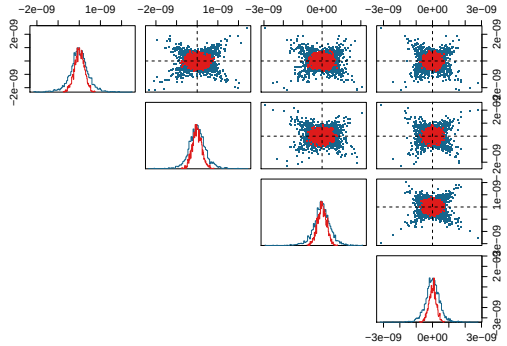
(a) Approximations of 23 rounds of PRESENT. The plot shows that the joint correlation distribution for PRESENT is close to normal.



(b) Approximations over 9 rounds of FLY. For each pair of approximations we observe four distinct clusters in the distributions.



(c) Approximations over 12 rounds of `GIFT-64`. For each pair of approximations we observe two distinct clusters in the distributions. This indicates a dependence between the approximations.



(d) Approximations over 14 rounds of RECTANGLE. For each pair of approximations we observe a significant deviation from normality, manifested by very long tails of the distributions.

Figure 6: Shown in blue, the pairwise joint linear correlation distributions for four linear approximations. The correlation distribution of an ideal cipher is shown in red.

5. Repeat for as many encryption keys as desired.

Clearly, this procedure only calculates a partial sum of $C^k_{(\alpha,\beta)}$. To obtain a better approximation of the actual value, we use the signal/noise decomposition technique proposed in [BT13]. This technique is summarised the in the following lemma.

Lemma 1 ([BT13])**.** *Let $\mathcal{S}$ be a set of strong linear trails for an approximation (α, β). Then $C^k_{(\alpha,\beta)}$ can be approximated by*

$$C^k_{(\alpha,\beta)} = \left(\sum_{U \in \mathcal{S}} (-1)^{s_U \oplus \langle U, \bar{k} \rangle} |C^k_U| \right) + \mathcal{N}(0, 2^{-n}),$$

where $\mathcal{N}(0, 2^{-n})$ denotes the normal distribution with mean 0 and variance 2^{-n}.

7.2 Results

We have applied the above technique to some of the ciphers we investigated in Section 6. That is, we calculated the partial sum of $C^k_{(\alpha,\beta)}$ for 10 000 randomly chosen encryption keys, and then added the noise distribution $\mathcal{N}(0, 2^{-n})$ to the resulting data sets. We

note that when doing this for only a few approximations, the process takes at most a few minutes, depending on the cipher. In light of the results of [BV17] we consider the joint distributions of four different ciphers.

Figure 6a shows the pairwise joint distributions of four linear approximations over 23 rounds of PRESENT. As a reference, the correlation distribution of an ideal cipher is shown, i.e. a bivariate normal distribution with marginals $\mathcal{N}(0, 2^{-n})$. In this case, the correlation distributions appear to be close to normal and entirely independent, resulting in a joint normal distribution. This matches the observations made in [BTV18].

Figure 6b shows the same picture but for 9 rounds of FLY. However, in this case, while the marginal correlation distributions appear the be close to normal, when considering the joint distributions, we can see that there are four clusters of observations for each pair of approximations. A similar situation occurs over 12 rounds of `GIFT-64`, as shown in Figure 6c, only here we only observe two clusters for each pair. As in [BV17], this would indicate that there is a heavy overlap in the trails of the approximations, resulting in a strong dependence between the signs of the correlations.

Finally, we consider approximations over 14 rounds of RECTANGLE in Figure 6d. Here, we observe even stranger behaviour, as the marginal distributions do not even appear to be normal. In fact, the distributions have much longer tails than expected, which would indicate that there is a large percentage of weak keys for which a linear attack would work better than expected.

The last three examples show that even if the ELP is close to the value expected from an ideal block cipher, the actual correlation distributions might exhibit additional behaviour which can be exploited in an attack. Attacks of this type warrant further investigation, and hopefully the algorithm presented in this work will make this line of research easier.

8 Future Work

The algorithm presented in this work has much potential for further extensions and improvements. First and foremost, it would be very useful to find improvements similar to those of Section 5 that apply to other types of ciphers, in particular Feistel designs and designs that are not based on S-boxes. This is closely related to the strategy for selecting edges, discussed in Section 4.1. As also pointed out there, it would be interesting to use the results of [BV14, KLT15] to develop an edge selection strategy for ARX and AND-RX designs.

In more general terms, it would also be highly interesting to explore different heuristics for the edge selection, as selecting the longest edges is not necessarily the best strategy. This consideration has two aspects: First, we might obtain globally better results by including very bad edges locally, and second, for all the ciphers we investigated, we end up only using a very small subset of the single round approximations/differentials we initially consider. As such, we waste much time and memory considering edges we are ultimately not interested in. A better heuristic that can filter out (some) of these edges early would potentially improve the algorithm.

Finally, we entertain the possibility that the general graph framework could be extended to other types of cryptanalysis. Indeed, we could describe any property that propagates through the round-function of a cipher as a path through a graph. As such, it might be possible to apply the technique to search for e.g. division properties.

References

[AAA+15] Mohamed Ahmed Abdelraheem, Javad Alizadeh, Hoda A. AlKhzaimi, Mohammad Reza Aref, Nasour Bagheri, and Praveen Gauravaram. Improved Linear

Cryptanalysis of Reduced-Round SIMON-32 and SIMON-48. In *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, pages 153–179, 2015.

[Abd12] Mohamed Ahmed Abdelraheem. Estimating the Probabilities of Low-Weight Differential and Linear Approximations on PRESENT-Like Ciphers. In *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, pages 368–382, 2012.

[ADK+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. Block Ciphers - Focus on the Linear Layer (feat. PRIDE). In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 57–76, 2014.

[AK18] Ralph Ankele and Stefan Kölbl. Mind the Gap - A Closer Look at the Security of Block Ciphers against Differential Cryptanalysis. In *Selected Areas in Cryptography - SAC 2018*, 2018.

[Ava17] Roberto Avanzi. The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 411–436, 2015.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 208–225, 2012.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153, 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, pages 450–466, 2007.

[BN16] Céline Blondeau and Kaisa Nyberg. Improved Parameter Estimates for Correlation and Capacity Deviates in Linear Cryptanalysis. *IACR Trans. Symmetric Cryptol.*, 2016(2):162–191, 2016.

[BN17] Céline Blondeau and Kaisa Nyberg. Joint Data and Key Distribution of Simple, Multiple, and Multidimensional Linear Cryptanalysis Test Statistic and Its Impact to Data Complexity. *Design, Codes and Cryptography*, 82(1-2):319–349, 2017.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 321–345, 2017.

[BR00] Paulo S.L.M. Barreto and Vincent Rijmen. The khazad legacy-level block cipher. *Primitive submitted to NESSIE*, 97, 2000.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, pages 2–21, 1990.

[BT13] Andrey Bogdanov and Elmar Tischhauser. On the Wrong Key Randomisation and Key Equivalence Hypotheses in Matsui's Algorithm 2. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 19–38, 2013.

[BTV18] Andrey Bogdanov, Elmar Tischhauser, and Philip S. Vejre. Multivariate Profiling of Hulls for Linear Cryptanalysis. *IACR Trans. Symmetric Cryptol.*, 2018(1):101–125, 2018.

[Bul13] Stanislav Bulygin. More on linear hulls of PRESENT-like ciphers and a cryptanalysis of full-round EPCBC-96. *IACR Cryptology ePrint Archive*, 2013:28, 2013.

[BV14] Alex Biryukov and Vesselin Velichkov. Automatic Search for Differential Trails in ARX Ciphers. In *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, pages 227–250, 2014.

[BV17] Andrey Bogdanov and Philip S. Vejre. Linear Cryptanalysis of DES with Asymmetries. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 187–216, 2017.

[CFG+14] Anne Canteaut, Thomas Fuhr, Henri Gilbert, María Naya-Plasencia, and Jean-René Reinhard. Multiple Differential Cryptanalysis of Round-Reduced PRINCE. In *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, pages 591–610, 2014.

[CHW08] Huiju Cheng, Howard M. Heys, and Cheng Wang. PUFFIN: A Novel Compact Block Cipher Targeted to Embedded Digital Systems. In *11th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2008, Parma, Italy, September 3-5, 2008*, pages 383–390, 2008.

[CMST15a] Jiageng Chen, Atsuko Miyaji, Chunhua Su, and Jesen Teh. Accurate Estimation of the Full Differential Distribution for General Feistel Structures. In *Information Security and Cryptology - 11th International Conference, Inscrypt 2015, Beijing, China, November 1-3, 2015, Revised Selected Papers*, pages 108–124, 2015.

[CMST15b] Jiageng Chen, Atsuko Miyaji, Chunhua Su, and Jesen Teh. Improved Differential Characteristic Searching Methods. In *IEEE 2nd International Conference on Cyber Security and Cloud Computing, CSCloud 2015, New York, NY, USA, November 3-5, 2015*, pages 500–508, 2015.

[DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Heuristic Tool for Linear Cryptanalysis with Applications to CAESAR Candidates. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 490–509, 2015.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Mathematical Cryptology*, 1(3):221–242, 2007.

[FLN07] Pierre-Alain Fouque, Gaëtan Leurent, and Phong Q. Nguyen. Automatic Search of Differential Path in MD4. *IACR Cryptology ePrint Archive*, 2007:206, 2007.

[FWG+16] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 268–288, 2016.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A New Family of Lightweight Block Ciphers. In *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, pages 1–18, 2011.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 326–341, 2011.

[HVLN15] Jialin Huang, Serge Vaudenay, Xuejia Lai, and Kaisa Nyberg. Capacity and Data Complexity in Multidimensional Linear Attack. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 141–160, 2015.

[KG16] Pierre Karpman and Benjamin Grégoire. The LITTLUN S-box and the FLY block cipher. In *Lightweight Cryptography Workshop*, pages 17–18, 2016.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON Block Cipher Family. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 161–185, 2015.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, pages 17–38, 1991.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, pages 386–397, 1993.

[Mat94] Mitsuru Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, pages 366–375, 1994.

[MP13] Nicky Mouha and Bart Preneel. Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. *IACR Cryptology ePrint Archive*, 2013:328, 2013.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, pages 57–76, 2011.

[Nyb94] Kaisa Nyberg. Linear approximation of block ciphers. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, pages 439–444, 1994.

[Ohk09] Kenji Ohkuma. Weak Keys of Reduced-Round PRESENT for Linear Cryptanalysis. In *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, pages 249–265, 2009.

[oST01] National Institute of Standards and Technology. 197: Advanced encryption standard (AES). *Federal information processing standards publication*, 197(441):0311, 2001.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 158–178, 2014.

[Ste13] Marc Stevens. New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 245–261, 2013.

[YDG89] S. H. Yen, David Hung-Chang Du, and Subbarao Ghanta. Efficient Algorithms for Extracting the K most Critical Paths in Timing Analysis. In *Proceedings of the 26th ACM/IEEE Design Automation Conference, Las Vegas, Nevada, USA, June 25-29, 1989.*, pages 649–654, 1989.

[YKPH11] Huihui Yap, Khoongming Khoo, Axel Poschmann, and Matt Henricksen. EPCBC - A Block Cipher Suitable for Electronic Product Code Encryption. In *Cryptology and Network Security - 10th International Conference, CANS 2011, Sanya, China, December 10-12, 2011. Proceedings*, pages 76–97, 2011.

[YML+17] Jun Yin, Chuyan Ma, Lijun Lyu, Jian Song, Guang Zeng, Chuangui Ma, and Fushan Wei. Improved Cryptanalysis of an ISO Standard Lightweight Block Cipher with Refined MILP Modelling. In *Information Security and Cryptology - 13th International Conference, Inscrypt 2017, Xi'an, China, November 3-5, 2017, Revised Selected Papers*, pages 404–426, 2017.

[ZBL+14] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: A Bit-slice Ultra-Lightweight Block Cipher Suitable for Multiple Platforms. *IACR Cryptology ePrint Archive*, 2014:84, 2014.

[ZDY18] Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. MILP-based Differential Attack on Round-reduced GIFT. Cryptology ePrint Archive, Report 2018/390, 2018. https://eprint.iacr.org/2018/390.