Heuristic Tool for Linear Cryptanalysis with Applications to CAESAR Candidates

Christoph Dobraunig, Maria Eichlseder, and Florian Mendel

Graz University of Technology, Austria
`christoph.dobraunig@iaik.tugraz.at`

Abstract. Differential and linear cryptanalysis are the general purpose tools to analyze various cryptographic primitives. Both techniques have in common that they rely on the existence of good differential or linear characteristics. The difficulty of finding such characteristics depends on the primitive. For instance, AES is designed to be resistant against differential and linear attacks and therefore, provides upper bounds on the probability of possible linear characteristics. On the other hand, we have primitives like SHA-1, SHA-2, and Keccak, where finding good and useful characteristics is an open problem. This becomes particularly interesting when considering, for example, competitions like CAESAR. In such competitions, many cryptographic primitives are waiting for analysis. Without suitable automatic tools, this is a virtually infeasible job. In recent years, various tools have been introduced to search for characteristics. The majority of these only deal with differential characteristics. In this work, we present a heuristic search tool which is capable of finding linear characteristics even for primitives with a relatively large state, and without a strongly aligned structure. As a proof of concept, we apply the presented tool on the underlying permutations of the first round CAESAR candidates Ascon, ICEPOLE, Keyak, Minalpher and Prøst.

Keywords: linear cryptanalysis · authenticated encryption · automated tools · guess-and-determine · CAESAR competition

1 Introduction

Research in symmetric cryptography in the last few years is mainly driven by dedicated high-profile open competitions such as NIST's AES and SHA-3 selection procedures, or ECRYPT's eSTREAM project. While these focused competitions in symmetric cryptography are generally viewed as having provided a tremendous increase in the understanding and confidence in the security of these cryptographic primitives, the impressive increase of submissions to such competitions reveal major problems related to the analytical effort for the cryptographic community. To better evaluate the security margin of the various submissions, automatic tools are needed to assist cryptanalysts with their work.

One important class of attacks is linear cryptanalysis [25,15]. The success of these attacks relies on the existence of suitable linear characteristics. The difficulty of finding such characteristics depends on the primitive. For example, the wide-trail design strategy [7] incorporated by AES provides lower bounds on the minimum number of active S-boxes in a linear characteristic and therefore, gives an upper bound on the highest possible bias. On the other hand, we have primitives with weak alignment [1], such as the winner of the SHA-3 competition Keccak, where finding good characteristics is an open problem, and heuristic search results are required to evaluate the security margin of the primitive. This is particularly interesting in the context of the CAESAR competition [26]. We noticed that many first round submissions focus their analysis on differential cryptanalysis, but provide only few results for linear cryptanalysis.

Our contribution. The main contribution of this paper is a dedicated automatic tool for linear cryptanalysis, which is available at github[1]. The tool performs heuristic searches for good linear characteristics in cryptographic primitives. It was designed for primitives based on substitution-permutation networks (SP networks).

The modular design of the tool allows easy extension to other cryptographic primitives. It also allows to easily develop and test new dedicated search strategies. To facilitate further improvements and analysis, the tool is publicly available and its source code is published together with this paper. Such a tool is particularly useful when designing new cryptographic primitives. It allows to easily explore the effects of, for instance, different S-boxes and linear layers on linear characteristics and reveals possible bad decisions in an early stage of the design process. Even in wide-trail designs with provable bounds, it can be useful to evaluate different choices for building blocks with respect to their long-term behaviour over a larger number of rounds, where the quality of the best characteristics can deviate significantly from the derived bounds (i.e., two algorithms with the same bounds may behave quite differently in a heuristic search, which can be a basis for the decision of choosing one design over the other).

As a proof of concept and to demonstrate the advantages of the tool, we have chosen the first round CAESAR candidates Ascon [9], ICEPOLE [19], Keyak [4], Minalpher [22] and PRØST [13] as analysis targets. Ascon, ICEPOLE, and Keyak are sponge-based authenticated encryption schemes. All three primitives use permutations that are not strongly aligned, making it hard to find good linear characteristics. We demonstrate the capability of our automated search tool by giving linear characteristics suitable for different attack scenarios. In comparison, the permutations used in Minalpher and PRØST provide more "structure" by incorporating an "AES-like" design strategy. Hence, the designers of these two primitives are able to give computer-aided bounds on the minimum number of active S-boxes by using mixed-integer linear programming (MILP) for a number of rounds sufficient to thwart attacks. For Minalpher and PRØST, we show that our tool is capable of finding linear characteristics which match the provided bounds. Our results are summarized in Table 1 (Sect. 4).

[1] https://github.com/iaikkrypto/lineartrails

Related Work. While several automatic tools for differential cryptanalysis have been published in the last few years [23,8,12,6,16,14,5,20], in particular for hash functions, the work on automatic tools dedicated to linear cryptanalysis is very limited. One example is a tool designed by Sun et al. [24], extending previous work of Mouha et al. [21]. They model the differential and linear behavior of a block cipher as a mixed-integer linear program (MILP) and use general-purpose MILP tools to solve the optimization problem (i.e., find the optimal characteristics for the – often simplified – model of the cipher). This approach works well for lightweight ciphers like Simon or Present, but faces problems when it comes to large-state and less structured ciphers such as Ascon, ICEPOLE, and Keyak. Hence, a dedicated search tool for linear characteristics will complement the existing tools.

Outline. This paper is divided into two main parts: the description of our new automated search tool for linear characteristics in Sect. 3, and its application to the CAESAR candidates in Sect. 4. However, first, we start with a short introduction to linear cryptanalysis and our notation in Sect. 2. Then, we deal with the propagation of linear masks in Sect. 3.2 and discuss the proposed search strategy for linear characteristics in Sect. 3.3. The application of the tool (Sect. 4) is first discussed in detail for Keyak in Sect. 4.1. Then, our results for the other ciphers are summarized and briefly discussed in Sect. 4.2 to 4.5. Finally, we conclude in Sect. 5.

2 Linear cryptanalysis

The goal of linear cryptanalysis [25,15] is to identify good affine linear approximations for the target cipher. More specifically, we want to find linear equations between the plaintext bits, ciphertext bits and key bits that hold with probability significantly different from $\frac{1}{2}$ (bias). Then, in the actual attack phase, these equations can be used to derive information on the key bits from known plaintext-ciphertext pairs.

For linear cryptanalysis, the operation of the cipher, or building blocks of the cipher, is considered as a vectorial boolean function $f : \mathbb{F}_2^m \to \mathbb{F}_2^n$ (where the key bits might be part of $\mathbb{F}_2^m$). A (probabilistic) linear relation between input and output bits of f is then characterized by two linear masks $\alpha \in \mathbb{F}_2^m, \beta \in \mathbb{F}_2^n$. For $x \in \mathbb{F}_2^m$, $y \in \mathbb{F}_2^n$ with $y = f(x)$, the masks represent the relation

$$\alpha^t \cdot x = \beta^t \cdot y,$$

where $v^t \cdot w$ denotes the natural inner product of vectors. The quality of a linear approximation α, β is measured by the probability that the corresponding relation holds; or more precisely, by how far this probability deviates from the average $\frac{1}{2}$. This deviation is referred to as the bias of the masks α, β:

$$\begin{aligned}\varepsilon_{\alpha,\beta} = \text{bias}_f(\alpha,\beta) &= \left|\mathbb{P}\left[\alpha^t \cdot x = \beta^t \cdot y \mid y = f(x)\right] - \frac{1}{2}\right| \\ &= \frac{1}{2^m} \cdot \left| \left|\left\{x \in \mathbb{F}_2^m \mid \alpha^t \cdot x = \beta^t \cdot f(x)\right\}\right| - 2^{m-1}\right|.\end{aligned}$$

If m is very small, the expression for $\varepsilon_{\alpha,\beta}$ can easily be evaluated explicitly for all masks α, β to determine the best masks. This information is summarized in the linear distribution table (LDT), where non-zero entries mark masks α, β with non-zero bias.

However, this is obviously infeasible for the complete cipher at once. To obtain an approximation of the complete cipher, it is split into smaller parts that are easier to analyze. Matsui's piling-up lemma [15] is used to combine the individual biases of multiple building blocks to derive the overall bias (under the assumption that the validity of the partial approximations is independent). If ε denotes the bias of the overall approximation of the block cipher, Matsui [15] showed that the necessary number of plaintext-ciphertext pairs to derive the bit of key information from the approximation is proportional to $\frac{1}{\varepsilon^2}$.

The difficult part is to find a network or "trail" of partial approximations that are compatible with each other and give a good overall bias. In particular, each involved approximation must have non-zero bias, otherwise the overall bias becomes zero. For this reason, we refer to non-zero entries in the individual LDTs as "valid transitions" of masks for this building block. In the the following, such a "trail" of partial linear approximations is called linear characteristic.

Several algorithms and improvements thereof have been proposed for finding characteristics with the highest overall bias, typically by a sort of branch-and-bound algorithms. For more complex, modern ciphers, such a complete search is not feasible. Two possible approaches to handle this situation are (a) to design ciphers in a way to allow to prove bounds on the best possible bias, and (b) to use heuristic search methods to find stronger biases (for reduced versions of the cipher) to make better predictions on the security margin of the complete cipher.

In the following, we will focus on the second approach, and heuristically search for good characteristics. Unlike the original, complete search algorithms, our search will not proceed in a "linear", round-by-round manner. Instead, we will take inspiration from similar searching tools for differential cryptanalysis [8], and randomize the search order. This naturally implies that we will often start building inconsistent characteristics, which will need to be fixed or discarded.

3 An automated tool for linear cryptanalysis

The proposed automated tool can be roughly split into two main parts. The first part is described in Sect. 3.2 and deals with the description of cryptographic primitives within the search tool, including the representation of linear approximations and, most importantly, their propagation. The other part of the tool is the choice of the search algorithm to find good linear characteristics (see Sect. 3.3). Before we start with the description of the tool, we take a look at the requirements we have for the design and implementation of such a heuristic search tool.

3.1 Implementation requirements for the search tool

In order for any automatic cryptanalysis tool to be useful for general application, for example to analyze the 57 first round CAESAR submissions, there are a number of flexibility and usability requirements:

– **Easy to add new primitives.** This is one of the main goals for the creation of this tool. To fulfill this requirement, we have decided to put the focus on primitives based on SP networks, i.e., with alternating S-box and linear layers. This simplifies the design process of the tool, since we did not have to consider every possible specialty, while still having a large group of applicable primitives. The programming interface should be designed to require as little effort as possible for converting, for example, a CAESAR reference implementation to a suitable cipher definition for the tool – ideally, it should possible to just copy the corresponding code fragments for the round transformation steps.
– **An easily adaptable, parameterized search algorithm.** The linear tool implements a heuristic guess-and-determine search algorithm. This algorithm delivers good results for various primitives. However, the success of the search is highly dependent on various different parameters, such as the configuration of the searching order and conflict-handling behavior. Therefore, it is crucial that these parameters can be adjusted easily. For this reason, our standard guess-and-determine algorithm is parameterizable via an XML-file. This XML-file specifies the search starting point and allows to configure various other parameters.
– **Easy to add other search algorithms.** The currently implemented, stack-based guess-and-determine algorithm is certainly not the only possible way to search for linear characteristics. To be open for new ideas and evaluate other algorithms, we have designed the tool in a way that the search algorithm is clearly separated from the description of the cipher and thus, can be replaced easily. This opens the door for experiments with various alternative search algorithms and will hopefully lead to new insights in this direction.
– **Portability of the code.** We do not want the tool to require a specific operating system or platform to run. Therefore, we have reduced the dependence from external libraries whenever possible, and omitted the use of platform-specific instructions.

3.2 Propagation of linear masks

Our overall search strategy is based on the “guess-and-determine” approach. We want to build a consistent linear characteristic with high bias step by step, starting from a "mostly unknown" (undetermined) characteristic of masks, and progressively deciding which bits should be selected (activated) by the final mask. For this purpose, we repeatedly "guess" the value of small parts of the masks, and then "determine" the consequences of this guess (in particular, whether this updated partial characteristic can still be completed to a "valid" characteristic). We refer to the "determining" step as propagation of information.

Representation of partial linear masks. The tool represents the linear masks on bit-level. During the search, we work with partially-determined search masks. We represent an active bit in the linear mask with `1` and an inactive bit of a linear mask with `0`. Mask bits that are not yet determined are represented by `?`.

Propagation in SP networks. We want to find linear characteristics for SP networks. Such a network consists of iterative applications of a substitution layer (consisting of relatively small S-boxes) and an (affine) linear layer (which typically covers larger parts of the state at once). We use different techniques for the propagation of information in these two layer types. The goal of the propagation step is to investigate whether the guess allows to derive explicit values for other ("neighbouring") bits, and in particular whether this explicit information is contradictory. The constraints that allow this propagation can be derived from the linear distribution table of the involved functions, since the characteristic must not contain any mask transitions with bias 0.

Propagation in the non-linear layer. We only deal with non-linear layers which can be represented by parallel applications of S-boxes. So the propagation of the linear masks at the input and the output of the S-boxes can be treated individually, since the parallel applications are considered independent of each other (any dependencies induced by the linking linear layers are treated separately). Therefore, we can do the propagation separately per S-box.

Many state-of-the-art ciphers use relatively small S-boxes. In many recent cipher proposals, the S-boxes map 4- to 5-bit inputs to outputs of the same size. Even the largest S-boxes hardly ever exceed a size of 8 bits. Therefore, the propagation of the linear masks can be done in a brute-force manner, based on the linear distribution table (LDT) of the S-box. The LDT is an exhaustive list of all valid (biased) mask transitions from mask α to mask β. Our current "knowledge" of the values of some input and output mask bits limits the set of available transitions. Depending on the concrete values of α and β and the remaining transition options, we have one of the following outcomes of the propagation:

1. **Contradiction:** The LDT reveals that no valid, biased transitions remain that satisfy the fixed mask bits; i.e., there is no linear relationship involving the bits currently marked by α and β as `1` (and optionally the `?` bits). In other words, we have a contradiction. This means that the current, partially

determined linear characteristic is in fact invalid. This situation has to be handled by the search algorithm by, e.g., backtracking and resolving the contradiction.

2. **Updated bits:** The LDT reveals that one or more biased transitions respecting the partially determined α and β remain. In addition, all remaining transitions share the same value (`0` or `1`) for one or more of the current `?` bits. Thus, we can refine some previously undefined bits in the masks to active or inactive bits by using information from the LDT. Before taking any further guesses, this newly-won information must in turn be propagated in all connected function components.
3. **No updates:** The LDT reveals that α and β are possible, but no additional explicit bit-wise information can be won. Nothing else happens.

Propagation in the linear layer. There are two main differences between the linear and non-linear layers from the propagation perspective: On the one hand, the linear layer typically involves significantly more variables than individual S-boxes. On other hand, propagating partial linear masks for linear functions can be achieved easily using basic linear algebra.

Assume that the function $f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ is linear, i.e., $f(x) = A \cdot x = y$ for some $A \in \mathbb{F}_2^{n \times m}$. Note that we can include affine linear functions in the same model, since the affine (constant) part is irrelevant for the bias of the linear model if we do not consider the sign of the bias. Then, for a fully determined mask α, β, the bias $\varepsilon_{\alpha,\beta}$ is either 0 (wrong model) or $\frac{1}{2}$ (exact, correct model). More specifically, α, β is a valid input-output mask if

$$\begin{aligned} \forall x :\ \alpha^t \cdot x = \beta^t \cdot f(x) \quad &\Leftrightarrow \quad \forall x :\ \alpha^t \cdot x = \beta^t \cdot (A \cdot x) \\ &\Leftrightarrow \quad \forall x :\ \alpha^t \cdot x = (A^t \cdot \beta)^t \cdot x \\ &\Leftrightarrow \quad \alpha = A^t \cdot \beta. \end{aligned}$$

If α and β are only partially determined, all propagation can be performed by propagating the information in the linear system $\alpha = A^t \cdot \beta$. For this purpose, we always keep the half-solved system in reduced row echelon form for all linear layers. Whenever mask values in α or β are updated, we perform partial Gaussian elimination to retain reduced row echelon form. If in the process, other bits of α or β are determined (case 1 or 2 from above), this information is extracted from the system and instead stored in the regular representation α, β of the mask bits that is also used for S-box propagation.

Update process. Every time the propagation step leads to new, explicit information in the linear masks (i.e., mask bits that were previously undetermined are now fixed, case 2), this information has to be propagated over the connected linear or non-linear layers, which share those updated mask bits. In other words, the propagation step needs to be iterated to update the neighbouring layers. Since we require that every linear layer is only connected to non-linear layers and vice versa, we can use a very simple update process scheduling: After each guess or update, we first perform propagation on all non-linear layers (with updated bits), then on all linear layers (with updated bits). This process is iterated until the propagation process has converged and no additional explicit information can be learned anymore, or a contradiction was detected.

3.3 Search for linear characteristics

In this section, we discuss our proposed search strategy. The search strategy guides the guessing behavior (choice of bits or bit sets to guess, and their values), as well as the backtracking behavior after detecting contradictions. We currently implement a simple stack-based search algorithm, similar to the strategy used in recent tools for differential characteristic search [16,17]. We first give an algorithmic overview, before detailing the choices made for individual ingredients.

Basic search algorithm. We start from a mostly-undetermined characteristic A_0 as a starting point, and incrementally guess more and more of its mask bits. We refer to the current characteristic as A, and keep a history of the guesses that led from A_0 to A in the stack S. For each guess, we select a guessable item X in the current characteristic A. Depending on the search strategy, this can be a single bit, or all bits of an S-box input-output mask (unlike some tools for differential characteristic search which only consider individual bits). The choice of X is guided by the search and backtracking strategy. The characteristics stored in S are used for backtracking, where some of the most recent guesses are undone to resolve conflicts, i.e., we return to an older status stored in S. The basic search algorithm is summarized in Algorithm 1.

Algorithm 1 Guess-and-determine search algorithm

```
choose characteristic $A_0$ as starting point
loop
    push $A_0$ to empty stack $S$
    repeat
        get the topmost characteristic $A$ from $S$
        select a guessable item (bit or S-box) $X$ in $A$
        for all most preferable possible values $x$ of $X$ do
            guess $X$ to $x$
            propagate information
            if contradiction detected then
                undo guess $x$ and all resulting updates
            else
                push $A$ to $S$ and break
        if no valid assignment $x$ was found then
            backtrack by popping characteristics from $S$
            mark $X$ critical
    until exhausted or solution characteristic found
```

Choice of the starting point. The starting point is a linear characteristic, in which most mask bits are still undetermined. The appearance of the starting point depends highly on the scenario in which the linear characteristic will be used, since it can be used to define which bits of the resulting characteristic must definitely be active or inactive.

For instance, consider the search for a linear characteristic for a block cipher or a permutation. In principle, every bit of the input or the respective output mask can be active in such a scenario. So, we can use a starting point where nearly every bit of the respective input and output linear masks is free for guessing during the search. On the other hand, if we consider sponge-like modes, we have more restrictions on the characteristic. Here, the attacker can only observe or control a fraction of the state on the input and the output. Depending on the actual attack, it can be necessary that bits belonging to unknown parts of the state remain inactive, and that only observable or controllable bits are active.

Besides defining the possible use-cases of the linear characteristic, the choice of the starting point also greatly influences the expected success of the search. By fixing parts of the starting point, it is possible to reduce the search space significantly, and thus accelerate the search to quickly find results that would otherwise be out of range. However, reducing the search space also has the potential to exclude classes of good characteristics. Thus, the starting point is usually not too much restricted at the beginning of the analysis of a certain cipher. Instead, the choice of the starting point is an adaptive process based on the cryptographer's intuition and the cipher's structure, using information from previous searches.

Guessing strategy. The guessing strategy specifies which undetermined bit or S-box is picked next for guessing, and how it will be refined. In S-box-based designs, the search success can profit significantly from guessing in an S-box-oriented manner; that is, by guessing the value of all bits in an S-box input-output mask at once. We refer to this as "guessing the S-box". Even if guesses are made S-box after S-box, the propagation procedure can produce half-guessed S-boxes with some bits fixed and others undetermined. It is also possible to mix S-box-wise and bit-wise guessing.

We refer to an S-box as "guessable" if the linear input and output masks contain at least one remaining undetermined ?-bit, and "fixed" or "not guessable" otherwise. In addition, the search configuration may limit the selection of S-boxes currently eligible for guessing, depending on the guessing progress. The most important example for this is the "critical" status that is assigned to an S-box after a failed attempt to find any valid assignment for this S-box, and assigns top priority to this S-box. Additionally, it can be useful to impose cipher-specific rules; for example, to demand that all S-boxes of the first few rounds must be fixed before we start guessing values in the following rounds.

To guide the guessing procedure, each guessable S-box is assigned a probability for being selected as the next guessing target, for example based on the criteria described above. In addition, all possible assignments for a guessable

S-box are ranked by how promising they are estimated to be for high-bias characteristics. Of course, the primary guess for potentially inactive S-boxes (i.e., only with bits `0` and `?` so far) is to keep them inactive (i.e., all `0`). If this is not possible, the S-box is marked as active. If the selected guessable S-box is already marked active, we rank all possible assignments of the linear masks according to their linear bias and the number of active bits. We pick a random optimally ranked assignment as primary guess. If the following propagation reveals that this assignment is in fact impossible, we try other assignments until no alternative is left, or we have reached a predefined threshold on the number of trials.

Backtracking. If all alternative assignments fail (or a predefined threshold of trials is reached), we need to backtrack. To resolve this conflict, we return to an earlier version of the linear characteristic as stored on the stack S. Again, we try to guess the same critical S-box that caused the conflict. If we cannot resolve the conflict here, we jump further back, until it can be resolved.

Restarts. To better randomize the search process and avoid being "stuck" with a few unhappy first guesses, it is helpful to occasionally restart the complete search. For this purpose, we define a limit of "credits" or resources for one search run. When this limit is exhausted before finding a valid, fully determined characteristic, we clear the stack S and restart from scratch with the starting point A_0. Additionally, the search is also restarted after completing a successful run, with the hope of finding new, better characteristics. If the cryptographer detects promising patterns in the preliminary results, these can serve as a basis for improved starting points for the next run.

4 Application to CAESAR candidates

In this section, we demonstrate the advantages of our tool for linear cryptanalysis by applying it to several first round CAESAR candidates: Keyak, Ascon, ICEPOLE, Prøst, and Minalpher. All the analyzed candidates are permutation-based (rather than based on block ciphers). This is, however, not a constraint of the linear tool, which works just as well for block ciphers, since the typical round-key additions do not influence the linear characteristics. Rather, it is representative of the trend that a significant portion of CAESAR candidates with new, dedicated SPN primitives are permutation-based, since most block-cipher modes employ AES.

For each candidate, we first consider linear characteristics for the (round-reduced) permutation. However, for many modes (in particular for sponges), an attacker cannot influence the complete input to the permutation, or cannot observe its complete output. For this reason, we also investigate characteristics with additional constraints, where parts of the linear masks are fixed beforehand. We define the following three types of linear characteristics:

– **Type I (permutation):** For this type of characteristics, we do not require any additional restrictions regarding the positions of active bits in the linear characteristic. Hence, a characteristic of this type might not be usable in a concrete attack on the duplex-like constructions of Keyak, Ascon, and ICEPOLE. Nevertheless, even for modes where Type-I characteristics allow no direct attacks, they still give insights in the resistance of the cryptographic primitive against linear attacks.
– **Type II (output constrained):** Linear characteristics of this type have the restriction that all active bits at the end of the characteristic have to be "observable". For duplex-like constructions, this means that all active mask bits have to be in the outer (rate) part of the state. Such linear characteristics can be used to create key-stream distinguishers in known-plaintext scenarios for duplex-like constructions, or even to perform key-recovery attacks.
– **Type III (input and output constrained):** In addition to Type-II characteristics, also all active bits of the input have to be in the outer (rate) part of the state. This type of linear characteristic can act as a key-stream distinguisher in known-plaintext scenarios for duplex-like constructions, targeting the encryption of the plaintext. A similar type of linear relations has been used for instance by Minaud [18] to detect linear biases in the key-stream of the CAESAR candidate AEGIS.

We first discuss our approach and our findings for Keyak in more detail, and then briefly present our results for Ascon, ICEPOLE, PRØST, and Minalpher.

4.1 Keyak

Brief description of Keyak. Keyak is a family of authenticated encryption algorithms designed by Bertoni et al. [4] and is one of the 57 submissions to the first round of the CAESAR competition. It is based on the round-reduced Keccak-f permutation and follows the duplex construction [2]. The designers have defined four instances of Keyak; all instances share the same capacity $c = 252$ and use 12 rounds of the Keccak-f permutation, but differ in their state size b and the degree of parallelism p:

– **River Keyak:** $b = 800$, $p = 1$ (serial),
– **Lake Keyak:** $b = 1600$, $p = 1$ (serial),
– **Sea Keyak:** $b = 1600$, $p = 2$ (parallel),
– **Ocean Keyak:** $b = 1600$, $p = 4$, (parallel).

The Keyak *duplex mode.* Fig. 1 sketches the encryption of serial Keyak without associated data: The initialization takes as input the secret key K and public nonce N, and applies the permutation f once. This ensures that one always starts with a random-looking state at the beginning of the encryption of the plaintext. Afterwards, the plaintext is processed by xoring it block-wise to the internal state, separated by invocations of the permutation f. The ciphertext blocks are extracted from the state after adding the plaintext. After all data is processed, the finalization applies the permutation f once more and returns the tag. For more details on KEYAK, including the rules for processing associated data, we refer to the specification [4].

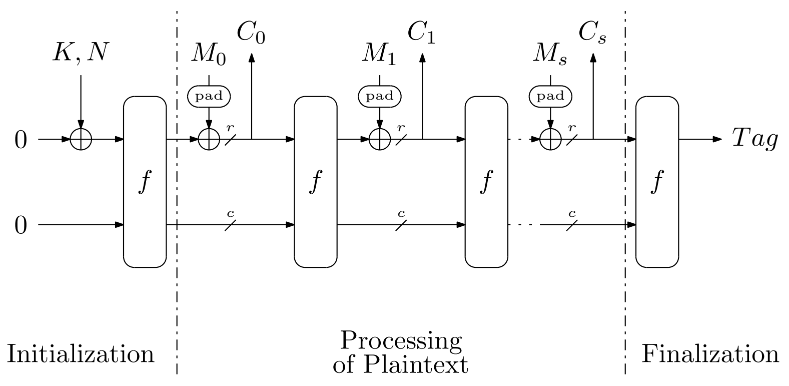

Fig. 1. Simplified sketch of LAKE KEYAK encryption (without associated data).

The KEYAK *permutation.* The KEYAK permutation is a round-reduced version of the KECCAK-f permutation, reduced to 12 rounds. It operates on the $5 \times 5 = 25$ w-bit words ("lanes") $S[x][y][*]$ of the state S, with $w = 32$ or 64. Each round applies the five steps $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$, where all steps except ι are equivalent for each round.

- Step θ adds to every bit of the state $S[x][y][z]$ the bitwise sum of the neighbouring columns $S[x-1][*][z]$ and $S[x+1][*][z-1]$. This procedure can be described by the following equation:

$$\theta: \quad S[x][y][z] \leftarrow S[x][y][z] + \sum_{y'=0}^{4} S[x-1][y'][z] + \sum_{y'=0}^{4} S[x+1][y'][z-1].$$

- Step ρ rotates the bits in every lane by a constant value,

$$\rho: \quad S[x][y][z] \leftarrow S[x][y][z + C(x,y)],$$

  where $C(x,y)$ is a constant value.
- Step π permutes the lanes using the following function:

$$\pi: \quad S[x][y][*] \leftarrow S[x'][y'][*], \qquad \text{where } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix}.$$

- Step χ is the only non-linear step in KECCAK and operates on each row of 5 bits:

$$\chi: \quad S[x][y][z] \leftarrow S[x][y][z] \oplus ((\neg S[x+1][y][z]) \wedge S[x+2][y][z]),$$

  which can be seen as applying a 5-bit S-box in parallel to all rows.

– Step ι adds a round-dependent constant to the state. For the actual values of the constants, we refer to the design document [4].

The designers provide some results on the linear properties of this permutation online, as part of the KeccakTools package [3].

Results for Keyak. For our analysis, we focus on the primary recommendation Lake Keyak using state size $b = 1600$. Since Lake Keyak, in contrast to Ascon and ICEPOLE, uses the same permutation (with the same number of rounds) in the initialization, finalization, and plaintext-processing phase, Type-III characteristics (to target plaintext-processing) offer no remarkable advantage over Type-II characteristics (to target the initialization). For this reason, we only consider Type-I and Type-II characteristics.

Type-I characteristics (for 3 and 4 rounds of the permutation). We first consider Type-I characteristics, i.e., linear characteristics for the underlying round-reduced Keyak permutation (Keccak-f) without any additional restrictions. We performed a search for linear characteristics for 4 rounds of the 1600-bit permutation. The best linear characteristic we found has 33 active S-boxes, which results in a bias of 2^{-34}. The best linear characteristic for 3 rounds with 13 active S-boxes and a bias of 2^{-14} can be obtained by omitting the first round of the 4-round linear characteristic. Our results are very similar to the characteristic given in the KeccakTools package [3].

Type-II characteristics (for 3 and 4 rounds of the initialization). The previous 3 and 4-round characteristics have active bits in the inner part (last four 64-bit words) of the state after round 4. Therefore, we cannot use this characteristic in an actual attack. For an attack on the initialization of round-reduced Keyak, we have to apply additional restrictions on the linear characteristics. Since we can only observe the outer (rate) part of the state at the output of the permutation after the initialization, we apply the restriction that only this part is allowed to contain active bits. Note that the input of the first permutation call is either known or constant. Therefore, we have no problems with active bits there.

For the initialization reduced to 3, or 4 rounds, we found characteristics which only have active S-boxes in the rate part of the state. Thus, considering a known-plaintext attack, we know all the output bits of these S-boxes and can invert them. This leads to the fact that the last round does not influence the bias. So we have an expected bias of 2^{-13} for the 3-round version, and 2^{-49} for the 4-round version of these characteristics. Taking the last S-box layer also into account, the bias of those characteristics would be 2^{-26} and 2^{-70}, respectively. When inverting the last S-boxes, both characteristics result in trivial key-stream distinguishers for round-reduced versions of Keyak with complexity 2^{26} and 2^{98}, respectively. Moreover, these distinguishers could also be used in a key-recovery attack on round-reduced Keyak, resulting in similar complexities.

Configuration of the search. As already mentioned, the proposed search tool is a heuristic one and thus, the quality of the results heavily depends on the applied heuristic search criteria, as well as on the definition of the starting points. For the search process that led to the Type-II characteristics for 3 and 4 rounds of Keyak, we used a quite natural starting point: For both starting points, the only restriction is that the S-boxes of the last round which "belong" to the inner part of the state must be inactive. In addition, one S-box in the second round is marked as active (to exclude the trivial, entirely inactive solution).

We split the search into two stages. In the first stage, we only pick potentially inactive guessable S-boxes, and set them to the best possible assignment (typically a completely inactive input and output linear mask). Which S-box is picked and refined is determined by a heuristic that picks the S-boxes according to a previously configured weight distribution. These weights can be manually assigned in the search configuration file (the same file in which the starting point is defined). In case of the search for the 3-round Type-II characteristic, the weights were assigned so that S-boxes of the first and second round have a 50 times higher chance to be picked compared to an S-box of the last round. The intention behind this distribution is that the majority of the active S-boxes of the resulting linear characteristic should be located in the last round, because their output is known in an attack. Hence, these active S-boxes can be inverted and do not contribute to the bias. Our heuristic for the 4-round Type-II characteristic prefers S-boxes from rounds 2 and 3 over S-boxes from rounds 1 and 4. Additionally, round 1 is favored over the last round 4. In the second stage, after every guessable and potentially inactive S-box is already determined, we continue by guessing active and yet not fully determined S-boxes until the linear characteristic is fully determined.

As can be seen above, the choice of the starting point and search heuristic depend on the structure of the target primitive, the planned use for the linear characteristic, and on the intuition of the cryptographer. Thus, better search strategies and starting points might exist, which may lead to better linear characteristics than those given in this paper.

4.2 Ascon

Brief description of Ascon. Ascon is a family of sponge-based candidates, designed by Dobraunig et al. [9]. Compared to Keyak, it features a significantly smaller state of 320 bits, and the linear layer is applied to each of the 5 64-bit words independently. The 5-bit S-box, on the other hand, is closely related (affine equivalent) to that of Keyak. The primary proposal Ascon-128 has a rate of 64 bits and hence, a capacity of 256 bits.

Results for Ascon. For the linear tool, the simple design of the linear layer means that its linear model can be split into 5 separate, independent matrices. Combined with a small state size, this property greatly reduces the cost for linear algebra needed to perform the propagation compared to Keyak.

Our findings for Ascon are summarized in Table 1. The number of active S-boxes of Type-I characteristics found with the help of this tool have already been included in work presented at CT-RSA 2015 [10]. Note that the characteristics given here are optimized for a minimum number of active S-boxes, rather than minimal bias. For Ascon-128, we additionally search for Type-II and Type-III characteristics. However, regarding Type-III characteristics, no meaningful results were obtained.

4.3 ICEPOLE

Brief description of ICEPOLE. ICEPOLE is a family of authenticated encryption schemes designed by Morawiecki et al. [19]. It consists of the three proposals ICEPOLE-128, ICEPOLE-128a, and ICEPOLE-256, which all use the same underlying 1280-bit permutation. All variants use 12 rounds of the permutation for initialization, and 6 rounds for processing of plaintext and finalization. However, they differ in details like size of the rate, key, nonce and tag.

The 1280-bit state of ICEPOLE is stored in $5 \times 4 = 20$ 64-bit words. For the linear layer, an MDS matrix over $\mathbb{F}_{2^5}$ is first applied 64 times in parallel (to each 20-bit slice of the state). Then, each word is rotated, and the words swap positions. The S-box layer applies 5-bit S-boxes (4 parallel row-wise applications for each 20-bit slice).

ICEPOLE's designers perform no dedicated linear analysis, but compare the cipher's resistance to linear cryptanalysis to its well-studied resistance against differential cryptanalysis. They conclude that the attack complexity after 5–6 rounds should be "completely intractable" [19]. At FSE 2015, Huang et al. [11] presented 3-round linear characteristics that they use in a differential-linear attack on ICEPOLE.

Results for ICEPOLE. The Type-II and Type-III characteristics given in Table 1 are constrained with respect to a capacity of 254 bits (due to padding, 256 bits are not observable), as defined for ICEPOLE-128 and ICEPOLE-128a. In the case of ICEPOLE, we do not have an immediate output of a ciphertext block right after the 12 rounds of the initialization. Before this happens, there is the option to process a secret message number and at least an empty associated data block is processed. Hence, 6 or even another 12 additional rounds have to be passed before an output suitable for our Type-II characteristic is accessible. Thus—in the worst case—our key-stream distinguisher using Type-II characteristics works for round-reduced versions of ICEPOLE-128, where the initialization plus the following processing is reduced to 5 out of 24 rounds with a complexity of about 2^{120}.

Type-III characteristics can be used to create distinguishers that target the processing of the plaintext. Here, every version of ICEPOLE uses the 6 round version of the ICEPOLE permutation. Thus, by using the Type-III characteristic in Table 1, the key-stream produced by round-reduced variants of ICEPOLE-128, where the permutation used in the plaintext processing is reduced to 4 (out of

6), rounds can be distinguished from a perfect randomly generated key-stream with a complexity of about 2^{88}. The bias of the 5-round Type-III characteristic is $2^{-87.08}$ and hence, the complexity of a resulting key-stream distinguisher cannot harm the 128-bit security of ICEPOLE-128. ICEPOLE-256a, on the other hand, claims a security level of 256 bits regarding the confidentiality. However, it has a higher capacity of 318 bits and therefore, the characteristics given in Table 1 cannot be used directly. Taking the higher capacity of ICEPOLE-256a into account, we get a Type-III characteristic with a bias of $2^{-89.49}$, which can be used to distinguish the key-stream of a round-reduced variant of ICEPOLE-256a, where the permutation used during the encryption is reduced to 5 (out of 6 rounds). Note that ICEPOLE-256a limits the number of blocks encrypted under a single key by 2^{62}. However, this type of key-stream distinguishers exploit relations between ciphertext block C_i and the key-stream used to generate the following ciphertext block C_{i+1}. Thus, distinguishers using Type-III characteristics in this way do not rely on the fact that always the same key is used.

Table 1 contains the results with the best bias, but not necessarily the minimal number of active S-boxes we found. For example, for 6 rounds, we also found a Type-I characteristic with only 103 active S-boxes, but a bias of $2^{-133.49}$ (compared to 104 active S-boxes with bias $2^{-126.32}$ as in the table).

4.4 Minalpher

Brief description of Minalpher. Minalpher is designed by Sasaki et al. [22]. In contrast to the previous 3 candidates, Minalpher is no sponge-based design. Instead, the permutation is applied in a new tweakable block cipher construction, called tweakable Even-Mansour. For this construction, the permutation size only needs to be twice the security level, so for 128-bit security, Minalpher has the smallest of all investigated permutation sizes with only 256 bits. This small state is further divided into two halves, whose only interaction in each of the 17.5 rounds is that one half is once xored to the other half, and the two halves swap places. Besides the interaction between the halves and some nibble reordering, the linear layer features a near-MDS matrix multiplication over $\mathbb{F}_{2^4}$. The S-box size of 4 bits is also nibble-oriented.

For Minalpher's construction, only Type-I characteristics are useful. We understand our results as an analysis of the underlying permutation. However, since Minalpher claims security in nonce misuse settings and under unverified plaintext release, the Type-I characteristics could also be useful for attacks on the cipher. In particular, for a fixed nonce, the construction allows to control the entire permutation input (at least differentially, due to the Even-Mansour construction, which xors a key- and nonce-dependent value before and after the permutation) and observe the entire output (again, differentially).

The designers analyze the minimum number of active S-boxes (for differential cryptanalysis) theoretically, and prove a minimum number of 22 S-boxes for 4 rounds. For up to 7 rounds, they extend the bounds with the help of mixed integer linear programming (MILP). The bounds obtained this way for the numbers of rounds r also covered by this work are 22 ($r = 4$), 41 ($r = 5$), and 58 ($r = 6$). The designers claim that the same bounds apply for linear cryptanalysis.

Results for Minalpher. The existing bounds serve as a kind of benchmark for our tool to check its capabilities. As shown in Table 1, we were able to match the given bounds for up to 6 rounds. For better comparability, we included our results with the minimal number of active S-boxes, but not necessarily the best bias, in the table. For example, for 6 rounds, we also found a Type-I characteristic with a smaller bias of 2^{-61}, but with 60 active S-boxes (compared to 58 active S-boxes with bias 2^{-62} in the table).

4.5 Prøst

Brief description of Prøst. PRØST, designed by Kavun et al. [13], offers both a sponge-based mode and two block-cipher-based modes, where the latter use the PRØST permutation in a single-key Even-Mansour construction. Each of the three modes offers two security levels: one based on the 256-bit PRØST-128 permutation, and one based on the 512-bit PRØST-256 permutation. The state is stored as 4×4 words of 16 or 32 bits, respectively. Both the 4-bit S-box (row-wise) and the 16-bit linear mixing function (MDS over $\mathbb{F}_{2^4}$ are applied in a bit-sliced way (using 1 bit of each word). Then, each word is rotated. The number of rounds per permutation call is $r = 16$ (PRØST-128) or $r = 18$ (PRØST-256), respectively.

We focus our analysis on PRØST-256 (formally offering 128-bit security). Like Minalpher, PRØST comes with a MILP-based proof for the minimum number of active S-boxes for differential and linear characteristics. For PRØST-256, the bounds for different round numbers are 25 ($r = 4$), 41 ($r = 5$), 105 ($r = 6$), and 169 ($r = 7$).

Results for Prøst. Again, we used the existing bounds as benchmarks for our linear tool. The tool is able to match each bound, mostly with optimal or near-optimal bias (2^{-26} for $r = 4$, 2^{-42} for $r = 5$, 2^{-107} for $r = 6$, and 2^{-175} for $r = 7$).

Table 1. Results for Keyak, Ascon, ICEPOLE, Minalpher, and PRØST. The corresponding linear characteristics can be found in the full version of this paper.

Cipher	Type	Rounds	Active S-boxes	Bias
Keyak	I	3	13	2^{-14}
		4	33	2^{-34}
	II	3*	12	2^{-13}
		4*	43	2^{-49}
Ascon	I	3	13	2^{-15}
		4	43	2^{-50}
		5	67	2^{-94}
	II	2	6	2^{-8}
		3	23	2^{-30}
		4	61	2^{-83}
ICEPOLE	I	5	38	$2^{-55.08}$
		6	104	$2^{-126.32}$
	II	4	22	$2^{-30.42}$
		5	38	$2^{-59.49}$
	III	3	10	$2^{-16.66}$
		4	22	$2^{-43.25}$
		5	42	$2^{-87.08}$
Minalpher	I	4	22	2^{-23}
		5	41	2^{-42}
		6	58	2^{-62}
PRØST-256	I	4	25	2^{-26}
		5	41	2^{-42}
		6	105	2^{-107}
		7	169	2^{-175}

* Last S-box layer inverted.

5 Conclusion

We presented a dedicated tool for the automatic linear cryptanalysis of substitution-permutation networks. The goal of the tool is to identify linear characteristics for a cryptographic function, which can subsequently be used by the cryptanalyst to mount key-recovery or distinguishing attacks. The heuristic search is based on an efficient guess-and-determine approach, which has previously been proven successful for searching differential characteristics. We described how to perform efficient propagation of linear masks in linear and non-linear building blocks of a cipher.

From the cryptanalyst's perspective, the tool is simple to use, flexible, and easy to extend with regard to search strategies and target ciphers. The open-source tool will be freely available to help analyze CAESAR candidates and other symmetric cryptographic primitives. We hope that our work will be a valuable contribution to get a better understanding of the security of these ciphers regarding linear cryptanalysis. In particular, we hope to encourage experiments with alternative, sophisticated search strategies optimized for different target ciphers.

We demonstrated the efficiency of our tool by applying it to several CAESAR candidates. The results obtained by searching for linear characteristics for the Minalpher and PRØST-256 permutation show that the presented heuristic search tool can keep pace with MILP-based approaches. However, due to the heuristic nature, we are not capable of providing bounds on the minimum number of active S-boxes.

On the other side, when looking at the results obtained for ASCON, ICEPOLE and KEYAK– all designs with weak alignment – we have been able to find new linear characteristics with a good bias that might be used in a key-recovery or distinguishing attack on round-reduced versions of the ciphers in the future. One highlight are the Type-III characteristics for round-reduced versions of ICEPOLE, which can be used to distinguish the key-stream of ICEPOLE in a nonce-respecting scenario.

Our results show that the existence of a publicly available analysis tool for linear characteristics will be of great help in the design of symmetric cryptographic primitives, in order to evaluate the resistance against linear attacks already in an early stage of the design. Thus, we think that this tool will facilitate new designs which are more balanced in their resistance against linear and differential attacks than some of today's designs.

Acknowledgments. The work has been supported in part by the Austrian Science Fund (project P26494-N15) and by the Austrian Research Promotion Agency (FFG) and the Styrian Business Promotion Agency (SFG) under grant number 836628 (SeCoS).

References

1. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On alignment in Keccak. `http://keccak.noekeon.org/KeccakAlignment.pdf`
2. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) Selected Areas in Cryptography – SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer (2011)
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: KECCAKTOOLS software. `http://keccak.noekeon.org/` (2014)
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: Keyak. Submission to the CAESAR competition: `http://competitions.cr.yp.to/round1/keyakv1.pdf` (2014)
5. Biryukov, A., Velichkov, V.: Automatic search for differential trails in ARX ciphers. In: Benaloh, J. (ed.) Topics in Cryptology – CT-RSA 2014. LNCS, vol. 8366, pp. 227–250. Springer (2014)

6. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Linearization framework for collision attacks: Application to CubeHash and MD6. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. LNCS, vol. 5912, pp. 560–577. Springer (2009)
7. Daemen, J., Rijmen, V.: AES and the wide trail design strategy. In: Knudsen, L.R. (ed.) Advances in Cryptology – EUROCRYPT 2002. LNCS, vol. 2332, pp. 108–109. Springer (2002)
8. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: General results and applications. In: Lai, X., Chen, K. (eds.) Advances in Cryptology – ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer (2006)
9. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon. Submission to the CAESAR competition: `http://competitions.cr.yp.to/round1/asconv1.pdf` (2014)
10. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Cryptanalysis of Ascon. In: Nyberg, K. (ed.) Topics in Cryptology – CT-RSA 2015. LNCS, vol. 9048, pp. 371–387. Springer (2015)
11. Huang, T., Tjuawinata, I., Wu, H.: Differential-linear cryptanalysis of ICEPOLE. In: Leander, G. (ed.) Fast Software Encryption – FSE 2015. LNCS, vol. 9054, pp. 243–263. Springer (2015)
12. Indesteege, S., Preneel, B.: Practical collisions for EnRUPT. In: Dunkelman, O. (ed.) Fast Software Encryption – FSE 2009. LNCS, vol. 5665, pp. 246–259. Springer (2009)
13. Kavun, E.B., Lauridsen, M.M., Leander, G., Rechberger, C., Schwabe, P., Yalçın, T.: Prøst. Submission to the CAESAR competition: `http://competitions.cr.yp.to/round1/proestv11.pdf` (2014)
14. Leurent, G.: Construction of differential characteristics in ARX designs: Application to Skein. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. LNCS, vol. 8042, pp. 241–258. Springer (2013)
15. Matsui, M.: Linear cryptoanalysis method for DES cipher. In: Helleseth, T. (ed.) Advances in Cryptology – EUROCRYPT '93. LNCS, vol. 765, pp. 386–397. Springer (1993)
16. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 characteristics: Searching through a minefield of contradictions. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology – ASIACRYPT 2011. LNCS, vol. 7073, pp. 288–307. Springer (2011)
17. Mendel, F., Nad, T., Schläffer, M.: Improving local collisions: New attacks on reduced SHA-256. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. LNCS, vol. 7881, pp. 262–278. Springer (2013)
18. Minaud, B.: Linear biases in AEGIS keystream. In: Joux, A., Youssef, A.M. (eds.) Selected Areas in Cryptography – SAC 2014. LNCS, vol. 8781, pp. 290–305. Springer (2014)
19. Morawiecki, P., Gaj, K., Homsirikamol, E., Matusiewicz, K., Pieprzyk, J., Rogawski, M., Srebrny, M., Wójcik, M.: ICEPOLE. Submission to the CAESAR competition: `http://competitions.cr.yp.to/round1/icepolev1.pdf` (2014)
20. Mouha, N., Preneel, B.: Towards finding optimal differential characteristics for ARX: Application to Salsa20. IACR Cryptology ePrint Archive, Report 2013/328 (2013), `http://eprint.iacr.org/2013/328`
21. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C., Yung, M., Lin, D. (eds.) Information Security and Cryptology – Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer (2011)

22. Sasaki, Y., Todo, Y., Aoki, K., Naito, Y., Sugawara, T., Murakami, Y., Matsui, M., Hirose, S.: Minalpher. Submission to the CAESAR competition: `http://competitions.cr.yp.to/round1/minalpherv1.pdf` (2014)
23. Schläffer, M., Oswald, E.: Searching for differential paths in MD4. In: Robshaw, M.J.B. (ed.) Fast Software Encryption – FSE 2006. LNCS, vol. 4047, pp. 242–261. Springer (2006)
24. Sun, S., Hu, L., Wang, M., Wang, P., Qiao, K., Ma, X., Shi, D., Song, L.: Automatic enumeration of (related-key) differential and linear characteristics with predefined properties and its applications. IACR Cryptology ePrint Archive, Report 2014/747 (2014), `http://eprint.iacr.org/2014/747`
25. Tardy-Corfdir, A., Gilbert, H.: A known plaintext attack of FEAL-4 and FEAL-6. In: Feigenbaum, J. (ed.) Advances in Cryptology – CRYPTO '91. LNCS, vol. 576, pp. 172–181. Springer (1991)
26. The CAESAR committee: CAESAR: Competition for authenticated encryption: Security, applicability, and robustness (2014), `http://competitions.cr.yp.to/caesar.html`