AutoDiVer: Automatically Verifying Differential Characteristics and Learning Key Conditions

Marcel Nageler[1], Shibam Ghosh[2,3], Marlene Jüttler[1] and Maria Eichlseder[1]

[1] Graz University of Technology, Graz, Austria
marcel.nageler@tugraz.at, marlene.juettler@student.tugraz.at,
maria.eichlseder@tugraz.at
[2] University of Haifa, Haifa, Israel
[3] Inria, Paris, France
shibam.ghosh@inria.fr

Abstract. Differential cryptanalysis is one of the main methods of cryptanalysis and has been applied to a wide range of ciphers. While it is very successful, it also relies on certain assumptions that do not necessarily hold in practice. One of these is the hypothesis of stochastic equivalence, which states that the probability of a differential characteristic behaves similarly for all keys. Several works have demonstrated examples where this hypothesis is violated, impacting the attack complexity and sometimes even invalidating the investigated prior attacks. Nevertheless, the hypothesis is still typically taken for granted. In this work, we propose AutoDiVer, an automatic tool that allows to thoroughly verify differential characteristics. First, the tool supports calculating the expected probability of differential characteristics while considering the key schedule of the cipher. Second, the tool supports estimating the size of the space of keys for which the characteristic permits valid pairs, and deducing conditions for these keys. AutoDiVer implements a custom SAT modeling approach and takes advantage of a combination of features of advanced SAT solvers, including approximate model counting and clause learning. To show applicability to many different kinds of block ciphers like strongly aligned, weakly aligned, and ARX ciphers, we apply AutoDiVer to GIFT, PRESENT, RECTANGLE, SKINNY, Midori, WARP, SPECK, and SPEEDY.[1]

Keywords: Differential cryptanalysis · Hypothesis of stochastic equivalence · Tool · SAT solver · GIFT · SKINNY · Midori · WARP · SPECK · SPEEDY

1 Introduction

Differential cryptanalysis is one of the main cryptanalytic techniques for block ciphers and other symmetric primitives [BS90, BS91]. It has been applied successfully to a range of targets. Differential cryptanalysis focuses on differences between two inputs rather than individual values. The reason behind this is that in many deterministic computations, differential propagation can be constructed even without knowing the value of the key. The main idea is to encrypt two similar messages and predict the intermediate differences and the output differences of the primitive. Crucially, we do not need to analyze the whole cipher at once. Instead, we use differentials for each component, i.e., predictions of how differences propagate through the component, and then stitch them together to form a differential characteristic. The probabilities for all the differentials can be multiplied to estimate the probability of the differential characteristic. However, this estimate might be inaccurate due to several assumptions it implicitly makes.

[1] This version contains fixed results for WARP and fixes a mistake in a characteristic for SPECK-48.

First, we rely on the *Dominant Trail Assumption* [LMM91]. It states that the probability of the differential characteristic is a good approximation for the probability of the differential. For the differential, we only care about observing the output difference given that the input difference is satisfied. Hence, its differential probability may be higher than that of the differential characteristic, which fixes all intermediate differences.

Additionally, we assume the analyzed cipher to be a *Markov cipher* [LMM91]. It states that the probability of a difference propagation through a cipher round is independent of the value of the round input if the round keys are chosen uniformly at random. For ciphers with a partial key addition, like SKINNY and GIFT, this is not the case. For Markov ciphers, one can calculate the expected probability of a differential characteristic by multiplying the probabilities of the individual rounds. However, this product of probabilities only states the expected probability for uniformly distributed and independent round keys. In a differential attack, we care only about the probability for the attacked key.

Therefore, we rely on the *Hypothesis of Stochastic Equivalence* [LMM91], which states that the expected probability over uniform round keys is approximately equal to the probability for a given sequence of round keys for almost all sequences of round keys. However, as we discuss in the following, substantial evidence in the existing literature suggests that this assumption may not hold in practice. Differential characteristics can be dependent on the specific values of the key. Therefore, verifying key dependencies in relation to a differential characteristic is of paramount importance. This necessity forms the core motivation of our work.

Related Work. The limitations of these assumptions in the differential cryptanalysis of different designs have been studied in several works.

Daemen and Rijmen examine the distribution of differential characteristic probability across different keys [DR07]. Introducing the concept of *plateau characteristics*, they describe a scenario where the probability is zero for a subset of keys and a fixed nonzero value for the other keys. By defining planar differentials and employing affine spaces, they demonstrate that for AES, all 2-round characteristics exhibit plateau behavior.

Leurent studies differential attacks on ARX constructions [Leu12]. He further generalizes the notion of generalized differences by De Canniere and Rechberger [CR06] by considering conditions on multiple bits, and proposes a tool to derive these multi-bit conditions and show inconsistencies. He applies these multi-bit conditions on differential characteristics for the hash function Skein, and points out problems in several of them.

Canteaut et al. [CLN+17] model necessary conditions for a pair following a differential characteristic in an unkeyed setting using affine spaces. By intersecting these spaces, they derive improved probability estimates for differentials in unkeyed Feistel or SPN ciphers.

Ankele and Kölbl [AK18] analyze the difference between differential characteristic probability and differential probability. Using SMT solvers, they enumerate many differential characteristics compatible with a differential to provide a better probability estimate. Additionally, they measure the differential probability for various keys experimentally.

Sun et al. [SWW18] analyze how differential characteristics of Midori64 interact with its key schedule. By modeling necessary conditions for a pair that follows the differential characteristic with affine subspaces, they find necessary conditions on the key. Furthermore, they enumerate all the solutions of characteristics for one STEP function of LED [GPPR11] to calculate the differential characteristic probability.

Liu et al. [LZS+20] propose using linear systems of equations to model the solution sets for differential characteristics of block ciphers with planar S-boxes. They combine this model with a linear system that describes the key schedule. In case the key schedule is partially non-linear, the linear system only covers the linear part. With this model they show that some characteristics they find are valid for at least one key of AES-128 but invalid for AES-192. Additionally, they show an invalid characteristic for Midori128 and

two characteristics for PRINCE where no key exists that leads to both characteristics being possible. Furthermore, they show a sufficient condition based for when the hypothesis of stochastic equivalence holds for a differential characteristic.

Liu et al. [LIM20] propose a method to find differential characteristics for the Gimli permutation that are not impossible. They propose to model the difference and value transitions simultaneously in one Mixed-Integer Linear Programming model. Therefore, a solution to this model includes a conforming pair in addition to a differential characteristic.

Beyne and Rijmen [BR22a] analyze the probability of differential characteristics in a fixed-key setting. They propose the theory of quasidifferential trails which track probabilistic linear equations on the values satisfying a differential characteristic. Based on these trails, the probability of a differential characteristic can be calculated as the sum of all compatible quasidifferential trails. They also provide a tool to find quasidifferential trails for several ciphers.

Peyrin and Tan [PT22] analyze the key dependencies arising from differential characteristics in GIFT and SKINNY. Their work provides algorithms for finding linear and nonlinear key dependencies (where this distinction primarily refers to the cause, not the representation, of conditions [Sun24]). The idea of their algorithm is modeling degrees of freedom, where constraints on cells are tracked through the linear layer. We note that their tool is challenging to adapt to other ciphers, especially for weakly aligned ciphers.

In summary, different approaches for the detection, representation, and analysis of key dependencies and related aspects in differential cryptanalysis have been proposed. Nevertheless, the study often remains limited to dedicated papers, while new cryptanalysis papers rarely take these effects and dependencies into account. This is at least in part due to the lack of easily useable, broadly applicable tools that can directly answer questions such as "For how many and for which keys is this characteristic valid? What is the impact of the key schedule on the expected probability?". Furthermore, many of the available approaches have different limitations in terms of the type of key conditions they can potentially identify.

Our Contribution. In this paper, we propose AutoDiVer, a precise, versatile, and usable tool designed to verify the probability of differential characteristics and identify key dependencies. Our approach is based on modeling the set of valid pairs and their associated keys for a given characteristic as a Boolean satisfiability (SAT) problem in conjunctive normal form (CNF). AutoDiVer then leverages the capabilities of advanced SAT solvers and model counters to provide the following functionalities:

- Estimate the probability of a differential characteristic averaged over all keys while considering the key schedule.
- Estimate the probability of a differential characteristic for a fixed key or for an unkeyed permutation.
- Estimate and bound the number of keys for which there exist valid pairs for the characteristic, also while considering the key schedule.
- Derive necessary linear and nonlinear conditions on the key such that valid pairs for the characteristic exist.

Some of the tool's functionalities yield deterministic results and upper bounds, while others lead to statistical estimates. For estimating the probability of a differential characteristic, our tool gives *probably approximately correct* estimates. These are within a certain range of the true value (based on a tolerance parameter ε) with a certain probability (based on a confidence parameter δ). We provide two statistical estimates for the number of keys that permit valid pairs. One of them gives similar probably approximately correct

guarantees, while the other is based on sampling keys and returns an interval based on a chosen confidence parameter. To complement these statistical estimates, we implement methods to derive linear and nonlinear conditions on keys, whose correctness is derived by a SAT solver. These conditions then lead to concrete upper bounds on the set of keys that permit valid pairs.

AutoDiVer is precise in the sense that we model the actual cipher, including the full details of the S-box and key schedule. This increased precision allows us to better capture the behavior of the differential characteristic. To ensure our formulas are correct, we compare the results we get by solving the CNF to the cipher reference implementations.

Finally, AutoDiVer provides good usability, as adding a new cipher is simple: one creates a subclass of our `SboxCipher` class, gives the S-box as a table and describes the linear layer and key schedules with XORS constraints and CNF clauses. We demonstrate the applications of AutoDiVer in the SPN-based ciphers GIFT-64, GIFT-128 [BPP+17], PRESENT [BKL+07], Midori64, Midori128 [BBI+15], SKINNY-64, SKINNY-128 [BJK+16], RECTANGLE [ZBL+15] and SPEEDY [LMMR21]. Along with SPN-based constructions, we also apply our tool to the Generalized Feistel-based construction WARP [BBI+20] and the ARX cipher SPECK [BSS+13, BSS+15]. A summary of our results is provided in Table 1 (Section 4). Furthermore, our tool produces clear output to minimize the need for manual post-processing. AutoDiVer is available on GitHub[2].

To estimate the probability of differential characteristics, we use approximate model counting [CMV13]. Approximate model counting gives a probably approximately correct estimate for the number of solutions of a CNF formula, with a confidence parameter δ and tolerance parameter ε. We apply projected model counting, a variant of model counting, to estimate the size of the set of valid keys. Additionally, we experimentally test whether some random key samples belong to the set of valid keys to estimate its size.

These statistical solver-based approaches introduce interesting advantages and limitations. They are often successfully applicable to characteristics where other experimental approaches fail due to the low probability of the characteristic, while still taking the complete specification including the key schedule into account. On the downside, having only statistical estimates with no easily verifiable evidence or interpretation is often unsatisfactory. The correctness of the results relies both on the correctness of the solver implementation (which is generally a reasonable assumption) and on the parameters δ, ε. To address these limitations of the purely statistical approach, we implement multiple approaches for learning key conditions to describe the set of valid keys. These necessary conditions provide concrete bounds for the number of valid keys.

As observed in previous work, most key conditions can be expressed as affine constraints. In our approach, we consider the affine hull of the set of valid keys. We derive candidate constraints on the affine hull, which are then confirmed using the SAT solver. This approach will identify all affine conditions on the set of valid keys and provide an upper bound on its size. In many cases, this upper bound appears to be tight as it closely matches our other methods for estimating the size of the set of valid keys. To fill any remaining gap between the estimate and bound, we also implement an approach to learn nonlinear conditions, again taking advantage of SAT solver features. Specifically, we use the clauses that the SAT solver learns during the statistical evaluation to deduce conditions on the key.

By combining the method for finding the affine hull of the set of valid keys with the learned CNF clauses about the key, we can give a good description of the set of valid keys in many cases. Thereby, we complement existing techniques that can be used to prove conditions on the key, such as constraint-propagation-based techniques [Leu12, PT22], quasidifferential trails [BR22a], and techniques based on linear systems of equations [DR07, CLN+17, SWW18, LZS+20]. Naturally, constraints on the key can be

[2] https://github.com/isec-tugraz/AutoDiVer

expressed and explained in multiple ways. In this paper, we will explain some constraints with value-based arguments (as used internally by our tool) as well as quasidifferential trails (for a succinct summary), to better show the connections between the different representations. Similarly, all other key conditions we list can also be represented using a handful of quasidifferential trails.

As an example, for a differential characteristic of Midori64, we find key dependencies that lead to a weak-key space of size 2^{111}, while the characteristic is impossible for all other keys. We list 17 linear constraints that describe this weak-key set. Furthermore, we can verify the expected differential probability of the same Midori64 characteristic to be 2^{-52} in just 1.5 seconds. Similarly, we find key dependencies for certain differential characteristics of all our analyzed ciphers.

We note that, since AutoDiVer is based on SAT solvers and model counters, it is sometimes hard to predict the runtime. In particular, our methods based on model counting become less efficient with more rounds. Furthermore, our tool only analyzes one differential characteristic at a time, so we still rely on the dominant trail assumption.

We believe AutoDiVer to be useful for a wide audience. Cryptanalysts can easily verify their differential characteristics and potentially improve them, while designers can use it to test various parameters of their block cipher, particularly different key schedules.

Outline. In Section 2, we recall the necessary background on differential cryptanalysis, SAT solvers, and model counters. In Section 3, we describe our method for verifying the probability and describing the set of valid keys. In Section 4, we summarize our results. In Section 5, we compare our tool to related work. We conclude in Section 6.

2 Background

In this section, we recall the necessary background. In Subsection 2.1, we recall differential cryptanalysis. In Subsection 2.2, we give an example of a key dependency on a toy cipher. In Subsection 2.3, we outline Boolean satisfiability (SAT) problems and algorithms to solve them. In Subsection 2.4, we recall the related problem of model counting and outline an algorithm for approximate model counting.

2.1 Differential Cryptanalysis

Differential cryptanalysis is a powerful statistical approach for attacking block ciphers, introduced by Biham and Shamir in 1990 [BS90]. It is a chosen plaintext attack. The idea is to look at differences between pairs of plaintext inputs rather than studying individual values. The rationale behind this is that the propagation of these differences can be predicted, even without knowing the value of the secret key used in the encryption process.

The attack starts by selecting two plaintext messages, denoted as x and x', that are processed by the same function f. Let $\Delta_x = x \oplus x'$ and $\Delta_y = f(x) \oplus f(x')$. The event where the difference Δ_x propagates to Δ_y, denoted as $\Delta_x \to \Delta_y$, is called a *differential* transition of f. The attacker's goal is to compute the probability of this event, i.e., $Pr(\Delta_x \to \Delta_y)$, and to find differences Δ_x and Δ_y with high differential probability.

For linear functions, the probability is always 1 or 0, however; for non-linear functions, it can be lower. The probability can be computed as the ratio of the number of inputs that produce the desired differential transition to the total number of inputs. For a function $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$, the probability can be computed as [NK92]:

$$Pr(\Delta_x \to \Delta_y) = \frac{\#\{x \in \mathbb{F}_2^n : f(x) \oplus f(x \oplus \Delta_x) = \Delta_y\}}{2^n}.$$

In order to apply differential cryptanalysis to an iterated block cipher, the attacker usually focuses on the *differential characteristics* through the round functions. Let us consider an iterated block cipher $\mathsf{E} = \mathsf{E}_{r-1} \circ \cdots \circ \mathsf{E}_0$. Let us denote input to the i-th *round* E_i as x_i and output as y_i. The differential characteristic is a series of differences that show the propagation of a difference through the round functions of an iterated cipher. An r-round differential characteristic with a series of intermediate differences $(\alpha_0, \alpha_1, \ldots, \alpha_n)$ is denoted as $\alpha_0 \xrightarrow{\mathsf{E}_0} \alpha_1 \xrightarrow{\mathsf{E}_1} \ldots \xrightarrow{\mathsf{E}_{r-1}} \alpha_r$, where α_0 is the difference in the input plaintexts and α_r is the difference between the outputs after r rounds. The attacker searches for differential characteristics that propagate with high probability through multiple rounds of the cipher, providing a means to recover the secret key. Thus, the attacker is interested in the conditional probability of a differential characteristic

$$\Pr(\alpha_0 \xrightarrow{\mathsf{E}_0} \alpha_1 \xrightarrow{\mathsf{E}_1} \ldots \xrightarrow{\mathsf{E}_{r-1}} \alpha_r) = \Pr[\Delta_{y_r} = \alpha_r, \Delta_{y_{r-1}} = \alpha_{r-1}, \ldots, \Delta_{y_1} = \alpha_1 | \Delta_{y_0} = \alpha_0].$$

The probability is taken over all choices of plaintext and key. The computation of this conditional probability is not straightforward, as the probability of $\alpha_i = \Delta_{y_i}$ depends on all the previous values. However, for a *Markov Cipher* [LMM91], this can be calculated from the one-round characteristics that compose the r-round cipher. For a Markov Cipher, the probability of a one-round differential, taken over all keys and a specific input value, is independent of the specific input value. Therefore, for a Markov cipher with uniformly random round keys, then the probability of an r-round characteristic can be computed as the product of the probabilities of each one-round characteristic. Specifically,

$$\Pr(\alpha_0 \xrightarrow{\mathsf{E}_0} \alpha_1 \xrightarrow{\mathsf{E}_1} \ldots \xrightarrow{\mathsf{E}_{r-1}} \alpha_r) = \prod_{i=1}^{r} \Pr[\Delta_{y_i} = \alpha_i | \Delta_{y_{i_1}} = \alpha_{i-1}].$$

Previously, we discussed the concept of a *differential* for any function, which measures how the output of a function changes in response to a certain input change. The probability of any differential from input difference α to output difference β can be computed by taking the sum over the probabilities of all possible characteristics of the form $\alpha_0 \xrightarrow{\mathsf{E}_0} \alpha_1 \xrightarrow{\mathsf{E}_1} \ldots \xrightarrow{\mathsf{E}_{r-1}} \alpha_r$ where $\alpha = \alpha_0$ and $\beta = \alpha_r$. Thus, if the function is an iterated Markov cipher then the probability of any differential $(\alpha \to \beta)$ can be computed as

$$\Pr(\alpha \to \beta) = \sum_{\alpha_1} \sum_{\alpha_2} \cdots \sum_{\alpha_{r-1}} \prod_{i=1}^{r} \Pr[\Delta_{y_i} = \alpha_i | \Delta_{y_{i_1}} = \alpha_{i-1}]\,.$$

However, the attacker needs to consider the average probabilities over all possible keys to calculate this probability, which is computationally expensive. Note that all the plaintext-ciphertext pairs that an attacker collects are usually encrypted with a fixed and unknown key. So, we assume that the probabilities are equivalent for almost all keys. This assumption is called the *hypothesis of stochastic equivalence* [LMM91], which states that

$$\begin{aligned}&\Pr[\Delta_{y_r} = \alpha_r, \Delta_{y_{r-1}} = \alpha_{r-1}, ..., \Delta_{y_1} = \alpha_1 | \Delta_{y_0} = \alpha_0, K = k] \\ &\approx \Pr[\Delta_{y_r} = \alpha_r, \Delta_{y_{r-1}} = \alpha_{r-1}, ..., \Delta_{y_1} = \alpha_1 | \Delta_{y_0} = \alpha_0],\end{aligned}$$

where K is a random variable for the key. However, it is important to note that this assumption is not always true; it is a common simplifying assumption made in cryptanalysis that usually, but not always, holds. For instance, the designers of Rijndael in [DR00] mentioned that block ciphers can have weak keys, which result in some differential characteristics having very high probabilities while others have very low probabilities. In [DR07], the authors proposed the concept of *plateau characteristics*, where the probability for a fixed key is either zero or a single nonzero value. It was demonstrated that for AES,

as well as many other block ciphers, all two-round differential characteristics are plateau characteristics. Furthermore, in the case of AES, most of the four-round characteristics are plateau characteristics. Thus, if an attacker suspects that the stochastic equivalence assumption does not hold for a specific cipher or key, then more complex techniques, such as experimental verification, are needed. However, for differentials with low probability, performing experimental verification is often infeasible. This challenge serves as the primary motivation for our work. In addition to the existing literature we have discussed, which suggests that differential characteristics can depend on specific key values, we present the following simple example to explore this issue.

2.2 Example of a Key Dependency

To illustrate how a key dependency can arise, we give an example in Figure 1. In this figure, we consider a toy cipher that uses the PRESENT S-box and a small version of PRESENT's bit permutation. To discuss the main issue, we consider a differential characteristic, with probability 2^{-10} when averaged over all keys. However, if the desired differential transition in the rightmost S-box in the first round happens, then the least significant output bit is always 0. Similarly, the least significant input bit of the rightmost S-box in the second round must be 0 for the transition to happen. Therefore, the key bit $K_{1,0}$, that connects the output and input bits, must be 0. Otherwise, the characteristic is impossible (i.e., the probability is 0). If the key bit is 0, the probability doubles to 2^{-9} as the least significant input bit in the last round always has the right value. Thus, the probabilities are not equivalent for all keys, i.e., the hypothesis of stochastic equivalence does not hold.

Based on this example, we use the term independent probability as in [PT22], which states the probability of the differential characteristic when only the set of valid keys, i.e., the set of keys which lead to non-zero probability, is considered. In this example, the independent probability would be doubled to 2^{-9}. Note that even within the set of valid keys, there might be variations of the probability depending on the exact value of the key.

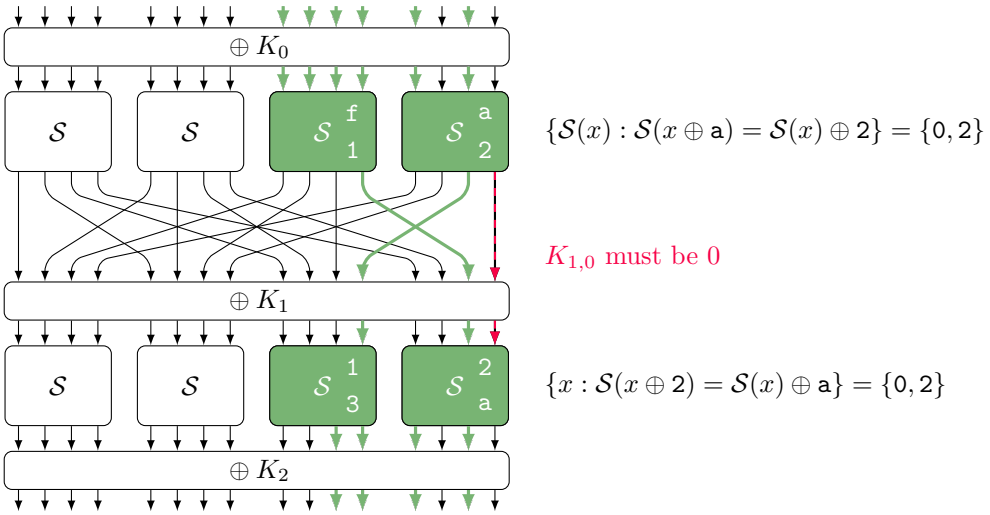


Figure 1: Key dependency in a differential characteristic of a toy cipher.

This example can also be explained in the framework of quasidifferential trails [BR22a]. A quasidifferential trail is a pair of a differential characteristic and associated linear masks. The linear masks specify linear equations over the valid pairs for the differential characteristic with a certain correlation. The total correlation is then calculated like in linear cryptanalysis, but for each S-box only the valid pairs for the differential transition are

considered. To show a contradiction, we consider two quasidifferential trails of maximum absolute correlation [BR22a, Theorem 4.2]. For both of them the differential part is given by our differential characteristic in Figure 1. The first one is the trivial one, where the linear part consists of only zero masks. The correlation of this trivial quasidifferential trail always equals the differential characteristic probability, i.e., 2^{-10}. The second quasidifferential trail has non-zero linear masks as indicated by the red/black dashed line in Figure 1. From before, we know that this linear approximation of the involved bits is always satisfied for valid pairs. Due to the key addition, this quasidifferential trail has correlation $(-1)^{K_{1,0}} \cdot 2^{-10}$. This implies that for $K_{1,0} = 1$, the characteristic is impossible [BR22a, Theorem 4.2]. Note that in the general case, one would consult the quasidifferential transition matrix to find the correlation.

2.3 SAT Solvers

Boolean satisfiability (SAT) solving is one of two major constraint programming paradigms used in the automated cryptanalysis of symmetric primitives. SAT solvers solve decision problems expressed using Boolean constraints in conjunctive normal form (CNF). Several popular generalizations are summarized under the term Satisfiability Modulo Theories (SMT) and provide more expressive languages for the constraints, such as bitvector operands or integer arithmetic. Given a problem, the solvers return either a solution satisfying all constraints or `UNSAT`, if such a solution does not exist. More advanced solvers additionally support features such as enumerating solutions. Internally, most solvers implement (extensions of) the Davis-Putnam-Logemann-Loveland (DPLL) algorithm for backtracking-based search [DLL62], particularly Conflict-Driven Clause Learning (CDCL) [SS96]. In CDCL, the solver derives new constraints learned during the search procedure; some solvers can also return these learned clauses via their user interface.

SAT solvers have been used for cryptanalysis for more than two decades [MM00, Mas99, SNC09], originally for executing straightforward key-recovery or preimage attacks based on given data. For this purpose, the cryptographic circuit can be translated into CNF with a generic transformation such as the Tseitin transformation [Tse70] and Boolean variables for the values of intermediate results, with the data of one or a few single plaintext-ciphertext pairs added as additional constraints.

In the last ten years, they have become more popular for finding distinguishers, particularly differential and linear distinguishers [MP13, Köl14, SHW+14]. Here, the Boolean variables instead represent differences/linear masks, and the constraints encode the propagation rules of differences/linear masks, either on a truncated level (e.g., for cell-wise activity patterns to derive bounds on the number of active S-boxes) or on a precise bitwise level based on the DDT or LAT (e.g., when searching for fully specified characteristics for an attack). Challenges include the modeling of large S-boxes [AST+17, BC20, SW23], efficient integer counters for bounding the probability of characteristics [SWW21a, EME22], modeling clustering and the differential effect [AK18], efficient problem partitioning and parallelization due to the single-threaded nature of most SAT solvers [EME22], combining characteristic search with solution search [SWW18], and more.

2.4 Model Counting

Model counting, also known as #SAT, is the problem of counting the number of solutions $\#F$ to a formula F in conjunctive normal form (CNF). Instead of enumerating all solutions, a number of dedicated model counters exist, most of which compete in the regular occurring model counting competitions [FHH21, KJ21, LM17, LMY21].

A feature of many model counters is the ability to perform *projected model counting*, where an additional input, the sampling set $\mathcal{V}$, is provided. Now, instead of counting all

solutions, the number of unique assignments to the variables in the sampling set such that the formula remains satisfiable is counted. We denote this projected count as $\#_{\mathcal{V}}F$.

To make these model counters more scalable, so called approximate model counters exist. We use ApproxMC [CMV13] which gives probabilistic guarantees: For a formula F with $\#F$ solutions, the following holds for the approximate count c:

$$\frac{1}{1+\varepsilon} \cdot \#F \leq c \leq (1+\varepsilon) \cdot \#F \qquad \text{with probability } p \geq 1-\delta,$$

where ε and δ parameterize the tolerance and confidence, respectively. The lower these values are, the better the approximation. However, better approximations also require more computational resources.

The main idea of ApproxMC is to divide the solutions space of the formula into many small cells by using a (noncryptographic) hash function and then counting the exact number of solutions for a subset of these cells. Assume F is a formula over n variables with $\#F$ solutions. To partition F into cells of approximate size $\#F \cdot 2^{-m}$, we pick masks $(a_1, a_2, \ldots a_m)$ uniformly at random from $\{0,1\}^n$. This defines a hash function $h(y) = h_1(y) \parallel h_2(y) \parallel \ldots \parallel h_m(y)$ where $h_i(y) = \bigoplus_{j=0}^{n} a_{i,j} \cdot y_j$. Each cell then corresponds to the subset of solutions where $h(y)$ takes a specific value $\alpha \in \{0,1\}^m$ denoted as $F \wedge h(y) = \alpha$. Thus, we arrive at a first estimate:

$$\#F \approx 2^m \cdot \#(F \wedge h(y) = \alpha)\,.$$

To provide the necessary guarantees, we calculate this estimate multiple times for non-empty cells of a certain size. To pick the number of XORs m, we initialize $m = 0$ and increase it until the cell size is less than a given threshold $t_{\#}$ based on the tolerance parameter ε, where $t_{\#} \in \mathcal{O}\left(1/\varepsilon^2\right)$. The iteration count t_{it} is based on the confidence parameter δ, where $t_{\text{it}} \in \mathcal{O}\left(\log_2\left(1/\delta\right)\right)$. Finally, we return the median of all t_{it} estimates.

Note that this explanation only covers the basic ideas as proposed at CP 2013 [CMV13]. Since then, a number of improvements have been implemented [SM19, SGM20, YM23]. For a full specification, we refer to these papers and the source code of ApproxMC[3].

3 SAT-Based Verification of Differential Characteristics

Now, we outline our approach for verifying differential characteristics. In Subsection 3.1, we show how to encode the set of valid pairs for a differential characteristic and associated keys in conjunctive normal form (CNF). In Subsection 3.2, we explain how to use that encoding to calculate the precise probability for that differential characteristic. In Subsection 3.3, we show how to estimate for how many keys this differential characteristic permits valid pairs. In Subsection 3.4, we explain how to obtain necessary conditions such that a key permits valid pairs for a certain differential characteristic. In Subsection 3.6, we give some implementation details of AutoDiVer, and outline how it can be extended for new ciphers.

3.1 Encoding valid pairs as a CNF

Given a differential characteristic, we aim to find a set of pairs that conform to the specified differential characteristic. To achieve this, we represent this set in conjunctive normal form (CNF). Essentially, we construct a CNF in which every solution is representative of the cipher execution adhering to the differential characteristic.

A naive approach involves encoding the execution of the cipher twice and adding constraints to ensure that, at each step, the differences follow the specified differential

[3] https://github.com/meelgroup/approxmc

characteristic. Such an approach is used in the existing literature [MZ06, SRB21]. Constraints are then created for all S-boxes and the linear layers in accordance with the cipher specification. The characteristic is then encoded by specifying that certain variables are equal or not equal. However, this encoding has a significant drawback: it requires specifying the cipher twice, resulting in twice as many variables and more than double the number of constraints.

Here, we use a different approach from the existing literature [NPE23] that is based on 3 key observations:

1. **Variable elimination:** We can eliminate all the variables associated with the second execution. Each variable in the second execution can be expressed as either identical to or the negation of its corresponding variable in the first execution, depending on the differential characteristic.

2. **Single encoding of the linear layer:** The linear layer needs to be encoded only once because the propagation of differences through the linear layer is deterministic.

3. **Optimized S-box encoding:** Instead of encoding each S-box twice (once for the normal variables and once for the variables with some bits flipped), we can directly analyze the solution set for the S-box $\{(x, y) : \mathcal{S}(x) = y \wedge \mathcal{S}(x \oplus \alpha) = y \oplus \beta\}$. Concretely, we create a dedicated small CNF using a logic minimizer like espresso [BHMS84] for a Boolean function f such that $f(x, y) = 1$ if and only if $\mathcal{S}(x) = y \wedge \mathcal{S}(x \oplus \alpha) = y \oplus \beta$.

To make our approach more comprehensive, we give an algorithmic description in Algorithm 1. In line 8, we use espresso to generate a small CNF that encodes the S-box constraints. In line 12, we record the constraints as Xor constraints. When passing this CNF to a solver that supports Xor constraints, such as CryptoMiniSAT, we pass the Xor constraints unmodified; otherwise, we convert them into standard CNF constraints.

Algorithm 1: Encoding valid pairs as a CNF formula.

Data: A differential characteristic $\alpha_0 \xrightarrow{\mathcal{R}} \alpha_1 \xrightarrow{\mathcal{R}} \dots \alpha_r$
Result: CNF Formula F that encodes the valid pairs and associated keys
1 **for** $j = 0$ *to* r **do**
2 Create variables for S-box inputs $x_{j,i}$ and outputs $y_{j,i}$ and the round keys $K_{j,i}$
3 Calculate output differences β_j of the S-box layer based on α_{j+1}
4 **end**
5 $F \leftarrow F \wedge$ constraints for the key schedule
6 **for** *every S-box* $y_{r,i} = \mathcal{S}(x_{r,i})$ **do**
7 Let $\alpha_{r,i}$ and $\beta_{r,i}$ denote the input and output differences of the S-box.
8 $F \leftarrow F \wedge (x_{r,i}, y_{r,i}) \in \{x, y : \mathcal{S}(x) = y \wedge \mathcal{S}(x \oplus \alpha_{r,i}) = y \oplus \beta_{r,i}\}$
9 **end**
10 Let $\mathfrak{c}_r$ denote the round constants
11 **for** *every linear layer* $x_{r+1} = \mathcal{L}(x_r, K_r, \mathfrak{c}_r)$ **do**
12 $F \leftarrow F \wedge x_{r+1} = \mathcal{L}(x_r, K_r, \mathfrak{c}_r)$
13 **end**
14 **return** F

Finally, when creating the CNF, it is crucial to ensure a one-to-one correspondence between solutions and valid pairs for the differential characteristic. For instance, if there was an unused variable with no constraints, each valid pair would correspond to two solutions. This would hinder our ability to count the number of solutions accurately to determine the number of valid pairs. Therefore, we take care not to introduce any unused variables into our model.

By solving this model with a SAT solver, we can verify whether the differential characteristic is possible. If the solver returns UNSAT, we know it is impossible. In the following, we propose various advanced applications of our model.

Remark. The applications of AutoDiVer, that we discuss below, are easily extendable to the *tweakey* framework. In this framework, we consider the tweakey instead of the key, as will be elaborated later for the SKINNY block cipher in Section 4.

3.2 Verifying the Probability

Often, differential characteristics have such low probabilities that verifying them experimentally is infeasible. To verify the probability of such low-probability characteristics, we use our CNF encoding by counting the number of valid pairs and keys. It is important to note that each solution of our CNF corresponds to a plaintext and key. Therefore, we can calculate the number of solutions as follows:

$$\Pr(\alpha_0 \xrightarrow{\mathcal{R}} \alpha_1 \xrightarrow{\mathcal{R}} \dots \alpha_r) = \frac{\#F}{2^{n+k}} ,$$

where F denotes the encoding of valid pairs, n denotes the block size and k denotes the key size in bits.

This method enables us to determine the exact probability of the differential characteristic, averaged over all 2^k keys of the block cipher. One advantage compared to evaluating the differential characteristic using the DDT is that our approach takes the key schedule into account. Of course, when we approximate $\#F$ using ApproxMC, we remain subject to its probabilistic (ε, δ) guarantees.

Having discussed the exact probability averaged over all keys of the block cipher, we now address the case of a fixed key. To calculate the differential characteristic probability for a specific key, we can add constraints to fix the key to a specific value and then avoid dividing by the key size, i.e.,

$$\Pr(\alpha_0 \xrightarrow{\mathcal{R}_{K_0}} \alpha_1 \xrightarrow{\mathcal{R}_{K_1}} \dots \alpha_r) = \frac{\#(F \wedge \mathcal{K} = K)}{2^n} ,$$

where $\mathcal{K}$ denotes the variables for the key and K denotes a fixed value.

3.3 Verifying the Number of Valid Keys

As a second application of our model, we discuss how to determine the size of the set of valid keys that satisfy a given differential characteristic. Some differential characteristics only permit a valid pair for a specific subset of keys, which we refer to as $\mathbb{K}^\star$. In extreme cases, the characteristic might not permit valid pairs for any key, meaning $\mathbb{K}^\star = \emptyset$. This phenomenon has been noted multiple times in the literature [AK18, SWW18, PT22, BR22a].

When the set of valid keys is non-empty, the differential characteristic can be leveraged in a weak key attack. In such cases, the probability of the differential characteristic is zero for most keys but significantly amplified for the set of valid keys. Therefore, it is important to estimate the size of this set and describe its elements. We propose two ways to do this using AutoDiVer:

1. One way to estimate the size of the set of valid keys is to use projected model counting. We set the sampling set to the key variables and count:

   $$\|\mathbb{K}^\star\| = \#_{\mathcal{K}} F ,$$

   where $\mathcal{K}$ denotes the variables for the key. Again, we use ApproxMC to approximate this count. This gives us a flexible way to estimate the size of the set of valid keys.

2. We can approximate the size of the set of valid keys by sampling many random keys and using a SAT solver to verify whether a valid pair exists for that key. If we perform this experiment for, say, t random samples and t_{sat} turn out to be satisfiable, we can approximate the size of the set of valid keys as

$$\|\mathbb{K}^\star\| \approx 2^k \cdot \frac{t_{\text{sat}}}{t}\,.$$

   To quantify the certainty of this estimate, we use the Wilson score interval [Wil27]. This interval defines an upper and lower limit of the true probability $\frac{\|\mathbb{K}^\star\|}{2^k}$ based on the observations t, t_{sat} with a certain confidence.

   One way to implement this approach is to add clauses to the CNF to constrain the key. However, we would leave some performance on the table, as the SAT solver would start a fresh search for each CNF. Instead, we do not modify the formula F and pass the random key as *assumptions* to CryptoMiniSAT. Assumptions allow us to fix certain variables (the key variables in our case) to a value without modifying the CNF. This way, the clauses the SAT solver learns for one key will be reused for subsequent keys.

   We can improve this process by instructing the solver to systematically search for clauses over the assumption variables, i.e., the key. To do so, whenever the solver returns UNSAT, we call the `find_conflict` method of CryptoMiniSAT. This method then returns a learned clause over the assumptions variables that excludes the current assignment of assumption variables. We find that explicitly calling `find_conflict` gives much better results than iterating over all learned clauses. After we have identified enough clauses, we minimize them using espresso [BHMS84] to get a better representation.

One important benefit of the second approach is that we can use the learned clauses to learn information about the set of valid keys $\mathbb{K}^\star$. The idea is to iterate over all learned clauses and look for clauses where all variables are bits of the key. Since the SAT solver only learns clauses that are implied by the original formula, these clauses serve as necessary conditions that a key belongs to the set of valid keys $\mathbb{K}^\star$. This approach works well for GIFT, where the linear layer and key schedule are simple. However, for ciphers with more complex key schedules we find that many clauses are necessary to describe the key dependency. In particular, XORs involving many variables can only be described using an amount of CNF clauses that is exponential in the number of variables. Therefore, we need a more systematic way to find linear conditions on the key, which we describe next.

3.4 Describing the Set of Valid Keys

Here, we present a systematic approach to describe the set of valid keys corresponding to a given characteristic. This method is based on the observation that in the existing literature most conditions on the key are affine conditions [DR07, CLN+17, SWW18, LZS+20, BR22a, PT22]. This usually occurs because most S-boxes are planar, i.e., the set of inputs/outputs for a given differential transition form an affine space. Still, SKINNY-128 also shows affine conditions [PT22] but uses a non-planar S-box.

In the existing literature such affine key conditions are often identified by modeling the solution sets of active S-boxes, the linear layer, and sometimes the key schedule as a system of linear equations. Here, we present a new method that also captures the effect of inactive S-boxes, by modeling the solution sets as a CNF (including the connection between input and output value). Note that the connection between input and output has been captured in different ways in the existing literature as well [BR22a, PT22].

We aim to describe the affine hull of the set of valid keys, namely $\text{aff}(\mathbb{K}^\star)$. Since $\mathbb{K}^\star \subseteq \text{aff}(\mathbb{K}^\star)$, if a key lies outside of $\text{aff}(\mathbb{K}^\star)$ it is also outside of $\mathbb{K}^\star$. Thus, we have a

necessary condition for a key to be valid. Furthermore, in Section 4, we will see that in many cases the affine hull of valid keys and the set of valid keys are nearly identical:

$$\mathrm{aff}(\mathbb{K}^\star) \approx \mathbb{K}^\star\,.$$

To construct the affine hull, we find $k+1$ solutions for our encoding of valid pairs F, where k denotes the key size in bits. We store these solutions in the set $\mathbb{K}_{\mathrm{sol}}$. To ensure these $k+1$ solutions are actually different, we take the following two steps:

1. Instruct the SAT solver to select the polarity of the variables it uses for branching decisions at random.
2. Use a different random seed every time.

From these $k+1$ solutions, we can extract $k+1$ values for the key which span an affine hull $\mathrm{aff}(\mathbb{K}_{\mathrm{sol}})$ of dimension $d \leq k$. This is definitely a subset of the affine hull of valid keys $\mathrm{aff}(\mathbb{K}^\star)$. But how do we know whether they are equal?

Affine hull as a set of linear equations. To answer that question, we need to restructure our affine hull $\mathrm{aff}(\mathbb{K}_{\mathrm{sol}})$ as a set of linear equations. We start with an arbitrary element $K^0 \in \mathbb{K}_{\mathrm{sol}}$ to serve as a constant offset. Then, we apply Gaussian reduction to $K^0 \oplus \mathbb{K}_{\mathrm{sol}}$ to receive a $k \times d$ basis matrix B. With this basis, we can represent any key $K \in \mathrm{aff}(\mathbb{K}_{\mathrm{sol}})$ as $K = K^0 + B \cdot v$ for some $v \in \mathbb{F}_2^d$. Let us consider $A = \ker(B)$ with $A \in \mathbb{F}_2^{(k-d)\times k}$ and $c = A \cdot K^0$ with $c \in \mathbb{F}_2^{k-d}$. Because $\ker(B) \cdot B = 0$, after multiplying with A, we get

$$A \cdot K = c\,.$$

This gives us a linear-equation-based representation of the affine hull $\mathrm{aff}(\mathbb{K}_{\mathrm{sol}})$. Crucially, the dimension d of our affine hull is likely close to k, so this way, we get a much shorter representation than using the basis matrix.

Finding a counterexample. With this linear-equation-based representation for the affine hull, we can use our SAT solver to solve the problem $F \wedge A \cdot \mathcal{K} \neq c$, where $\mathcal{K}$ denotes the SAT variables used for the key. If this new problem is satisfiable, we retrieve the key and add it to our basis B and start over. Otherwise, if it is unsatisfiable, we have proved that no valid key exists outside of the affine hull, i.e., $\mathrm{aff}(\mathbb{K}_{\mathrm{sol}}) = \mathrm{aff}(\mathbb{K}^\star)$. This also implies that the conditions $A \cdot K = c$ are necessary conditions for a valid key.

Combining the techniques. While many key dependencies for differential characteristics can be described using linear equations, nonlinear key dependencies do sometimes exist. Therefore, we combine the technique to find the affine hull of the set of valid keys with the SAT-based experimental verification from Subsection 3.3. Concretely, instead of sampling uniformly from the full key space, we sample uniformly from the affine hull of valid keys. This way we can verify whether there is a significant number of invalid keys within the affine hull of valid keys. Furthermore, if there are invalid keys, the clauses returned by the SAT solver when we call `get_conflict` will describe the nonlinear key conditions.

Impact on probability. With the technique from Subsection 3.2, we can verify the expected differential probability p, i.e. the probability of the differential characteristic when averaged over all 2^k keys of the block cipher. If we find that the probability is nonzero only for a set of 2^{k-x} keys (for example, if we find x linear constraints on the key), then we can conclude that the independent probability, i.e. the probability averaged over the 2^{k-x} valid keys of the block cipher, is $p \cdot 2^x$. The independent probability predicts the data complexity of a weak key attack.

3.5 Explaining Contradictions

In some cases a differential characteristic may be impossible. With a CNF encoding of the solution set, we can quickly identify such impossible characteristics as the SAT solver returns UNSAT. However, to find better characteristics, we would like to learn why the characteristic is impossible.

Now, we augment our tool to automatically identify the source(s) of impossible characteristics. To do so, we first add extra *assumption* variables $s_{r,i}$ for all (active and inactive) S-boxes. We use this assumption variable to selectively toggle the constraints for this S-box on or off. We achieve this by modeling a logical implication:

$$s_{r,i} \to (x_{r,i}, y_{r,i}) \in \{x, y : \mathcal{S}(x) = y \wedge \mathcal{S}(x \oplus \alpha_{r,i}) = y \oplus \beta_{r,i}\},$$

where $x_{r,i}$ and $y_{r,i}$ denote the input and output values and $\alpha_{r,i}$ and $\beta_{r,i}$ denote the input and output difference. In conjunctive normal form this implication can be modeled in a straightforward way, by appending $\vee \neg s_{r,i}$ to each clause of the existing S-box model. Now, if we constrain the new variables $s_{r,i}$ to be true, then this model is equivalent to the existing one.

To learn which S-boxes cause the contradiction, we pass all $s_{r,i}$ variables as assumptions to the SAT solver. Remember that assumptions allow us to fix certain variables without modifying the CNF. Additionally, if the solving process returns UNSAT, we can call `get_conflict` to learn a clause over the assumption variables. This learned clause then lists the S-boxes that cause a contradiction.

We can apply the same idea to explain which S-boxes are responsible for which affine conditions on the key. To do so, we first find the linear conditions on the key as explained in Subsection 3.4. Then, for each identified constraint, we add the inverse to our model, making it unsatisfiable, and use the method outlined above to find the relevant S-boxes. Similarly, we also apply this method to explain nonlinear constraints by explaining each clause separately. We also apply this method to ARX ciphers, where we add an assumption variable for each full adder, i.e., n assumption variables for each n-bit addition.

To illustrate the method, we apply it to the toy cipher example (Figure 1, page 7). First, we define the variables $s_{r,i}$ with $r \in \{0, 1\}$ and $i \in \{0, 1, 2, 3\}$ where $i = 0$ denotes the rightmost S-box. Then, we apply the method from Subsection 3.4 (with $s_{r,i}$ as assumption variables) to find that $K_{1,0} = 0$. Next, we add the constraint that $K_{1,0} = 1$ making the problem UNSAT and the call to `get_conflict` returns $\neg s_{0,0} \vee \neg s_{1,0}$ indicating that one of the two rightmost S-boxes must be removed for the model to be satisfiable. Hence, we conclude that the two rightmost S-boxes cause the conflict.

3.6 Implementation of the Approach

The methods outlined above are all implemented in a Python library. We use CryptoMiniSAT as a SAT solver and ApproxMC as an approximate model counter. Furthermore, we use espresso to minimize the formulas for the S-boxes.

We ensure that our cipher descriptions match the actual ciphers as specified by using an extensive suite of automated tests. For each cipher, we test our CNF descriptions against a reference implementation. Furthermore, we also test that the solutions for a given characteristic match that characteristic.

Our library can be readily extended to include more ciphers. To implement a new cipher, one has to describe the linear layer and the key schedule in terms of CNF clauses and/or XOR constraints. Note that describing the S-boxes is handled automatically, based on a table description of the S-box.

4 Results

We apply AutoDiVer to the ciphers GIFT-64, GIFT-128, SKINNY-64, SKINNY-128, Midori64, Midori128, RECTANGLE, WARP, SPEEDY, and SPECK. Note that this list includes SPN ciphers, generalized Feistel ciphers, strongly aligned ciphers, and weakly aligned ciphers. We particularly focused on reproducing results from related work to verify the applicability of the tool but also provide new results not included in prior work. This diverse set of ciphers highlights the broad applicability of AutoDiVer.

We list our results for estimating the size of the set of valid keys in Table 1. 'EDP' lists the expected differential characteristic probability calculated using the DDT. Key set size denotes the size of the set of keys that permits valid pairs for the differential characteristic calculated using various methods: 'AMC' denotes the estimate using projected approximate model counting with ApproxMC with $\delta = 0.2$ and $\varepsilon = 0.8$. 'Exp. est.' denotes the estimate based on picking many keys independently uniformly at random and verifying whether valid pairs exist; the range shows the 80% confidence interval. The last columns summarize the number of key conditions in our and prior work: 'LC' denotes the number of linear conditions found with AutoDiVer, with * denoting cases where we found additional nonlinear constraints. This gives an upper bound on the size of the set of valid keys. 'RW' lists the number of conditions, expressed as the key space reduction measured in bits (i.e., n corresponds to a key space reduction by a factor of 2^{-n}).

We find that the most efficient method is to calculate the affine hull. Furthermore, we note that in many cases the upper bound given by the affine hull matches very well with the experimental estimate of the key size. In these cases, the linear constraints discovered by AutoDiVer provide a very accurate description of the set of valid keys.

When applying approximate model counting, we always use the default parameters of $\delta = 0.2$ and $\varepsilon = 0.8$. This guarantees, that with 80% probability the result is within a factor of $2^{\pm 0.85}$. However, we observe that even with these parameters the results match very closely with our experimental verification. That is, the results from approximate model counting seem to be more accurate than the guarantees given by the proof.

When applying our method for verifying the probability of a differential characteristic, we find that the runtime is only acceptable for Midori64, where we are able to verify a characteristic of probability of 2^{-54} in only 1.5 seconds. Furthermore, we verify the probabilities of reduced variants of GIFT and WARP characteristics. For GIFT, we can verify characteristics with probabilities of about 2^{-40}, while for WARP, we can verify characteristics with probabilities as low as 2^{-78}.

4.1 Results for GIFT

GIFT is a family of SPN-based block ciphers proposed by Banik et al. at CHES 2017 [BPP+17]. We give a specification in Appendix A.1.

Results. We present our findings on the size of the set of valid keys for various characteristics of GIFT from existing literature in Table 1. For GIFT, we find a key dependency in most characteristics. The affine hull also serves as a very good upper bound for the size of the set of valid keys.

Our results for GIFT mostly match those published by Peyrin and Tan [PT22]. However, for a characteristic of GIFT-128 [ZDY19, Tab. 14], we find an additional linear key constraint compared to Peyrin and Tan. This additional linear constraint also matches the experimental estimate. Furthermore, we manually analyze the involved S-boxes and confirm the key constraints.

For the characteristics from [LWZZ19, Table 7 and Table 8], we observe additional effects that increase the independent probability as listed in [PT22, Table 3]. Because these

Table 1: Our results for estimated key set size that permits valid pairs for the differential characteristic. EDP: expected differential probability for characteristic. AMC: approximate model counting with $\delta = 0.2$, $\varepsilon = 0.8$. Experimental estimate denotes the 80% confidence interval. LC: number of linear conditions on the key. RW: key space reduction in related work in bits (red: new constraints on the key). ⊘: impossible characteristic. ⋆: extra nonlinear constraints found. $: independent round keys (as opposed to actual key schedule). †: mistake in characteristic fixed.

Cipher	Characteristic	#R	EDP	Key set size			
				AMC	exp. est.	LC	RW
GIFT-64	[LWZZ19, Tab. 2]	9	2^{-42}	$2^{124.00}$	$[3.92, 4.08]$	4	4
	[LWZZ19, Tab. 7]	12	2^{-58}	$2^{124.00}$	$[3.96, 4.12]$	4	4
	[LWZZ19, Tab. 8]	13	2^{-62}		$[3.93, 4.09]$	4	4
	[SWW21b, Tab. 8.1]	13	2^{-64}			1	1
	[SWW21b, Tab. 8.2]	13	2^{-64}		$[4.79, 4.97]$	5	5
	[SWW21b, Tab. 8.3]	13	2^{-64}			3	3
	[ZDY19, Tab. 4]	12	2^{-59}		$[2.92, 3.03]$	3	3
	[ZDY19, Tab. 6]	12	2^{-60}			0	0
GIFT-128	[LLL+21, Tab. 5]	12	$2^{-60.4}$	$2^{127.00}$	$[0.96, 1.00]$	1	1
	[ZDY19, Tab. 13]	12	$2^{-62.4}$	$2^{127.00}$	$[0.97, 1.01]$	1	1
	[LLL+21, Tab. 6]	13	$2^{-67.8}$	$2^{127.00}$	$[0.97, 1.01]$	1	1
	[ZDY19, Tab. 14]	14	$2^{-85.0}$	$2^{125.00}$	$[2.92, 3.03]$	3	2
	[ZDY19, Tab. 10]	18	2^{-109}	0	–	⊘	⊘
PRESENT-80	[Wan08, Tab. 5]	4	2^{-18}		$[0.00, 0, 00]$	0	–
	[Wan08, Tab. 7]	14	2^{-62}			0	–
RECTANGLE$	[ZBL+14, App. E]	14	2^{-63}			1	1
	[ZBL+14, App. E]	14	2^{-66}			2	2
SKINNY-64	[DDH+21, Tab. 9]	15	2^{-54}	$2^{185.81}$	$[6.11, 6.48]$	5⋆	6.2
SKINNY-128	[DDH+21, Tab. 12†]	17	2^{-110}	$2^{376.67}$	$[7.39, 7.98]$	6⋆	–
Midori64	[ZHWW20, Tab. 5]	5	2^{-52}	$2^{111.00}$	$[17.1, 18.0]$	17	–
Midori128	[TAY16, Tab. 3]	11	2^{-123}		$[5.11, 5.29]$	4⋆	–
Midori128	[CXTQ23, Tab. 7]	10	2^{-115}			4	–
WARP	[KY21, Tab. 8]	18	2^{-122}			4	–
	[KY21, Tab. 9]	18	2^{-122}			7	–
SPECK-32$	[BR22b, Tab. 18.1]	6	2^{-13}	$2^{96.00}$	$[0.00, 0.00]$	0	0
	[BR22b, Tab. 18.2]	7	2^{-18}	$2^{112.00}$	$[0.00, 0.00]$	0	0
	[BR22b, Tab. 19.1]	8	2^{-24}	$2^{126.00}$	$[1.96, 2.16]$	2	2
	[BR22b, Tab. 19.2]	8	2^{-27}	$2^{125.00}$	$[2.87, 3.18]$	3	3
	[BR22b, Tab. 20.1]	9	2^{-30}	$2^{142.93}$	$[0.98, 1.10]$	1⋆	1
	[BR22b, Tab. 20.2]	9	2^{-33}	$2^{141.55}$	$[2.49, 2.75]$	1⋆	1
	[BR22b, Tab. 20.3]	9	2^{-33}	$2^{141.29}$	$[2.40, 2.65]$	2⋆	2
SPECK-48$	[ALLW14, Tab. 7†]	10	2^{-40}	$2^{240.00}$	$[0.00, 0.00]$	0	–
SPECK-64$	[BR22b, Tab. 22.2]	15	2^{-62}			3	3
SPECK-96$	[BR22b, Tab. 22.3]	15	2^{-81}			6	6
SPECK-128$	[BR22b, Tab. 23.1]	20	2^{-128}			5	5
	[BR22b, Tab. 23.2]	20	2^{-128}			7	7
	[BR22b, Tab. 23.3]	20	2^{-128}			5	5
	[BR22b, Tab. 23.4]	20	2^{-128}			3	3
SPEEDY-192	[BDBN23, Fig. 4]	5.5	2^{-171}	0	–	⊘	⊘

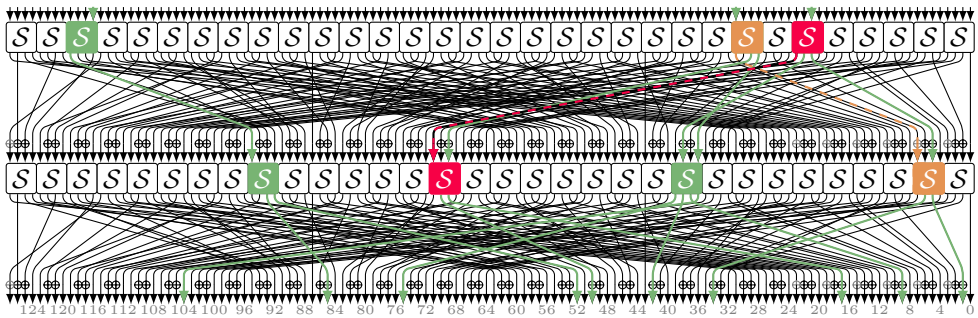
are based on a 4-round iterative characteristic and due to the simple key schedule of GIFT-64, we find that certain key constraints are created twice. We now analyze the 13-round characteristic [LWZZ19, Table 8]. By using our method for explaining key constraints, we find that the constraint $K_{8,1} \oplus K_{10,1} = 0$ is created due to $(\mathcal{S}_{1,4}, \mathcal{S}_{1,6}, \mathcal{S}_{2,1}, \mathcal{S}_{2,9})$ and due to $(\mathcal{S}_{9,4}, \mathcal{S}_{9,6}, \mathcal{S}_{10,1}, \mathcal{S}_{10,9})$, where $\mathcal{S}_{r,i}$ denotes the S-box i in round r and both r and i start from 0. Similarly, the constraint $K_{24,0} \oplus K_{26,0} = 0$, is created due to $(\mathcal{S}_{3,0}, \mathcal{S}_{3,2}, \mathcal{S}_{4,0}, \mathcal{S}_{4,8})$ and due to $(\mathcal{S}_{11,0}, \mathcal{S}_{11,2}, \mathcal{S}_{12,0}, \mathcal{S}_{12,8})$. Therefore, we conclude that if these two key constraints are met, the overall probability of the characteristic becomes 2^4 times as likely. Combined with the other 2 linear constraints, the independent probability is 2^6 higher than the probability as calculated by the DDT. The analysis for the 12-round characteristic [LWZZ19, Table 7] is very similar as it is a prefix of the 13-round characteristic. Here, the constraint generated by the S-boxes in the last round does not exist. Therefore, the independent probability is increased by a factor of 2^5 compared to the probability calculated by the DDT.

We notice that our results for GIFT-128 cover more rounds than for GIFT-64. While the 18-round result is an outlier as the contradiction is due to a local effect, we show results on 14 rounds of GIFT-128 compared to 13 rounds for GIFT-64. We conjecture this is because of the increased degrees of freedom available when solving GIFT-128 models. That is, for GIFT-64 the probability of the characteristics is very close to the generic probability of 2^{-64}, while for GIFT-128, the margin is larger.

As Peyrin and Tan, we find that the 18-round characteristic by Zhu et al. [ZDY19, Tab. 10] is impossible. By applying our tool we find that the contradiction occurs in round 16–17 as illustrated in Figure 2. The relevant S-boxes in round 16 and 17 both have differential transitions of $\mathtt{2} \to \mathtt{6}$. The most significant bit (MSB) at the output of the relevant S-box in round 16 connects to the MSB at the input of the relevant S-box in the next round without key addition.

To explain the contradiction, we provide a value-based explanation and the relevant quasidifferential trails. The solution set for the S-boxes highlighted in red in Figure 2 is $\{(x, \mathcal{S}(x)) : \mathcal{S}(x \oplus \mathtt{2}) = \mathcal{S}(x) \oplus \mathtt{6}\} = \{(\mathtt{1}, \mathtt{a}), (\mathtt{3}, \mathtt{c}), (\mathtt{5}, \mathtt{f}), (\mathtt{7}, \mathtt{9})\}$. We see that at the output the MSB is always 1, while at the input it is always 0. Since they are connected without a key addition, the characteristic is impossible.

A very similar effect happens at the S-boxes highlighted in yellow. We identified these S-boxes by modeling a GIFT-128 variant with a full key addition each round. We let our tool then search for key dependencies and explain the source. We find that the involved



$\{\mathcal{S}(x) : \mathcal{S}(x \oplus \mathtt{2}) = \mathcal{S}(x) \oplus \mathtt{6}\} = \{\mathtt{9}, \mathtt{a}, \mathtt{c}, \mathtt{f}\}$ $\quad$ $\{x : \mathcal{S}(x \oplus \mathtt{2}) = \mathcal{S}(x) \oplus \mathtt{6}\} = \{\mathtt{1}, \mathtt{3}, \mathtt{5}, \mathtt{7}\}$

Figure 2: Contradiction in differential characteristic for GIFT-128 [ZDY19, Tab. 10, rounds 16–17]. The dashed red line shows the inactive bit that causes the contradiction. Our tool identifies the red S-boxes as the source of the contradiction. The yellow dashed line shows an inactive bit that leads to a contradiction if the involved round constant bit is 1.

round constant must be 0, which is the case when the characteristic is used for rounds 0–17 of the block cipher. With the same model, we find a total of 8 linear constraints. One of them is always satisfied due to the partial key addition of GIFT-128, effectively doubling the probability. And two of them span over 5 rounds of the cipher. These constraints are also discussed in [Sun24].

In the framework of quasidifferential trails, the red/black dashed line corresponds to the linear masks for a quasidifferential trail with (key independent) correlation -2^{-109}. Therefore, the differential characteristic has probability 0. Similarly, the yellow/black dashed line corresponds to the linear masks for a quasidifferential trail with correlation $(-1)^{rc} \cdot 2^{-109}$, where rc denotes the involved round constant bit. Hence, if $rc = 1$, the characteristic has probability 0.

Finally, we verify the expected differential probability of round-reduced variants of differential characteristics for GIFT-64. We find that the probability closely matches the estimate using the differential distribution table. We detail our results in Table 2.

Table 2: Estimated and measured probability for reduced variants of GIFT-64 characteristics. DDT: negative logarithm of the probability estimate using differential distribution table. AMC: probability measurement with approximate model counting ($\delta = 0.2$, $\varepsilon = 0.8$).

	Rounds	0–7	1–8	2–9	3–10	4–11	5–12
[SWW21b, Table 8.1]	DDT	40.0	40.0	40.0	40.0	40.0	40.0
	AMC	39.91	39.87	39.98	39.91	40.09	39.89
	Time	17s	11s	4s	13s	4s	1m
[SWW21b, Table 8.2]	DDT	38.0	40.0	42.0	40.0	42.0	40.0
	AMC	37.96	40.02	42.05	39.91	42.05	39.96
	Time	3s	3s	7s	5s	2s	2s
[SWW21b, Table 8.3]	DDT	40.0	40.0	42.0	40.0	42.0	40.0
	AMC	40.09	39.85	42.14	39.96	42.00	40.00
	Time	3s	2s	37s	16s	4s	43s

4.2 Results for SKINNY

SKINNY is a family of tweakable block ciphers proposed by Beierle et al. at CRYPTO 2016 [BJK+16]. We give a specification in Appendix A.2.

Results. The results on the size of the set of valid keys for SKINNY are given in Table 1. For SKINNY, we notice a tweakey dependency for both analyzed characteristics. In contrast to GIFT, we notice a slight discrepancy between the counted size of the set of valid tweaks and the upper bound given by the affine hull. This indicates that there are some nonlinear dependencies that cannot be captured by the affine hull. For SKINNY-64, the nonlinear constraints explain a key space reduction by a factor of $2^{1.18}$, leaving only a gap of $2^{0.01}$. For SKINNY-128, the nonlinear constraints explain a key space reduction by a factor of $2^{1.29}$, leaving only a gap of $2^{0.04}$. Note that we fixed a typo in the first round of the characteristic where `0x32` was stated instead of `0x20` (see Figure 6 on page 32).

In many applications, we do not expect this tweakey dependency to affect a differential attack. This is because, usually, the tweakey is split into 3 parts: TK1, TK2, and TK3, where TK2 and TK3 or just TK3 serve as a tweak that the attacker can control. This allows the attacker to alternate the tweak and work around the dependency. Still, once a valid pair is found, the constraints listed in Appendix C.3 could accelerate key recovery.

Our constraints for SKINNY match those published by Peyrin and Tan [PT22]. However, our tool does not predict the probability distribution across valid keys.

4.3 Results for Midori

Midori is a family of block ciphers proposed by Banik et al. at ASIACRYPT 2015 [BBI+15]. We give a specification in Appendix A.3.

Results. The characteristic we analyze for Midori64 is depicted in Figure 9a. In Figure 9b we show the solution sets that we model for active S-boxes; we model inactive S-boxes according to the S-box definition. For this characteristic, we have verified the expected differential probability of 2^{-52} in only 1.5 seconds. Furthermore, we find a strong key dependence as only for a fraction of 2^{-17} of all keys valid pairs exist. However, the *independent probability*, i.e. the probability of the characteristic for the set of valid keys, is $2^{-52} \cdot 2^{17} = 2^{-35}$. Therefore, this characteristic can be used for a weak key attack where a much lower data complexity is traded for a small weak key space. We list the conditions on the key in Appendix C.2. Furthermore, we analyze whether there are any additional nonlinear constraints on the key by selecting 100 000 keys at random from the affine hull of valid keys. We find that all 100 000 keys permit an assignment of plaintext/ciphertext that follows the differential characteristic.

For the 11-round characteristic of Midori128 [TAY16, Tab. 3], we find 4 linear constraints and an additional 36 CNF clauses that further reduce the key space by $2^{-1.22}$. We find these extra constraints in a few seconds by analyzing rounds 4–8 of the block cipher.

4.4 Results for WARP

WARP is a lightweight block cipher proposed by Banik et al. at SAC 2020 [BBI+20]. We give a specification in Appendix A.4. In this updated version, we have fixed a bug in the loading of characteristics for WARP that lead to incorrect results.

Results. For WARP, when using the characteristics by Kumar and Yadav [KY21], we find that solving our CNF takes some time. Still, with our method of finding the affine hull of valid keys, we are able to give an upper bound of 2^{112} on the size of valid keys using only 130 SAT calls, taking about 1 hour each. This highlights the effectiveness of the idea to describe the affine hull. We list the linear constraints we find for WARP in Appendix C.4

While applying approximate model counting to the full characteristics turns out to be infeasible, we can still verify reduced versions very efficiently. We find that we can verify up to 8 rounds with probabilities as low as 2^{-78}. Our results are listed in Table 3.

Table 3: Estimated and measured probability for reduced variants of WARP characteristics. DDT: negative logarithm of the probability estimate using differential distribution table. AMC: probability measurement with approximate model counting ($\delta = 0.2$, $\varepsilon = 0.8$).

(a) Results for characteristic from [KY21, Table 8].

Rounds	0–7	1–8	2–9	3–10	4–11	5–12	6–13	7–14	8–15	9–16	10–17
DDT	36.0	32.0	36.0	40.0	50.0	66.0	72.0	78.0	76.0	76.0	74.0
AMC	36.09	32.00	35.93	39.89	50.09	65.85	72.05	78.05	76.00	76.00	73.96
Time	14s	1s	2s	3s	10s	2s	2s	7s	1s	1s	2s

(b) Results for characteristic from [KY21, Table 9].

Rounds	0–7	1–8	2–9	3–10	4–11	5–12	6–13	7–14	8–15	9–16	10–17
DDT	36.0	32.0	36.0	40.0	50.0	66.0	72.0	78.0	76.0	76.0	74.0
AMC	35.89	32.00	35.93	39.85	50.09	66.07	72.00	77.91	76.00	76.00	74.05
Time	2s	1s	2s	5s	10s	3s	2s	11s	2s	1s	1s

4.5 Results for SPEEDY

SPEEDY was proposed as an ultra-low-latency block cipher by Leander et al. at CHES 2021 [LMMR21]. We give a specification in Appendix A.5.

Related Work. Boura et al. analyzed SPEEDY and presented a differential attack on all 7 rounds [BDBN23] with a time complexity of $2^{187.84}$. Subsequently, Beyne and Neyt provided a note that shows that the main characteristic used in the attack is impossible [BN24]. They used quasidifferential trails to show a contradiction in the second round of the main characteristic.

Results. We reproduce this result by modeling the set of valid pairs for the main characteristic and find that the resulting CNF is indeed unsatisfiable.

The implications of these findings on the attack are not entirely clear since the attack uses additional characteristics with the same input and output differences that contribute $2^{-171.49}$ to the total differential probability. If these additional characteristics were valid, the attack would likely still be applicable, albeit the data and time complexity would be higher. These additional characteristics are not published, so we cannot analyze them.

In general, the impossibility of one characteristic does not serve as a security argument for a block cipher. While the dependency leads to a contradiction in this case, it could increase the overall probability in other cases. With more careful and complex analysis, we believe such improved characteristics can be identified.

4.6 Results for PRESENT

PRESENT is a lightweight block cipher proposed by Bogdanov et al. at CHES 2007 [BKL+07]. We give a specification in Appendix A.6.

Results. We apply approximate model counting to measure the probability of the 4-round iterative characteristic [Wan08, Tab. 5]. We find that the probability averaged over all 2^{80} keys is $2^{-17.98}$ ($\delta = 0.2$, $\varepsilon = 0.8$). We do not find key dependencies for this characteristic.

To analyze the 14-round characteristic, we look for linear conditions on the round keys of PRESENT, i.e., we analyze PRESENT with independent round keys. We find no linear key dependencies in this setting. Additionally, we verify the first and last 10 rounds of the characteristic separately with the PRESENT-80 key schedule. We do not find any key dependencies here as we try 1000 random keys and all of them permit valid pairs.

4.7 Results for RECTANGLE

RECTANGLE is a block cipher proposed by Zhang et al. in 2015 [ZBL+15, ZBL+14]. We give a specification in Appendix A.7.

Related Work. Beyne and Rijmen analyze the characteristics provided by the designers [BR22a, ZBL+14] and find key dependencies in both. They also predict some variation in probability based on an unspecified linear combination of round key bits.

Results. By applying our tool to rounds 9–11 of the characteristic with $p = 2^{-63}$ from [ZBL+14, App. E], we find 1000 SAT clauses that explain the key space reduction of 2^{-1}. To ease analysis, we also model RECTANGLE with independent round keys. We find that the following key-condition is caused by the S-boxes $\mathcal{S}_{9,6}$, $\mathcal{S}_{10,2}$, $\mathcal{S}_{10,3}$, and $\mathcal{S}_{11,3}$:

$$RK^{10}_{2,2} \oplus RK^{10}_{3,3} \oplus RK^{11}_{3,0} \oplus RK^{11}_{3,1} = 1\,,$$

where $RK^i_{c,r}$ denotes the bit at column c and row r of the round key for round i.

With the knowledge of the involved S-boxes, we provide a value-based explanation of the key dependency. We trace the parity, i.e., the Xor, of the bits highlighted in red through the S-boxes shown in Figure 3. At the output of $\mathcal{S}_{9,3}$ the parity of the bits highlighted in red is 1, as they must be different. Due to the key addition, at the inputs of $\mathcal{S}_{10,2}$ and $\mathcal{S}_{10,3}$, the parity is $1 \oplus RK^{10}_{2,2} \oplus RK^{10}_{3,3}$. The two S-boxes do not change the parity, but we add two keys and arrive at a parity of $1 \oplus RK^{10}_{2,2} \oplus RK^{10}_{3,3} \oplus RK^{11}_{3,1} \oplus RK^{11}_{3,0}$ at the input of $\mathcal{S}_{11,3}$. For this S-box, the parity of the two highlighted input bits must be 0; hence, we arrive at the key condition.

We can phrase this explanation in terms of quasidifferential trails like in previous work [BR22a]. In Figure 3, the red lines correspond to the linear mask for the quasidifferential trail. The same linear masks have been derived by Beyne and Rijmen [BR22a, Table 3]. This quasidifferential trail has a correlation of $(-1)^{RK^{10}_{2,2}+RK^{10}_{3,3}+RK^{11}_{3,0}+RK^{11}_{3,1}+1} \cdot 2^{-63}$. Thus, [BR22a, Theorem 4.2] leads to the same key condition.

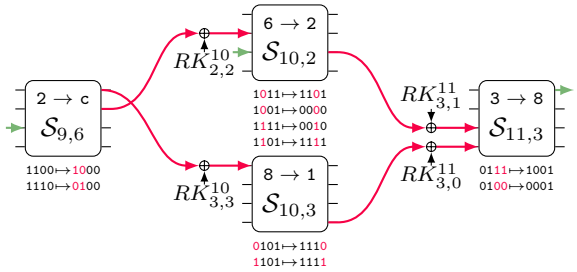


Figure 3: Part of a differential characteristic of RECTANGLE that leads to a key dependency. Next to the S-boxes we list the values the inputs/outputs can take if the differential characteristic is followed.

4.8 Results for SPECK

SPECK is a lightweight block cipher proposed by the National Security Agency [BSS+13]. We give a specification in Appendix A.8. In our work, we only consider 'long-key' variants of SPECK, i.e., SPECK with independent round keys. For the characteristic for SPECK-48 [ALLW14, Tab. 7], we fix an invalid difference bit in the first round which also improves the probability from the stated 2^{-41} to 2^{-40}.

Related Work. The characteristics from [BR22b] have already been analyzed by Beyne and Rijmen. For those characteristics, we list the original attribution in Table 4 on page page 36. With our tool we find the same conditions on the keys as them. [4] Additionally, they find some dependencies that only affect the probability which we cannot reproduce with our tool. However, AutoDiVer can prove some nonlinear key dependencies that were previously only reported to lead to a probability of approximately zero. Furthermore, we can capture and measure the non-local effect of SPECK-32 where the characteristic is unsatisfiable because its probability is too close to 2^{-32}.

Results. In all cases, we can find the linear conditions on the key in only a few seconds. For the SPECK-32 characteristics with probability close to 2^{-32}, we find many non-local nonlinear conditions. These conditions probably arise from the fact that the probability of some characteristics are very close to 2^{-32}. The SAT solver then finds many long SAT

[4] In the published version [BR22a] there are some wrong key conditions due to a mistake in a formula which are fixed in the newest eprint version [BR22b].

clauses that only exclude a tiny fraction of keys each. Hence, we do not list them. However, as visible in Table 1, we can still learn the fraction of keys that permit valid pairs.

For the characteristic [BR22b, Table 20.2], we find some nonlinear key dependencies. In addition to 1 linear constraint, we find 4 nonlinear constraints which we reformulate as

$$(RK_8^3 \oplus RK_9^3 = 1) \vee (RK_2^4 \oplus RK_3^4 = 1)\,.$$

Hence, the space of valid keys is reduced by a factor of $3/4 \approx 2^{-0.42}$. While such nonlinear constraints can be identified with quasidifferential trails, proving them is more difficult. In [BR22a], these conditions were reported to lead to a probability of approximately zero.

5 Comparison with Existing Work and Discussion

Now, we discuss our tool in comparison with related work, particularly the work on quasidifferential trails by Beyne and Rijmen and the work by Peyrin and Tan.

Quasidifferential trails [BR22a] provide an accurate theoretical model to calculate the exact probability of differentials and differential characteristics. A quasidifferential trail is defined by a combination of a differential characteristic and a set of linear masks that describe the solution sets for the differential transitions. The exact probability of a differential (characteristic) can be calculated by summing the correlations of all quasidifferential trails compatible with a given differential (characteristic). Since summing all compatible quasidifferential trails seems computationally infeasible, the authors prove a sufficient condition for a differential characteristic to be impossible [BR22a, Theorem 4.2]. Concretely, they show that if any number of quasidifferential trails with maximum absolute correlation sum to zero, the associated differential characteristic is impossible. Since the sign of quasidifferential trails is often key-dependent, Beyne and Rijmen use this result to find key dependencies. They also use quasidifferential trails to explain a key dependency for SPECK-64 that affects only the probability. Furthermore, Beyne and Neyt use quasidifferential trails with key-independent signs to show that a characteristic for SPEEDY is impossible [BN24].

Quasidifferential trails have the advantage of allowing more detailed analysis to explain dependencies that only affect probability. Beyne and Rijmen focus on providing a theoretical framework, but they also publish code to find quasidifferential trails with weight and sign. Additional analysis is deferred to the user of the code. While quasidifferential trails of maximum absolute correlation allow straight-forward deduction of the implied key dependency, there is no similarly easy way to prove key dependencies that are due to quasidifferential trails of lower absolute correlation. We see this in our application on SPECK-32 where we prove nonlinear key dependencies that were only reported to lead to a probability of approximately zero by [BR22a]. Furthermore, AutoDiVer directly deduces and prints the constraints on the key as we list them in our appendices, taking the key schedule into account (if desired), and identifies the S-boxes causing the constraint.

The work by Sun, Wang, and Wang analyzes how differential characteristics of Midori64 interact with its key schedule [SWW18]. They find conditions on the key by modeling the solution set of the active S-boxes as linear equations and applying Gaussian elimination. This approach is very efficient and excels at finding conditions that are due to two rounds interacting. However, conditions that are caused by more than two rounds interacting cannot be found with this method. Furthermore, they use SAT solvers to combine the sets of valid keys of many characteristics that correspond to a differential. In contrast, AutoDiVer does not explicitly support analysis of differentials. However, we do find key conditions over many rounds as evident by the results on RECTANGLE and GIFT-128.

The work by Peyrin and Tan [PT22] finds linear and nonlinear key dependencies by analyzing how constraints on the internal values due to active S-boxes diffuse through the cipher. In their framework, an S-box is called active if the set of inputs and outputs

is restricted to a particular subset. This can happen because of a differential transition or because the linear layer only combines active S-boxes when calculating a new S-box input. An active S-box is called a half constraint. When two half constraints meet at a key addition, they form a full constraint on the involved key. With this diffusion-based approach, they identify key dependencies and impossibilities on the keys for characteristics on SKINNY and GIFT. This includes an analysis yielding nonlinear constraints, although in a different format than our CNF-based analysis. Furthermore, they analyze the identified constraints to evaluate which percentage of keys leads to which probability.

The work by Peyrin and Tan has the advantage of calculating the probability distribution over the keys for SKINNY. However, we find that their work is quite tailored towards GIFT and SKINNY. It seems there is no shared code between their analysis of the two ciphers. In contrast, for AutoDiVer, all presented methods apply to all implemented ciphers. We believe our tool to be easier to adapt for new ciphers as shown by broad cipher support.

One limitation of our tool is that the effectiveness of the counting-based probability verification seems to depend on the key schedule. We found it to be very effective for WARP, Midori64, and GIFT-64, which use relatively simple key schedules. There, it worked even for characteristics with very low probability, such as 2^{-78} for WARP. For other ciphers, the counting became infeasible after few rounds. However, the detection of (non)linear key conditions still provides bounds for the key space and independent probability.

In summary, we believe that AutoDiVer will yield useful results for cryptanalysts, as we are providing a practical and extensible tool that complements existing theoretical frameworks. While our tool is unable to detect key dependencies that only lead to variations in probability, it excels at finding key conditions. Furthermore, we verify the probability of characteristics for WARP and Midori128 that are infeasible to verify experimentally.

6 Conclusion

We have shown various methods to analyze differential characteristics very closely. Our approach includes a method to estimate the probability of a differential characteristic, considering the block cipher's actual key schedule. Further, we describe how to estimate the size of the set of valid keys for a characteristic. Finally, based on the linear and nonlinear conditions we derive, we can upper bound the size of the set of valid keys.

All these methods are packaged into AutoDiVer, an open-source tool designed with a focus on extensibility and usability. AutoDiVer can be easily adapted for new ciphers by describing the linear layer and the key schedule. It also provides detailed output in human and machine-readable formats to minimize the need for manual post-processing.

AutoDiVer allows us to test the hypothesis of stochastic equivalence. We have seen that for many characteristics, this hypothesis does, in fact, not hold. This was demonstrated by many examples where the set of valid keys is restricted to a small subset of all keys. Crucially, the analyzed cipher does not need to be a Markov cipher, and we do not need to assume independent round keys since we model the block cipher's actual key schedule.

However, we still rely on the dominant trail assumption as our tool can only analyze one differential characteristic at a time. In future work, the tool could be combined with automatic differential characteristic search to analyze differentials [LPS21, BdSF+21, SII23, WMEJ24]. Alternative applications include finding many differential characteristics with the same input difference and weak key spaces that cover a large fraction of keys.

Acknowledgments. This research was funded in part by the Austrian Science Fund (FWF) SFB project SPyCoDe (doi:10.55776/F85) and the European Research Council (ERC) starting grant KEYLESS (#101165216). The second author was partially supported by the Center for Cyber, Law, and Policy in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office and the ERC starting grant ReSCALE (#101041545).

References

[AK18] Ralph Ankele and Stefan Kölbl. Mind the gap – A closer look at the security of block ciphers against differential cryptanalysis. In *SAC 2018*, volume 11349 of *LNCS*, pages 163–190. Springer, 2018. doi:10.1007/978-3-030-10970-7_8.

[ALLW14] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential cryptanalysis of round-reduced Simon and Speck. In *FSE 2014*, volume 8540 of *LNCS*, pages 525–545. Springer, 2014. doi:10.1007/978-3-662-46706-0_27.

[AST+17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Transactions on Symmetric Cryptology*, 2017(4):99–129, 2017. doi:10.13154/TOSC.V2017.I4.99-129.

[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 *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015. doi:10.1007/978-3-662-48800-3_17.

[BBI+20] Subhadeep Banik, Zhenzhen Bao, Takanori Isobe, Hiroyasu Kubo, Fukang Liu, Kazuhiko Minematsu, Kosei Sakamoto, Nao Shibata, and Maki Shigeri. WARP : Revisiting GFN for lightweight 128-bit block cipher. In *SAC 2020*, volume 12804 of *LNCS*, pages 535–564. Springer, 2020. doi:10.1007/978-3-030-81652-0_21.

[BC20] Christina Boura and Daniel Coggia. Efficient MILP modelings for sboxes and linear layers of SPN ciphers. *IACR Transactions on Symmetric Cryptology*, 2020(3):327–361, 2020. doi:10.13154/TOSC.V2020.I3.327-361.

[BDBN23] Christina Boura, Nicolas David, Rachelle Heim Boissier, and María Naya-Plasencia. Better steady than Speedy: Full break of SPEEDY-7-192. In *EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 36–66. Springer, 2023. doi:10.1007/978-3-031-30634-1_2.

[BdSF+21] Alex Biryukov, Luan Cardoso dos Santos, Daniel Feher, Vesselin Velichkov, and Giuseppe Vitto. Automated truncation of differential trails and trail clustering in ARX. In *SAC 2021*, volume 13203 of *LNCS*, pages 286–307. Springer, 2021. doi:10.1007/978-3-030-99277-4_14.

[BHMS84] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*, volume 2 of *The Kluwer International Series in Engineering and Computer Science*. Springer, 1984. doi:10.1007/978-1-4613-2821-6.

[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 *CRYPTO 2016*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016. doi:10.1007/978-3-662-53008-5_5.

[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 *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007. doi:10.1007/978-3-540-74735-2_31.

[BN24] Tim Beyne and Addie Neyt. Note on the cryptanalysis of Speedy. IACR Cryptology ePrint Archive, Report 2024/262, 2024. URL: https://eprint.iacr.org/2024/262.

[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 *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, 2017. doi:10.1007/978-3-319-66787-4_16.

[BR22a] Tim Beyne and Vincent Rijmen. Differential cryptanalysis in the fixed-key model. In *CRYPTO 2022*, volume 13509 of *LNCS*, pages 687–716. Springer, 2022. doi:10.1007/978-3-031-15982-4_23.

[BR22b] Tim Beyne and Vincent Rijmen. Differential cryptanalysis in the fixed-key model. IACR Cryptology ePrint Archive, Report 2022/837, 2022. URL: https://eprint.iacr.org/2022/837.

[BRV14] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential analysis of block ciphers SIMON and SPECK. In *FSE 2014*, volume 8540 of *LNCS*, pages 546–570. Springer, 2014. doi:10.1007/978-3-662-46706-0_28.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In *CRYPTO '90*, volume 537 of *LNCS*, pages 2–21. Springer, 1990. doi:10.1007/3-540-38424-3_1.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of Feal and n-hash. In *EUROCRYPT '91*, volume 547 of *LNCS*, pages 1–16. Springer, 1991. doi:10.1007/3-540-46416-6_1.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. IACR Cryptology ePrint Archive, Report 2013/404, 2013. URL: http://eprint.iacr.org/2013/404.

[BSS+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *DAC 2015*, pages 175:1–175:6. ACM, 2015. doi:10.1145/2744769.2747946.

[BSS+17] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. Notes on the design and analysis of SIMON and SPECK. IACR Cryptology ePrint Archive, Report 2017/560, 2017. URL: http://eprint.iacr.org/2017/560.

[CLN+17] Anne Canteaut, Eran Lambooij, Samuel Neves, Shahram Rasoolzadeh, Yu Sasaki, and Marc Stevens. Refined probability of differential characteristics including dependency between multiple rounds. *IACR Transactions on Symmetric Cryptology*, 2017(2):203–227, 2017. doi:10.13154/TOSC.V2017.I2.203-227.

[CMV13] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *CP 2013*, volume 8124 of *LNCS*, pages 200–216. Springer, 2013. doi:10.1007/978-3-642-40627-0_18.

[CR06] Christophe De Cannière and Christian Rechberger. Finding SHA-1 characteristics: General results and applications. In *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006. doi:10.1007/11935230_1.

[CXTQ23] Yaxin Cui, Hong Xu, Lin Tan, and Wenfeng Qi. Sat-aided differential cryptanalysis of lightweight block ciphers Midori, MANTIS and QARMA. In *ICICS 2023*, volume 14252 of *LNCS*, pages 3–18. Springer, 2023. doi:10.1007/978-981-99-7356-9_1.

[DDH+21] Stéphanie Delaune, Patrick Derbez, Paul Huynh, Marine Minier, Victor Mollimard, and Charles Prud'homme. Efficient methods to search for best differential characteristics on SKINNY. In *ACNS 2021*, volume 12727 of *LNCS*, pages 184–207. Springer, 2021. doi:10.1007/978-3-030-78375-4_8.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962. doi:10.1145/368273.368557.

[DR00] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *AES Candidate Conference*, pages 343–348. National Institute of Standards and Technology„ 2000.

[DR07] Joan Daemen and Vincent Rijmen. Plateau characteristics. *IET Information Security*, 1(1):11–17, 2007. doi:10.1049/IET-IFS:20060099.

[EME22] Johannes Erlacher, Florian Mendel, and Maria Eichlseder. Bounds for the security of Ascon against differential and linear cryptanalysis. *IACR Transactions on Symmetric Cryptology*, 2022(1):64–87, 2022. doi:10.46586/TOSC.V2022.I1.64-87.

[FHH21] Johannes Klaus Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. *ACM Journal of Experimental Algorithmics*, 26:13:1–13:26, 2021. doi:10.1145/3459080.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011. doi:10.1007/978-3-642-23951-9_22.

[KJ21] Tuukka Korhonen and Matti Järvisalo. Integrating tree decompositions into decision heuristics of propositional model counters (short paper). In *CP 2021*, volume 210 of *LIPIcs*, pages 8:1–8:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CP.2021.8.

[Köl14] Stefan Kölbl. CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives, 2014. URL: https://github.com/kste/cryptosmt.

[KY21] Manoj Kumar and Tarun Yadav. MILP based differential attack on round reduced WARP. In *SPACE 2021*, volume 13162 of *LNCS*, pages 42–59. Springer, 2021. doi:10.1007/978-3-030-95085-9_3.

[Leu12] Gaëtan Leurent. Analysis of differential attacks in ARX constructions. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 226–243. Springer, 2012. doi:10.1007/978-3-642-34961-4_15.

[LIM20] Fukang Liu, Takanori Isobe, and Willi Meier. Automatic verification of differential characteristics: Application to reduced gimli. In *CRYPTO 2020*, volume 12172 of *LNCS*, pages 219–248. Springer, 2020. doi:10.1007/978-3-030-56877-1_8.

[LLL+21] Yu Liu, Huicong Liang, Muzhou Li, Luning Huang, Kai Hu, Chenhe Yang, and Meiqin Wang. STP models of optimal differential and linear trail for s-box based ciphers. *Sci. China Inf. Sci.*, 64(5), 2021. doi:10.1007/S11432-018-9772-0.

[LM17] Jean-Marie Lagniez and Pierre Marquis. On preprocessing techniques and their impact on propositional model counting. *Journal of Automated Reasoning*, 58(4):413–481, 2017. doi:10.1007/S10817-016-9370-8.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *EUROCRYPT '91*, volume 547 of *LNCS*, pages 17–38. Springer, 1991. doi:10.1007/3-540-46416-6_2.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):510–545, 2021. doi:10.46586/TCHES.V2021.I4.510-545.

[LMY21] Yong Lai, Kuldeep S. Meel, and Roland H. C. Yap. The power of literal equivalence in model counting. In *AAAI 2021*, pages 3851–3859. AAAI Press, 2021. doi:10.1609/AAAI.V35I5.16503.

[LPS21] Gaëtan Leurent, Clara Pernot, and André Schrottenloher. Clustering effect in Simon and simeck. In *ASIACRYPT 2021*, volume 13090 of *LNCS*, pages 272–302. Springer, 2021. doi:10.1007/978-3-030-92062-3_10.

[LWZZ19] Lingchen Li, Wenling Wu, Yafei Zheng, and Lei Zhang. The relationship between the construction and solution of the MILP models and applications. IACR Cryptology ePrint Archive, Report 2019/049, 2019. URL: https://eprint.iacr.org/2019/049.

[LZS+20] Yunwen Liu, Wenying Zhang, Bing Sun, Vincent Rijmen, Guoqiang Liu, Chao Li, Shaojing Fu, and Meichun Cao. The phantom of differential characteristics. *Designs, Codes and Cryptography*, 88(11):2289–2311, 2020. doi:10.1007/S10623-020-00782-3.

[Mas99] Fabio Massacci. Using walk-sat and rel-sat for cryptographic key search. In *IJCAI 99*, pages 290–295. Morgan Kaufmann, 1999. URL: http://ijcai.org/Proceedings/99-1/Papers/043.pdf.

[MM00] Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24(1/2):165–203, 2000. doi:10.1023/A:1006326723002.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for ARX: Application to Salsa20. IACR Cryptology ePrint Archive, Report 2013/328, 2013. URL: https://eprint.iacr.org/2013/328.

[MZ06] Ilya Mironov and Lintao Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In *SAT 2006*, volume 4121 of *LNCS*, pages 102–115. Springer, 2006. doi:10.1007/11814948_13.

[NK92] Kaisa Nyberg and Lars R. Knudsen. Provable security against differential cryptanalysis. In *CRYPTO '92*, volume 740 of *LNCS*, pages 566–574. Springer, 1992. doi:10.1007/3-540-48071-4_41.

[NPE23] Marcel Nageler, Felix Pallua, and Maria Eichlseder. Finding collisions for round-reduced Romulus-h. *IACR Transactions on Symmetric Cryptology*, 2023(1):67–88, 2023. doi:10.46586/TOSC.V2023.I1.67-88.

[PT22] Thomas Peyrin and Quan Quan Tan. Mind your path: On (key) dependencies in differential characteristics. *IACR Transactions on Symmetric Cryptology*, 2022(4):179–207, 2022. doi:10.46586/TOSC.V2022.I4.179-207.

[SGM20] Mate Soos, Stephan Gocht, and Kuldeep S. Meel. Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling. In *CAV 2020*, volume 12224 of *LNCS*, pages 463–484. Springer, 2020. doi:10.1007/978-3-030-53288-8_22.

[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 *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 158–178. Springer, 2014. doi:10.1007/978-3-662-45611-8_9.

[SHY16] Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In *ACISP 2016*, volume 9723 of *LNCS*, pages 379–394. Springer, 2016. doi:10.1007/978-3-319-40367-0_24.

[SII23] Kosei Sakamoto, Ryoma Ito, and Takanori Isobe. Parallel SAT framework to find clustering of differential characteristics and its applications. In *SAC 2023*, volume 14201 of *LNCS*, pages 409–428. Springer, 2023. doi:10.1007/978-3-031-53368-6_20.

[SM19] Mate Soos and Kuldeep S. Meel. BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *AAAI 2019*, pages 1592–1599. AAAI Press, 2019. doi:10.1609/AAAI.V33I01.33011592.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *SAT 2009*, volume 5584 of *LNCS*, pages 244–257. Springer, 2009. doi:10.1007/978-3-642-02777-2_24.

[SRB21] Sadegh Sadeghi, Vincent Rijmen, and Nasour Bagheri. Proposing an MILP-based method for the experimental verification of difference-based trails: application to speck, SIMECK. *Designs, Codes and Cryptography*, 89(9):2113–2155, 2021. doi:10.1007/S10623-021-00904-5.

[SS96] João P. Marques Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In *ICCAD 1996*, pages 220–227. IEEE Computer Society / ACM, 1996. doi:10.1109/ICCAD.1996.569607.

[Sun24] Ling Sun. A linearisation method for identifying dependencies in differential characteristics: Examining the intersection of deterministic linear relations and nonlinear constraints. IACR Cryptology ePrint Archive, Report 2024/1849, 2024. URL: https://eprint.iacr.org/2024/1849.

[SW23] Ling Sun and Meiqin Wang. Sok: Modeling for large s-boxes oriented to differential probabilities and linear correlations. *IACR Transactions on Symmetric Cryptology*, 2023(1):111–151, 2023. doi:10.46586/TOSC.V2023.I1.111-151.

[SWW18] Ling Sun, Wei Wang, and Meiqin Wang. More accurate differential properties of LED64 and Midori64. *IACR Transactions on Symmetric Cryptology*, 2018(3):93–123, 2018. doi:10.13154/TOSC.V2018.I3.93-123.

[SWW21a] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Transactions on Symmetric Cryptology*, 2021(1):269–315, 2021. doi:10.46586/TOSC.V2021.I1.269-315.

[SWW21b] Ling Sun, Wei Wang, and Meiqin Wang. Improved attacks on GIFT-64. IACR Cryptology ePrint Archive, Report 2021/1179, 2021. URL: https://eprint.iacr.org/2021/1179.

[TAY16] Mohamed Tolba, Ahmed Abdelkhalek, and Amr M. Youssef. Truncated and multiple differential cryptanalysis of reduced round Midori128. In *ISC 2016*, volume 9866 of *LNCS*, pages 3–17. Springer, 2016. doi:10.1007/978-3-319-45871-7_1.

[Tse70] Grigori S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics*, pages 115–125, 1970. Translated from Russian: Zapiski Nauchnykh Seminarov LOMI 8 (1968), pp. 234–259.

[Wan08] Meiqin Wang. Differential cryptanalysis of reduced-round PRESENT. In *AFRICACRYPT 2008*, volume 5023 of *LNCS*, pages 40–49. Springer, 2008. doi:10.1007/978-3-540-68164-9_4.

[Wil27] Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. doi:10.1080/01621459.1927.10502953.

[WMEJ24] Dachao Wang, Alexander Maximov, Patrik Ekdahl, and Thomas Johansson. A new stand-alone MAC construct called SMAC. IACR Cryptology ePrint Archive, Report 2024/819, 2024. URL: https://eprint.iacr.org/2024/819.

[YM23] Jiong Yang and Kuldeep S. Meel. Rounding meets approximate model counting. In *CAV 2023*, volume 13965 of *LNCS*, pages 132–162. Springer, 2023. doi:10.1007/978-3-031-37703-7_7.

[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, Report 2014/084, 2014. URL: http://eprint.iacr.org/2014/084.

[ZBL+15] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *Sci. China Inf. Sci.*, 58(12):1–15, 2015. doi:10.1007/S11432-015-5459-7.

[ZDY19] Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. MILP-based differential attack on round-reduced GIFT. In *CT-RSA 2019*, volume 11405 of *LNCS*, pages 372–390. Springer, 2019. doi:10.1007/978-3-030-12612-4_19.

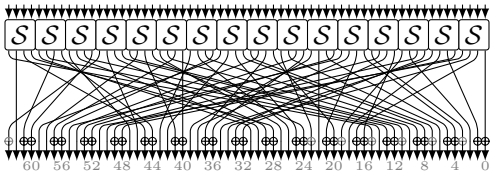
[ZHWW20] Hongluan Zhao, Guoyong Han, Letian Wang, and Wen Wang. MILP-based differential cryptanalysis on round-reduced Midori64. *IEEE Access*, 8:95888–95896, 2020. doi:10.1109/ACCESS.2020.2995795.

A Cipher Descriptions

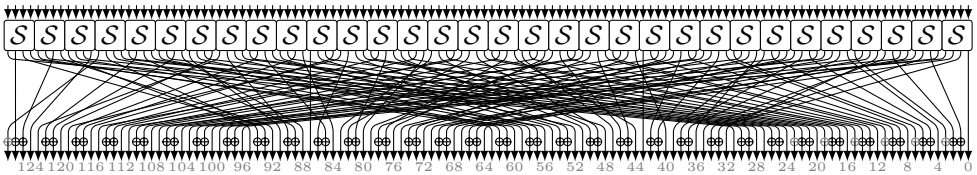
A.1 GIFT

GIFT is a family of SPN-based block ciphers proposed by Banik et al. at CHES 2017 [BPP+17]. This family includes two variants: GIFT-64 and GIFT-128. Both variants use a 128-bit key, but they differ in state size and the number of rounds. GIFT-64 has a 64-bit state size and consists of 28 rounds, while GIFT-128 features a 128-bit state size and 40 rounds. The round function in both GIFT-64 and GIFT-128 employs a 4-bit S-box, combined with a bit permutation and partial key addition. Specifically, for GIFT-64, the round keys are added to the 0th and 1st bits of each 4-bit nibble, whereas for GIFT-128, they are added to the 1st and 2nd bits. The round function and S-box are depicted in Figure 4. One of the characteristics we analyze is depicted in Figure 5.

The key schedule is defined as follows. For the n-bit version, the round key of $n/2$ bits is divided into two halves as $U \parallel V$. U is added to bits b_{4i+1} (GIFT-64) or b_{4i+2} (GIFT-128), $0 \leq i < n/4$, and V is added to bits b_{4i} (GIFT-64) or b_{4i+1} (GIFT-128), $0 \leq i < n/4$. The 7-bit round constants are added to bits b_i, $i \in \{n-1, 23, 19, 15, 11, 7, 3\}$. Each round key is a subset of the permuted bits of the 128-bit master key $K = k_7 \parallel \ldots \parallel k_0$.



(a) GIFT-64 round function.



(b) GIFT-128 round function.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

(c) GIFT S-box $\mathcal{S}$.

Figure 4: The round function of GIFT-64 and GIFT-128.

A.2 SKINNY

SKINNY's round function operates on a 4×4 state of nibbles or bytes. The variants we analyze, SKINNY-64-192 and SKINNY-128-384, apply 40 and 56 rounds, respectively. Each round comprises a SubCells operation, an AddConstants operation, a partial tweakey addition on the top 2 rows, and a lightweight MixColumns operation. The tweakey schedule is entirely linear and based on a cell shuffle and application of two linear feedback shift registers. The round function is depicted in Figure 7. A characteristic we analyze is depicted in Figure 6.

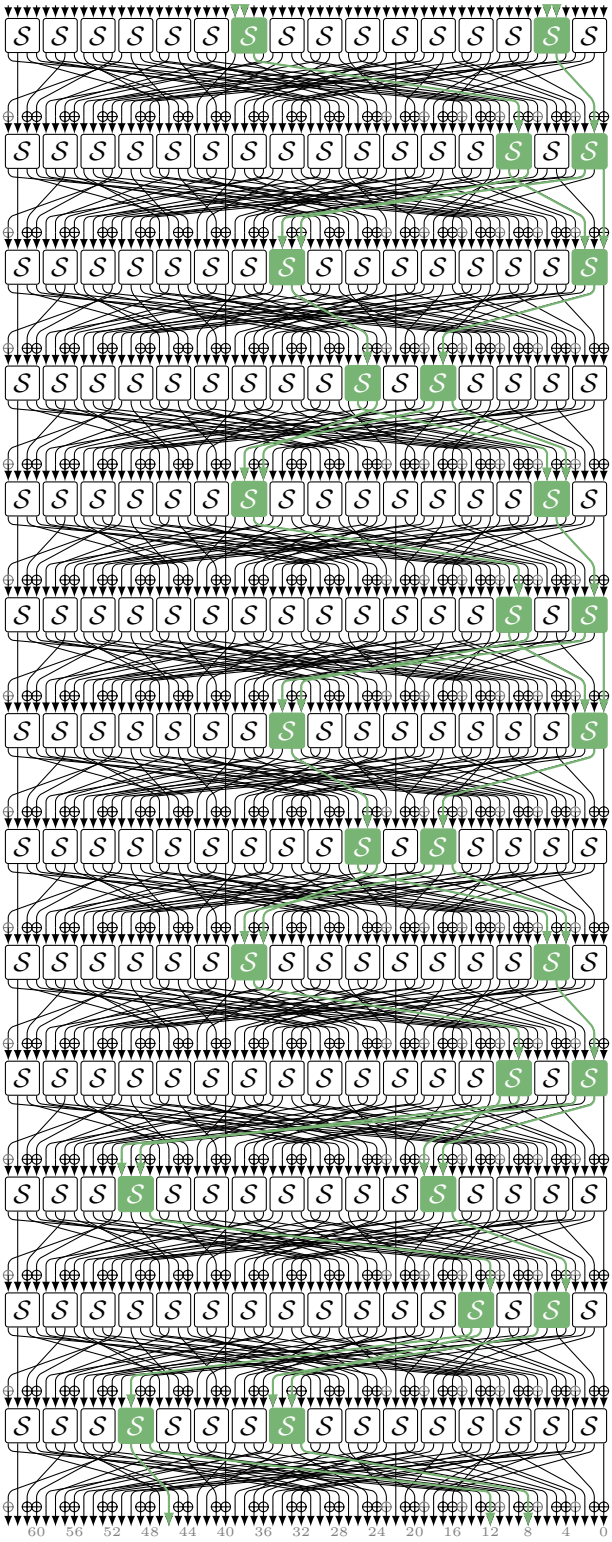


Figure 5: Differential characteristic for 13-round GIFT-64 [SWW21b, Table 8.2].

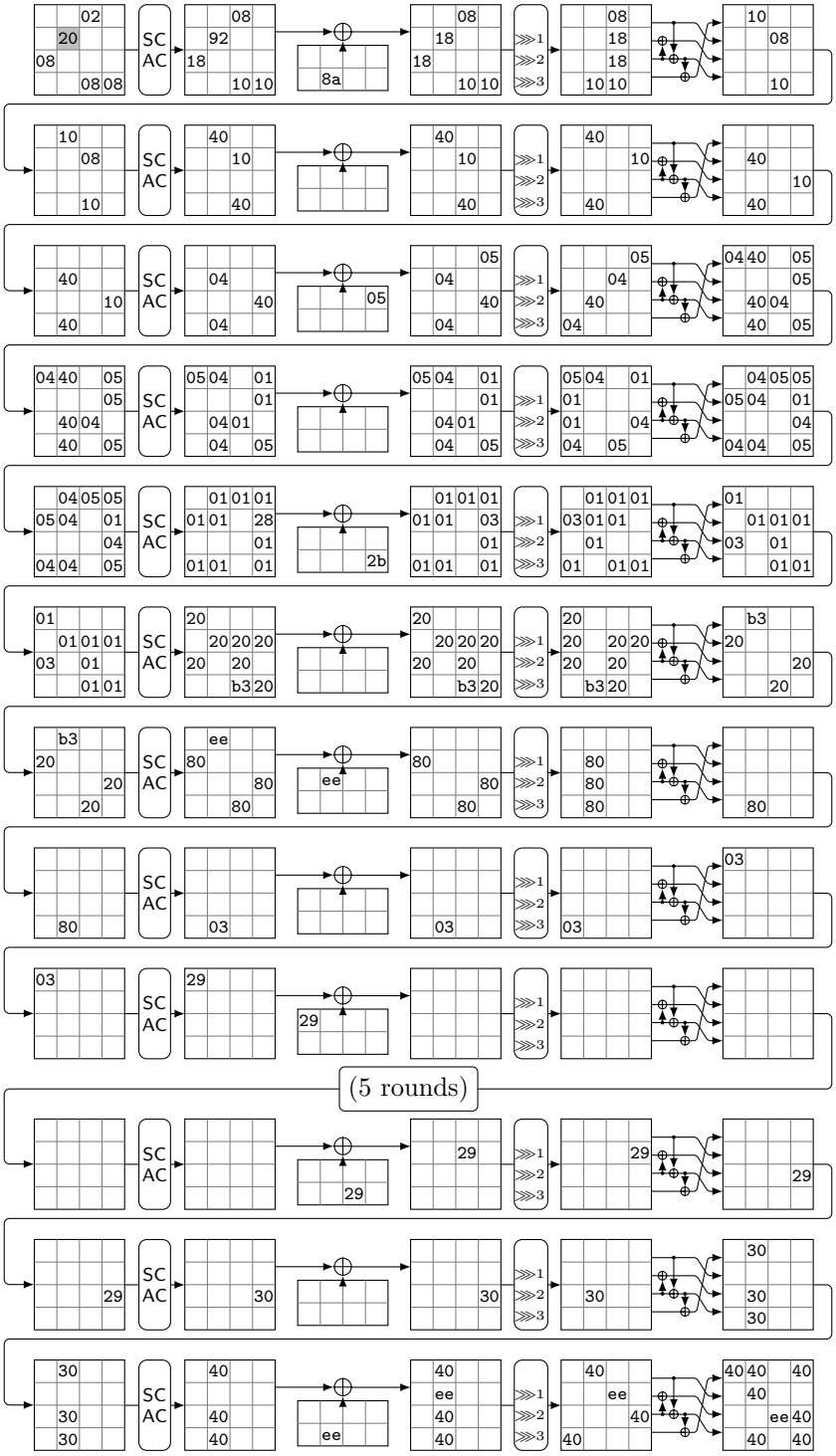


Figure 6: Differential characteristic for 17-round SKINNY-128 [DDH+21, Table 12]. The highlighted cell in the first round shows a fixed typo.

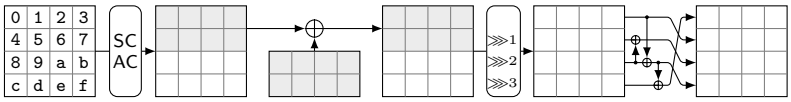


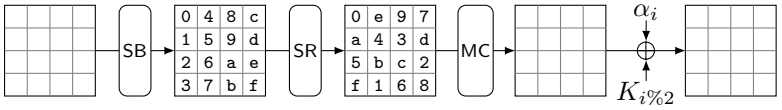
Figure 7: The round function of SKINNY.

A.3 Midori

Midori is a family of block ciphers proposed by Banik et al. at ASIACRYPT 2015 [BBI+15]. The members Midori64 and Midori128 both accept a 128-bit key but differ in block size and number of rounds. Midori64 and Midori128 apply 16 and 20 iterations of the round function, respectively.

Midori's round function is depicted in Figure 8. It operates on a 4×4 state of nibbles or bytes. First, SubCell (SC in Figure 8) applies a 4- or 8-bit S-box to each cell. Next, ShuffleCell (SR) permutes the cells of state. Then, MixColumn (MC) multiplies each column of the state with the Near-MDS matrix M.

Midori's key schedule is very simple. For Midori64, the master key K is split into two halves: $K = K_0 \parallel K_1$. At the first round and after the last round, a whitening key $WK = K_0 \oplus K_1$ is XORED. All other round keys are determined by one of the two halves and a constant: $RK_i = K_{i \bmod 2} \oplus \alpha_i$. For Midori128, the key schedule is even simpler. The whitening keys equal the master key $WK = K$, and the round keys are determined based on some constants: $RK_i = K \oplus \beta_i$.

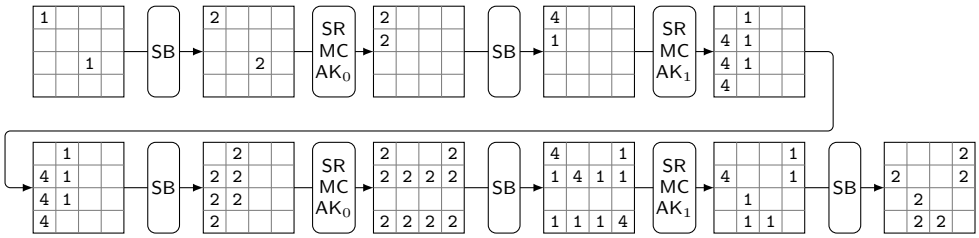


(a) Round function.

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

(b) MC matrix M.

Figure 8: The round function of Midori.



(a) Differential characteristic, first to last S-box layer.

Differential	Solutions $(x, \mathcal{S}(x))$	Differential	Solutions $(x, \mathcal{S}(x))$
$1 \to 2$	$\{(c,0),(d,2),(e,4),(f,6)\}$	$2 \to 1$	$\{(0,c),(2,d),(4,e),(6,f)\}$
$4 \to 2$	$\{(0,c),(2,d),(4,e),(6,f)\}$	$2 \to 4$	$\{(c,0),(d,2),(e,4),(f,6)\}$

(b) Solution sets of differential transitions.

Figure 9: Differential characteristic for 5-round Midori64 [ZHWW20, Table 5].

A.4 WARP

WARP is a lightweight block cipher proposed by Banik et al. at SAC 2020 [BBI+20]. It encrypts a 128-bit plaintext block using a 128-bit key. Its structure is based on a generalized Feistel construction with 32 branches.

WARP applies 41 rounds in total. For each round, a 4-bit S-box is applied to all even branches of the state. Then, the output of the S-boxes and a 64-bit round key is XORED onto all odd branches. Finally, a nibble permutation is performed as depicted in Figure 10.

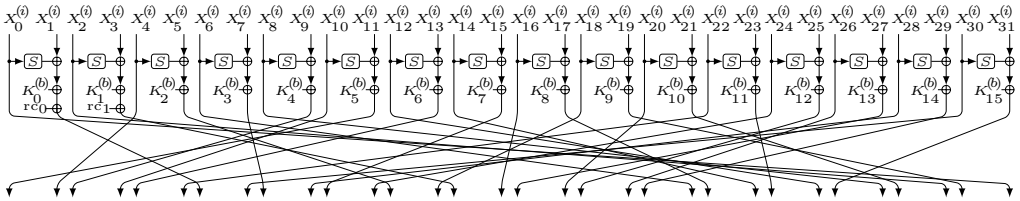


Figure 10: The round function of WARP, where the round key index is $b = i \bmod 2$.

A.5 SPEEDY

SPEEDY was proposed as an ultra-low-latency block cipher by Leander et al. at CHES 2021 [LMMR21]. Three versions exist: SPEEDY-5-192, SPEEDY-6-192, and SPEEDY-7-192 all of which accept a 192-bit key and a 192-bit plaintext input. SPEEDY-5-192 uses 5 rounds and claims 128 bits of security when data is limited to 2^{64}. SPEEDY-6-192 uses 6 rounds and claims 128 bits of security, while SPEEDY-7-192 uses 7 rounds and claims full 192 bits of security.

The round function of SPEEDY applies to a 32×6 matrix of bits and comprises 7 operations: AddRoundKey, which performs a full key addition with the round key. SubBox, which applies a 6-bit S-box to each state row. ShiftColums, which rotates the j-th column of the state upwards by j bits. Another SubBox. Another ShiftColumns. MixColumns, which multiplies each row with a cyclic binary matrix. And finally, AddConstants, which XORS a dense constant onto the state. For the last round, the second ShiftColumns as well as the MixColumns and the AddConstants operations are omitted, while a post-whitening key is added to the state. One full round function excluding constant addition is depicted in Figure 11. The key schedule of SPEEDY is based on repeatedly applying a bit permutation to the key.

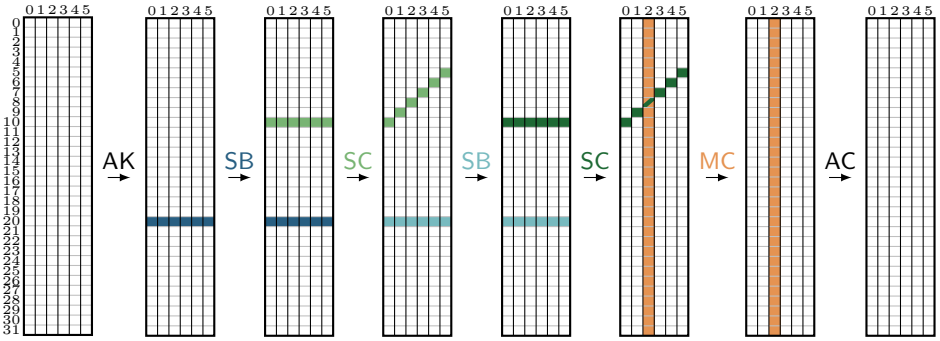
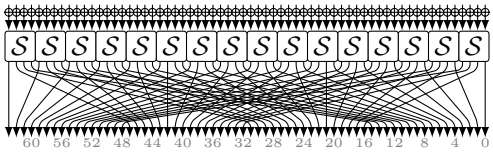


Figure 11: The round function of SPEEDY.

A.6 PRESENT

PRESENT is a lightweight SPN-based block cipher proposed by Bogdanov et al. at CHES 2007 [BKL+07]. It encrypts a 64-bit plaintext block using either an 80-bit or 128-bit key and consists of 31 rounds. The round function comprises three operations: First addRoundKey, which XORs a round key to the state, then sBoxLayer, which applies the 4-bit S-box specified in Figure 12b to the state, and finally pLayer, which performs a bit permutation on the state. The round function is illustrated in Figure 12. The key schedule is nonlinear and based on a bit rotation, partial application of the PRESENT S-box and XOR of a round counter.



(a) Round function.

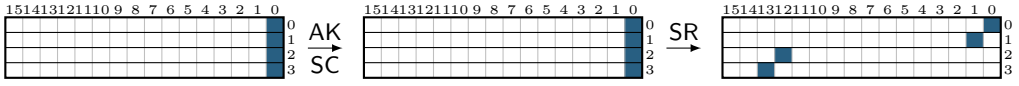
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

(b) PRESENT S-box $\mathcal{S}$.

Figure 12: The round function of PRESENT.

A.7 RECTANGLE

RECTANGLE is a lightweight SPN-based block cipher proposed by Zhang et al. [ZBL+14, ZBL+15]. Its block length is 64 bits, and the key is either 80 or 128 bits long. Over 25 rounds, it applies three operations: AddRoundKey (AK), which XORs the round key to the state, SubColumn (SC), which applies 4-bit S-boxes to the columns of the state, and ShiftRow (SR), which applies a left rotation over each row, with different shifts. The key schedule consists of four applications of the RECTANGLE S-box, a generalized Feistel transformation and an XOR with a round constant.



(a) Round function.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(x)$	6	5	c	a	1	e	7	9	b	0	3	d	8	f	4	2

(b) RECTANGLE S-box $\mathcal{S}$.

Figure 13: The round function of RECTANGLE.

A.8 SPECK

SPECK is a lightweight block cipher designed by the NSA and published at DAC 2015 [BSS+13, BSS+15, BSS+17]. The family consists of several members SPECK-$2n/mn$ with

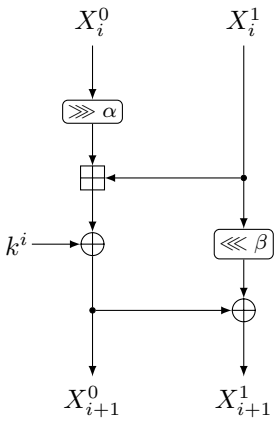


Figure 14: The round function of SPECK.

different block sizes of $2n$ bits and key sizes of mn bits for word sizes $n \in \{16, 24, 32, 48, 64\}$ and parameters $m \in \{2, 3, 4\}$. The round function is a pure ARX function operating on two n-bit words X^0, X^1 and one n-bit round key. Let (X_i^0, X_i^1) denote the input and (X_{i+1}^0, X_{i+1}^1) the output of the ith round, and k_i the corresponding round key. Then the round function is defined as follows, as illustrated in Figure 14:

$$(X_{i+1}^0, X_{i+1}^1) = (((X_i^0 \ggg \alpha) \boxplus X_i^1) \oplus k^i, \quad (X_i^1 \lll \beta) \oplus (((X_i^0 \ggg \alpha) \boxplus X_i^1) \oplus k^i)),$$

where the rotation parameters (α, β) vary depending on the family member:

$$(\alpha, \beta) = \begin{cases} (7, 2) & \text{for block size } 2n = 32, \\ (8, 3) & \text{otherwise.} \end{cases}$$

The key schedule is nonlinear, using the same operations as the round function to generate round keys.

Table 4 provides an overview of selected characteristics for SPECK.

Table 4: Characteristics analyzed by [BR22b] and original source.

Characteristic	Attribution	Comment
[BR22b, Tab. 18.1]	[ALLW14]	Table 7, rounds 1–7
[BR22b, Tab. 18.2]	[ALLW14]	Table 7, rounds 0–7
[BR22b, Tab. 19.1/2]	[ALLW14]?	not found in [ALLW14]
[BR22b, Tab. 20.1]	[BRV14]	Table 6
[BR22b, Tab. 20.2/3]	[SHY16]	not explicitly listed
[BR22b, Tab. 23.2]	[SHY16]	Table 6
[BR22b, Tab. 23.1/3/4]	[SHY16]	not explicitly listed

Figure 15 shows the differential characteristic where we find additional nonlinear key dependencies. With our tool, we can automatically identify the full adders which cause the key dependency. We highlight the inputs and outputs of the relevant full adders in the figure.

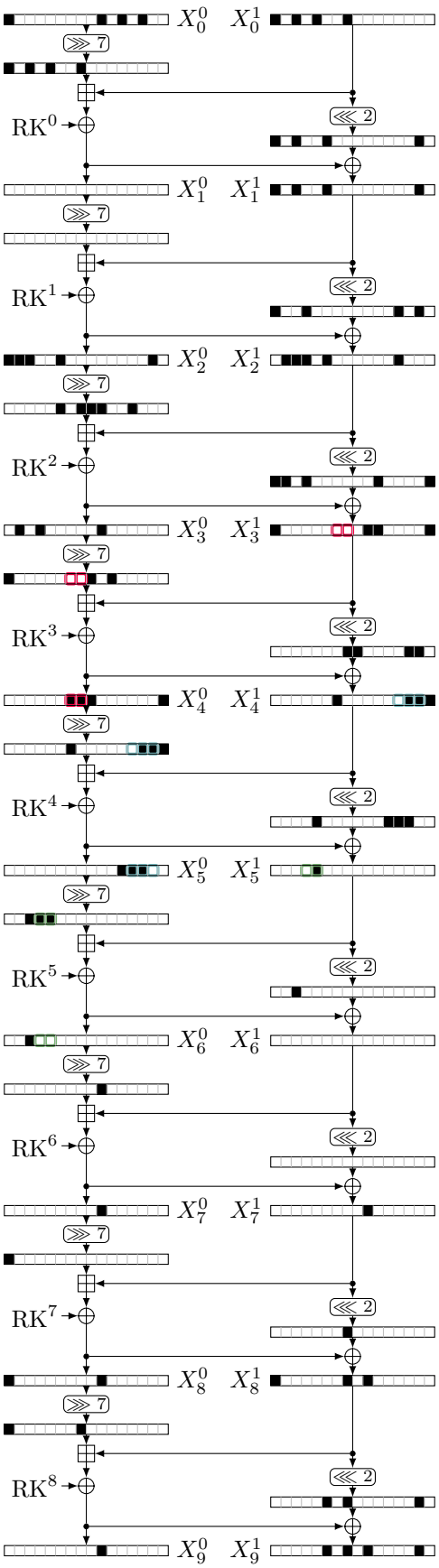


Figure 15: Characteristic for 9 rounds of SPECK-32 [BR22b, Tab. 20.2]. We highlight the inputs/outputs of the full adders involved in the nonlinear key-dependency we find.

B Runtime of our Tool

We list the runtime of our tool in Table 5. All experiments were performed on a single core of an AMD EPYC 9754 CPU.

We find that finding the affine hull of the set of valid keys is the most efficient method in all cases. In particular, when modeling independent round keys, this method is very efficient, as the associated SAT problems become very easy. For example, the slow result for Midori128 can be reproduced in a minute.

For the experimental estimate, we often find that the UNSAT cases are much quicker since the contradiction is often local. In general, the SAT problems for finding valid pairs for a given key are harder to solve since we lose the degrees of freedom of the key.

For model counting, we find that counting the set of valid keys is more efficient than counting all solutions to estimate the probability. From Subsection 2.4, we know that ApproxMC adds Xor clauses over the projection variables. For counting the keys there are fewer projection variables leading to shorter Xor clauses. In general, model counting is slower than SAT solving, as it uses many internal SAT solver calls with modified problems.

Table 5: The necessary runtime of our results. All runtimes are single-core numbers. Note that the experimental estimate was performed in parallel, so the wall clock time is lowered by the number of cores available. †: mistake in characteristic fixed

Cipher	Reference	#R	EDP	key set	Runtime		
				LC	AMC	exp. est.	LC
GIFT-64	[LWZZ19, Tab. 2]	9	2^{-42}	4	6s	3s ($n = 8$k)	1.7s
	[LWZZ19, Tab. 7]	12	2^{-58}	4	7h	4h ($n = 8$k)	3.9s
	[LWZZ19, Tab. 8]	13	2^{-62}	4		34h ($n = 8$k)	4.6s
	[SWW21b, Tab. 8.1]	13	2^{-64}	1			6.1s
	[SWW21b, Tab. 8.2]	13	2^{-64}	5		185h ($n = 12$k)	5.0s
	[SWW21b, Tab. 8.3]	13	2^{-64}	3			5.3s
	[ZDY19, Tab. 4]	12	2^{-59}	3		68h ($n = 8$k)	3.4s
	[ZDY19, Tab. 6]	12	2^{-60}	0			4.0s
GIFT-128	[LLL+21, Tab. 5]	12	$2^{-60.4}$	1	23m	2h ($n = 8$k)	48s
	[ZDY19, Tab. 13]	12	$2^{-62.4}$	1	5m	28h ($n = 8$k)	45s
	[LLL+21, Tab. 6]	13	$2^{-67.8}$	1	7h	26h ($n = 8$k)	7m
	[ZDY19, Tab. 14]	14	$2^{-85.0}$	3	25h	9h ($n = 8$k)	6m
	[ZDY19, Tab. 10]	18	2^{-109}	⊘	3s	1s ($n = 8$k)	3s
SKINNY-64	[DDH+21, Tab. 9]	15	2^{-54}	5	4h	8m ($n = 8$k)	2m
SKINNY-128	[DDH+21, Tab. 12†]	17	2^{-110}	6	8h	7m ($n = 8$k)	46m
Midori64	[ZHWW20, Tab. 5]	5	2^{-52}	17	1s	4m ($n = 4$M)	1.2s
Midori128	[TAY16, Tab. 3]	11	2^{-123}	4		1122h ($n = 14$M)	4m
Midori128	[CXTQ23, Tab. 7]	10	2^{-115}	4			101h
WARP	[KY21, Tab. 8]	18	2^{-122}	16			52m
	[KY21, Tab. 9]	18	2^{-122}	16			61m
SPECK-32	[BR22b, Tab. 18.1]	6	2^{-13}	0	2s	7s ($n = 5$k)	1s
	[BR22b, Tab. 18.2]	7	2^{-18}	0	10s	32s ($n = 5$k)	1s
	[BR22b, Tab. 19.1]	8	2^{-24}	2	15s	22s ($n = 4$k)	1s
	[BR22b, Tab. 19.2]	8	2^{-27}	3	13s	14s ($n = 4$k)	1s
	[BR22b, Tab. 20.1]	9	2^{-30}	1	19s	1m ($n = 12$k)	1s
	[BR22b, Tab. 20.2]	9	2^{-33}	1	16s	2m ($n = 40$k)	1s
	[BR22b, Tab. 20.3]	9	2^{-33}	2	23s	1m ($n = 10$k)	1s
SPECK-48	[ALLW14, Tab. 7†]	10	2^{-40}	0	13h	65h ($n = 2$k)	1s
SPECK-64	[BR22b, Tab. 22.2]	15	2^{-62}	3			6s
SPECK-96	[BR22b, Tab. 22.3]	15	2^{-81}	6			16s
SPECK-128	[BR22b, Tab. 23.1]	20	2^{-128}	5			1m
	[BR22b, Tab. 23.2]	20	2^{-128}	7			1m
	[BR22b, Tab. 23.3]	20	2^{-128}	5			1m
	[BR22b, Tab. 23.4]	20	2^{-128}	3			1m
SPEEDY-192	[BDBN23, Fig. 4]	5.5	2^{-171}	⊘	2s		2s

C Constraints on the Key

Here we list the linear and nonlinear constraints on the key as discovered by our tool.

C.1 Linear Constraints for GIFT

For GIFT, we find linear constraints on the key for almost all characteristics. We use $K_{n,i}$ to denote bit i of nibble n of the key. The constraints marked with * occur two times caused by two independent sets of S-boxes. Therefore these constraints increase the independent probability by 2^2 instead of 2^1. Below we list the constraints we find on characteristics for GIFT-64:

$$\left.\begin{aligned} K_{8,1} \oplus K_{10,1} &= 0^\star \\ K_{9,1} \oplus K_{11,1} &= 0 \\ K_{24,0} \oplus K_{26,0} &= 0 \\ K_{25,0} \oplus K_{27,0} &= 0 \end{aligned}\right\} \text{ for char. [LWZZ19, Table 7]}$$

$$\left.\begin{aligned} K_{8,1} \oplus K_{10,1} &= 0^\star \\ K_{9,1} \oplus K_{11,1} &= 0 \\ K_{24,0} \oplus K_{26,0} &= 0^\star \\ K_{25,0} \oplus K_{27,0} &= 0 \end{aligned}\right\} \text{ for char. [LWZZ19, Table 8]}$$

$$\left.\begin{aligned} K_{8,1} \oplus K_{10,1} &= 0 \\ K_{9,1} \oplus K_{11,1} &= 0 \\ K_{24,0} \oplus K_{26,0} &= 0 \\ K_{25,0} \oplus K_{27,0} &= 0 \end{aligned}\right\} \text{ for char. [LWZZ19, Table 2]}$$

$$\left.\begin{aligned} K_{25,0} &= 1 \end{aligned}\right\} \text{ for char. [SWW21b, Table 8.1]}$$

$$\left.\begin{aligned} K_{8,0} \oplus K_{10,0} &= 0 \\ K_{9,0} \oplus K_{11,0} &= 0 \\ K_{24,1} \oplus K_{26,1} &= 0 \\ K_{25,0} &= 1 \\ K_{25,1} \oplus K_{27,1} &= 0 \end{aligned}\right\} \text{ for char. [SWW21b, Table 8.2]}$$

$$\left.\begin{aligned} K_{8,0} \oplus K_{10,0} &= 0 \\ K_{24,1} \oplus K_{26,1} &= 0 \\ K_{25,0} &= 1 \end{aligned}\right\} \text{ for char. [SWW21b, Table 8.3]}$$

$$\left.\begin{aligned} K_{8,0} \oplus K_{10,0} &= 0 \\ K_{24,1} \oplus K_{26,1} &= 0 \\ K_{25,1} \oplus K_{27,1} &= 0 \end{aligned}\right\} \text{ for char. [ZDY19, Table 4]}$$

Now, we list the constraints we find on characteristics for GIFT-128.

$$\left.\begin{aligned} K_{3,1} \oplus K_{6,1} &= 0 \end{aligned}\right\} \text{ for char. [LLL}^+\text{21, Table 5]}$$

$$\left.\begin{aligned} K_{25,1} \oplus K_{30,3} &= 0 \end{aligned}\right\} \text{ for char. [ZDY19, Table 13]}$$

$$\left.\begin{aligned} K_{2,0} \oplus K_{5,0} &= 0 \end{aligned}\right\} \text{ for char. [ZDY19, Table 6]}$$

$$\left.\begin{aligned} K_{24,0} \oplus K_{29,2} &= 0 \\ K_{26,3} &= 1 \\ K_{27,3} &= 1 \end{aligned}\right\} \text{ for char. [ZDY19, Table 14]}$$

C.2 Constraints for Midori

For Midori128, we use $K_{n,b}$ to denote nibble n and bit b of the key. We list the identified constraints below.

$$\left.\begin{aligned} K_{16,3} &= 1 \\ K_{20,3} \oplus K_{22,3} &= 1 \\ K_{24,3} \oplus K_{30,3} &= 0 \\ K_{28,3} \oplus K_{30,3} &= 0 \end{aligned}\right\} \text{for char. [TAY16, Table 3]}$$

$$\left.\begin{aligned} K_{18,3} \vee K_{20,3} \vee K_{21,1} \vee \neg K_{22,3} \\ \neg K_{18,3} \vee K_{20,3} \vee K_{21,1} \vee K_{22,3} \\ K_{19,1} \vee \neg K_{20,3} \vee K_{21,1} \vee K_{23,1} \\ K_{19,1} \vee K_{20,3} \vee \neg K_{21,1} \vee \neg K_{23,1} \\ \neg K_{19,1} \vee K_{20,3} \vee \neg K_{21,1} \vee K_{23,1} \\ \neg K_{19,1} \vee \neg K_{20,3} \vee K_{21,1} \vee \neg K_{23,1} \\ K_{18,3} \vee \neg K_{19,1} \vee \neg K_{21,1} \vee K_{22,3} \vee K_{23,1} \\ K_{18,3} \vee K_{19,1} \vee \neg K_{21,1} \vee K_{22,3} \vee \neg K_{23,1} \\ \neg K_{18,3} \vee \neg K_{19,1} \vee \neg K_{21,1} \vee \neg K_{22,3} \vee K_{23,1} \\ \neg K_{18,3} \vee K_{19,1} \vee \neg K_{21,1} \vee \neg K_{22,3} \vee \neg K_{23,1} \\ K_{26,3} \vee K_{27,1} \vee \neg K_{28,3} \vee K_{29,1} \vee K_{30,3} \vee \neg K_{31,1} \\ K_{26,3} \vee K_{27,1} \vee \neg K_{28,3} \vee \neg K_{29,1} \vee K_{30,3} \vee K_{31,1} \\ K_{26,3} \vee \neg K_{27,1} \vee \neg K_{28,3} \vee K_{29,1} \vee K_{30,3} \vee K_{31,1} \\ K_{26,3} \vee \neg K_{27,1} \vee \neg K_{28,3} \vee \neg K_{29,1} \vee K_{30,3} \vee \neg K_{31,1} \\ \neg K_{26,3} \vee K_{27,1} \vee \neg K_{28,3} \vee K_{29,1} \vee \neg K_{30,3} \vee \neg K_{31,1} \\ \neg K_{26,3} \vee K_{27,1} \vee \neg K_{28,3} \vee \neg K_{29,1} \vee \neg K_{30,3} \vee K_{31,1} \\ \neg K_{26,3} \vee \neg K_{27,1} \vee \neg K_{28,3} \vee K_{29,1} \vee \neg K_{30,3} \vee \neg K_{31,1} \\ \neg K_{26,3} \vee \neg K_{27,1} \vee \neg K_{28,3} \vee \neg K_{29,1} \vee \neg K_{30,3} \vee \neg K_{31,1} \\ K_{3,1} \vee K_{7,1} \vee K_{16,3} \vee K_{17,0} \vee K_{18,3} \vee K_{20,3} \vee K_{21,0} \\ K_{3,1} \vee K_{7,1} \vee K_{16,3} \vee \neg K_{17,0} \vee K_{18,3} \vee K_{20,3} \vee \neg K_{21,0} \\ K_{3,1} \vee \neg K_{7,1} \vee K_{16,3} \vee K_{17,0} \vee K_{18,3} \vee K_{20,3} \vee \neg K_{21,0} \\ K_{3,1} \vee \neg K_{7,1} \vee K_{16,3} \vee \neg K_{17,0} \vee K_{18,3} \vee K_{20,3} \vee K_{21,0} \\ \neg K_{3,1} \vee K_{7,1} \vee K_{16,3} \vee K_{17,0} \vee K_{18,3} \vee K_{20,3} \vee \neg K_{21,0} \\ \neg K_{3,1} \vee K_{7,1} \vee K_{16,3} \vee \neg K_{17,0} \vee K_{18,3} \vee K_{20,3} \vee K_{21,0} \\ \neg K_{3,1} \vee \neg K_{7,1} \vee K_{16,3} \vee K_{17,0} \vee K_{18,3} \vee K_{20,3} \vee K_{21,0} \\ K_{3,1} \vee K_{7,1} \vee \neg K_{16,3} \vee K_{17,0} \vee \neg K_{18,3} \vee \neg K_{20,3} \vee K_{21,0} \\ \neg K_{3,1} \vee \neg K_{7,1} \vee K_{16,3} \vee \neg K_{17,0} \vee K_{18,3} \vee K_{20,3} \vee \neg K_{21,0} \\ K_{3,1} \vee K_{7,1} \vee \neg K_{16,3} \vee \neg K_{17,0} \vee \neg K_{18,3} \vee \neg K_{20,3} \vee \neg K_{21,0} \\ K_{3,1} \vee \neg K_{7,1} \vee \neg K_{16,3} \vee K_{17,0} \vee \neg K_{18,3} \vee \neg K_{20,3} \vee \neg K_{21,0} \\ K_{3,1} \vee \neg K_{7,1} \vee \neg K_{16,3} \vee \neg K_{17,0} \vee \neg K_{18,3} \vee \neg K_{20,3} \vee K_{21,0} \\ \neg K_{3,1} \vee K_{7,1} \vee \neg K_{16,3} \vee K_{17,0} \vee \neg K_{18,3} \vee \neg K_{20,3} \vee \neg K_{21,0} \\ \neg K_{3,1} \vee K_{7,1} \vee \neg K_{16,3} \vee \neg K_{17,0} \vee \neg K_{18,3} \vee \neg K_{20,3} \vee K_{21,0} \\ \neg K_{3,1} \vee \neg K_{7,1} \vee \neg K_{16,3} \vee K_{17,0} \vee \neg K_{18,3} \vee \neg K_{20,3} \vee K_{21,0} \\ \neg K_{3,1} \vee \neg K_{7,1} \vee \neg K_{16,3} \vee \neg K_{17,0} \vee \neg K_{18,3} \vee \neg K_{20,3} \vee \neg K_{21,0} \end{aligned}\right\} \text{for char. [TAY16, Table 3]}$$

$$\left.\begin{aligned} K_{1,3} \oplus K_{5,3} \oplus K_{7,3} &= 0 \\ K_{12,0} &= 1 \\ K_{18,0} &= 0 \\ K_{24,0} \oplus K_{30,0} &= 1 \end{aligned}\right\} \text{for char. [CXTQ23, Table 7]}$$

For the characteristic for Midori64 [ZHWW20, Table 5], we find the following conditions, where $K_{i,r,c,b}$ denotes bit b, row r, column c of K_i:

$$\left.\begin{aligned}
K_{0,0,0,3} \oplus K_{0,1,0,3} &= 1\\
K_{0,0,3,3} \oplus K_{0,1,3,3} \oplus K_{0,3,3,3} &= 0\\
K_{0,1,1,3} \oplus K_{0,3,1,3} &= 0\\
K_{0,1,2,3} \oplus K_{0,3,2,3} &= 1\\
K_{0,3,0,3} &= 0\\
K_{1,0,1,2} \oplus K_{1,1,1,2} \oplus K_{1,3,1,2} &= 0\\
K_{1,0,1,3} \oplus K_{1,1,1,3} \oplus K_{1,3,1,3} &= 0\\
K_{1,0,3,2} \oplus K_{1,1,3,2} &= 0\\
K_{1,0,3,3} \oplus K_{1,1,3,3} &= 0\\
K_{1,1,0,0} &= 1\\
K_{1,1,0,3} &= 0\\
K_{1,2,0,0} \oplus K_{1,3,0,0} &= 0\\
K_{1,2,0,3} \oplus K_{1,3,0,3} &= 0\\
K_{1,2,1,2} \oplus K_{1,3,1,2} &= 0\\
K_{1,2,1,3} \oplus K_{1,3,1,3} &= 0\\
K_{1,3,2,2} &= 0\\
K_{1,3,2,3} &= 0
\end{aligned}\right\} \text{for char. [ZHWW20, Table 5] / Figure 9a}$$

C.3 Constraints for SKINNY

For SKINNY-64, we find the following linear constraints. We use $K^t_{r,c,b}$ to denote row r, column c, and bit b of tweakey t.

$$\left.\begin{aligned}
K^1_{2,1,0} \oplus K^1_{2,1,1} \oplus K^1_{2,1,2} \oplus K^2_{2,1,0} \oplus K^2_{2,1,1} \oplus K^2_{2,1,3} \oplus K^3_{2,1,0} \oplus K^3_{2,1,2} &= 1\\
K^1_{2,1,3} \oplus K^2_{2,1,1} \oplus K^3_{2,1,0} \oplus K^3_{2,1,1} \oplus K^3_{2,1,3} &= 0\\
K^1_{2,2,0} \oplus K^2_{2,2,0} \oplus K^2_{2,2,2} \oplus K^3_{2,2,0} \oplus K^3_{2,2,1} \oplus K^3_{2,2,2} \oplus K^3_{2,2,3} &= 1\\
K^1_{2,2,1} \oplus K^1_{2,2,3} \oplus K^2_{2,2,0} \oplus K^2_{2,2,3} \oplus K^3_{2,2,1} &= 0\\
K^1_{2,2,2} \oplus K^2_{2,2,0} \oplus K^2_{2,2,2} \oplus K^2_{2,2,3} \oplus K^3_{2,2,1} \oplus K^3_{2,2,2} \oplus K^3_{2,2,3} &= 0
\end{aligned}\right\} \text{for char. [DDH+21, Table 9]}$$

In addition to the linear constraints above, we find the following nonlinear constraints on the round tweakeys. We use $RTK^i_{r,c,b}$ to denote row r, column c, and bit b of the round tweakey used in round i.

$$\left.\begin{aligned}
\neg RTK^2_{0,1,0} \vee RTK^2_{0,1,2} \vee RTK^3_{1,1,3}\\
RTK^2_{0,1,0} \vee RTK^2_{0,1,2} \vee \neg RTK^3_{1,1,3}\\
RTK^2_{0,1,0} \vee RTK^2_{0,1,2} \vee RTK^3_{1,1,0} \vee RTK^3_{1,1,1} \vee RTK^3_{1,1,2}\\
RTK^2_{0,1,0} \vee RTK^2_{0,1,2} \vee RTK^3_{1,1,0} \vee \neg RTK^3_{1,1,1} \vee \neg RTK^3_{1,1,2}\\
RTK^2_{0,1,0} \vee RTK^2_{0,1,2} \vee \neg RTK^3_{1,1,0} \vee RTK^3_{1,1,1} \vee \neg RTK^3_{1,1,2}\\
RTK^2_{0,1,0} \vee RTK^2_{0,1,2} \vee \neg RTK^3_{1,1,0} \vee \neg RTK^3_{1,1,1} \vee RTK^3_{1,1,2}\\
\neg RTK^2_{0,1,0} \vee \neg RTK^2_{0,1,2} \vee RTK^3_{1,1,0} \vee RTK^3_{1,1,1} \vee \neg RTK^3_{1,1,2}\\
\neg RTK^2_{0,1,0} \vee \neg RTK^2_{0,1,2} \vee RTK^3_{1,1,0} \vee \neg RTK^3_{1,1,1} \vee RTK^3_{1,1,2}\\
\neg RTK^2_{0,1,0} \vee \neg RTK^2_{0,1,2} \vee \neg RTK^3_{1,1,0} \vee RTK^3_{1,1,1} \vee RTK^3_{1,1,2}\\
\neg RTK^2_{0,1,0} \vee \neg RTK^2_{0,1,2} \vee \neg RTK^3_{1,1,0} \vee \neg RTK^3_{1,1,1} \vee \neg RTK^3_{1,1,2}\\
RTK^2_{0,1,0} \vee RTK^3_{1,1,0} \vee RTK^3_{1,1,1} \vee RTK^3_{1,1,2} \vee \neg RTK^3_{1,1,3}\\
RTK^2_{0,1,0} \vee \neg RTK^3_{1,1,0} \vee \neg RTK^3_{1,1,1} \vee RTK^3_{1,1,2} \vee \neg RTK^3_{1,1,3}\\
RTK^2_{0,1,0} \vee RTK^3_{1,1,0} \vee \neg RTK^3_{1,1,1} \vee \neg RTK^3_{1,1,2} \vee \neg RTK^3_{1,1,3}\\
RTK^2_{0,1,0} \vee \neg RTK^3_{1,1,0} \vee RTK^3_{1,1,1} \vee \neg RTK^3_{1,1,2} \vee \neg RTK^3_{1,1,3}\\
\neg RTK^2_{0,1,2} \vee RTK^3_{1,1,0} \vee \neg RTK^3_{1,1,1} \vee RTK^3_{1,1,2} \vee RTK^3_{1,1,3}\\
\neg RTK^2_{0,1,2} \vee RTK^3_{1,1,0} \vee RTK^3_{1,1,1} \vee \neg RTK^3_{1,1,2} \vee RTK^3_{1,1,3}\\
\neg RTK^2_{0,1,2} \vee \neg RTK^3_{1,1,0} \vee RTK^3_{1,1,1} \vee RTK^3_{1,1,2} \vee RTK^3_{1,1,3}\\
\neg RTK^2_{0,1,2} \vee \neg RTK^3_{1,1,0} \vee \neg RTK^3_{1,1,1} \vee \neg RTK^3_{1,1,2} \vee RTK^3_{1,1,3}
\end{aligned}\right\} \text{for char. [DDH+21, Table 9]}$$

For SKINNY-128, we find the following linear constraints.

$$\left.\begin{aligned} K^1_{0,2,0} \oplus K^2_{0,2,0} \oplus K^3_{0,2,0} &= 0 \\ K^1_{0,3,3} \oplus K^2_{0,3,1} \oplus K^3_{0,3,5} &= 0 \\ K^1_{1,0,3} \oplus K^2_{1,0,1} \oplus K^3_{1,0,5} &= 0 \\ K^1_{2,3,1} \oplus K^2_{2,3,5} \oplus K^2_{2,3,7} \oplus K^3_{2,3,3} &= 0 \\ K^1_{3,3,3} \oplus K^2_{3,3,1} \oplus K^3_{3,3,5} &= 1 \\ K^1_{3,3,7} \oplus K^2_{3,3,6} \oplus K^3_{3,3,0} \oplus K^3_{3,3,6} &= 0 \end{aligned}\right\} \text{for char. [DDH+21, Table 12]}$$

and the following nonlinear constraints on the round tweakeys:

$$\left.\begin{aligned} \neg RTK^5_{0,0,4} \vee RTK^5_{0,0,6} \\ RTK^3_{0,3,4} \vee RTK^3_{0,3,6} \vee RTK^3_{0,3,7} \\ RTK^4_{0,3,4} \vee RTK^4_{0,3,6} \vee RTK^4_{0,3,7} \\ RTK^4_{0,2,4} \vee RTK^4_{0,2,6} \vee RTK^4_{0,2,7} \\ \neg RTK^4_{0,1,4} \vee RTK^4_{0,1,6} \vee RTK^4_{0,1,7} \\ RTK^3_{0,3,3} \vee RTK^3_{0,3,4} \vee \neg RTK^3_{0,3,5} \vee RTK^3_{0,3,7} \\ \neg RTK^3_{0,3,3} \vee RTK^3_{0,3,4} \vee RTK^3_{0,3,5} \vee RTK^3_{0,3,7} \end{aligned}\right\} \text{for char. [DDH+21, Table 12]}$$

C.4 Linear Constraints for WARP

For WARP, we use $K_{n,i}$ to denote nibble n and bit i of the key. Below, we list the linear constraints we have identified.

$$\left.\begin{aligned} K_{6,0} \oplus K_{6,2} &= 1 \\ K_{11,0} \oplus K_{11,2} &= 1 \\ K_{19,0} \oplus K_{19,2} \oplus K_{25,0} \oplus K_{25,2} &= 0 \\ K_{28,0} \oplus K_{28,2} &= 1 \end{aligned}\right\} \text{for char. [KY21, Table 8]}$$

$$\left.\begin{aligned} K_{6,0} \oplus K_{6,2} &= 1 \\ K_{6,1} \oplus K_{6,2} \oplus K_{6,3} &= 1 \\ K_{11,0} \oplus K_{11,2} &= 1 \\ K_{19,0} \oplus K_{19,2} \oplus K_{25,0} \oplus K_{25,2} &= 0 \\ K_{19,1} \oplus K_{19,2} \oplus K_{19,3} \oplus K_{25,1} \oplus K_{25,2} \oplus K_{25,3} &= 1 \\ K_{28,0} \oplus K_{28,2} &= 0 \\ K_{28,1} \oplus K_{28,3} &= 0 \end{aligned}\right\} \text{for char. [KY21, Table 9]}$$

C.5 Linear Constraints for RECTANGLE

For RECTANGLE, we find the following linear constraints on the key. For the main characteristic from the design paper [ZBL+14, App. E, $p = 2^{-63}$], we find the following constraint. We use $RK^i_{r,c}$ to denote row r, column c of the key used in round i.

$$RK^{10}_{2,2} \oplus RK^{10}_{3,3} \oplus RK^{11}_{3,0} \oplus RK^{11}_{3,1} = 1\,.$$

For the secondary characteristic from the design paper [ZBL+14, App. E, $p = 2^{-66}$], we find the following constraints:

$$\begin{aligned} RK^{10}_{2,2} \oplus RK^{10}_{3,3} \oplus RK^{11}_{3,0} \oplus RK^{11}_{3,1} &= 0\,, \\ RK^{11}_{0,3} \oplus RK^{11}_{3,0} \oplus RK^{12}_{0,0} \oplus RK^{12}_{0,3} &= 1\,. \end{aligned}$$

C.6 Constraints for SPECK

In addition to the linear and non-local nonlinear constraints, we find some local nonlinear conditions for the characteristic [BR22b, Table 20.2].

$$\left.\begin{aligned} RK_8^3 \vee RK_9^3 \vee RK_2^4 \vee RK_3^4 \\ \neg RK_8^3 \vee \neg RK_9^3 \vee RK_2^4 \vee RK_3^4 \\ RK_8^3 \vee RK_9^3 \vee \neg RK_2^4 \vee \neg RK_3^4 \\ \neg RK_8^3 \vee \neg RK_9^3 \vee \neg RK_2^4 \vee \neg RK_3^4 \end{aligned}\right\} \text{for characteristic [BR22b, Table 20.2]}$$

Note that these 4 constraints can be reformulated as $(RK_8^3 \oplus RK_9^3 = 1) \vee (RK_2^4 \oplus RK_3^4 = 1)$.

Now, we list the linear constraints we identified.

$$\left.\begin{aligned} RK_9^1 \oplus RK_{10}^1 = 0 \\ RK_{11}^2 \oplus RK_{12}^2 = 1 \end{aligned}\right\} \text{for characteristic [BR22b, Table 19.1]}$$

$$\left.\begin{aligned} RK_9^1 \oplus RK_{11}^1 = 0 \\ RK_{10}^1 \oplus RK_{11}^1 = 0 \\ RK_{11}^2 \oplus RK_{12}^2 = 1 \end{aligned}\right\} \text{for characteristic [BR22b, Table 19.2]}$$

$$\left.\begin{aligned} RK_2^4 \oplus RK_3^4 = 1 \end{aligned}\right\} \text{for characteristic [BR22b, Table 20.1]}$$

$$\left.\begin{aligned} RK_{13}^1 \oplus RK_{14}^1 = 1 \end{aligned}\right\} \text{for characteristic [BR22b, Table 20.2]}$$

$$\left.\begin{aligned} RK_{10}^1 \oplus RK_{11}^1 = 0 \\ RK_2^4 \oplus RK_3^4 = 1 \end{aligned}\right\} \text{for characteristic [BR22b, Table 20.3]}$$

$$\left.\begin{aligned} RK_{25}^6 \oplus RK_{26}^6 = 0 \\ RK_{28}^7 \oplus RK_{30}^7 = 1 \\ RK_{29}^7 \oplus RK_{30}^7 = 1 \end{aligned}\right\} \text{for characteristic [BR22b, Table 22.2]}$$

$$\left.\begin{aligned} RK_{25}^5 \oplus RK_{26}^5 = 1 \\ RK_{27}^5 \oplus RK_{29}^5 = 0 \\ RK_{28}^5 \oplus RK_{29}^5 = 0 \\ RK_{30}^6 \oplus RK_{34}^6 = 1 \\ RK_{32}^6 \oplus RK_{34}^6 = 1 \\ RK_{33}^6 \oplus RK_{34}^6 = 1 \end{aligned}\right\} \text{for characteristic [BR22b, Table 22.3]}$$

$$\left.\begin{aligned} RK_1^2 \oplus RK_2^2 = 1 \\ RK_4^3 \oplus RK_5^3 = 0 \end{aligned}\right\} \text{for characteristic [BR22b, Table 23.1/2]}$$

$$\left.\begin{aligned} RK_{13}^6 \oplus RK_{14}^6 = 1 \\ RK_{16}^7 \oplus RK_{17}^7 = 0 \end{aligned}\right\} \text{for characteristic [BR22b, Table 23.2/3]}$$

$$\left.\begin{aligned} RK_{25}^{10} \oplus RK_{26}^{10} = 1 \\ RK_{28}^{11} \oplus RK_{30}^{11} = 1 \\ RK_{29}^{11} \oplus RK_{30}^{11} = 1 \end{aligned}\right\} \text{for characteristic [BR22b, Table 23.1/2/3/4]}$$