A Bit-Vector Differential Model for the Modular Addition by a Constant and its Applications to Differential and Impossible-Differential Cryptanalysis

Seyyed Arash Azimi[1], Adrián Ranea[2], Mahmoud Salmasizadeh[3], Javad Mohajeri[3], Mohammad Reza Aref[1], and Vincent Rijmen[2,4]

[1] Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran
arash_azimi@ee.sharif.edu, aref@sharif.edu
[2] imec-COSIC, KU Leuven, Belgium
{adrian.ranea,vincent.rijmen}@esat.kuleuven.be
[3] Electronic Research Institute, Sharif University of Technology, Tehran, Iran
{salmasi,mohajer}@sharif.edu
[4] Department of Informatics, UiB, Norway

Abstract. ARX algorithms are a class of symmetric-key algorithms constructed by Addition, Rotation, and XOR. To evaluate the resistance of an ARX cipher against differential and impossible-differential cryptanalysis, the recent automated methods employ constraint satisfaction solvers to search for optimal characteristics or impossible differentials. The main difficulty in formulating this search is finding the differential models of the non-linear operations. While an efficient bit-vector differential model was obtained for the modular addition with two variable inputs, no differential model for the modular addition by a constant has been proposed so far, preventing ARX ciphers including this operation from being evaluated with automated methods.

In this paper, we present the first bit-vector differential model for the n-bit modular addition by a constant input. Our model contains $O(\log_2(n))$ basic bit-vector constraints and describes the binary logarithm of the differential probability. We describe an SMT-based automated method that includes our model to search for differential characteristics of ARX ciphers including constant additions. We also introduce a new automated method for obtaining impossible differentials where we do not search over a small pre-defined set of differences, such as low-weight differences, but let the SMT solver search through the space of differences. Moreover, we implement both methods in our open-source tool `ArxPy` to find characteristics and impossible differentials of ARX ciphers with constant additions in a fully automated way. As some examples, we provide related-key impossible differentials and differential characteristics of TEA, XTEA, HIGHT, LEA, SHACAL-1, and SHACAL-2, which achieve better results compared to previous works.

Keywords: modular addition, ARX, SMT, automated tool, differential cryptanalysis, impossible differential

---

Parts of this paper were presented at the Asiacrypt 2020 conference [4].

1 Introduction

Low-end devices such as RFID tags, sensor networks, and the Internet of Things (IoT) are becoming ubiquitous. In 2018, Gartner, Inc. forecasted that there would be more than 25 billion connected devices forming the IoT by 2021 [1], and following the COVID-19 lockdowns Gartner also revealed that the unprecedented event led IoT implementers to increase IoT investments to reduce costs [2]. Traditional cryptographic algorithms are not suitable for these resource-constrained devices, and several lightweight cryptographic algorithms have been recently proposed to meet this growing demand. In this regard, the National Institute of Standards and Technology (NIST) has started a process to evaluate and standardize lightweight cryptographic algorithms [3].

ARX primitives, composed exclusively of modular Additions, cyclic Rotations, and XORs, are a promising class of lightweight cryptographic algorithms with the most efficient software implementations on low-end microcontrollers [5]. There are many noteworthy ARX algorithms, such as the hash function BLAKE [6], the stream cipher Salsa20 [7], the MAC algorithm Chaskey [8] and notable block ciphers like HIGHT [9], LEA [10], SPECK [11], SPARX [12] or CHAM [13]. Usually, ciphers that are exclusively composed of ARX operations and other common bit-vector operations (e.g., modular multiplication or logical shifts) are also considered in the class of ARX ciphers, such as IDEA [14], TEA [15], or XTEA [16].

The security of ARX ciphers is evaluated by analysing their robustness against various attacks. Some of the most successful attacks applied to ARX algorithms are differential cryptanalysis and their variants, such as boomerang or related-key differential attacks [10, 13]. These attacks exploit differences in the inputs that propagate through the cipher with high probability. Another powerful attack based on non-random propagation of differences is impossible-differential cryptanalysis [17, 18], which exploits input differences propagating to differences in the outputs with zero probability.

The standard approach to show an ARX cipher is secure against differential and impossible-differential attacks is by finding the optimal characteristics (i.e., trails of differences with the highest probabilities) and the longest impossible differentials and checking that no high-probability characteristic and no impossible differential cover most rounds of the cipher [9, 10]. When the best attack in the design stage is a differential or an impossible-differential attack, the number of rounds of the cipher is determined by the longest observed high-probability characteristic or impossible differential. Thus, searching for optimal characteristics and impossible differentials is a crucial step in the design and security analysis of a cipher.

Two main techniques have been applied to search for optimal characteristics of ARX algorithms: branch-and-bound algorithms [19, 20] based on Matsui's algorithm [21], and the recent automated methods based on Constraint Satisfaction Problems (CSP), such as SMT (Satisfiability Modulo Theories) or MILP (Mixed Integer Linear Programming) problems [22, 23]. CSP-based methods have also been recently applied to find impossible differentials [24, 25, 26]. These automated

methods formulate the characteristic or impossible-differential search problem as a CSP and delegate the solving task to one of the powerful off-the-shelf CSP solvers available nowadays [27, 28]. While some CSP-based open-source tools automate the search of ARX characteristics (e.g., CryptoSMT [29]), no CSP-based open-source tool has been published to search for impossible differentials of ARX ciphers.

The main difficulty in formulating a CSP-based search problem lies in the *differential models* of the non-linear operations, that is, the constraints describing the differential probability of the non-linear operations of the cipher. ARX ciphers can be efficiently described using the bit-vector theory of SMT, and several bit-vector differential models have been proposed so far [30, 31, 32]. For the modular addition with two n-bit operands, the foremost non-linear operation in ARX primitives, Lipmaa and Moriai found a bit-vector algorithm for computing the differential probability with complexity $O(\log_2 n)$ [30]. This algorithm can be straightforwardly translated to a bit-vector differential model, and it has been used in several SMT-based methods to search for characteristics [22, 33, 32] and impossible differentials [25] of ARX ciphers.

However, no CSP-based differential model has been proposed for the modular addition with a constant input, preventing from searching for characteristics or impossible differentials of ARX ciphers that contain constant additions. Lipmaa's algorithm is restricted to the modular addition with two operands, and it cannot be applied when one of the inputs is fixed to a constant, as we will discuss later. Machado proposed an algorithm to compute the differential probability of the constant addition [34], but it cannot be translated to an efficient bit-vector differential model due to its recursive nature and the use of floating-point arithmetic.

Contributions. We propose an efficient bit-vector differential model for the modular addition by an n-bit constant. Our model contains $O(\log_2 n)$ basic bit-vector constraints and it is composed of a bit-vector formula that determines whether a differential over the constant addition has non-zero probability, and a bit-vector function that computes the binary logarithm of the differential probability. Our bit-vector model exploits the properties of the carry chain of the modular addition and relies on efficient well-known bit-vector functions, such as the hamming weight or the bit-reversal, and new bit-vector functions that we have developed for the binary logarithm.

Furthermore, we describe an SMT-based automated method to search for characteristics of ARX ciphers, including constant additions. Our method is composed of an iterated search of optimal characteristics of round-reduced versions of the cipher and an automated encoding technique that formulates the SMT problems from the cipher's Single Static Assignment (SSA) form. Moreover, we describe a new automated method to search for impossible differentials of ARX ciphers which does not depend on any pre-defined sets of input and output differences.

We have implemented our methods in an SMT-based open-source tool `ArxPy`[5], which fully automated the search of ARX characteristics and impossible differentials. `ArxPy` is the first open-source tool that can search for the characteristics of ARX ciphers with constant additions, and it is also the first CSP-based open-source tool that automates the search of ARX impossible differentials. `ArxPy` offers a simple interface to represent any ARX cipher, different types of characteristics and impossible differentials to search, and a complete documentation.

We have applied our characteristic and impossible-differential search methods to several ARX ciphers containing constant additions to provide some examples. In particular, we have searched for different types of related-key characteristics alongside related-key impossible differentials of TEA, XTEA, HIGHT, LEA, SHACAL-1, and SHACAL-2. With our automated approach, we have revisited results previously found with manual and ad-hoc techniques. We have obtained better characteristics in terms of probability and number of rounds, and longer impossible differentials.

With our bit-vector model for the constant addition, the SMT-based automated methods, and our open-source tool `ArxPy`, we provide cipher designers with the resources to design ARX ciphers, including constant additions that are secure against differential and impossible-differential attacks. Thus, cipher designers can choose the best constants for the modular additions and optimize the number of rounds to balance security and efficiency.

Differences to the conference version This paper is an extended full version of the conference paper [4]. Thus, the content of the conference paper is included in this paper, namely the bit-vector differential model for the constant addition, the SMT-based method and tool to search for ARX differential characteristics and the related-key differential characteristics found for TEA, XTEA, HIGHT, and LEA. Apart from this content, the rest of this paper is new material. This new content includes the SMT-based method to search for impossible differentials of ARX ciphers, the tool to search for ARX impossible differentials, the related-key differential characteristics found for SHACAL-1 and SHACAL-2 and the related-key impossible differentials found for TEA, XTEA, HIGHT, LEA, SHACAL-1 and SHACAL-2. Furthermore, this paper enhances the description of the bit-vector differential model of the constant addition with improved proofs and new examples.

Outline. The notations and preliminaries are introduced in Section 2, and the bit-vector model for the modular addition by a constant is described in Section 3. Section 4 illustrates the formulation of the search of characteristics and impossible differentials as sequences of SMT problems. Section 5 presents the characteristics and impossible differentials found for TEA, XTEA, HIGHT, LEA, SHACAL-1, and SHACAL-2 using our automated approaches. Finally, Section 6 concludes the paper and addresses future works.

[5] https://github.com/ranea/ArxPy

2 Preliminaries

2.1 Notations

Let x be an integer such that its n-bit vector representation when $0 \leq x < 2^n$ is $x = (x[n-1], \ldots, x[0])$, where $x[0]$ and $x[n-1]$ denote respectively the least and the most significant bit. For ease of notation, we define $x[i] = 0$ when $i < 0$ and the symbol $*$ stands for an undetermined bit. The usual integer operations are denoted by $(+, -, \times, /)$ and the *basic* bit-vector operations are gathered in Table 1.

A mathematical expression only involving bit-vector variables and basic bit-vector operations is called a bit-vector expression. A bit-vector formula is a bit-vector expression returning `True` or `False`, such as Equals, whereas an n-bit vector function is a bit-vector expression returning an n-bit vector. In order to measure the complexity of the bit-vector differential model that we propose in this paper, we define the *bit-vector complexity* of a bit-vector expression as the number of basic bit-vector operations that the expression is composed of.

Table 1. Basic bit-vector operations for n-bit vectors.

$x[i, j]$	the bit-vector $(x[i], \ldots, x[j])$, $n > i \geq j \geq 0$
$\neg x$	bit-wise NOT of x
$x \parallel y$	concatenation of x and y
$x \wedge y$	bit-wise AND of x and y
$x \vee y$	bit-wise OR of x and y
$x \oplus y$	bit-wise XOR of x and y
$x \ll i$	(logical) left shift of x by i bits
$x \gg i$	right shift of x by i bits
$x \lll i$	left cyclic rotation of x by i bits
$x \ggg i$	right cyclic rotation of x by i bits
$x \boxplus y$	modular addition of x and y
$x \boxminus y$	modular subtraction of x and y
$\mathsf{Equals}(x, y)$	bit-vector equality of x and y, returning `True` if x and y are the same, otherwise `False`

In the literature of the bit-vector theory, the set of basic bit-vector operations usually includes the operations gathered in Table 1 and few additional operations, such as modular multiplication or modular division [35]. However, modular multiplication and modular division are much more costly than the other operations in practice, and we have excluded them from our set of basic bit-vector operations, which resembles the unit-cost RAM model used in [30].

Apart from the basic bit-vector operations listed in Table 1, we will also employ the following well-known bit-vector functions: Carry, Rev, RevCarry, HW and LZ. The carry function $c = \mathsf{Carry}(x, y)$ returns the n-bit carry chain of

the n-bit modular addition $x \boxplus y$. It is defined iteratively as $c[0] = 0$ and $c[i+1] = (x[i] \wedge y[i]) \oplus (x[i] \wedge c[i]) \oplus (y[i] \wedge c[i])$ for $0 < i < n-1$. Note that the carry has bit-vector complexity $O(1)$, since $\mathsf{Carry}(x, y) = x \oplus y \oplus (x \boxplus y)$. The carry function is an efficient function that allows propagating information from the least significant bits to the most significant bits, a property that we will exploit for our bit-vector differential model.

The bit-reversal function $\mathsf{Rev}(x)$ reverses the order of bits of x, i.e., $\mathsf{Rev}(x) = (x[0], x[1], \ldots, x[n-1])$. This function can be computed using a divide and conquer method with bit-vector complexity $O(\log_2 n)$ [36, Figure 7-1]. We will use this function to define the reverse carry, $\mathsf{RevCarry}(x, y) = \mathsf{Rev}(\mathsf{Carry}(\mathsf{Rev}(x), \mathsf{Rev}(y)))$, which allows to propagate information from right to left and also has bit-vector complexity $O(\log_2 n)$.

The hamming weight $\mathsf{HW}(x)$ returns an n-bit vector denoting the number of non-zero bits of the n-bit input x. Similar to the bit-reversal, the hamming weight can be computed using a divide and conquer approach with bit-vector complexity $O(\log_2 n)$ [36, Figure 5-2]. The hamming weight will be one of the main building blocks to obtain an efficient bit-vector representation of the binary logarithm.

The last bit-vector function we will consider is the leading zeros function $\mathsf{LZ}(x)$. This function marks the leading zeros of an n-bit input x, that is, for $0 \leq i < n$, $\mathsf{LZ}(x)[i] = 1 \iff x[n-1, i] = 0$. This function is used as a subroutine for the well-known function to compute the number of leading zeros. Similar to the previous bit-vector functions, LZ can be computed with bit-vector complexity $O(\log_2 n)$ [36, Figure 5-16].

2.2 Differential and Impossible-Differential Cryptanalysis

A block cipher is a family of permutations parametrized by a κ-bit key k, mapping n-bit plaintexts p to n-bit ciphertexts c. Most block ciphers consist of a key scheduling algorithm KS, which derives round keys $k_1, \ldots, k_r$ from the master key k, and an encryption algorithm E_k, which processes the plaintext by iterating a round function f and injecting a round key at each round, i.e., $E_k = f_{k_r} \circ \cdots \circ f_{k_1}$.

Block ciphers are shown to be secure by analysing their resistance against all known attacks. One of the most potent attacks, especially to ARX primitives, is differential cryptanalysis [37]. It exploits the non-random propagation of differences in the input to recover the secret key.

Let F be an n-bit to n-bit function and (Δ_p, Δ_c) be the XOR of a pair of inputs (p, p') and their corresponding outputs (c, c'), i.e., $\Delta_p = p \oplus p'$ and $\Delta_c = c \oplus c'$. The pair (Δ_p, Δ_c) is called a differential and its probability is defined as

$$\Pr[\Delta_p \xrightarrow{F} \Delta_c] = \frac{\#\{p : \ F(p) \oplus F(p \oplus \Delta_p) = \Delta_c\}}{2^n} .$$

A differential is valid if it has non-zero probability. In this case, its weight is defined as

$$\mathsf{weight}_F(\Delta_p, \Delta_c) = -\log_2(\Pr[\Delta_p \xrightarrow{F} \Delta_c]) .$$

The differential $0 \xrightarrow{F} 0$ has probability 1 for any function F, and a differential with non-zero input difference over a random n-bit permutation has probability 2^{-n}. Differential cryptanalysis [37] exploits a differential over the n-bit block cipher with probability $p > 2^{-n}$ to recover the secret key with roughly $O(p^{-1})$ encryption calls.

Related-key differential cryptanalysis [38] extends differential cryptanalysis by considering key differences. A related-key differential is given by a pair of differentials over the key schedule and the encryption function respectively,

$$(\Delta_k \xrightarrow{\mathsf{KS}} (\Delta_{k_1}, \ldots, \Delta_{k_r})), \quad (\Delta_p \xrightarrow{E} \Delta_c)\,,$$

where the ciphertext difference is computed using the related round-key pairs,

$$\Delta_c = (f_{k_r} \circ \cdots \circ f_{k_1})(p) \oplus (f_{k_r \oplus \Delta_{k_r}} \circ \cdots \circ f_{k_1 \oplus \Delta_{k_1}})(p \oplus \Delta_p)\,.$$

The probability of a related-key differential is the product of the probability of key schedule differential p_{KS} and the probability of encryption differential p_E.

A related-key attack exploits a related-key differential with $p_{\mathsf{KS}} > 2^{-\kappa}$ and $p_E > 2^{-n}$ to recover the secret key with complexity $O((p_{\mathsf{KS}} \times p_E)^{-1})$. The attacker takes about p_{KS}^{-1} key pairs to find one key, on average, that satisfies the key schedule differential. Next and for each key pair, the attacker runs a differential attack over the encryption using $O(p_E^{-1})$ encryption calls.

Related-key differential cryptanalysis requires a very powerful attacker that can query the encryption function $E_{k \oplus \Delta_k}$ for many keys $k \oplus \Delta_k$. In fact, if an adversary can query $E_{k \oplus \Delta_k}$ for 2^m key differences Δ_k, any block cipher is vulnerable to a related-key attack with complexity $O(2^m + 2^{n-m})$ [39]. Thus, we distinguish between weak related-key differentials (i.e., $p_{\mathsf{KS}} < 1$) and strong related-key differentials (i.e., $p_{\mathsf{KS}} = 1$), which can be exploited in practice with a single related-key pair. Furthermore, we call equivalent keys as pairs of related keys $(k, k \oplus \Delta_k)$ such that $\forall p,\ E_k(p) = E_{k \oplus \Delta_k}(p \oplus \Delta_p) \oplus \Delta_c$, for some (Δ_p, Δ_c). Note that a related-key differential with $p_E = 1$ leads to $2^{\kappa} p_{\mathsf{KS}}$ pairs of equivalent keys.

Lastly, we consider (related-key) impossible differentials. A differential (Δ_p, Δ_c) over a function F is called impossible if its probability is zero, and a related-key differential $(\Delta_k, \Delta_p \to \Delta_c)$ over a block cipher is called impossible if its probability is zero for all keys. Impossible-differential cryptanalysis [17] is an attack on block ciphers that exploits an impossible differential over the block cipher holding for every key. Related-key impossible-differential cryptanalysis is a combination of impossible-differential cryptanalysis and related-key cryptanalysis. Using the known difference of the key pairs and the input and the output of the impossible differential, the attacker discards the wrong keys to obtain the correct key.

Searching for characteristics and impossible differentials. The most challenging step to launch a differential attack is finding a differential with high probability. The main approach is to analyse how differences traverse through

the round function and search for a characteristic, that is, a trail of differences

$$\Omega = (\Delta_p = \Delta_{x_0} \xrightarrow{f_{k_1}} \Delta_{x_1} \to \cdots \to \Delta_{x_{r-1}} \xrightarrow{f_{k_r}} \Delta_{x_r} = \Delta_c)\,.$$

Similar to differentials, a characteristic Ω is valid if it has non-zero probability and its weight is defined as $-\log_2(\Pr[\Omega])$. Furthermore, we denote a related-key characteristic by a pair of characteristics $(\Omega_{\mathsf{KS}}, \Omega_E)$, where Ω_{KS} is the key schedule characteristic containing the trail of differences from the master key to the round keys and Ω_E is the encryption characteristic containing the trail of differences through the encryption.

Obtaining the exact probability of a characteristic is computationally infeasible. Thus, two assumptions are commonly made. First, it is assumed that the differential probabilities over each round are independent, which allows computing the weight of a characteristic by summing the round weights, i.e.,

$$\mathsf{weight}(\Omega) = \sum_{i=0}^{r} \mathsf{weight}(\Delta_{x_i} \to \Delta_{x_{i+1}})\,.$$

Second, it is assumed that the probability of a characteristic does not strongly depend on the choice of the secret key, also known as the hypothesis of stochastic equivalence [40], which allows computing the weight of a characteristic by averaging over all keys.

On top of that, designers also assume that the probability of a differential (Δ_p, Δ_c) is close to the probability of the best characteristic $(\Delta_p \to \cdots \to \Delta_c)$, and they prove a cipher is secure against differential cryptanalysis by showing that characteristics with high probability cannot cover most rounds of the cipher. While these assumptions do not always hold, currently this is the best systematic approach to argue security against differential cryptanalysis, and this heuristic approach is widely used for ARX ciphers in practice [22, 41, 31, 23, 33, 42].

Searching for characteristics is usually dependent on some assumptions, as mentioned earlier. In contrast, the process of obtaining an impossible differential typically results in a sound proof, guaranteeing that the probability of the achieved differential is equal to zero. Therefore, most of the impossible-differential search methods are sound but not complete. In other words, any differential found by these methods is assuredly impossible, yet there may be many impossible differentials that the search methods cannot detect.

SMT solvers. A recent approach to search for characteristics and impossible differentials of ARX ciphers is by formulating the search problem as an SMT problem in the bit-vector theory [22, 43, 31, 33, 32, 25]. Satisfiability Modulo Theories (SMT) refers to the problem of determining whether a first order formula is satisfiable with respect to some logical theory. SMT problems are a generalization of SAT problems; while the latter problems are expressed in propositional logic, SMT formulas can be expressed in richer logics, such as the theory of bit-vectors or the theory of integers.

SMT has grown in recent years into a very active research field, and several off-the-shelf SMT solvers are available nowadays [27]. Most SMT solvers can determine the satisfiability of a problem and obtain an assignment of the variables that satisfies the problem. This feature allows SMT solvers to be applied in search problems.

An SMT problem in the bit-vector theory is given by a set of bit-vector variables and a set of bit-vector formulas or constraints. The constraints can be defined with the usual logical operations (e.g., Equals, $\mathsf{NotEquals}$, $\mathsf{Implies}$, etc.) and the usual bit-vector operations (e.g., $\oplus, \boxplus, \lll$, etc.).

For example, a bit-vector SMT problem to find an 8-bit preimage of $y = f(x) = x \oplus ((x \boxplus x) \vee 1)$, given the 8-bit image $y = 3 = \texttt{00000011}$, is the following:

$$\exists x \in \{0,1\}^8 : \ \mathsf{Equals}(\texttt{00000011}, x \oplus ((x \boxplus x) \vee 1)) .$$

This problem is satisfiable and the only assignment that satisfies the problem is $x = \texttt{11111110}$.

2.3 Differential Models

To represent a characteristic in a constraint satisfaction problem, it is necessary to find a *differential model* of the round function f. For an SMT problem in the bit-vector theory, a differential model of a function $y = f(x)$ is given by a bit-vector formula $\mathsf{valid}_f(\Delta_x, \Delta_y)$ and a bit-vector function $\mathsf{weight}_f(\Delta_x, \Delta_y)$. The formula $\mathsf{valid}_f(\Delta_x, \Delta_y)$ is $\texttt{True}$ if and only if the differential $(\Delta_x \to \Delta_y)$ over f is valid, and the function $\mathsf{weight}_f(\Delta_x, \Delta_y)$ returns the weight of a valid differential $(\Delta_x \to \Delta_y)$.

Characteristics over ARX ciphers are usually defined by considering the difference after each ARX operation. The differential models of the XOR and the cyclic rotations are very simple since these operations propagate differences deterministically, that is,

$$\Delta_{x_1}, \Delta_{x_2} \xrightarrow{f(x_1,x_2)=x_1 \oplus x_2} \Delta_{x_1} \oplus \Delta_{x_2} , \qquad \Delta_x \xrightarrow{f_a(x)=x \oplus a} \Delta_x ,$$
$$\Delta_x \xrightarrow{f_a(x)=x \lll a} \Delta_x \lll a , \qquad \Delta_x \xrightarrow{f_a(x)=x \ggg a} \Delta_x \ggg a .$$

For the modular addition with two n-bit inputs, $y = f(x_1, x_2) = x_1 \boxplus x_2$, the algorithm by Lipmaa and Moriai [30] can be translated into the following differential model with bit-vector complexity $O(\log_2 n)$.

Theorem 1. *Let $((\Delta_{x_1}, \Delta_{x_2}), \Delta_y)$ be a differential over the modular addition $y = x_1 \boxplus x_2$ and denote $\overleftarrow{x} = x \ll 1$ and $\mathrm{eq}(a,b,c) = (\neg a \oplus b) \wedge (\neg a \oplus c)$. Then, the differential is valid if and only if the bit-vector formula*

$$\mathsf{valid}_{\boxplus}((\Delta_{x_1}, \Delta_{x_2}), \Delta_y) = \mathsf{Equals}(0, \mathrm{eq}(\overleftarrow{\Delta_{x_1}}, \overleftarrow{\Delta_{x_2}}, \overleftarrow{\Delta_y}) \wedge (\Delta_{x_1} \oplus \Delta_{x_2} \oplus \Delta_y \oplus \overleftarrow{\Delta_{x_2}}))$$

is $\texttt{True}$. In this case, the differential weight is given by the bit-vector function

$$\mathsf{weight}_{\boxplus}((\Delta_{x_1}, \Delta_{x_2}), \Delta_y) = \mathsf{HW}(\neg \mathrm{eq}(\Delta_{x_1}, \Delta_{x_2}, \Delta_y) \ll 1) .$$

For the modular addition with a constant input $\boxplus_a(x) = x \boxplus a$, Machado obtained the following algorithm to compute the differential probability [34].

Theorem 2. *Let (u, v) be a differential over the n-bit constant addition $\boxplus_a$. Then, the differential probability is given by*

$$\Pr[u \xrightarrow{\boxplus_a} v] = \varphi_0 \times \cdots \times \varphi_{n-1}\,,$$

where φ_i depends on the δ_{i-1} and S_i, each one defined for $0 \leq i < n$ by

$$S_i = (u[i-1], v[i-1], u[i] \oplus v[i])\,,$$

$$\delta_i = \begin{cases} (a[i-1] + \delta_{i-1})/2, & S_i = \mathit{000} \\ 0, & S_i = \mathit{001} \\ a[i-1], & S_i \in \{\mathit{010}, \mathit{100}, \mathit{110}\} \\ \delta_{i-1}, & S_i \in \{\mathit{011}, \mathit{101}\} \\ 1/2, & S_i = \mathit{111} \end{cases}$$

$$\varphi_i = \begin{cases} 1, & S_i = \mathit{000} \\ 0, & S_i = \mathit{001} \\ 1/2, & S_i \in \{\mathit{010}, \mathit{011}, \mathit{100}, \mathit{101}\} \\ 1 - (a[i-1] + \delta_{i-1} - 2a[i-1]\delta_{i-1}), & S_i = \mathit{110} \\ (a[i-1] + \delta_{i-1} - 2a[i-1]\delta_{i-1}), & S_i = \mathit{111}, \end{cases}$$

For $i = -1$, S_i and δ_i are defined by $S_{-1} = \perp$ and $\delta_{-1} = 0$.

Unfortunately, the algorithm illustrated in Theorem 2 is not suitable for constraint satisfaction problems due to its recursive nature and the use of floating-point arithmetic.

Some authors [44, Corollary 2] [45] have adapted the differential model of the 2-input addition (i.e., the modular addition with two independent inputs) for the constant addition by setting the difference of the second operand to zero, that is,

$$\begin{aligned} \mathsf{valid}_{\boxplus_a}(\Delta_x, \Delta_y) &\leftarrow \mathsf{valid}_{\boxplus}((\Delta_x, 0), \Delta_y)\,, \\ \mathsf{weight}_{\boxplus_a}(\Delta_x, \Delta_y) &\leftarrow \mathsf{weight}_{\boxplus}((\Delta_x, 0), \Delta_y)\,. \end{aligned} \tag{1}$$

The approximation given by Equation (1) models the differential $(\Delta_x \xrightarrow{\boxplus_a} \Delta_y)$ by averaging over all a. While this approach can be used to model the constant addition by a round key, since the characteristic probability is also computed by averaging over all keys, for a fixed constant this approach is rather inaccurate.

Surprisingly, the differential properties of the 2-input addition and the constant addition are very different. The 2-input addition was shown to be CCZ-equivalent to a quadratic function [46], that is, the differential properties of the 2-input addition are the same as some quadratic functions. In particular, the set of inputs (x_1, x_2) satisfying a differential $((\Delta_{x_1}, \Delta_{x_2}) \to \Delta_y)$ over the 2-input addition forms a subspace of $\mathbb{F}_2^n$, which allows to describe its differential model using few basic operations.

On the other hand, the constant addition is not CCZ-equivalent to a quadratic function, since the set of inputs (x_1, x_2) satisfying a differential (Δ_x, Δ_y) over $\boxplus_a$ does not form a subspace for many a. In other words, the probability of a differential over the constant addition is not necessarily of the form $2^{-\alpha}$ for a positive integer α, and finding a differential model for the constant input addition is a much harder problem.

We experimentally checked the accuracy of the approximation given by Equation (1) for 8-bit constants a. For most values of a, validity formulas differ roughly in 2^{13} out of all 2^{16} differentials. For those differentials where they did not differ, the difference between their weights was significantly high on average.

Consequently, no differential model of the constant addition suitable for constraint satisfaction problems has been proposed so far. In the next section, we present the first differential model of the constant addition for SMT problems in the bit-vector theory.

3 Bit-Vector Differential Model of the Constant Addition

We present a bit-vector differential model of the constant addition, composed of a bit-vector formula to determine whether a given differential is valid and a bit-vector function that computes the weight of the valid differential. Our model takes benefit from Theorem 2 [34]; however, we avoid bit iterations, floating-point arithmetic, multiplications and look-up tables, in order to obtain efficient bit-vector constraints to be used in bit-vector SMT problems.

Before we illustrate our model, we remark an essential property of Theorem 2. When the state S_i is not `110` or `111`, the probability of the step i, φ_i, depends exclusively on S_i; otherwise, φ_i depends on S_i and δ_{i-1}. When $S_i = \texttt{11*}$, $S_{i-1} \in \{\texttt{010}, \texttt{100}, \texttt{110}, \texttt{000}\}$ and for the first three cases, δ_{i-1} is equal to $a[i-2]$. However, considering the forth case, i.e., $S_{i-1} = \texttt{000}$, δ_{i-1} depends on δ_{i-2} and this dependency will proceed until we obtain a state $S_{i-\ell_i} \neq \texttt{000}$ for some positive integer ℓ_i. Thus, δ_{i-1} has the following expression when $S_i = \texttt{11*}$,

$$\delta_{i-1} = \frac{a[i-\ell_i-1]}{2^{\ell_i-1}} + \sum_{j=2}^{\ell_i} \frac{a[i-j]}{2^{j-1}}\,. \tag{2}$$

Therefore, when $S_i = \texttt{11*}$, φ_i also depends on the previous states $S_{i-1}, \cdots, S_{i-\ell_i}$, which motivates the following definition.

Definition 1. *Let $S_i = \texttt{11*}$. The chain Γ_i is defined as the smallest set of previous states $\{S_{i-1}, S_{i-2}, \cdots, S_{i-\ell_i}\}$ that completely determine φ_i, and the positive integer ℓ_i is called the length of Γ_i.*

Given a chain $\Gamma_i = \{S_{i-1}, S_{i-2}, \cdots, S_{i-\ell_i}\}$, note that $S_{i-\ell_i} \neq \texttt{000}$ and the remaining states in the chain (if any) are all equal to `000`.

In the next example, we illustrate how to calculate the differential probability using the iterative method of Theorem 2 and we learn more about the intermediate variables used for obtaining the probability.

Example 1. Consider the differential $(u, v) = (\texttt{1000101110}, \texttt{1010001110})$ over the modular addition by the 10-bit constant $a = \texttt{1000101110}$. According to Theorem 2, the differential probability of (u, v) is given by

$$\Pr[u \xrightarrow{\boxplus_a} v] = \frac{\#\{x : (x\boxplus_a) \oplus ((x \oplus u)\boxplus_a) = v\}}{2^{10}} = \prod_{i=0}^{9} \varphi_i \,.$$

Table 2 displays the variables we need to compute to obtain the differential probability. As we mentioned earlier, if $S_i = (u_{i-1}, v_{i-1}, u_i \oplus v_i) \neq \texttt{11*}$, each φ_i can be computed in a straightforward way without any further dependencies of previous states.

For the remaining states equal to $\texttt{110}$ or $\texttt{111}$, we first obtain their associated chains as

$$\begin{aligned}
&S_2 = \texttt{111},\ \Gamma_2 = \{S_1 = \texttt{000}, S_0 = \texttt{000}, S_{-1} = \perp\}, && \ell_2 = 3\,,\\
&S_4 = \texttt{110},\ \Gamma_4 = \{S_3 = \texttt{100}\}, && \ell_4 = 1\,,\\
&S_8 = \texttt{110},\ \Gamma_8 = \{S_7 = \texttt{000}, S_6 = \texttt{000}, S_5 = \texttt{000}, S_4 = \texttt{110}\}, && \ell_8 = 4\,.
\end{aligned}$$

Then, we compute the associated δ_{i-1} using Equation (2), and finally we obtain φ_i from the values of $a[i-1]$ and the computed δ_{i-1}.

Table 2. The intermediate variables for finding the differential probability of Example 1.

i	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**−1**
$a[i]$	1	0	0	0	1	0	1	1	1	0	0
$u[i]$	1	0	1	0	0	0	1	1	1	0	0
$v[i]$	1	0	1	0	0	0	1	0	1	0	0
S_i	000	110	000	000	000	110	100	111	000	000	$\perp$
δ_i	0	0	$\frac{3}{8}$	$\frac{3}{4}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	0	0	0
φ_i	1	$\frac{5}{8}$	1	1	1	1	$\frac{1}{2}$	1	1	1	

Multiplying each φ_i listed in Table 2 leads to the differential probability,

$$\Pr[u \xrightarrow{\boxplus_a} v] = \prod_{i=0}^{9} \varphi_i = \frac{5}{16}\,.$$

3.1 Validity

Let (u, v) be a differential over $\boxplus_a$, the modular addition by n-bit constant a. According to Theorem 2, the differential probability of (u, v) can be expressed as $\varphi_0 \times \cdots \times \varphi_{n-1}$. Thus, (u, v) is a valid differential, i.e., with non-zero probability, if and only if all φ_i are non-zero. If $\varphi_i = 0$, note that S_i must be $\texttt{001}, \texttt{110}$ or $\texttt{111}$. While $S_i = \texttt{001}$ always implies $\varphi_i = 0$, the other two cases require an extra condition to result in $\varphi_i = 0$, as shown in the next lemma.

Lemma 1. *Let the state S_i be* `11b`*, for* $\mathtt{b} \in \{0, 1\}$*. Then, φ_i is equal to 0 if and only if* $\neg \mathtt{b} \oplus a[i-1] = a[i-2] = \cdots = a[i-\ell_i-1]$.

Proof. Having $S_i = \mathtt{11b}$, $\varphi_i = 0$ if and only if $\neg \mathtt{b} = \delta_{i-1} \oplus a[i-1]$. Let ℓ_i be the chain length of S_i. The case for $\ell_i = 1$ is trivial, since $\delta_{i-1} = a[i-2]$. To achieve $\delta_{i-1} = a[i-1] \oplus \neg b$ when $\ell_i > 1$, the non-negative rational number δ_{i-1} must be equal to 0 or 1. Since δ_{i-1} is a monotonically increasing function of $(a[i-2], \ldots, a[i-\ell_i-1])$ regarding Equation (2), δ_{i-1} reaches its extrema in $(0, \ldots, 0)$ and $(1, \ldots, 1)$, that is,

$$\delta_{i-1} = c \iff a[i-2] = a[i-3] = \cdots = a[i-\ell_i-1] = c\,, \quad \forall c \in \{0, 1\}\,,$$

Thus, $\delta_{i-1} = a[i-1] \oplus \neg \mathtt{b} \iff \delta_{i-1} = a[i-2] = \cdots = a[i-\ell_i]$. □

The next lemma provides a bit-vector expression to check Lemma 1 by exploiting the fact that the carry chain allows a bit to affect the bits to its left.

Lemma 2. *Consider the following n-bit values,*

$$s_{00*} = \neg(u \ll 1) \wedge \neg(v \ll 1), \quad s_{**1} = u \oplus v, \quad a' = (a \oplus (a \ll 1)) \ll 1,$$
$$c = \mathsf{Carry}\left(s_{00*} \wedge \neg a', \neg(s_{00*} \ll 1)\right), \quad g = (s_{**1} \oplus a') \wedge (c \vee \neg(s_{00*} \ll 1))\,.$$

Then, for all states $S_i =$ `11*`*, we have $\varphi_i = 0$ if and only if $g[i] = 1$.*

Proof. Let $S_i = \mathtt{11b}$ with chain length ℓ_i. Note that $a'[i] = a[i-1] \oplus a[i-2]$ and that $s_{\mathtt{00*}}[i] = 1$ (resp. $s_{\mathtt{**1}}[i] = 1$) if and only if $S_i = \mathtt{00*}$ (resp. $S_i = \mathtt{**1}$).

The first operand of $g[i]$, i.e., $(s_{\mathtt{**1}} \oplus a')[i]$, is equal to one if and only if $\mathtt{b} = \neg(a[i-1] \oplus a[i-2])$. For $\ell_i = 1$ it is easy to see that $S_{i-1} \neq \mathtt{00*}$; therefore, the second operand of $g[i]$ is 1, and by Lemma 1 $g[i] = 1$ if and only if $\varphi_i = 0$.

When $\ell_i > 1$, $S_{i-1} = \mathtt{000}$ and the second major operand of $g[i]$ reduces to c. In particular, the two major operands of the Carry function of c are given by

$$(s_{\mathtt{00*}} \wedge \neg a')[i, i-\ell_i] = (\neg(a[i-1] \oplus a[i-2]), \ldots, \neg(a[i-\ell_i] \oplus a[i-\ell_i-1]), 0)\,,$$
$$\neg(s_{\mathtt{00*}} \ll 1)[i, i-\ell_i] = (0, \ldots, 0, 1, *)\,.$$

Thus, $c[i] = c[i-1] \wedge \neg a'[i-1]$ and $c[i-\ell_i+1] = c[i-\ell_i] \wedge \neg s_{\mathtt{00*}}[i-\ell_i-1] = 0$; otherwise, for $0 \leq j \leq i-\ell_i-1$ we will obtain $s_{\mathtt{00*}}[j] = 0$ which does not conform to $S_0 = \mathtt{00*}$. By unrolling the recursive definition of $c[i]$, we see that $c[i] = \neg a'[i-1] \wedge \cdots \wedge \neg a'[i-\ell_i+1]$. In other words, $c[i] = 1$ if and only if $a[i-2] = \cdots = a[i-\ell_i-1]$. Together with the condition for $(s_{\mathtt{**1}} \oplus a')[i] = 1$, we have that $g[i] = 1$ exactly when $\varphi_i = 0$, regarding Lemma 1. □

Lemma 2 provides a bit-vector variable g that detects the states $S_i = \mathtt{11*}$ leading to invalidity. The next theorem presents the final bit-vector formula for the validity by taking into account the states $S_i = \mathtt{001}$ as well.

Theorem 3. *Let (u, v) be a differential over the n-bit constant addition $\boxplus_a$. Consider the n-bit value g defined in Lemma 2 and the following n-bit values*

$$s_{001} = \neg(u \ll 1) \wedge \neg(v \ll 1) \wedge (u \oplus v)\,, \qquad s_{11*} = (u \ll 1) \wedge (v \ll 1)\,.$$

Then, the bit-vector formula $\mathsf{valid}_{\boxplus_a}(u, v) = \mathsf{Equals}(s_{001} \vee (s_{11*} \wedge g), 0)$ *is* `True` *if and only if the differential (u, v) is valid.*

Proof. By the definition of s_{001} and s_{11*}, $s_{001}[i] = 1$ (respectively $s_{11*}[i] = 1$) if and only if $S_i = \texttt{001}$ (respectively $S_i = \texttt{11*}$). Moreover, $\varphi_i = 0$ exactly when $S_i = 001$, or when $S_i = 11*$ and $g[i] = 1$ (Lemma 2). Thus, $\varphi_i = 0$ if and only if $s_{001} \vee (s_{11*} \wedge g)[i] = 1$. □

Since the number of basic bit-vector operations of our bit-vector validity formula is independent of the bit-size of the inputs, the bit-vector complexity of $\mathsf{valid}_{\boxplus_a}$ is $O(1)$.

Example 2. Consider the valid differential of Example 1, i.e. $a = \texttt{1000101110}$, $u = \texttt{1010001110}$, and $v = \texttt{1010001010}$. Previously, we showed that its differential probability is non-zero and equal to 5/16. In this example, we will illustrate our bit-vector validity formula step by step.

Table 3 provides some of the essential bit-vector values used in Theorem 3. Since there is no state equal to `001`, s_{001} is the all-zero bit-vector. As we have shown in Example 1, there are three states equal to `11*`, and the associated bit of s_{11*} is equal to one in the corresponding bits. In this example, no state `11*` leads to invalidity, and g is equal to the all-zero bit-vector. Thus, $s_{001}[i] \vee (s_{11*}[i] \wedge g[i]) = 0$ for all i, and our validity formula

$$\mathsf{valid}_{\boxplus_a}(u, v) = \mathsf{Equals}(s_{001} \vee (s_{11*} \wedge g), 0)$$

evaluates to `True`.

Table 3. The intermediate variables for evaluating the bit-vector validity formula of Example 2.

i	9	8	7	6	5	4	3	2	1	0
$a[i]$	1	0	0	0	1	0	1	1	1	0
$u[i]$	1	0	1	0	0	0	1	1	1	0
$v[i]$	1	0	1	0	0	0	1	0	1	0
$g[i]$	0	0	0	0	0	0	0	0	0	0
$s_{11*}[i]$	0	1	0	0	0	1	0	1	0	0
$s_{001}[i]$	0	0	0	0	0	0	0	0	0	0

3.2 Weight of a Valid Differential

In this section, we propose a bit-vector function that computes the weight of a valid differential over the constant addition. Working with differential weights has the advantage that multiple differential weights can be combined by adding them up, while probabilities need to be multiplied, a very costly operation in a bit-vector SMT problem.

The weight of a valid differential over the constant addition is an irrational value in general, and it cannot be represented as a fixed-sized bit-vector. Thus, our bit-vector function computes a close approximation of the weight, and we provide almost tight bounds for the approximation error.

Through the rest of the section, let (u, v) be a valid differential over the n-bit constant addition $\boxplus_a$. According to Theorem 2, the weight can be obtained by

$$\mathsf{weight}_{\boxplus_a}(u, v) = -\log_2\left(\prod_{i=0}^{n-1} \varphi_i\right) = -\sum_{i=0}^{n-1} \log_2(\varphi_i)\,. \tag{3}$$

Let $\mathcal{I}$ denote the set of indices corresponding to the states `11*` with chain length bigger than one, i.e., $\mathcal{I} = \{1 \le i \le n-1 \mid S_i = \texttt{11*},\ \ell_i > 1\}$. For $i \notin \mathcal{I}$, the probability φ_i only depends on the current state S_i and φ_i is either 1 or $1/2$. Based on the aforementioned fact, we show how to acquire the summation of all $\log_2(\varphi_i)$ when $i \notin \mathcal{I}$ using bit-vector expressions.

Lemma 3. *Let* $\mathcal{I} = \{1 \le i \le n-1 \mid S_i = \texttt{11*},\ \ell_i > 1\}$. *Then,*

$$-\sum_{i \notin \mathcal{I}} \log_2(\varphi_i) = \mathsf{HW}((u \oplus v) \ll 1)\,.$$

Proof. To prove the lemma, we divide the set $\{i \mid i \notin \mathcal{I}\}$ into two parts as $\{i \mid S_i \neq \texttt{11*}\}$ and $\{i \mid S_i = \texttt{11*}, \ell_i = 1\}$. For each state $S_i \neq 11*$, there are two possible cases. If S_i is equal to `000`, the corresponding step probability is $\varphi_i = 1$. Otherwise, $S_i \in \{010, 011, 100, 101\}$ and we obtain $\varphi_i = 1/2$. Considering these two cases leads to

$$\begin{aligned}\sum_{\substack{i \\ S_i \neq 11*}} \log_2(\varphi_i) &= \sum_{\substack{i \\ S_i = 000}} \log_2(1) \quad + \sum_{\substack{i \\ S_i \in \{010,011,100,101\}}} \log_2(1/2)\,,\\ &= -\,\#\{S_i \in \{010, 011, 100, 101\} : 0 \le i < n\}\,.\end{aligned}$$

Since the second case $S_i \in \{010, 011, 100, 101\}$ occurs when $u[i-1] \oplus v[i-1] = 1$, we can use $\mathsf{HW}((u \oplus v) \ll 1)$ to compute the number of times this case happens.

Now for $S_i = \texttt{11*}$ when $\ell_i = 1$, we know that $\delta_{i-1} = a[i-2] \in \{0, 1\}$. Since the probability is not equal to zero, we obtain $\varphi_i = 1$. Thus we get

$$\sum_{\substack{i \\ S_i = 11* \\ \ell_i = 1}} \log_2(\varphi_i) = 0\,.$$

By and large, the sum of $\log_2(\varphi_i)$ when $i \notin \mathcal{I}$ is

$$\sum_{i \notin \mathcal{I}} \log_2(\varphi_i) = \sum_{\substack{i \\ S_i \neq 11*}} \log_2(\varphi_i) \ + \sum_{\substack{i \\ S_i = 11* \\ \ell_i = 1}} \log_2(\varphi_i) = -\,\mathsf{HW}((u \oplus v) \ll 1)\,. \qquad \square$$

Lemma 3 describes the sum of $\log_2(\varphi_i)$ when $i \notin \mathcal{I}$ as a bit-vector expression with complexity $O(\log_2 n)$. To describe the logarithmic summation when $i \in \mathcal{I}$ as a bit-vector, we will first show how to split φ_i as the quotient of two integers.

Lemma 4. *Let $i \in \mathcal{I}$ and let p_i be the positive integer defined by*

$$p_i = \begin{cases} a[i-2, i-\ell_i] + a[i-\ell_i-1], & u[i] \oplus v[i] \oplus a[i-1] = 1 \\ 2^{\ell_i - 1} - (a[i-2, i-\ell_i] + a[i-\ell_i-1]), & u[i] \oplus v[i] \oplus a[i-1] = 0 \end{cases}$$

where $\ell_i > 1$ is the chain length of the state $S_i = \texttt{11*}$. Then, $\varphi_i = \frac{p_i}{2^{\ell_i - 1}}$.

Proof. Considering the definition of φ_i when $S_i = \texttt{11*}$,

$$\varphi_i = \begin{cases} \delta_{i-1}, & u[i] \oplus v[i] \oplus a[i-1] = 1 \\ 1 - \delta_{i-1}, & u[i] \oplus v[i] \oplus a[i-1] = 0 \end{cases}$$

and following the definition of δ_{i-1} given by Equation (2),

$$2^{\ell_i - 1}\delta_i = \sum_{j=0}^{\ell_i - 2} 2^j a[i - \ell_i + j] \; + a[i-\ell_i-1] = a[i-2, i-\ell_i] + a[i-\ell_i-1]\,,$$

we obtain that $\varphi_i = p_i / 2^{\ell_i - 1}$. Moreover, having $0 < \varphi_i \leq 1$ and $\ell_i > 1$ results in $0 < p_i \leq 2^{\ell_i - 1}$. Thus, p_i is always a positive integer. □

Due to Lemma 4, we can decompose the logarithmic summation over $\mathcal{I}$ as

$$\sum_{i \in \mathcal{I}} \log_2(\varphi_i) = \sum_{i \in \mathcal{I}} \log_2(p_i) \; - \sum_{i \in \mathcal{I}} (\ell_i - 1)\,.$$

The next lemma shows how to describe the summation involving the chain lengths with basic bit-vector operations.

Lemma 5. *Consider the n-bit vector $s_{000} = \neg(u \ll 1) \wedge \neg(v \ll 1)$. Then,*

$$\sum_{i \in \mathcal{I}} (\ell_i - 1) = \mathsf{HW}\left(s_{000} \wedge \neg\, \mathsf{LZ}(\neg s_{000})\right).$$

Proof. Recall that there are exactly $(\ell_i - 1)$ states in each chain Γ_i such that

$$S_{i-1} = S_{i-2} = \cdots = S_{i-(\ell_i - 1)} = \texttt{000}.$$

Therefore, we have $\sum_{i \in \mathcal{I}} (\ell_i - 1) = \#\{S_j | S_j = \texttt{000} \text{ and } \exists i \in \mathcal{I} \; s.t. \; S_j \in \Gamma_i\}$. When $S_j = \texttt{000}$, the next state S_{j+1} will be a member of the set $\{\texttt{000}, \texttt{11*}\}$. As a result, it is easy to see that for an arbitrary j, if S_j is equal to $\texttt{000}$, then either S_j is included in some chain $\Gamma_i, i \in \mathcal{I}$, or S_j belongs to the set Γ' defined by

$$\Gamma' = \{S_{n-1} = \texttt{000}, \cdots, S_{n-k} = \texttt{000}\}\,,$$

for some $k > 0$, where $S_{n-k-1} \neq 000$. Concerning Definition 1, one can observe that Γ' is not a chain. Therefore, $\sum_{i \in \mathcal{I}} (\ell_i - 1) = \#\{S_j | S_j = \texttt{000} \text{ and } S_j \notin \Gamma'\}$.

Since we are assuming that the differential is valid, there are no states $S_j = \texttt{001}$, and $s_{000}[j] = 1$ if and only if $S_j = \texttt{000}$. On the other hand, the function LZ can be used to detect the states from the set Γ'. In particular, $\mathsf{LZ}(\neg s_{000})[i]$ is equal to $\texttt{1}$ if and only if $S_i \in \Gamma'$. Therefore, we obtain

$$\sum_{i \in \mathcal{I}} (\ell_i - 1) = \mathsf{HW}\left(s_{000} \wedge (\neg\, \mathsf{LZ}(\neg s_{000}))\right).$$ □

Representing the sum of $\log_2(p_i)$ by a bit-vector expression is the most complex and challenging part of our differential model. Thus, we will proceed in several steps. First, we will show how to obtain a bit-vector w that contains all the p_i as some sub-vectors.

Lemma 6. *Consider the following n-bit values,*

$$\begin{aligned}
&s_{000} = \neg(u \ll 1) \wedge \neg(v \ll 1)\,, && s'_{000} = s_{000} \wedge \neg\,\mathsf{LZ}(\neg s_{000})\,,\\
&t = \neg s'_{000} \wedge (s'_{000} \gg 1)\,, && t' = s'_{000} \wedge (\neg(s'_{000} \gg 1))\,,\\
&s = ((a \ll 1) \wedge t) \boxplus (a \wedge (s'_{000} \gg 1))\,, && q = \big((\neg((a \ll 1) \oplus u \oplus v)) \gg 1\big) \wedge t'\,,\\
&d = \mathsf{RevCarry}(s'_{000}, q) \vee q\,, && w = (q \boxminus (s \wedge d)) \vee (s \wedge \neg d)\,.
\end{aligned}$$

Then, for all states $S_i = \texttt{11*}$ with $i \in \mathcal{I}$, $w[i-1, i-\ell_i] = p_i$.

Proof. For each $i \in \mathcal{I}$ and $0 \le j < n$, note that $s'_{000}[j] = 1$ exactly when $S_j = \texttt{000}$ and $S_j \in \Gamma_i$, and $t[j] = 1$ (resp. $t'[j] = 1$) if and only if $S_j = S_{i-\ell_i}$ (resp. $S_j = S_{i-1}$). Denoting $s = s_1 \boxplus s_2$, where $s_1 = (a \ll 1) \wedge t$ and $s_2 = a \wedge (s'_{000} \gg 1)$, when $i \in \mathcal{I}$ the sub-vectors

$$\begin{aligned}
s_1[i-1, i-\ell_i-1] &= (0,\, 0,\, \ldots,\, 0,\, a[i-\ell_i-1],\, 0)\,,\\
s_2[i-1, i-\ell_i-1] &= (0,\, a[i-2],\, \ldots,\, a[i-\ell_i+1],\, a[i-\ell_i],\, 0)\,,
\end{aligned}$$

result in $s[i-1, i-\ell_i] = a[i-2, i-\ell_i] + a[i-\ell_i-1]$. In particular, $s[i-1, i-\ell_i] \le 2^{\ell_i - 1}$ and the equality holds when $s[i-1, i-\ell_i] = \texttt{10...0}$.

It is easy to see that $q[i-1] = \neg(a[i-2] \oplus u[i-1] \oplus v[i-1])$ when $i \in \mathcal{I}$ and q is zero elsewhere. Then, the sub-vectors $d[i-1, i-\ell_i]$ are composed of repeated copies of $q[i-1]$ when $i \in \mathcal{I}$, as shown by the following sub-vectors

$$\begin{aligned}
s'_{000}[i, i-\ell_i-1] &= (0,\, 1,\, 1,\, \ldots,\, 1,\, 0,\, *)\,,\\
q[i, i-\ell_i-1] &= (0,\, q[i-1],\, 0,\, \ldots,\, 0,\, 0,\, *)\,,\\
\mathsf{RevCarry}(s'_{000}, q)[i, i-\ell_i-1] &= (*,\, 0,\, q[i-1],\, \ldots,\, q[i-1],\, q[i-1],\, 0)\,,\\
d[i, i-\ell_i-1] &= (*,\, q[i-1],\, q[i-1],\, \ldots,\, q[i-1],\, q[i-1],\, *)\,.
\end{aligned}$$

The only exception for the above equations is when $i - \ell_i = -1$, where the two least significant bits of the above sub-vectors will be equal to zero.

Let $w = w_1 \wedge w_2$, where $w_1 = q \boxminus (s \wedge d)$ and $w_2 = s \wedge \neg d$. Regarding the acquired patterns for q and d, we prove the following inequalities for $i \in \mathcal{I}$

$$\begin{aligned}
(s \wedge d)[i-1, i-\ell_i] &\le q[i-1, i-\ell_i]\,,\\
(s \wedge d)[i-\ell_i-1, 0] &\le q[i-\ell_i-1, 0]\,,
\end{aligned}$$

which imply the identity $w_1[i-1, i-\ell_i] = q[i-1, i-\ell_i] \boxminus (s \wedge d)[i-1, i-\ell_i]$.

The first inequality can be derived from the fact that $s[i-1, i-\ell_i] \le \texttt{10...0}$. For the second inequality, consider the index set $\mathcal{J} = \{j | \forall i \in I, S_j \notin \Gamma_i\}$. Then, the second inequality holds since for $j \in \mathcal{J}$ and $c \in \{0, 1\}$ we can see that

$$s'_{000}[j+1-c] = 0 \implies s_1[j-c] = s_2[j-c] = 0\,.$$

We are now ready to evaluate $w[i-1, i-\ell_i]$ when $i \in \mathcal{I}$. If $q[i-1] = 0$, then $d[i-1, i-\ell_i] = (0, \ldots, 0)$, $w_1[i-1, i-\ell_i]$ reduces to 0, and

$$w[i-1, i-\ell_i] = w_2[i-1, i-\ell_i] = a[i-2, i-\ell_i] + a[i-\ell_i - 1]\,.$$

If $q[i-1] = 1$, then $d[i-1, i-\ell_i] = (1, \ldots, 1)$, $w_2[i-1, i-\ell_i]$ reduces to 0, and

$$\begin{aligned} w[i-1, i-\ell_i] = w_1[i-1, i-\ell_i] &= (1, 0, \ldots, 0) \boxminus s[i-1, i-\ell_i] \\ &= 2^{\ell_i - 1} - (a[i-2, i-\ell_i] + a[i-\ell_i - 1])\,. \end{aligned}$$

Hence, for $q[i-1] = \neg(a[i-1] \oplus u[i] \oplus v[i])$ and regarding Lemma 4, we obtain that $w[i-1, i-\ell_i] = p_i$. □

Recall that both LZ and RevCarry have bit-vector complexity $O(\log_2 n)$. Therefore, w can be described with $O(\log_2 n)$ basic bit-vector operations.

Since p_i is not always a power of two, $\log_2(p_i)$ cannot be represented by a fixed-sized bit-vector. Thus, we will use the following approximation for the binary logarithm of a positive integer x,

$$\mathsf{apxlog}_2(x) \triangleq m + \frac{\mathsf{Truncate}(x[m-1, 0])}{2^4}\,, \tag{4}$$

where $m = \lfloor \log_2(x) \rfloor$ and $\mathsf{Truncate}(z)$ for an m-bit vector z is defined by

$$\mathsf{Truncate}(z) = \begin{cases} z[m-1, m-4], & m \geq 4 \\ z[m-1, 0] \parallel (\overbrace{0, \ldots, 0}^{4-m}), & m < 4 \end{cases}$$

In other words, apxlog_2 includes the integer part of the logarithm and takes the four bits right after the most significant one as the "fraction" bits. While Truncate can be generalized to consider more fraction bits, we will show later that four fraction bits are enough to minimize the bounds of our approximation error.

To describe $\sum_{i \in \mathcal{I}} \mathsf{apxlog}_2(p_i)$ with basic bit-vector operations, we will introduce in the next proposition two new bit-vector functions ParallelLog and ParallelTrunc. Given a bit-vector x with sub-vectors delimited by a bit-vector y, $\mathsf{ParallelLog}(x, y)$ computes the sum of the integer part of the logarithm of the delimited sub-vectors, whereas $\mathsf{ParallelTrunc}(x, y)$ calculates the sum of the four most significant bits of the delimited sub-vectors.

Proposition 1. *Let x and y be n-bit vectors such that y has r sub-vectors*

$$y[i_t, j_t] = (1, 1, \ldots, 1, 0), \quad t = 1, \ldots, r$$

where $i_1 > j_1 > i_2 > j_2 > \cdots > i_r > j_r \geq 0$, and y is equal to zero elsewhere.

We define the bit-vector functions ParallelLog *and* ParallelTrunc *by*

$$\begin{aligned} \mathsf{ParallelLog}(x, y) &= \mathsf{HW}(\mathsf{RevCarry}(x \wedge y, y)) \\ \mathsf{ParallelTrunc}(x, y) &= (\mathsf{HW}(z_0) \ll 3) \boxplus (\mathsf{HW}(z_1) \ll 2) \boxplus (\mathsf{HW}(z_2) \ll 1) \boxplus \mathsf{HW}(z_3) \end{aligned}$$

where $z_\lambda = x \wedge (y \gg 0) \wedge \cdots \wedge (y \gg \lambda) \wedge \neg(y \gg (\lambda + 1))$.

a) If $x[i_t, j_t] > 0$ *for* $t = 1, \ldots, r$*, then*

$$\sum_{t=1}^{r} \lfloor \log_2(x[i_t, j_t]) \rfloor = \mathsf{ParallelLog}(x, y)\,.$$

b) If at least $\lfloor \log_2(n) \rfloor + 4$ *bits are dedicated to* $\mathsf{ParallelTrunc}(x, y)$*, then*

$$\sum_{t=1}^{r} \mathsf{Truncate}(x[i_t, j_t + 1]) = \mathsf{ParallelTrunc}(x, y)\,.$$

Proof. Case a) Let $m = \lfloor \log_2(x[i_1, j_1]) \rfloor$ and $c = \mathsf{RevCarry}(x \wedge y, y)$. Note that $c[n-1, i_1] = 0$, since $y[n-1, i_1 + 1] = 0$. For $m \geq 1$, we obtain the sub-vectors

$$\begin{array}{rl} & i_1, \ldots, j_1 + m + 1,\ j_1 + m,\ j_1 + m - 1,\ \ldots,\ j_1 + 1,\ j_1,\ j_1 - 1 \\ y[i_1, j_1 - 1] = (1, \ldots, & 1, \quad 1, \quad 1, \quad \ldots, \quad 1, \quad 0, \quad *)\,, \\ (x \wedge y)[i_1, j_1 - 1] = (0, \ldots, & 0, \quad 1, \quad *, \quad \ldots, \quad *, \quad 0, \quad *)\,, \\ c[i_1, j_1 - 1] = (0, \ldots, & 0, \quad 0, \quad 1, \quad \ldots, \quad 1, \quad 1, \quad 0)\,. \end{array}$$

In particular, $c[i_1, j_1]$ has m bits set to one. If $m = 0$, $x[i_1, j_1 + 1] = 0$ and $y[j_1] = 0$, which implies that there is no carry chain, i.e., $c[i_1, j_1] = 0$. Therefore, in both cases $\mathsf{HW}(c)[i_1, j_1]) = m = \lfloor \log_2(x[i_1, j_1]) \rfloor$.

Note that the reversed carry chain stops at j_1, and $c[j_1 - 1, i_2] = \mathtt{0} \cdots \mathtt{0}$. Therefore, the same argument can be applied for $t = 2, \ldots, r$, obtaining

$$\mathsf{HW}(c[i_t, j_t]) = \lfloor \log_2(x[i_t, j_t]) \rfloor\,, \quad c[j_t - 1, i_{t+1}] = 0\,.$$

Finally, it is easy to see that $c[j_r - 1, 0] = 0$, concluding the proof for this case.

Case b) First note that for $\lambda = 0, \ldots, 3$ and $t = 1, \ldots, r$, the variable z_λ is

$$z_\lambda[i] = \begin{cases} x[i], & \text{if } i = i_t - \lambda > j_t \\ 0, & \text{otherwise} \end{cases}$$

Therefore, the hamming weight of z_λ computes the following summation:

$$\mathsf{HW}(z_\lambda) = \sum_{\substack{t \\ i_t - \lambda > j_t}} x[i_t - \lambda]\,.$$

While we define HW as a bit-vector function returning an n-bit output given an n-bit input, $\lfloor \log_2(n) \rfloor + 1$ bits are sufficient to represent the output of HW. Therefore, by representing each $\mathsf{HW}(z_\lambda) \ll (3 - \lambda)$ in a $(\lfloor \log_2(n) \rfloor + 4)$-bit variable h_λ, the bit-vector expression $h_0 \boxplus h_1 \boxplus h_2 \boxplus h_3$ does not overflow, and we obtain

$$\sum_{t=1}^{r} \mathsf{Truncate}(x[i_t, j_t + 1]) = \sum_{t=1}^{r} \sum_{\substack{\lambda=0 \\ i_t - \lambda > j_t}}^{3} x[i_t - \lambda] \times 2^{3-\lambda} = h_0 \boxplus h_1 \boxplus h_2 \boxplus h_3\,,$$

which concludes the proof. □

Since both HW and Rev have $O(\log_2 n)$ bit-vector complexities, so do the functions $\mathsf{ParallelLog}$ and $\mathsf{ParallelTrunc}$. The next lemma applies $\mathsf{ParallelLog}$ and $\mathsf{ParallelTrunc}$ to provide a bit-vector expression of the sum of $\mathsf{apxlog}_2(p_i)$.

Lemma 7. *Let r and f be the bit-vectors given by*

$$r = \mathsf{ParallelLog}((w \wedge s'_{000}) \ll 1, s'_{000} \ll 1)\,,$$
$$f = \mathsf{ParallelTrunc}(w \ll 1, \mathsf{RevCarry}((w \wedge s'_{000}) \ll 1, s'_{000} \ll 1))\,.$$

If at least $\lfloor \log_2(n) \rfloor + 5$ bits are dedicated to r and f, then

$$2^4 \sum_{i \in \mathcal{I}} \mathsf{apxlog}_2(p_i) = (r \ll 4) \boxplus f\,.$$

Proof. Regarding Lemma 6, $w[i-1, i-\ell_i]$ represents the ℓ_i-bit vector of p_i and $s'_{000}[i-1, i-\ell_i]$ conforms to the pattern $(1, \cdots, 1, 0)$ for any $i \in \mathcal{I}$. Therefore,

$$\sum_{i \in \mathcal{I}} \lfloor \log_2(p_i) \rfloor = \mathsf{HW}\big(\mathsf{RevCarry}((w \wedge s'_{000}) \ll 1, s'_{000} \ll 1)\big)\,,$$

following Proposition 1. For the second case, let c be the n-bit vector given by $c = \mathsf{RevCarry}((w \wedge s'_{000}) \ll 1, s'_{000} \ll 1)$. Denoting by $j = i - l_i$ and $m = \lfloor \log_2(p_i) \rfloor$ for a given $i \in \mathcal{I}$, note that $p_i[m]$ is the most significant active bit of p_i and

$$\begin{array}{rccccccccc} & i{+}1, \ldots, & j{+}m{+}2, & j{+}m{+}1, & j{+}m, & \ldots, & j{+}2, & j{+}1, & j \\ (w \ll 1)[i{+}1, j] = & (0, \ldots, & 0 & p_i[m], & p_i[m{-}1], \ldots, & p_i[1], & p_i[0] & 0)\,, \\ c[i{+}1, j] = & (0, \ldots, & 0 & 0, & 1, \ldots, & 1, & 1 & 0)\,. \end{array}$$

Thus $c[j+m, j]$ conforms to the pattern $(1, \cdots, 1, 0)$ and Proposition 1 leads to

$$\sum_{\substack{i \in \mathcal{I} \\ m = \lfloor \log_2(p_i) \rfloor}} \mathsf{Truncate}(p_i[m-1, 0]) = \mathsf{ParallelTrunc}(w \ll 1, c)\,.$$

For any n-bit variables x and y, it is easy to see that $\mathsf{ParallelLog}(x, y) < n$. Thus, $\lfloor \log_2(n) \rfloor + 4$ bits are sufficient to represent $(r \ll 4)$, and f can also be represented with the same number of bits following Proposition 1. Therefore, by representing $(r \ll 4)$ and f in $(\lfloor \log_2(n) \rfloor + 5)$-bit variables, the bit-vector expression $(r \ll 4) \boxplus f$ does not overflow. □

Recall that the differential weight of constant addition can be decomposed as

$$\mathsf{weight}_{\boxplus_a}(u, v) = -\sum_{i \notin \mathcal{I}} \log_2(\varphi_i) - \sum_{i \in \mathcal{I}} \log_2\left(\frac{1}{2^{\ell_i - 1}}\right) - \sum_{i \in \mathcal{I}} \log_2(p_i)\,.$$

If the binary logarithm of p_i is replaced by our approximation of the binary logarithm $\mathsf{apxlog}_2(p_i)$, we obtain the following approximation of the weight,

$$\mathsf{apxweight}_{\boxplus_a}(u, v) \triangleq -\sum_{i \notin \mathcal{I}} \log_2(\varphi_i) - \sum_{i \in \mathcal{I}} \log_2\left(\frac{1}{2^{\ell_i - 1}}\right) - \sum_{i \in \mathcal{I}} \mathsf{apxlog}_2(p_i)\,. \quad (5)$$

Our weight approximation can be computed with the bit-vector function $\mathsf{BvWeight}$ described in Algorithm 1, as shown in the lemma.

Algorithm 1 Bit-vector function $\mathsf{BvWeight}(u, v, a)$.

Input: (u, v, a)
Output: $\mathsf{BvWeight}(u, v, a)$

$s_{000} \leftarrow \neg(u \ll 1) \wedge \neg(v \ll 1)$
$s'_{000} \leftarrow s_{000} \wedge \neg\,\mathsf{LZ}(\neg s_{000})$
$t \leftarrow \neg s'_{000} \wedge (s'_{000} \gg 1)$
$t' \leftarrow s'_{000} \wedge (\neg(s'_{000} \gg 1))$
$s \leftarrow ((a \ll 1) \wedge t) \boxplus (a \wedge (s'_{000} \gg 1))$
$q \leftarrow \big((\neg((a \ll 1) \oplus u \oplus v)) \gg 1\big) \wedge t'$
$d \leftarrow \mathsf{RevCarry}(s'_{000}, q) \vee q$
$w \leftarrow (q \boxminus (s \wedge d)) \vee (s \wedge \neg d)$
$int \leftarrow \mathsf{HW}((u \oplus v) \ll 1) \boxplus \mathsf{HW}(s'_{000}) \boxminus \mathsf{ParallelLog}((w \wedge s'_{000}) \ll 1, s'_{000} \ll 1)$
$frac \leftarrow \mathsf{ParallelTrunc}(w \ll 1, \mathsf{RevCarry}((w \wedge s'_{000}) \ll 1, s'_{000} \ll 1))$
return $(int \ll 4) \boxminus frac$

Lemma 8. *If at least* $\lfloor \log_2(n) \rfloor + 5$ *bits are dedicated to* $\mathsf{BvWeight}(u, v, a)$*, then*

$$2^4\, \mathsf{apxweight}_{\boxplus_a}(u, v) = \mathsf{BvWeight}(u, v, a)\,.$$

Proof. Regarding Lemma 3 and Lemmas 5 and 7 we respectively obtain

$$-\sum_{i \notin \mathcal{I}} \log_2(\varphi_i) = \mathsf{HW}((u \oplus v) \ll 1)\,, \quad -\sum_{i \in \mathcal{I}} \log_2\left(\frac{1}{2^{\ell_i - 1}}\right) = \mathsf{HW}(s'_{000})\,,$$

$$2^4 \sum_{i \in \mathcal{I}} \mathsf{apxlog}_2(p_i) = (\mathsf{ParallelLog}((w \wedge s'_{000}) \ll 1, s'_{000} \ll 1) \ll 4) \boxplus frac\,.$$

All in all, we get the following identities,

$$2^4\, \mathsf{apxweight}_{\boxplus_a}(u, v) = 2^4((int \ll 4) \boxminus frac) = \mathsf{BvWeight}(u, v, a)\,. \qquad \square$$

Note that the four least significant bits of $\mathsf{BvWeight}(u, v, a)$ correspond to the fraction bits of the approximate weight. In other words, the output of $\mathsf{BvWeight}(u, v, a)$ represents the rational value

$$\sum_{i=0}^{\lfloor \log_2(n) \rfloor + 4} 2^{i-4}\, \mathsf{BvWeight}(u, v, a)[i]\,.$$

The bit-vector complexity of BvWeight is dominated by the complexity of LZ, Rev, HW, ParallelLog, and ParallelTrunc. Since these operations can be computed with $O(\log_2 n)$ basic bit-vector operations, so does BvWeight.

Theorem 4 shows that BvWeight leads to a close approximation of the differential weight and provides explicit bounds for the approximation error.

Theorem 4. *Let* (u, v) *be a valid differential over the* n*-bit constant addition* $\boxplus_a$*, let* $\mathsf{weight}_{\boxplus_a}(u, v)$ *be the differential weight of* (u, v)*, and let* BvWeight *be the bit-vector function defined by Algorithm 1. Then, the approximation error,*

$$E = \mathsf{weight}_{\boxplus_a}(u, v) - \mathsf{apxweight}_{\boxplus_a}(u, v) = \mathsf{weight}_{\boxplus_a}(u, v) - 2^{-4}\, \mathsf{BvWeight}(u, v, a)$$

is bounded by $-0.029(n-1) \leq E \leq 0$.

Section 3.3 is devoted to the proof of Theorem 4, where we will also analyse the error caused by our approximated binary logarithm. Before proving Theorem 4, we will describe an example to understand this theorem and Algorithm 1.

Example 3. Consider the same conditions as defined in Example 1, i.e., $(u, v) = (\texttt{1010001110}, \texttt{1010001010})$ and the constant input $a = \texttt{1000101110}$. The weight for our differential is

$$\mathsf{weight}_{\boxplus_a}(u, v) = -\log_2(\Pr[u \xrightarrow{\boxplus_a} v]) = -\log_2(\frac{5}{16}) \approx 1.678\,.$$

Let's find the approximate weight $\mathsf{apxweight}_{\boxplus_a}$ based on Algorithm 1. Table 4 presents some of the variables we obtain to compute the aforementioned approximate weight.

Table 4. Intermediate variables for computing $\mathsf{apxweight}_{\boxplus_a}(u, v)$ of Example 3.

i	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
$a[i]$	1	0	0	0	1	0	1	1	1	0
$u[i]$	1	0	1	0	0	0	1	1	1	0
$v[i]$	1	0	1	0	0	0	1	0	1	0
$s_{000}[i]$	1	0	1	1	1	0	0	0	1	1
$s'_{000}[i]$	0	0	1	1	1	0	0	0	1	1
$w[i]$	0	0	0	1	0	1	0	0	1	0
$\mathsf{BvWeight}(u, v, a)[i]$	0	0	0	0	0	1	1	1	0	0

The set $\mathcal{I} = \{1 \leq i \leq n-1 \mid S_i = \texttt{11*},\ \ell_i > 1\}$ in this example is equal to $\mathcal{I} = \{2, 8\}$. The variable *int* in Algorithm 1 consists of three parts. The first part of the variable *int* calculated in Algorithm 1 is

$$\mathsf{HW}((u \oplus v) \ll 1) = \texttt{0000000001} = 1\,,$$

which is equal to $-\sum_{i \notin \mathcal{I}} \log_2(\varphi_i) = -\sum_{i \in \{0,1,3,4,5,6,7,9\}} \log_2(\varphi_i) = -\log_2(\varphi_3)$.

The second part uses the variable s'_{000} which is the same as s_{000} except that the leading one bits of s_{000} are replaced in s'_{000} by zeros. In other words, $s'_{000}[9] = 0$ but the remaining bits of these two bit-vectors are exactly equal. Computing the second part results in

$$\mathsf{HW}(s'_{000}) = \texttt{0000000101} = 5\,,$$

and it is equal to $\sum_{i \in \mathcal{I}}(\ell_i - 1) = (\ell_2 - 1) + (\ell_8 - 1)$.

For the third and last part of int, we compute w, obtaining $w = \texttt{0001010010}$. We remark that the bit-vector w for $i \in \mathcal{I}$ in fact includes p_i as some subvectors, i.e.

$$i = 2: \; w[2-1, 2-\ell_2] = w[1,-1] = \texttt{100} = 4 = p_2\,,$$
$$i = 8: \; w[8-1, 8-\ell_8] = w[7,4] = \texttt{0101} = 5 = p_8\,.$$

Hence, the third part of int is computed as

$$\mathsf{ParallelLog}((w \wedge s'_{\texttt{000}}) \ll 1, s'_{\texttt{000}} \ll 1) = \texttt{0000000100} = 4\,,$$

which is equal to $\sum_{i\in\mathcal{I}} \lfloor \log_2(p_i) \rfloor = \lfloor \log_2(p_2) \rfloor + \lfloor \log_2(p_8) \rfloor$. Considering all three parts, the bit-vector int is obtained by

$$int = \texttt{0000000001} \boxplus \texttt{0000000101} \boxminus \texttt{0000000100} = \texttt{0000000010} = 2\,.$$

Moreover, the $frac$ bit-vector in Algorithm 1 is calculated by

$$\begin{aligned} frac &= \mathsf{ParallelTrunc}(w \ll 1, \mathsf{RevCarry}((w \wedge s'_{\texttt{000}}) \ll 1, s'_{\texttt{000}} \ll 1))\,, \\ &= \mathsf{HW}(\texttt{0000100000}) \ll 2 = 4\,. \end{aligned}$$

Considering the third major operand of int and the bit-vector $frac$, we can obtain the summation of all approximate logarithms

$$\sum_{i\in\mathcal{I}} \mathsf{apxlog}_2(p_i) = \mathsf{apxlog}_2(p_2) + \mathsf{apxlog}_2(p_8) = 4 + \frac{4}{2^4} = 4.25\,.$$

To summarize, the output of Algorithm 1 is

$$\begin{aligned} \mathsf{BvWeight}(u, v, a) &= (int \ll 4) \boxminus frac\,, \\ &= \texttt{0000100000} \boxminus \texttt{0000000100} = \texttt{0000011100} = 28\,. \end{aligned}$$

Therefore, the approximate weight will be equal to

$$\mathsf{apxweight}_{\boxplus_a}(u, v) = 2^{-4}\,\mathsf{BvWeight}(u, v, a) = \frac{28}{2^4} = 1.75\,.$$

The total error of our approximation is

$$E = \mathsf{weight}_{\boxplus_a}(u, v) - \mathsf{apxweight}_{\boxplus_a}(u, v) \approx 1.678 - 1.75 = -0.072\,,$$

which is a negative value and lower bounded by $-0.029(n-1) = -0.261$ as suggested by Theorem 4.

3.3 Error Analysis - Proof of Theorem 4

In this subsection, we will prove Theorem 4 by gradually analysing the error produced by our approximation of the binary logarithm. As we can see from Equations (3) and (5), the gap between $\mathsf{weight}_{\boxplus_a}(u, v)$ and $\mathsf{apxweight}_{\boxplus_a}(u, v)$ is

$$\mathsf{weight}_{\boxplus_a}(u, v) - \mathsf{apxweight}_{\boxplus_a}(u, v) = -\sum_{i\in\mathcal{I}} \left(\log_2(p_i) - \mathsf{apxlog}_2(p_i)\right).$$

Note that the integer part of apxlog_2 is equal to the integer part of $\log_2$ and the error is caused by the fraction part of the logarithm.

Given a positive integer x and the corresponding $m = \lfloor \log_2(x) \rfloor$, we define apxlog_2^κ as

$$\mathsf{apxlog}_2^\kappa(x) = \begin{cases} m + x[m-1,0]/2^m, & m \leq \kappa \\ m + x[m-1, x-\kappa]/2^\kappa, & m > \kappa \end{cases}$$

The non-negative integer κ is called the precision of the fraction part. Note that apxlog_2^κ is the generalization of apxlog_2, which considers $\kappa = 4$ bits for the fraction part. While Theorem 4 only focuses on $\kappa = 4$, we will use apxlog_2^κ in this section to additionally prove that our error bound also applies to $\kappa \geq 4$.

The following lemma bounds the approximation error of apxlog_2 when $\kappa \geq \lfloor \log_2(x) \rfloor$, with a similar proof as [47] for the sake of completeness. The main idea is that after extracting integer part of the logarithm in base 2, one can estimate $\log_2(1+\gamma)$ by γ when $0 \leq \gamma < 1$.

Lemma 9. *Consider a positive integer x and the binary logarithm approximation $\log_2(x) \approx m + x[m-1,0]/2^m$, where $m = \lfloor \log_2(x) \rfloor$. Then, the approximation error $e = \log_2(x) - (m + x[m-1,0]/2^m)$ is bounded by $0 \leq e \leq B$, where B is given by*

$$B = 1 - \big(1 + \ln(\ln(2))\big) / \ln(2) \approx 0.086\,.$$

Proof. Let $x = 2^m + b$, where b is a non-negative integer such that $0 \leq b < 2^m$. Therefore, $x[m-1,0] = x - 2^m = b$ and the error is given by

$$e = \log_2(x) - (m + \frac{x[m-1,0]}{2^m}) = \log_2(2^m + b) - (m + \frac{b}{2^m}) = \log_2(1 + \frac{b}{2^m}) - \frac{b}{2^m}\,.$$

For $\gamma = b/2^m$, we obtain $0 \leq \gamma < 1$ and $e = \log_2(1+\gamma) - \gamma$. Note that e is a concave function of γ where $e \geq 0$ if and only if $0 \leq \gamma \leq 1$. By deriving $e = e(\gamma)$, one can see that $\max(e) = B = 1 - \big(1 + \ln(\ln(2))\big) / \ln(2) \approx 0.086$ is reached when $\gamma = 1/\ln(2) - 1 \approx 0.44$. □

The bound B is an almost tight bound, e.g., when $x = 3$, the obtained error is $\log_2(3) - (1 + \frac{1}{2}) \cong 0.085$. The following example sheds more light on our binary logarithm approximation.

Example 4. Consider the positive integer $x = \texttt{11101}$. Note that $\log_2(x) \cong 4.85798$ and $m = \lfloor (\log_2(x)) \rfloor = 4$. In order to obtain the approximation defined in Lemma 9, we first find and omit the greatest "one" in binary representation of x, and we get $x[m-1,0] = \texttt{1101}$. By interpreting the remaining bits as a binary fraction, we have $x[m-1,0]/2^m = \texttt{0.1101}$. Therefore, the approximated binary logarithm of $x = \texttt{11101}$ in binary representation is

$$m + \frac{x[m-1,0]}{2^m} = \texttt{100.1101}\,,$$

which is equal to 4.8125. In addition, the corresponding error of such approximation is $e \approx 0.04548$, which is a positive value and upper bounded by $B \approx 0.086$.

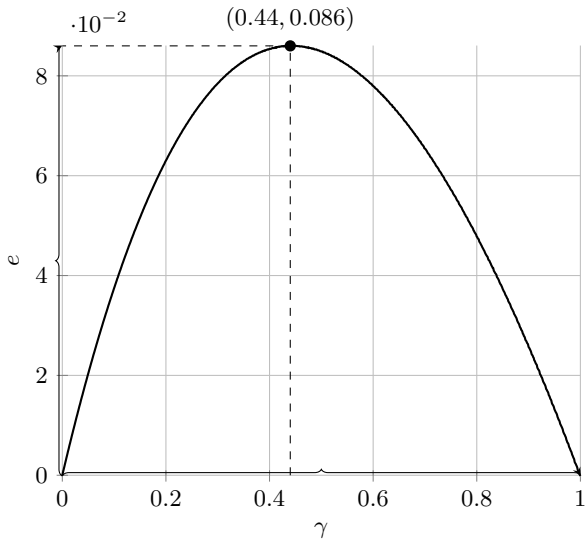


Fig. 1. The error $e = \log_2(1+\gamma) - \gamma$, over $0 \le \gamma < 1$.

Finally, we can now prove Theorem 4, which basically states that if we dedicate 4 bits to the fraction precision κ, the approximation error E is bounded by $-0.029 \cdot (n-1) \le E \le 0$. While Theorem 4 focuses on $\kappa = 4$, we will show in the proof that we can also bound the error for $\kappa \ge 4$. To this end, we generalize the approximated weight $\mathsf{apxweight}_{\boxplus_a}$ and the approximated weight error E_κ as follows

$$\mathsf{apxweight}^{\kappa}_{\boxplus_a}(u,v) = -\Big(\sum_{i\in\mathcal{I}} \mathsf{apxlog}^{\kappa}_2(p_i) + \sum_{i\in\mathcal{I}} \log_2(\tfrac{1}{2^{\ell_i-1}}) + \sum_{i\notin\mathcal{I}} \log_2(\varphi_i)\Big)$$
$$E_\kappa = \mathsf{weight}_{\boxplus_a}(u,v) - \mathsf{apxweight}^{\kappa}_{\boxplus_a}(u,v)\,,$$

where $\mathsf{apxweight}^{4}_{\boxplus_a}(u,v) = \mathsf{apxweight}_{\boxplus_a}(u,v)$ is defined by Equation (5) and $E_4 = E$ is defined in Theorem 4.

Proof (Proof (Theorem 4)). First, we mention that $\log_2(\varphi_i)$ is an integer number when $S_i \neq \texttt{11*}$ or for $S_i = \texttt{11*}$ we see $\ell_i < 3$. For these cases, $\log_2(\varphi_i) = \lfloor \log_2(\varphi_i) \rfloor$ and the approximation error is equal to zero.

Next, for each $i \in \mathcal{I}$ when $\ell_i \ge 3$, let $p_i = 2^{m_i} + b_i$ such that m_i and b_i are two non-negative integers, $m_i \le \ell_i - 2$ and $0 \le b_i < 2^{m_i}$. If $\ell_i \le \kappa + 2$, we obtain $m_i \le \kappa$ and $\mathsf{apxlog}^{\kappa}_2(p_i) = m_i + b_i \cdot 2^{-m_i}$. Thus, the resulting error

$$e_i = \log_2(p_i) - \mathsf{apxlog}^{\kappa}_2(p_i) = \log_2(p_i) - (m_i + b_i \cdot 2^{-m_i})$$

is exactly the same as the error defined in Lemma 9, and $0 \le e_i \le B \approx 0.086$.

On the other hand, for $m_i > \kappa$, i.e., $\ell_i \ge \kappa + 3$, let $p_i = 2^{m_i} + t_i \cdot 2^{m_i-\kappa} + \zeta_i$, where t_i and ζ_i are two non-negative integers such that $0 \le t_i < 2^\kappa$ as well as

$0 \leq \zeta_i < 2^{m_i-\kappa}$. In this case, the approximated binary logarithm is $\mathsf{apxlog}_2^{\kappa}(p_i) = m_i + t_i \cdot 2^{-\kappa}$. We now define a new error e'_i as

$$e'_i = \log_2(p_i) - \mathsf{apxlog}_2^{\kappa}(p_i) = \log_2(1 + t_i \cdot 2^{-\kappa} + \zeta_i \cdot 2^{-m_i}) - t_i \cdot 2^{-\kappa}\,.$$

Due to the fact that $\zeta_i \geq 0$, we can see that

$$e'_i = \log_2(p_i) - (m_i + t_i \cdot 2^{-\kappa}) \geq \log_2(p_i) - (m_i + t_i \cdot 2^{-\kappa} + \zeta_i \cdot 2^{-m_i}) = e_i \geq 0\,.$$

Since $\zeta_i < 2^{m_i-\kappa}$ and by reforming the error, we obtain the upper bound of e'_i

$$e'_i \leq \log_2(1 + t_i \cdot 2^{-\kappa} + 2^{-\kappa}) - t_i \cdot 2^{-\kappa} = (\log_2(1+\gamma'_i) - \gamma'_i) + 2^{-\kappa}\,,$$

where $\gamma'_i = (t_i + 1) \cdot 2^{-\kappa}$ and $2^{-\kappa} \leq \gamma'_i < 1$. Regarding Lemma 9, the new error e'_i is bounded by $0 \leq e'_i \leq B + 2^{-\kappa}$.

Note that for a valid differential (u, v) over the constant addition $\boxplus_a$ we can see that $S_0 = \mathtt{000}$ which for some i^* belongs to the chain

$$\Gamma_{i^*} = \{S_{i^*-1}, \cdots, S_0, S_{-1} = \perp\}, \quad \ell_{i^*} = i^* + 1\,.$$

Hence, we obtain

$$\delta_{i^*-1} = \frac{a[i^* - \ell_{i^*} - 1]}{2^{\ell_{i^*}-1}} + \sum_{j=2}^{\ell_{i^*}} \frac{a[i^* - j]}{2^{j-1}} = \sum_{j=2}^{\ell_{i^*}-1} \frac{a[i^* - j]}{2^{j-1}}\,.$$

Similar to the proof of Lemma 4 we have

$$\varphi_{i^*} = \frac{p_{i^*}}{2^{\ell_{i^*}-1}} = \frac{p^*_{i^*}}{2^{\ell_{i^*}-2}}\,,$$

where $p^*_{i^*} = p_{i^*}/2$ is an integer. Therefore, by replacing ℓ_{i^*} with $\ell^*_{i^*} = \ell_{i^*} - 1$, the previous statements considering the error bounds of our approximation for the state $S_{i^*} = \mathtt{11*}$ and its new $\ell^*_{i^*}$ are still correct. We now define two bits $\mathtt{b}$ and $\mathtt{b}'$ as

$$\mathtt{b} = \begin{cases} 1, & 3 \leq \ell^*_{i^*} \leq \kappa + 2 \\ 0, & o.w. \end{cases}, \qquad \mathtt{b}' = \begin{cases} 1, & \ell^*_{i^*} > \kappa + 2 \\ 0, & o.w. \end{cases}$$

Finally, by defining the conditional index set $\mathcal{I}_{\alpha}^{\beta} = \{i \in \mathcal{I} - \{i^*\} \mid \alpha \leq \ell_i \leq \beta\}$ we obtain

$$\begin{aligned}
E_\kappa &= \mathsf{weight}_{\boxplus_a}(u, v) - \mathsf{apxweight}^{\kappa}_{\boxplus_a}(u, v) \\
&= -\sum_{i \in \mathcal{I}} (\log_2(p_i) - \mathsf{apxlog}_2^{\kappa}(p_i)) \\
&= -\Big(\sum_{i \in \mathcal{I}_3^{\kappa+2}} e_i + \sum_{i \in \mathcal{I}_{\kappa+3}^{n}} e'_i + \mathtt{b} e_{i^*} + \mathtt{b}' e'_{i^*} \Big) \\
&\geq -\Big(B \sum_{i \in \mathcal{I}_3^{\kappa+2}} 1 \;+\; (B + 2^{-\kappa}) \sum_{i \in \mathcal{I}_{\kappa+3}^{n}} 1 + \mathtt{b} B + \mathtt{b}'(B + 2^{-\kappa}) \Big) \\
&\geq -\Big(\frac{B}{3} \sum_{i \in \mathcal{I}_3^{\kappa+2}} \ell_i \;+\; (\frac{B + 2^{-\kappa}}{\kappa + 3}) \sum_{i \in \mathcal{I}_{\kappa+3}^{n}} \ell_i + \mathtt{b} \frac{B}{3} \ell^*_{i^*} + \mathtt{b}' (\frac{B + 2^{-\kappa}}{\kappa + 3}) \ell^*_{i^*} \Big)\,.
\end{aligned}$$

For $\kappa \geq 4$, we can see that $\frac{B + 2^{-\kappa}}{\kappa + 3} \leq \frac{B}{3}$, resulting in

$$0 \geq E_\kappa \geq -\big(\frac{B}{3}\sum_{i \in \mathcal{I}_3^n} \ell_i + \frac{B}{3}\ell_{i^*}^*\big) = -\big(\frac{B}{3}\sum_{i \in \mathcal{I}_3^n} \ell_i + \frac{B}{3}(\ell_{i^*} - 1)\big)$$
$$\geq -\frac{B}{3}(n-1) \approx -0.029(n-1)\,.$$

Since for $\kappa = 4$, we have $E_4 = E = \mathsf{weight}_{\boxplus_a}(u, v) - \mathsf{apxweight}_{\boxplus_a}(u, v)$, the above inequalities hold for the approximation error E as well. □

While dedicating $\kappa = 4$ bits as the fraction precision is enough to obtain the same error bounds as $\kappa > 4$, considering $\kappa < 4$ creates a trade-off between the lower bound of the error and the complexity of Algorithm 1. As an example, choosing $\kappa = 3$ removes one HW call in Algorithm 1. However, by following the proof of Theorem 4 for $\kappa = 3$, the error will be lower bounded by $-0.035(n-1)$, which potentially is an acceptable trade-off.

The differential model of the constant addition as well as the approximation error will be used in the automated method that we will present in the next section to search for characteristics and impossible differentials of ARX ciphers.

4 SMT-based Search of Characteristics and Impossible Differentials

In this section, we describe how to formulate the search of optimal characteristics and impossible differentials as a sequence of SMT problems, which can be solved by an off-the-shelf SMT solver such as Boolector [48] or STP [49]. Our methods are inspired by the approach of Mouha and Preneel to search for single-key characteristics of Salsa20 [22] and the approach by Sasaki and Todo to search for impossible differentials using Mixed Integer Linear Programming (MILP) [24].

4.1 Searching for Characteristics

To search for characteristics up to probability 2^{-n}, the probability space is decomposed into n intervals $I_w = \left(2^{-w-1}, 2^{-w}\right]$, where $w = 0, 1, \ldots, n-1$, and for each interval, the decision problem of whether there exists a characteristic with probability $p \in I_w$ is encoded as an SMT problem. Note that a characteristic Ω has probability $p \in I_w$ if and only if its integer weight $\lfloor \mathsf{weight}(\Omega) \rfloor$ is equal to w. Section 4.2 describes the encoding process for an ARX cipher.

The SMT problems are provided to the SMT solver, which checks their satisfiability in increasing weight order. When the SMT solver finds the first satisfiable problem, an assignment of the variables that makes the problem satisfiable is obtained, and the search finishes. The assignment contains a characteristic with integer weight $\hat{w}$, and it is optimal in the sense that there are no characteristics with integer weight strictly smaller than $\hat{w}$. If the n SMT problems are found to

be unsatisfiable, then it is proved there are no characteristics with probability higher than 2^{-n}.

To speed up the search, we perform the search iteratively on round-reduced versions of the cipher. First, we search for an optimal characteristic for a small number of rounds r; let $\hat{w}$ denote its integer weight. Then, we search for an optimal $(r+1)$-round characteristic, but skipping the SMT problems with weight strictly less than $\hat{w}$. Since these SMT problems were found to be unsatisfiable for r rounds, they will also be unsatisfiable for $r+1$ rounds. This procedure is repeated until the total number of rounds is reached. Algorithm 2 describes in pseudo-code this search strategy. Our strategy prioritises SMT problems with low weight and small number of rounds, which are faster to solve. In addition, our search also finds optimal characteristics of round-reduced versions, which can be used in other differential-based attacks, such as rectangle or rebound attacks [50, 51].

Algorithm 2 SMT-based optimal characteristic search

```
Ω ← ∅
ŵ ← 0
for r = 1, ..., max_rounds do
    for w = ŵ, ŵ + 1, ..., n − 1 do
        P ← CreateSMTProblem(rounds = r, target_weight = w)
        if SMTSolver.IsSatisfiable(SMT_problem = P) then
            ŵ ← w
            Ω ← SMTSolver.GetAssignment(SMT_problem = P)
            Print(Ω)                          ▷ optimal r-round characteristic
            break
return Ω, ŵ
```

This automated method can be used to search for either single-key or related-key characteristics. Furthermore, additional SMT constraints can be added to the SMT problems in order to search for different types of characteristics. For related-key characteristics and by default, this method searches characteristics minimizing the total weight $\mathsf{weight}(\Omega) = \mathsf{weight}(\Omega_{KS}) + \mathsf{weight}(\Omega_E)$. Strong related-key characteristics can be searched by adding the constraint $\mathsf{weight}(\Omega_{KS}) = 0$ in the SMT problems. Similarly, equivalent keys can be found by adding the constraint $\mathsf{weight}(\Omega_E) = 0$.

Algorithm 2 returns the characteristic with the minimum SMT integer weight, obtained from the bit-vector differential models used within the SMT problems. When some of these models compute approximations of the intermediate weights, the SMT integer weight and the actual integer weight of the characteristic might differ, and the returned characteristic might not be optimal. However, if we have alternative models that compute the exact intermediate weights (that cannot be represented in the SMT problems) and a bound for the error of the SMT integer weight, Algorithm 2 can be adapted to obtain the optimal characteristic as follows.

First, Algorithm 2 is used to obtain the characteristic Ω with the minimum SMT integer weight $\hat{w}$. Then, one finds all[6] characteristics with SMT integer weights $\{\hat{w}, \hat{w}+1, \ldots, \hat{w}+\lfloor\epsilon\rfloor\}$, where ϵ is the absolute bound for the error of the SMT integer weight. Finally, the weights of the found characteristics are recomputed with the alternative models, and the characteristic with the minimum integer weight is returned. This adaptation can be used for ARX ciphers with constant additions, as the error bound can be computed from Theorem 4 and Machado's algorithm [34] can be used to compute the exact weights of the constant additions.

This method only ensures optimality if the differential probabilities over each round are independent and the characteristic probability does not strongly depend on the choice of the secret key. When these assumptions do not hold for a cipher, we empirically compute the weight of each characteristic found by sampling many input pairs satisfying the input difference and counting those satisfying the difference trail. In this case, this method provides a practical heuristic to find characteristics with high probability, and it is one of the best systematic approaches for some families of ciphers, such as ARX.

4.2 Encoding the SMT problems

In this section, we explain how to formulate the decision problem of determining whether a characteristic Ω exists with integer weight W of an ARX cipher as an SMT problem in the bit-vector theory.

First, the ARX cipher is represented in Static Single Assignment (SSA) form, that is, as an ordered list of instructions $y \leftarrow f(x)$ such that each variable is assigned exactly once and each instruction is a modular addition, a rotation, or an XOR.

For each variable x in the SSA representation, a bit-vector variable Δ_x denoting the difference of x is defined in the SMT problem. Then, for every instruction $y \leftarrow f(x)$, the weight and the differential model of f are added to the SMT problem as a bit-vector variable w and bit-vector constraints $\mathsf{valid}_{f_i}(\Delta_x, \Delta_y)$ and $\mathsf{Equals}(w, \mathsf{weight}_{f_i}(\Delta_x, \Delta_y))$, following Table 5.

Table 5. Bit-vector differential models of ARX operations.

$y = f_a(x)$	Validity	Weight
$y = x_1 \oplus x_2$	$\mathsf{Equals}(\Delta_y, \Delta_{x_1} \oplus \Delta_{x_2})$	0
$y = x \oplus a$	$\mathsf{Equals}(\Delta_y, \Delta_x)$	0
$y = x \lll a$	$\mathsf{Equals}(\Delta_y, \Delta_x \lll a)$	0
$y = x \ggg a$	$\mathsf{Equals}(\Delta_y, \Delta_x \ggg a)$	0
$y = x_1 \boxplus x_2$	Theorem 1	Theorem 1
$y = x \boxplus a$	Theorem 3	Theorem 4

[6] It is possible to get another solution of an SMT problem by solving it again with an additional constraint that excludes the first solution. By repeating this process, one can find all the solutions.

Finally, the following bit-vector constraints are added to the SMT problem,

$$\mathsf{NotEquals}(\Delta_p, 0)\,,\ \mathsf{Equals}(W, w_1 \boxplus \cdots \boxplus w_r)\,,$$

where Δ_p denotes the input difference and $(w_1, \ldots, w_r)$ denote the weight of each operation. The first constraint excludes the trivial characteristic with zero input difference, while the second constraint fixes the weight of the characteristic to the target weight. Note that the bit-size of the weights might need to be increased to prevent an overflow in the modular addition of the last constraint.

Example 5. Consider the keyed function f_k with key k and input $p = (p_1, p_2)$,

$$f_k(p_1, p_2) = (((p_2 \boxplus 1) \oplus k) \boxplus p_1, p_1 \lll 1)\,.$$

This function can be written as a list of simple instructions (SSA form) as

$$\begin{aligned} x_1 &\leftarrow p_2 \boxplus 1\,,\\ x_2 &\leftarrow x_1 \oplus k\,,\\ x_3 &\leftarrow x_2 \boxplus p_1\,,\\ x_4 &\leftarrow p_1 \lll 1\,, \end{aligned}$$

where the output is the pair (x_3, x_4). Figure 2 depicts the function f_k together with its intermediate variables.

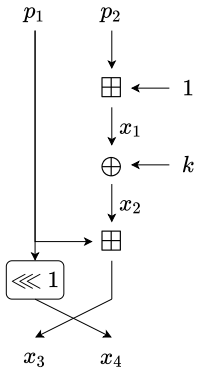


Fig. 2. The function f_k.

An SMT problem in the bit-vector theory denoting whether f_k has a characteristic with integer weight W is as follows:

$$\begin{aligned}
&\exists \Delta_{p_1}, \Delta_{p_2}, \Delta_{x_1}, \Delta_{x_2}, \Delta_{x_3}, \Delta_{x_4}, w_1, w_2, w_3, w_4 : \\
&\quad \mathsf{valid}_{\boxplus_1}(\Delta_{p_2}, \Delta_{x_1}) , \\
&\quad \mathsf{Equals}(w_1, \mathsf{BvWeight}(\Delta_{p_2}, \Delta_{x_1}, 1)) , \\
&\quad \mathsf{Equals}(\Delta_{x_2}, \Delta_{x_1}) , \\
&\quad \mathsf{Equals}(w_2, 0) , \\
&\quad \mathsf{valid}_{\boxplus}((\Delta_{x_2}, \Delta_{p_1}), \Delta_{x_3}) , \\
&\quad \mathsf{Equals}(w_3, \mathsf{weight}_{\boxplus}((\Delta_{x_2}, \Delta_{p_1}), \Delta_{x_3})) , \\
&\quad \mathsf{Equals}(\Delta_{x_4}, \Delta_{d_{p_1}} \lll 1) , \\
&\quad \mathsf{Equals}(w_4, 0) , \\
&\quad \mathsf{NotEquals}((\Delta_{p_1}, \Delta_{p_2}), 0) , \\
&\quad \mathsf{Equals}(W, (w_1 \boxplus ((w_2 \boxplus w_3 \boxplus w_4) \ll 4) \gg 4)) .
\end{aligned}$$

The shifts in the last constraint are due to the fact that the last four bits of w_1 denote fraction bits. Furthermore, depending on the bit-size of f_k, it might be necessary to extend the bit-size of the weights in order to prevent an overflow in the last modular additions.

4.3 Searching of Impossible Differentials

In [24], Sasaki and Todo propose an MILP-based method to search for impossible differentials that employs the MILP problems used to search for characteristics. Since this method can also be adapted to SMT problems, we will explain the method within the SMT context.

The method's main idea is that one can check whether a particular differential (Δ_p, Δ_c) is impossible by querying a simple SMT problem. While it is unfeasible to check all differentials, one can check those with a low number of active bits since most of the known impossible differentials have this property.

The subroutine to check whether a particular differential $(\Delta_{p_0}, \Delta_{c_0})$ is impossible can be done as follows. First, the SMT problem of whether there exists a characteristic over the cipher is encoded as in Section 4.2. However, only the validity constraints are added; the weight constraints and the target weight W are ignored. Second, the constraints that fix the input and output differences (Δ_p, Δ_c) to $(\Delta_{p_0}, \Delta_{c_0})$ are added to the SMT problem, that is,

$$\mathsf{Equals}(\Delta_p, \Delta_{p_0}), \quad \mathsf{Equals}(\Delta_c, \Delta_{c_0}) .$$

Then, the SMT solver checks the satisfiability of the SMT problem. If the problem is found to be unsatisfiable, the differential is impossible.

This method can be used to search for single-key and related-key impossible differentials. For the former case, the validity constraints of the key schedule are ignored, while for the latter case they are included in the SMT problems.

As opposed to the previous SMT-based characteristic search method, the impossible check subroutine is a sound method. In other words, a characteristic found by Algorithm 2 could be invalid due to the independence assumptions, but a differential found impossible by the check subroutine is always impossible. While the check subroutine is a sound method, it is not complete; there are some impossible differentials that cannot be detected by the check subroutine.

While the check subroutine is fast, checking all differentials is unfeasible and only a small subset can be checked with the method by [24]. Thus, we propose a new automated method to search for impossible differentials that does not restrict the search over any pre-defined small subset and let the SMT solver efficiently search through the space of differentials. Our automated method proceeds as follows.

First, we split the cipher $E = E_2 \circ E_1 \circ E_0$ into three parts E_2, E_1 and E_0. Let $\Omega = (\Delta_{x_0}, \Delta_{x_1}, \Delta_{x_2}, \Delta_{x_3})$ denote a *partial characteristic* over E, that is, any characteristic verifying

$$\Pr(\Delta_{x_0} \xrightarrow{E_0} \Delta_{x_1}) = 1, \quad \Pr(\Delta_{x_2} \xrightarrow{E_2} \Delta_{x_3}) = 1\,.$$

Note that no relation is imposed between Δ_{x_1} and Δ_{x_2}.

Then, we search for all partial characteristics using our SMT-based method from Section 4.1. For each partial characteristic $\Omega = (\Delta_{x_0}, \Delta_{x_1}, \Delta_{x_2}, \Delta_{x_3})$, we apply the check subroutine to the differential $(\Delta_{x_1}, \Delta_{x_2})$ over E_1. If $(\Delta_{x_1}, \Delta_{x_2})$ is found to be impossible over E_1, then $(\Delta_{x_0}, \Delta_{x_3})$ is an impossible differential over E, since $(\Delta_{x_0}, \Delta_{x_1})$ and $(\Delta_{x_2}, \Delta_{x_3})$ are differentials with probability one (see Figure 3).

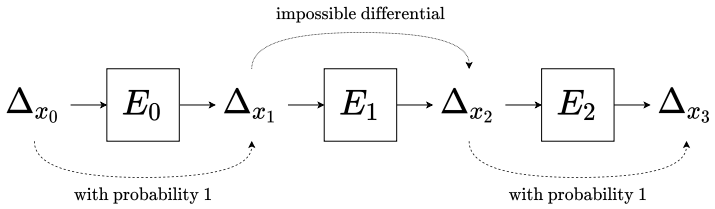


Fig. 3. The partial characteristic $\Omega = (\Delta_{x_0}, \Delta_{x_1}, \Delta_{x_2}, \Delta_{x_3})$ over $E = E_2 \circ E_1 \circ E_0$, alongside the condition that the inner part $(\Delta_{x_1}, \Delta_{x_2})$ over E_1 is an impossible differential.

Like the characteristic search method, we start searching for impossible differentials over a round-reduced version of the cipher and keep increasing the number of rounds iteratively. This procedure is described in Algorithm 3. Impossible differentials starting after a few rounds are useful in practice, and our method can easily be adapted by splitting the cipher into four parts, $E = E_2 \circ E_1 \circ E_0 \circ E_{-1}$, where E_{-1} denotes the skipped rounds.

Algorithm 3 SMT-based impossible-differential search

```
for r = 1, ..., max_rounds do
  for r_1 = 1, ..., r − 2 do
    for r_0 = 1, ..., r − r_1 − 1 do
      r_2 = r − r_0 − r_1
      for Ω ∈ FindPartialCh(rounds_{E_0} = r_0, rounds_{E_1} = r_1, rounds_{E_2} = r_2) do
        (Δ_{x_0}, Δ_{x_1}, Δ_{x_2}, Δ_{x_3}) ← Ω
        if IsImpossible(inputΔ = Δ_{x_1}, outputΔ = Δ_{x_2}, rounds_{E_1} = r_1) then
          Print(Δ_{x_0}, Δ_{x_3})                ▷ impossible differential over r rounds
          break                                  ▷ break three inner loops and increase r
return (Δ_{x_0}, Δ_{x_3})
```

The main advantage of our method is that the subset of differentials to check does not need to be specified. Thus, it can find impossible differentials that other methods cannot. Moreover, the search of partial characteristics is quite fast, as for many operations f (including the modular addition and the constant addition) the constraint $\mathsf{Equals}(0, \mathsf{weight}_f(\Delta_x, \Delta_y))$ is much simpler than the constraint for the general case $\mathsf{Equals}(w, \mathsf{weight}_f(\Delta_x, \Delta_y))$.

As opposed to the search of characteristics, the search of $r+1$-round impossible differentials cannot reuse information obtained from the search of r-round impossible differentials. In other words, Algorithm 2 exploits the fact that if no r-round characteristics were found with weight w, then no $r+1$-round characteristics can be found with the same weight. However, for some key schedules, Algorithm 2 might find $r+1$-round impossible differentials even if no r-round impossible differentials were found.

4.4 Implementation

We have developed an open-source tool `ArxPy`[7] to find characteristics and impossible differentials of ARX ciphers implementing the methods described earlier. Originally, `ArxPy` was a tool to search for rotational-XOR characteristics using SMT solvers [52]. However, we have extended it to support (related-key) differential characteristics and impossible differentials containing the constant addition. `ArxPy` provides high-level functions that automate the search, a simple interface to represent ARX ciphers, and complete documentation in HTML format, among other features.

`ArxPy` workflow is represented in Figure 4. The user first defines the ARX cipher using the interface provided by `ArxPy` and chooses the parameters of the search (e.g., the type of the characteristic to search, the SMT solver to use, etc.). Then, `ArxPy` automatically translates the python implementation of the ARX cipher into SSA form, encodes the SMT problems associated to the type of search selected by the user, and solves the SMT problems by querying the SMT solver. When searching for characteristics, for each satisfiable SMT problem

[7] https://github.com/ranea/ArxPy

found, ArxPy reconstructs the characteristic from the assignment of the variables that satisfies the problem and empirically verifies the weight of the characteristic. Finally, ArxPy returns the results of the search to the user.

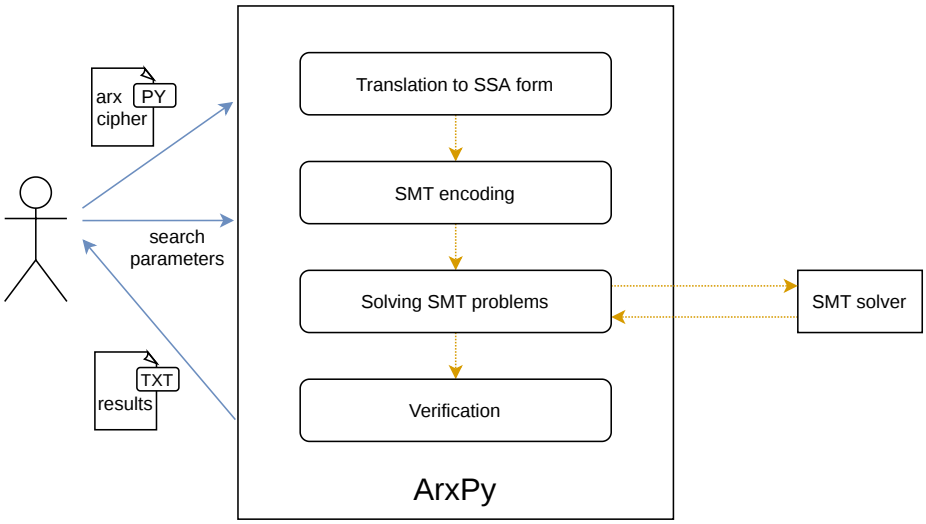


Fig. 4. Workflow of ArxPy

Internally, ArxPy is implemented in Python 3 and uses the libraries SymPy [53] to obtain the SSA representation through symbolic execution and PySMT [54] for the communication with the SMT solvers. Thus, all the SMT solvers supported by PySMT can be directly used for ArxPy.

5 Experiments

We have applied our methods for finding characteristics and impossible differentials to some ARX ciphers that include constant additions. In particular, we have searched for related-key characteristics and related-key impossible differentials of TEA, XTEA, HIGHT, LEA, SHACAL-1, and SHACAL-2.

Due to the difficulty of searching for characteristics of ciphers with constant additions this far, cipher designers have avoided constant additions in the encryption functions so that they could search for single-key characteristics, the most threatening ones. Only a few ciphers include constant additions in the encryption function, and their ad-hoc structures make them more suitable to be analysed with other types of differences, such as additive differences in the case of TEA [19]. As a result, we have focused on searching related-key characteristics and impossible differentials of some well-known ciphers.

Regarding the search for characteristics, we used Algorithm 2 to find related-key characteristics starting from the first round of each cipher. For the case

of impossible differentials, we applied Algorithm 3 to search for related-key impossible differentials but skipping the first rounds of the cipher. To this end, we repeatedly call Algorithm 3 while increasing the number of skipped rounds in each call.

For related-key characteristics, the usual assumptions (i.e., round independence and the hypothesis of stochastic equivalence) do not always hold. Thus, we empirically verify each characteristic and stopped each round-reduced search after the first valid characteristic is found.

To verify a related-key characteristic Ω, we split Ω in smaller characteristics $\Omega_i = (\Delta_{x_i} \to \cdots \to \Delta_{y_i})$ with weight w_i lower than 20, and empirically compute the probability of each differential $(\Delta_{x_i}, \Delta_{y_i})$ by sampling a small multiple of 2^{w_i} input pairs for 2^{10} related-key pairs. After combining the probability of each differential, we obtain 2^{10} characteristic probabilities, one for each related-key pair. If the characteristic probability is non-zero for several key pairs, we consider the characteristic valid and we define its empirical probability (resp. weight) as the arithmetic mean of the 2^{10} characteristic probabilities (resp. weights), but excluding those key pairs with zero probability.

Thus, for each characteristic that we have found, Table 6 provides: (1) the theoretical key schedule and encryption weights (w_{KS}, w_E), computed by summing the weight of each ARX operation; (2) the empirical key schedule and encryption weights $(\overline{w_{\mathsf{KS}}}, \overline{w_E})$, computed by sampling input pairs as explained before; and (3) the percentage of key pairs that lead to non-zero probability in the weight verification. In the appendix, we provide the round weights and the round differences for the characteristics covering the most rounds.

When searching for impossible differentials with skipped rounds, Algorithm 3 splits the cipher into four parts. More specifically, the cipher E is represented as $E = E_2 \circ E_1 \circ E_0 \circ E_{-1}$, where E_{-1} denotes the skipped rounds, E_1 stands for the rounds of the inner impossible differential, and E_0 and E_2 respectively denote the backward and the forward rounds of the partial characteristics. In Table 7 we provide the number of rounds of each part for the best related-key impossible differentials that we found, and in Table 8 we provide the input and output differences of our longest impossible differentials.

We also implemented and searched impossible differentials using the automated method of Sasaki and Todo [24] to compare the results with the ones observed by Algorithm 3. While for XTEA, LEA, and HIGHT, both methods find impossible differentials with the same number of rounds, for SHACAL-1 and SHACAL-2 Algorithm 3 achieves impossible differentials covering more rounds. For XTEA and HIGHT, the longest impossible differentials found by Algorithm 3 include a few active bits, and thus they could also be found by the other method. However, for SHACAL-1 and SHACAL-2, our algorithm found impossible differentials containing multiple active bits, which cannot be obtained by other methods that restrict to predefined differential subsets with a low number of active bits.

For the experiments, we have used `ArxPy` equipped with the SMT solver Boolector [48], winner of the SMT competition SMT-COMP 2019 in the bit-

vector track [55]. We run the characteristic search for one week on a single core of an Intel Xeon 6244 at 3.60GHz. The search of impossible differentials was done on similar hardware during one week as well. Note that better characteristics and impossible differentials could be found if the round-reduced searches are not stopped after the first valid characteristic or if more time is employed.

Table 6. Best related-key differential characteristics of XTEA, HIGHT, LEA, SHACAL-1, and SHACAL-2.

Cipher	Ch. Type	Rounds	$(w_{\mathsf{KS}}, \overline{w_{\mathsf{KS}}})$	$(w_E, \overline{w_E})$	% valid keys	Reference
XTEA	Strong	16	0	32	-	[56]
		16	(0,0)	(37, 32.02)	46%	This paper
		18	(0,0)	(57, 49.79)	48%	This paper
	Weak	18	17	19	-	[57, 56]
		18	(4.83, 3.15)	(16, 14.46)	100%	This paper
		27	(6.89, 5.74)	(40, 39.39)	7%	This paper
HIGHT	Strong	10	0	12	-	[58]
		10	(0, 0)	(12, 9.97)	34%	This paper
		15	(0, 0)	(45, 42.59)	8%	This paper
	Weak	12	2	19	-	[59]
		12	(2.48, 3.19)	(19, 17.65)	40%	This paper
		14	(13.09, 9.85)	(14, 11.46)	17%	This paper
LEA	Weak	11	-	-	-	[10]
		6	(1.56, 1.22)	(24, 22.50)	100%	This paper
		7	(2.56, 4.68)	(36, 34.35)	100%	This paper
SHACAL-1	Strong	27	0	29	-	[60]
		30	(0, 0)	(63, 47.36)	97%	This paper
	Weak	35	10	29	-	[61, 62]
		25	(1, 0.87)	(22, 13.31)	47%	This paper
SHACAL-2	Strong	24	0	$38^{\dagger}$	-	[63]
		23	(0, 0)	(58, 48.06)	100%	This paper
	Weak	24	-	52	-	[64]
		22	(6.67, 2.38)	(29, 24.03)	100%	This paper

†: Imposes extra conditions on plaintext values or intermediate values.

TEA. Designed by Wheeler and Needham, TEA [15] is a block cipher with 64-bit block size and 128-bit key size. It iterates 64 times an ARX round function, including constant additions and logical shifts, depicted in Figure 5. Since the logical shifts propagate XOR differences deterministically, the encoding method presented in Section 4.2 can be easily extended to include these operations.

Table 7. Best related-key impossible differentials of XTEA, HIGHT, LEA, SHACAL-1, and SHACAL-2.

Cipher	Rounds	Start	Backward	Inner	ID	Forward	Reference
XTEA	$25(19 \sim 43)$	19	-	-	-		[65]
	$25(0 \sim 24)$	0	7	13	5		This paper
HIGHT	$22(6 \sim 27)$	6	-	-	-		[66]
	$22(0 \sim 21)$	0	4	14	4		This paper
LEA	$4(0 \sim 3)$	0	0	4	0		This paper
SHACAL-1	$30(20 \sim 49)$	20	2	16	12		This paper
SHACAL-2	$18(0 \sim 17)$	0	-	-	-		[67]
	$24(0 \sim 23)$	0	1	12	11		This paper

Table 8. Input, output, and key differences of our longest related-key impossible differentials.

Cipher	Differences
XTEA	Δ_{mk} (0x80000000, 0x00000000, 0x00000000, 0x00000000)
	Δ_p (0x80000000, 0x00000000)
	Δ_c (0x00000000, 0x80000000)
HIGHT	Δ_{mk} (0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00)
	Δ_p (0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00)
	Δ_c (0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)
LEA	Δ_{mk} (0x00000001, 0x00000000, 0x00000000, 0x00000000)
	Δ_p (0x00000000, 0x00000000, 0x00000000, 0x00000000)
	Δ_c (0x00000000, 0x00000000, 0x00000000, 0x00000000)
SHACAL-1	Δ_{mk} (0xa0000000, 0x80000000, 0xa0000000, 0x00000000, 0x80000000, 0x00000000, 0xc0000000, 0x00000000, 0x80000000, 0x80000000, 0x40000000, 0x00000000, 0x80000000, 0x80000000, 0x40000000, 0x00000000)
	Δ_p (0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000)
	Δ_c (0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)
SHACAL-2	Δ_{mk} (0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x94857ee6, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)
	Δ_p (0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)
	Δ_c (0x80000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000, 0x00000000)

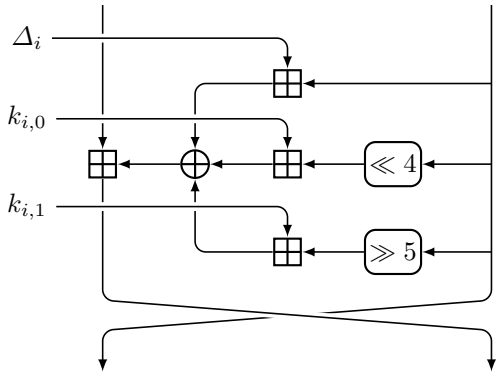


Fig. 5. The i-th round of TEA, $i = 0, 1 \ldots, 63$. The master key mk is split into four 32-bit words (mk_0, mk_1, mk_2, mk_3) and the i-th round key is defined as $(k_{i,0}, k_{i,1}) = (mk_0, mk_1)$ if i is even and $(k_{i,0}, k_{i,1}) = (mk_2, mk_3)$ if i is odd. The i-th round constant is defined as $\Delta_i = \Delta_{i-2} \boxplus \Delta_0$, where $\Delta_{-1} = \Delta_0 = 2654435769$.

Kelsey, Schneier, and Wagner presented the best related-key characteristics in [68]. They found a 2-round iterative strong related-key characteristic Ω with weight $(w_k, w_e) = (0, 1)$, which they extended to a 60-round characteristic with weight (0, 30). They also discovered in [38] that each TEA key has three other equivalent keys.

Using `ArxPy`, we revisited the results by Kelsey, Schneier, and Wagner, but in a fully automated way. We found three related-key characteristics with weight zero over the full cipher, confirming that each key is equivalent to exactly three other keys. Excluding these three characteristics, we also obtained a 60-round strong related-key characteristic with weight (0, 30), and all the 60-round SMT problems with smaller weights were found to be unsatisfiable. Since a 60-round related-key characteristic is sufficient to mount the related-key differential cryptanalysis on full-round TEA [68], there is no need to search for characteristics containing more rounds of TEA, and we stop at 60 rounds.

There is also no need to search for related-key impossible differentials of TEA, as each of the three full-round zero-weight related-key characteristics induces roughly 2×2^{64} full-round related-key impossible differentials, simply by alternating either the plaintext or the ciphertext difference.

XTEA. The block cipher XTEA [16] is designed by the same authors of TEA to fix the weakness of the former cipher (in the related-key setting). XTEA has a 64-bit block size and 128-bit key size, and it iterates 64 times the round function depicted in Figure 6. Like TEA, the round function also includes logical shifts, but the constant additions are included in the key schedule.

The longest related-key characteristics found so far are the 16-round strong related-key differential with weight 32, manually found by Lu in [56], and the

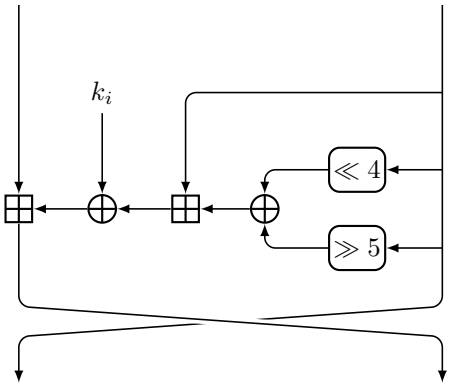


Fig. 6. The i-th round of XTEA, $i = 0, 1 \ldots, 63$.. The master key mk is split into four 32-bit words (mk_0, mk_1, mk_2, mk_3) and the i-th round key is defined as $k_i = s_i \boxplus mk_{s_i \wedge 3}$ if i is even and $k_i = s_i \boxplus mk_{(s_i \ll 11) \wedge 3}$ if i is odd. The i-th constant s_i is defined as $s_i = s_{i-2} \boxplus s_0$, where $s_{-1} = s_0 = 2654435769$.

18-round weak related-key characteristic with weights $(w_{\mathsf{KS}}, w_E) = (19, 19)$, manually found by Lee, Hong, Chang, Hong, and Lim [57] but later improved to (17, 19) by Lu [56].

The results of our automated search for related-key characteristics are listed in Table 6. In the strong related-key search, we found an 18-round characteristic with weight 57; all the SMT problems for 19 rounds were found to be unsatisfiable. In the weak related-key search, we found characteristics up to 27 rounds, where the 27-round characteristic has total weight $6 + 40 = 46$. No equivalent keys were found for XTEA.

In our automated search for related-key impossible differentials of XTEA, we observed impossible differentials spanning 25 rounds, similar to the best impossible differential found this far by Darbuka [65]. Denoting the cipher XTEA with 25 rounds by $E = E_2 \circ E_1 \circ E_0$, our related-key impossible differential contains a 13-round inner impossible differential over E_1, extended by a deterministic 7-round backward trail over E_0 and a deterministic 5-round forward trail over E_2 as depicted in Table 7. Our automated tool was also able to complete the search of related-key impossible differentials up to 31 rounds, but no impossible differentials spanning more than 25 rounds were found.

HIGHT. Adopted as an international standard by ISO/IEC [69], HIGHT [9] is a lightweight cipher with a block size of 64 bits and a key size of 128 bits. The encryption function performs initial and final key whitening transformations, and iterates 32 times a round function including XORs, 2-input additions and rotations; the constant additions are performed in the key schedule.

The longest related-key characteristics found for HIGHT are a 10-round strong characteristic with weight 12 found by Lu [58], and a 12-round weak characteristic

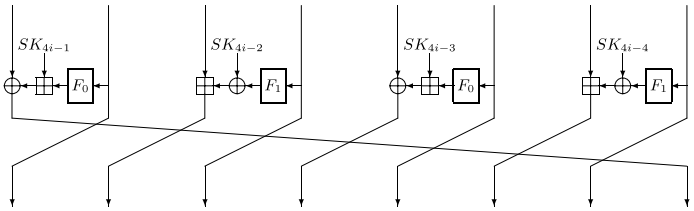


Fig. 7. The i-th round function of HIGHT, $i = 0, 1 \ldots, 31$ [9]. The i-th round key is denoted by $k_i = (SK_{4i-1}, SK_{4i-2}, SK_{4i-3}, SK_{4i-4})$ and the functions F_0 and F_1 are defined as $F_0(x) = (x \lll 1) \oplus (x \lll 2) \oplus (x \lll 7)$ and $F_1(x) = (x \lll 3) \oplus (x \lll 4) \oplus (x \lll 6)$.

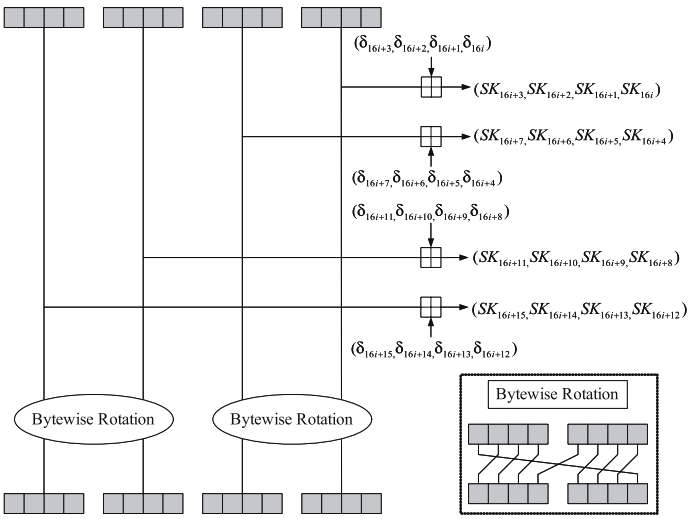


Fig. 8. The key schedule of HIGHT [9]. The round key words are denoted by SK_i and the key schedule constants are denoted by δ_j.

with weights $(w_{\mathsf{KS}}, w_E) = (2, 19)$ found by Koo, Hong, and Kwon [59]. In our automated search, we found related-key characteristics up to 15 rounds, listed in Table 6. The longest strong related-key characteristic we found covered 15 rounds with weights (0, 45), whereas the longest weak related-key characteristic covered 14 rounds with total weight $13 + 14 = 27$.

Özen, Varıcı, Tezcan, and Kocair introduced the best-known 22-round related-key impossible differential for HIGHT [66]. Using `ArxPy`, we found a new impossible differential covering the same number of rounds, as shown in Table 7. Our impossible differential consists of a 14-round inner impossible differential, extended by two zero-weight 4-round backward and forward related-key trails. The mentioned 22-round impossible differential is the longest related-key impossible

differential that `ArxPy` could obtain in one week by checking up to 32 rounds of HIGHT.

LEA. Among the family of ARX ciphers LEA [10], we analysed LEA-128, the version with 128-bit block size, 24 rounds, and 128-bit key size. The encryption round function of LEA performs 2-input additions, rotations, and XORs, whereas the key schedule contains constant additions and rotations.

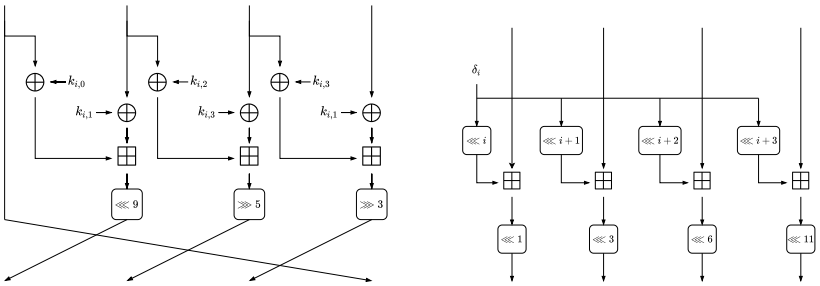


Fig. 9. The i-th round function of the encryption (left) and the key schedule (right) of LEA-128, $i = 0, 1 \ldots, 23$. The tuple $(k_{i,0}, \ldots, k_{i,3})$ denotes the i-th round key and δ_i denotes the i-th key schedule constant.

The designers of LEA found related-key characteristics up to 11 rounds, but only specifying that the 11-round characteristics are valid for a small part of the key space and without providing the weights of such characteristics [10]. Excluding these characteristics, there are no others examples of related-key characteristics of LEA. Our automated search found weak related-key characteristics up to 7 rounds valid for the full key space, listed in Table 6. Strong characteristics with weight smaller than 128 were found up to 4 rounds, and all the strong related-key SMT problems for 5 rounds were found unsatisfiable. No equivalent keys were found for LEA.

We applied `ArxPy` on LEA to automatically search for related-key impossible differentials. While our method completed the search for a large number of rounds, only impossible differentials with non-zero key difference spanning up to four rounds were found. The lack of related-key impossible differentials, with low hamming weight or with key schedule transitions with probability 1, seems to be due to the heavy and robust key schedule of LEA. By and large, ciphers with lightweight key schedule algorithms tend to have longer related-key impossible differentials in comparison to their single-key counterparts. However, the key schedule and the round function of LEA are of the same complexity. Thus, finding an impossible differential that covers more rounds in related-key setting instead of single-key setting seems unfeasible. We confirmed this by applying our tool to LEA in the single-key setting, obtaining multiple 10-round single-key impossible differentials within a few hours.

SHACAL-1. Based on the compression function of the NIST standard hash function SHA-1 [70], the block cipher SHACAL-1 was initially suggested in [71] and submitted by Handschuh and Naccache to the NESSIE project [72]. SHACAL-1 uses 160-bit block size and 80 rounds, where its round function is similar to the SHA-1 compression function. The key size can be variable from 0 to 512 bits, although a minimum of 128-bit key size is required in [73] and we analysed SHACAL-1 for 512-bit keys.

There are some ad-hoc differential characteristics presented in [60, 74, 61]. However, Wang, Keller, and Dunkelman [62] indicated that many of the previous characteristics are invalid. The longest valid XOR differential characteristic is a 35-round weak related-key characteristic that appeared in [61] and was later corrected in [62] to obtain the corrected weights $(w_{\mathsf{KS}}, w_E) = (10, 29)$. Moreover, the longest strong related-key XOR differential characteristic that is not found to be invalid by [62] spans 27 rounds of SHACAL-1 with the corresponding weights $(w_{\mathsf{KS}}, w_E) = (0, 29)$ [60].

These characteristics do not necessarily start from the first round of SHACAL-1 since they are not used for a differential attack but rather for a rectangle attack [51]. Note that the round function of SHACAL-1 changes in different rounds (see Figure 10), and these characteristics could take advantage of the variable definition of the round function. However, we are only looking for the characteristics starting from the first round and for the particular case of SHACAL-1 with variable round functions, we do not necessarily obtain the best possible characteristics.

Our automated tool `ArxPy` obtained a weak-key 25-round characteristic with $(w_{\mathsf{KS}}, w_E) = (1, 22)$. Moreover, the tool could also find a 30-round characteristic in the strong-key setting with the corresponding weights $(w_{\mathsf{KS}}, w_E) = (0, 63)$. In our search, a large amount of SHACAL-1 characteristics found by the SMT solver did not pass our empirical validation test, which significantly increased the running time for finding each valid characteristic. More specifically and in the weak-key setting, we found more than 100 empirically invalid characteristics until we detected a valid 25-round one; we also obtained multiple 26-round weak characteristics within a week, but they were found invalid by our empirical test. We discarded more than 900 empirically invalid characteristics for the strong-key setting before finding a valid 30-round characteristic, and none of the 31-round trails obtained in a week could pass the test.

One of the main reasons for the large number of empirically invalid characteristics is the 5-input modular addition within the round function of SHACAL-1. Since the differential model of the modular addition with three or more inputs is unknown, we had to approximate the differential model of the 5-input addition with a chain of 2-input addition models. In other words, to model the 5-input addition $y = x_1 \boxplus x_2 \boxplus x_3 \boxplus x_4 \boxplus x_5$, we split it into four 2-input additions

$$z_1 = x_1 \boxplus x_2,\ z_2 = z_1 \boxplus x_3,\ z_3 = z_2 \boxplus x_4,\ y = z_3 \boxplus x_5\,,$$

and we model the four 2-input additions independently. Thus, we are approximating the differential probability of the 5-input addition

$$\Pr[(\Delta_{x_1}, \ldots, \Delta_{x_5}) \xrightarrow{\boxplus} \Delta_y]$$

with the multiplication of the differential probabilities of the four 2-input additions

$$\Pr[(\Delta_{x_1}, \Delta_{x_2}) \xrightarrow{\boxplus} \Delta_{z_1}] \times \Pr[(\Delta_{z_1}, \Delta_{x_3}) \xrightarrow{\boxplus} \Delta_{z_2}] \times \Pr[(\Delta_{z_2}, \Delta_{x_4}) \xrightarrow{\boxplus} \Delta_{z_3}] \times \Pr[(\Delta_{z_3}, \Delta_{x_5}) \xrightarrow{\boxplus} \Delta_y] .$$

For many differentials, this approximation is not accurate, and this caused the appearance of many empirically invalid characteristics in our search.

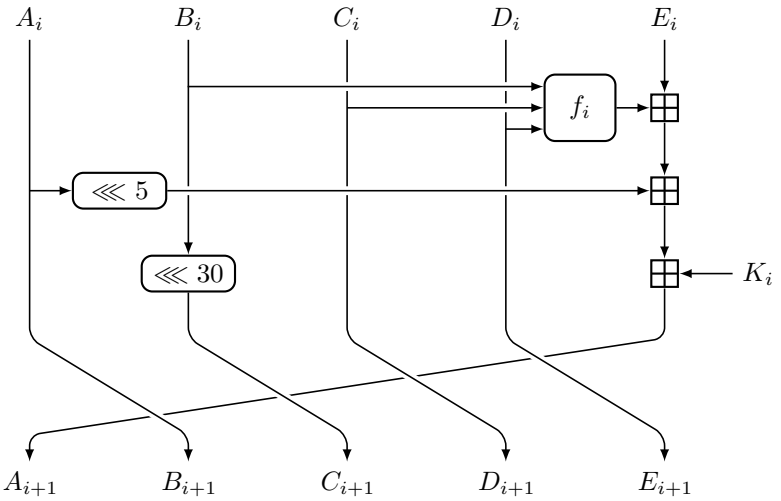
As depicted in Table 7, our automated tool found the first known related-key impossible differential of SHACAL-1, extending to 30 rounds of the cipher from rounds 20 to 49. The backward and forward trails respectively traverse 2 and 12 rounds of SHACAL-1, delimiting the inner 16-round impossible differential. The search for 31-round impossible differentials did not finish after one week, and we stopped the search. Thus, we expect that dedicating more time to the search may result in obtaining longer impossible differentials.

SHACAL-2. Similar to SHACAL-1, the block cipher SHACAL-2 [73] was designed based on the compression function of the NIST standard hash function SHA-256 [75]. The cipher was submitted to the NESSIE project [72] and was approved as one of the NESSIE final selections. SHACAL-2 is a 256-bit block cipher, has 64 rounds, and supports a variable key size up to 512 bits. We analysed SHACAL-2 for 512-bit keys.

The longest ad-hoc related-key XOR differential characteristic in the strong-key setting is a 24-round characteristic presented in [63] with $(w_{\mathsf{KS}}, w_E) = (0, 38)$, which relies on some additional conditions on specific values alongside the differences to improve the weights. Moreover, in the weak-key setting, Biryukov, Lamberger, Mendel, and Nikolić [64] provided two 24-round related-key XOR differential characteristics of SHACAL-2, each has encryption weight $w_E = 52$. However, they did not explicitly mention the key schedule weight w_{KS} for each characteristic.

Our automated search resulted in a 23-round characteristic in the strong-key setting with encryption weight $w_E = 58$ and a 22-round characteristic in the weak-key setting with total weight $w_{\mathsf{KS}} + w_E = 6 + 29 = 35$. Like SHACAL-1, our automated search could not find longer characteristics of SHACAL-2 within a week due to the large number of empirically invalid characteristics found. The round function of SHACAL-2 also contains a modular addition with multiple inputs (i.e., seven operands), and modelling it with 2-input additions is one of the main reasons for the inaccurate differential behaviour for some special differences.

Table 7 lists the results of the best related-key impossible differentials of SHACAL-2. The 18-round impossible differential presented by Yang, Hu, and Zhong in [67] has been the longest known related-key impossible differential for SHACAL-2 so far. Our automated tool `ArxPy` obtained a 24-round impossible differential, improving the previous best result by 6 rounds. This impossible differential includes a 12-round inner impossible differential, extended by two deterministic 1-round backward and 11-round forward trails. We checked up to 28-rounds of SHACAL-2 in one week, and the longest impossible differential we observed was the 24-round related-key impossible differential.



$$f_i(X,Y,Z) = \begin{cases} (X \wedge Y) \vee (\neg Y \wedge Z), & 0 \le i \le 19\,; \\ X \oplus Y \oplus Z, & 20 \le i \le 39,\ 60 \le i \le 79\,; \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & 40 \le i \le 59\,; \end{cases}$$

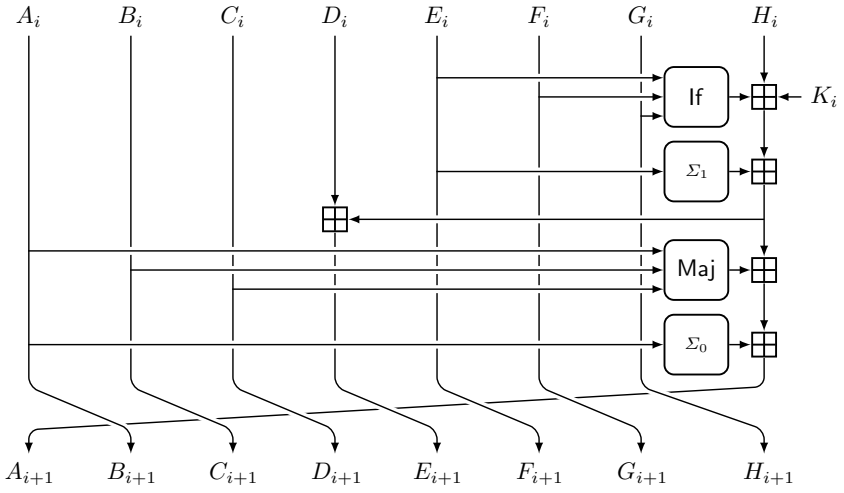
$$K_i = \begin{cases} M_i \boxplus W_i, & 0 \le i \le 15\,; \\ \big((K_{i-3} \oplus K_{i-8} \oplus K_{i-14} \oplus K_{i-16}) \lll 1\big) \boxplus W_i, & 16 \le i \le 79\,. \end{cases}$$

Fig. 10. The i-th round of SHACAL-1, $i = 0, 1 \ldots, 79$. The 160-bit input is divided into five 32-bit words A_i, B_i, C_i, D_i, and E_i. The function f_i significantly changes regarding the round number. For a given 512-bit master key $mk = (M_0, M_1, \cdots, M_{15})$, the round keys K_i are computed as described above, where W_i is the round constant.

6 Conclusion

In this paper, we proposed the first bit-vector differential model of the n-bit modular addition with a constant. We described a bit-vector formula, with bit-vector complexity $O(1)$, that determines whether a differential is valid and a bit-vector function, with complexity $O(\log_2 n)$, that provides a close approximation of the differential weight. In this regard, we carefully studied our approximation error and obtained almost tight bounds. Moreover, we described two new SMT-based automated methods to search for characteristics and impossible differentials of ARX ciphers including constant additions, respectively.

Each of our methods formulates the search problem as a sequence of bit-vector SMT problems, encoded from the cipher's SSA representation and the bit-vector differential models of each operation. We have implemented our methods in `ArxPy`, an open-source tool to find characteristics and impossible differentials of ARX



$$\mathsf{If}(X,Y,Z) = (X \wedge Y) \oplus (\neg X \wedge Z)\,;$$

$$\mathsf{Maj}(X,Y,Z) = (X \wedge Y) \oplus (Y \wedge Z) \oplus (X \wedge Z)\,;$$

$$\Sigma_0(X) = (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22)\,;$$

$$\Sigma_1(X) = (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25)\,;$$

$$K_i = \begin{cases} M_i \boxplus W_i, & 0 \le i \le 15\,; \\ \sigma_1(K_{i-2}) \boxplus K_{i-7} \boxplus \sigma_0(K_{i-15}) \boxplus K_{i-16} \boxplus W_i, & 16 \le i \le 79\,; \end{cases}$$

$$\sigma_0(X) = (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3)\,;$$

$$\sigma_1(X) = (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10)\,.$$

Fig. 11. The i-th round of SHACAL-2, $i = 0, 1 \ldots, 63$. The 256-bit input is divided into eight 32-bit words A_i, B_i, C_i, D_i, E_i, F_i, G_i, and H_i. The special operators used in the round function of SHACAL-2 are If, Maj, Σ_0, and Σ_1 that are defined as above. For a given 512-bit master key $mk = (M_0, M_1, \cdots, M_{15})$, the key schedule generates round keys K_i as described in above, where W_i is the round constant.

ciphers in a fully automated way. To show some examples, we have applied our automated methods to search for equivalent keys, related-key characteristics, and related-key impossible differentials of TEA, XTEA, HIGHT, LEA, SHACAL-1, and SHACAL-2.

Regarding the characteristic results, for TEA we revisited previous results obtained in a manual approach. In contrast, for XTEA, HIGHT, and LEA, we improved the previous best-known related-key characteristics in both the strong-key and the weak-key settings. Our characteristic results of SHACAL-1

and SHACAL-2 did not outperform previous works in all settings due to the presence of modular additions with more than two inputs, for which no efficient differental model has been proposed yet.

Concerning the impossible differentials, our results for TEA, XTEA, and HIGHT are of the same length, compared to the best-known related-key impossible differentials. On the other hand, we obtained the longest related-key impossible differentials for LEA, SHACAL-1, and SHACAL-2.

Our differential model relies on a bit-vector-friendly approximation on the binary logarithm. Thus, future works could explore other approximations improving the bit-vector complexity or the approximation error, which could also be applied to other SMT problems involving the binary logarithm. While we have focused on the modular addition by a constant, there are other simple operations for which no differential model has been proposed so far, such as the modular multiplication, and the modular addition with more than two inputs. Obtaining differential models for more operations will allow designing ciphers with more flexibility, leading to new designs that potentially are more efficient.

Acknowledgments Seyyed Arash Azimi and Mohammad Reza Aref were partially supported by Iran National Science Foundation (INSF) under contract No. 96/53979. Adrián Ranea is supported by a PhD Fellowship from the Research Foundation – Flanders (FWO).

References

[1] Gartner. *Gartner Identifies Top 10 Strategic IoT Technologies and Trends*. 2018. URL: https://www.gartner.com/en/newsroom/press-releases/2018-11-07-gartner-identifies-top-10-strategic-iot-technologies-and-trends.

[2] Gartner. *Gartner Survey Reveals 47 percent of Organizations Will Increase Investments in IoT Despite the Impact of COVID-19*. 2020. URL: https://www.gartner.com/en/newsroom/press-releases/2020-10-29-gartner-survey-reveals-47-percent-of-organizations-will-increase-investments-in-iot-despite-the-impact-of-covid-19-.

[3] National Institute of Standards and Technology. *Lightweight Cryptography Project*. URL: https://csrc.nist.gov/Projects/Lightweight-Cryptography.

[4] Seyyed Arash Azimi, Adrián Ranea, Mahmoud Salmasizadeh, Javad Mohajeri, Mohammad Reza Aref, and Vincent Rijmen. "A bit-vector differential model for the modular addition by a constant". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2020, pp. 385–414.

[5] Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. "Triathlon of lightweight block ciphers for the Internet of things". In: *J. Cryptographic Engineering* 9.3 (2019).

[6] Jean Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C W Phan. "SHA-3 proposal BLAKE". In: *Submission to NIST (round 3)* 92 (2008).

[7] Daniel J Bernstein. "The Salsa20 family of stream ciphers". In: *New stream cipher designs*. Springer, 2008.

[8] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. "Chaskey: an efficient MAC algorithm for 32-bit microcontrollers". In: *International Conference on Selected Areas in Cryptography*. Springer. 2014.

[9] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. "HIGHT: A New Block Cipher Suitable for Low-Resource Device". In: *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*. 2006.

[10] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Donggeon Lee. "LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors". In: *WISA*. Vol. 8267. Lecture Notes in Computer Science. Springer, 2013.

[11] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. "The SIMON and SPECK Families of Lightweight Block Ciphers". In: *IACR Cryptol. ePrint Arch.* 2013 (2013), p. 404.

[12] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. "Design Strategies for ARX with Provable Bounds: Sparx and LAX". In: *ASIACRYPT (1)*. Vol. 10031. Lecture Notes in Computer Science. 2016.

[13] Bonwook Koo, Dongyoung Roh, Hyeonjin Kim, Younghoon Jung, Donggeon Lee, and Daesung Kwon. "CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices". In: *Information Security and Cryptology - ICISC 2017 - 20th International Conference, Seoul, South Korea, November 29 - December 1, 2017, Revised Selected Papers*. 2017.

[14] Xuejia Lai and James L. Massey. "A Proposal for a New Block Encryption Standard". In: *EUROCRYPT*. Vol. 473. Lecture Notes in Computer Science. Springer, 1990.

[15] David J. Wheeler and Roger M. Needham. "TEA, a Tiny Encryption Algorithm". In: *FSE*. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994.

[16] Roger Needham and David Wheeler. *Tea extensions*. Tech. rep. Computer Laboratory, University of Cambridge, 1997.

[17] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *EUROCRYPT*. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999.

[18] Lars Knudsen. "DEAL-a 128-bit block cipher". In: *complexity* 258.2 (1998), p. 216.

[19] Alex Biryukov and Vesselin Velichkov. "Automatic search for differential trails in ARX ciphers". In: *Cryptographers' Track at the RSA Conference*. Springer. 2014.

[20] Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. "Automatic search for the best trails in ARX: application to block cipher speck". In: *International Conference on Fast Software Encryption*. Springer. 2016, pp. 289–310.

[21] Mitsuru Matsui. "On correlation between the order of S-boxes and the strength of DES". In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1994.

[22] Nicky Mouha and Bart Preneel. "Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20". In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 328.

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

[24] Yu Sasaki and Yosuke Todo. "New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers". In: *EUROCRYPT (3)*. Vol. 10212. Lecture Notes in Computer Science. 2017.

[25] Jiongjiong Ren and Shaozhen Chen. "Cryptanalysis of reduced-round SPECK". In: *IEEE Access* 7 (2019), pp. 63045–63056.

[26] Tingting Cui, Shiyao Chen, Kai Fu, Meiqin Wang, and Keting Jia. "New automatic tool for finding impossible differentials and zero-correlation linear approximations". In: *SCIENCE CHINA-INFORMATION SCIENCES* 64.2 (2021).

[27] Clark Barrett and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[28] Andrea Lodi. "Mixed Integer Programming Computation". In: *50 Years of Integer Programming*. Springer, 2010.

[29] Stefan Kölbl and Hosein Hadipour. *CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives based on SMT/SAT solvers*. URL: https://github.com/kste/cryptosmt.

[30] Helger Lipmaa and Shiho Moriai. "Efficient Algorithms for Computing Differential Properties of Addition". In: *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*. 2001.

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

[32] Yunwen Liu, Glenn De Witte, Adrián Ranea, and Tomer Ashur. "Rotational-XOR Cryptanalysis of Reduced-round SPECK". In: *IACR Trans. Symmetric Cryptol.* 2017.3 (2017).

[33] Ling Song, Zhangjie Huang, and Qianqian Yang. "Automatic Differential Analysis of ARX Block Ciphers with Application to SPECK and LEA". In: *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II*. 2016.

[34] Alexis Warner Machado. "Differential Probability of Modular Addition with a Constant Operand". In: *IACR Cryptology ePrint Archive* 2001 (2001), p. 52.

[35] Gergely Kovásznai, Andreas Fröhlich, and Armin Biere. "Complexity of Fixed-Size Bit-Vector Logics". In: *Theory Comput. Syst.* 59.2 (2016).

[36] Jr. Henry S. Warren. *Hacker's delight*. Addison-Wesley, 2003.

[37] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *J. Cryptology* 4.1 (1991).

[38] John Kelsey, Bruce Schneier, and David A. Wagner. "Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES". In: *CRYPTO*. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996.

[39] Robert S. Winternitz and Martin E. Hellman. "Chosen-Key Attacks on a Block Cipher". In: *Cryptologia* 11.1 (1987).

[40] Xuejia Lai, James L. Massey, and Sean Murphy. "Markov Ciphers and Differential Cryptanalysis". In: *EUROCRYPT*. Vol. 547. Lecture Notes in Computer Science. Springer, 1991.

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

[42] Siwei Sun, David Gerault, Pascal Lafourcade, Qianqian Yang, Yosuke Todo, Kexin Qiao, and Lei Hu. "Analysis of AES, SKINNY, and Others with Constraint Programming". In: *IACR Trans. Symmetric Cryptol.* 2017.1 (2017).

[43] Jean Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. "Analysis of NORX: Investigating Differential and Rotational Properties". In: *LATINCRYPT*. Vol. 8895. Lecture Notes in Computer Science. Springer, 2014.

[44] Helger Lipmaa. "On Differential Properties of Pseudo-Hadamard Transform and Related Mappings". In: *Progress in Cryptology - INDOCRYPT 2002, Third International Conference on Cryptology in India, Hyderabad, India, December 16-18, 2002*. Ed. by Alfred Menezes and Palash Sarkar. Vol. 2551. Lecture Notes in Computer Science. Springer, 2002.

[45] Elnaz Bagherzadeh and Zahra Ahmadian. "MILP-Based Automatic Differential Searches for LEA and HIGHT". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 948.

[46] Ernst Schulte-Geers. "On CCZ-equivalence of addition mod 2^n". In: *Designs, Codes and Cryptography* 66.1-3 (2013).

[47] John N Mitchell. "Computer multiplication and division using binary logarithms". In: *IRE Transactions on Electronic Computers* 4 (1962).

[48] Aina Niemetz, Mathias Preiner, and Armin Biere. "Boolector 2.0 system description". In: *Journal on Satisfiability, Boolean Modeling and Computation* 9 (2015), pp. 53–58.

[49] Vijay Ganesh and David L. Dill. "A Decision Procedure for Bit-Vectors and Arrays". In: *CAV*. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007.

[50] David A. Wagner. "The Boomerang Attack". In: *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*. 1999.

[51] Eli Biham, Orr Dunkelman, and Nathan Keller. "The Rectangle Attack - Rectangling the Serpent". In: *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*. 2001.

[52] Adrián Ranea, Yunwen Liu, and Tomer Ashur. "An Easy-to-Use Tool for Rotational-XOR Cryptanalysis of ARX Block Ciphers". In: *Proceedings of the Romanian Academy, Series A* 18.3 (2017).

[53] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992.

[54] Marco Gario and Andrea Micheli. "PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms". In: *SMT Workshop 2015*. 2015.

[55] Liana Hadarean, Antti Hyvarinen, Aina Niemetz, and Giles Reger. *14th International Satisfiability Modulo Theories Competition (SMT-COMP 2019)*. 2019. URL: https://smt-comp.github.io/2019/.

[56] Jiqiang Lu. "Related-key rectangle attack on 36 rounds of the XTEA block cipher". In: *Int. J. Inf. Sec.* 8.1 (2009).

[57] Eunjin Lee, Deukjo Hong, Donghoon Chang, Seokhie Hong, and Jongin Lim. "A Weak Key Class of XTEA for a Related-Key Rectangle Attack". In: *VIETCRYPT*. Vol. 4341. Lecture Notes in Computer Science. Springer, 2006.

[58] Jiqiang Lu. "Cryptanalysis of Reduced Versions of the HIGHT Block Cipher from CHES 2006". In: *Information Security and Cryptology - ICISC 2007, 10th International Conference, Seoul, Korea, November 29-30, 2007, Proceedings*. 2007.

[59] Bonwook Koo, Deukjo Hong, and Daesung Kwon. "Related-Key Attack on the Full HIGHT". In: *Information Security and Cryptology - ICISC 2010 -*

13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers. 2010.

[60] Jongsung Kim, Guil Kim, Seokhie Hong, Sangjin Lee, and Dowon Hong. "The related-key rectangle attack, application to SHACAL-1". In: *Australasian Conference on Information Security and Privacy*. Springer. 2004.

[61] Orr Dunkelman, Nathan Keller, and Jongsung Kim. "Related-key rectangle attack on the full SHACAL-1". In: *International Workshop on Selected Areas in Cryptography*. Springer. 2006.

[62] Gaoli Wang, Nathan Keller, and Orr Dunkelman. "The delicate issues of addition with respect to XOR differences". In: *International Workshop on Selected Areas in Cryptography*. Springer. 2007.

[63] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. "Related-key rectangle attack on 42-round SHACAL-2". In: *International Conference on Information Security*. Springer. 2006.

[64] Alex Biryukov, Mario Lamberger, Florian Mendel, and Ivica Nikolić. "Second-order differential collisions for reduced SHA-256". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2011.

[65] Asli Darbuka. "Related-key attacks on block ciphers. Master's Thesis". MA thesis. Middle East Technical University, 2009.

[66] Onur Özen, Kerem Varıcı, Cihangir Tezcan, and Çelebi Kocair. "Lightweight block ciphers revisited: Cryptanalysis of reduced round PRESENT and HIGHT". In: *Australasian Conference on Information Security and Privacy*. Springer. 2009.

[67] Shao Ping Yang, Yu Pu Hu, and Ming Fu Zhong. "Related-key impossible differential attacks on 31-round SHACAL-2". In: *Journal on Communications* 28.11A (2006), pp. 54–58.

[68] John Kelsey, Bruce Schneier, and David A. Wagner. "Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA". In: *ICICS*. Vol. 1334. Lecture Notes in Computer Science. Springer, 1997.

[69] *Information technology, Security techniques, Encryption algorithms, Part 3: Block ciphers*. Standard. International Organization for Standardization, Mar. 2010.

[70] FIPS. "Secure Hash Standard". In: *Federal Information Processing Standards Publication 180-1* (1995).

[71] Helena Handschuh, Lars R Knudsen, and Matthew J Robshaw. "Analysis of SHA-1 in encryption mode". In: *Cryptographers' Track at the RSA Conference*. Springer. 2001.

[72] NESSIE. *New European Schemes for Signatures, Integrity and Encryption*. URL: https://www.cosic.esat.kuleuven.be/nessie/index.html.

[73] Helena Handschuh and David Naccache. "SHACAL: a family of block ciphers". In: *Submission to the NESSIE project* (2002).

[74] Seokhie Hong, Jongsung Kim, Sangjin Lee, and Bart Preneel. "Related-key rectangle attacks on reduced versions of SHACAL-1 and AES-192". In: *International Workshop on Fast Software Encryption*. Springer. 2005.

[75] FIPS. “Secure Hash Standard”. In: *Federal Information Processing Standards Publication 180-4* (2015).

A Characteristics

We describe the characteristics covering most rounds that we obtained in Section 5. For each characteristic, we provide the difference of the master key words Δ_{mk}, the difference of the plaintext words Δ_p and the difference of the ciphertext words Δ_c. Furthermore, for each round $i = 0, 1, \ldots$ of the cipher, we provide the difference of the i-th round key words, the output difference of the i-th round function Δ_{x_i}, the (cumulative) weight of the operations that compute the i-th round key words w_{k_i} and the weight of the i-th round function w_{x_i}. The differences are given in hexadecimal values.

Table 9. The 60-round strong related-key characteristic of TEA.

i-th round	Δ_{x_i}	w_{x_i}
0	(0x00000000, 0x80000000)	0
1	(0x80000000, 0x00000000)	1
2	(0x00000000, 0x80000000)	0
3	(0x80000000, 0x00000000)	1
4	(0x00000000, 0x80000000)	0
5	(0x80000000, 0x00000000)	1
6	(0x00000000, 0x80000000)	0
7	(0x80000000, 0x00000000)	1
8	(0x00000000, 0x80000000)	0
9	(0x80000000, 0x00000000)	1
10	(0x00000000, 0x80000000)	0
11	(0x80000000, 0x00000000)	1
12	(0x00000000, 0x80000000)	0
13	(0x80000000, 0x00000000)	1
14	(0x00000000, 0x80000000)	0
15	(0x80000000, 0x00000000)	1
16	(0x00000000, 0x80000000)	0
17	(0x80000000, 0x00000000)	1
18	(0x00000000, 0x80000000)	0
19	(0x80000000, 0x00000000)	1
20	(0x00000000, 0x80000000)	0
21	(0x80000000, 0x00000000)	1
22	(0x00000000, 0x80000000)	0
23	(0x80000000, 0x00000000)	1
24	(0x00000000, 0x80000000)	0
25	(0x80000000, 0x00000000)	1
26	(0x00000000, 0x80000000)	0
27	(0x80000000, 0x00000000)	1
28	(0x00000000, 0x80000000)	0
29	(0x80000000, 0x00000000)	1
30	(0x00000000, 0x80000000)	0
31	(0x80000000, 0x00000000)	1
32	(0x00000000, 0x80000000)	0
33	(0x80000000, 0x00000000)	1
34	(0x00000000, 0x80000000)	0
35	(0x80000000, 0x00000000)	1
36	(0x00000000, 0x80000000)	0
37	(0x80000000, 0x00000000)	1
38	(0x00000000, 0x80000000)	0
39	(0x80000000, 0x00000000)	1
40	(0x00000000, 0x80000000)	0
41	(0x80000000, 0x00000000)	1
42	(0x00000000, 0x80000000)	0
43	(0x80000000, 0x00000000)	1
44	(0x00000000, 0x80000000)	0
45	(0x80000000, 0x00000000)	1
46	(0x00000000, 0x80000000)	0
47	(0x80000000, 0x00000000)	1
48	(0x00000000, 0x80000000)	0
49	(0x80000000, 0x00000000)	1
50	(0x00000000, 0x80000000)	0
51	(0x80000000, 0x00000000)	1
52	(0x00000000, 0x80000000)	0
53	(0x80000000, 0x00000000)	1
54	(0x00000000, 0x80000000)	0
55	(0x80000000, 0x00000000)	1
56	(0x00000000, 0x80000000)	0
57	(0x80000000, 0x00000000)	1
58	(0x00000000, 0x80000000)	0
59	(0x80000000, 0x00000000)	1
Total		30
Δ_p	(0x80000000, 0x00000000)	
Δ_c	(0x80000000, 0x00000000)	
Δ_{mk}	(0x00000000, 0x00000000, 0x00000000, 0x84000000)	

Table 10. The three full-round related-key characteristics with total weight 0 of TEA.

Δ_{mk}	Δ_p	Δ_c
(0x80000000, 0x80000000, 0x80000000, 0x80000000)	(0x00000000, 0x00000000)	(0x00000000, 0x00000000)
(0x00000000, 0x00000000, 0x80000000, 0x80000000)	(0x00000000, 0x00000000)	(0x00000000, 0x00000000)
(0x80000000, 0x80000000, 0x00000000, 0x00000000)	(0x00000000, 0x00000000)	(0x00000000, 0x00000000)

Table 11. The 18-round strong related-key characteristic of XTEA.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
0	0x00000000	0	(0x00010000, 0x44200000)	9
1	0x00000000	0	(0x44200000, 0x04000000)	6
2	0x00000000	0	(0x04000000, 0x80000000)	6
3	0x80000000	0	(0x80000000, 0x00000000)	2
4	0x80000000	0	(0x00000000, 0x00000000)	0
5	0x00000000	0	(0x00000000, 0x00000000)	0
6	0x00000000	0	(0x00000000, 0x00000000)	0
7	0x00000000	0	(0x00000000, 0x00000000)	0
8	0x00000000	0	(0x00000000, 0x00000000)	0
9	0x00000000	0	(0x00000000, 0x00000000)	0
10	0x00000000	0	(0x00000000, 0x00000000)	0
11	0x00000000	0	(0x00000000, 0x00000000)	0
12	0x80000000	0	(0x00000000, 0x80000000)	0
13	0x80000000	0	(0x80000000, 0x04000000)	2
14	0x00000000	0	(0x04000000, 0x44200000)	6
15	0x00000000	0	(0x44200000, 0x00010000)	6
16	0x00000000	0	(0x00010000, 0xc4310800)	9
17	0x00000000	0	(0xc4310800, 0x01010040)	11
Total		0		57
Δ_p	(0xc4310800, 0x00010000)			
Δ_c	(0xc4310800, 0x01010040)			
Δ_{mk}	(0x00000000, 0x00000000, 0x80000000, 0x00000000)			

Table 12. The 27-round weak related-key characteristic of XTEA.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
0	0x00000000	0	(0x00000000, 0x00000000)	0
1	0x80000000	0	(0x00000000, 0x80000000)	0
2	0x80000000	0	(0x80000000, 0x04000000)	2
3	0x40200000	1.179	(0x04000000, 0x04000000)	4
4	0x40200000	0	(0x04000000, 0x80000000)	4
5	0x80000000	0	(0x80000000, 0x00000000)	2
6	0x80000000	0	(0x00000000, 0x00000000)	0
7	0x00000000	0	(0x00000000, 0x00000000)	0
8	0x00000000	0	(0x00000000, 0x00000000)	0
9	0x00000000	0	(0x00000000, 0x00000000)	0
10	0x80000000	0	(0x00000000, 0x80000000)	0
11	0x80000000	0	(0x80000000, 0x04000000)	2
12	0x40200000	1.006	(0x04000000, 0x04000000)	4
13	0x40200000	0.734	(0x04000000, 0x80000000)	4
14	0x80000000	0	(0x80000000, 0x00000000)	2
15	0x80000000	0	(0x00000000, 0x00000000)	0
16	0x00000000	0	(0x00000000, 0x00000000)	0
17	0x00000000	0	(0x00000000, 0x00000000)	0
18	0x80000000	0	(0x00000000, 0x80000000)	0
19	0x00000000	0	(0x80000000, 0x84000000)	2
20	0x40600000	2.067	(0x84000000, 0x80000000)	4
21	0x80000000	0	(0x80000000, 0x80000000)	2
22	0x80000000	0	(0x80000000, 0x84000000)	2
23	0xc0600000	1.907	(0x84000000, 0x80000000)	4
24	0x00000000	0	(0x80000000, 0x00000000)	2
25	0x80000000	0	(0x00000000, 0x00000000)	0
26	0x80000000	0	(0x00000000, 0x80000000)	0
Total		6.893		40
Δ_p	(0x00000000, 0x00000000)			
Δ_c	(0x80000000, 0x00000000)			
Δ_{mk}	(0x00000000, 0x80000000, 0xc0200000, 0x80000000)			

Table 13. The 15-round strong related-key characteristic of HIGHT. The round -1 corresponds to the initial key whitening.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
-1	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x09, 0x20, 0xb8, 0xe9, 0x80, 0x00)	1
0	(0x00, 0x00, 0x80, 0x00)	0	(0x00, 0x00, 0x20, 0xb8, 0xe9, 0x80, 0x00, 0x00)	3
1	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0xb8, 0x2c, 0x80, 0x00, 0x00, 0x00)	6
2	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x2c, 0x80, 0x00, 0x00, 0x00, 0x00)	3
3	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00)	3
4	(0x00, 0x80, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	0
5	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	0
6	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	0
7	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	0
8	(0x80, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80)	0
9	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0xd4, 0x80, 0x00)	5
10	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x90, 0xd4, 0x80, 0x00, 0x00)	1
11	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0xe9, 0x90, 0x95, 0x80, 0x00, 0x00, 0x00)	7
12	(0x00, 0x00, 0x00, 0x00)	0	(0xe9, 0x00, 0x95, 0x80, 0x00, 0x00, 0x00, 0x80)	1
13	(0x00, 0x00, 0x00, 0x80)	0	(0x00, 0xe9, 0x80, 0x00, 0x00, 0xa4, 0x80, 0xe9)	9
14	(0x00, 0x00, 0x00, 0x00)	0	(0x80, 0xe9, 0x80, 0x00, 0x89, 0xa4, 0x2b, 0xe9)	6
Total		0		45
Δ_p	(0x00, 0x00, 0x09, 0x20, 0xb8, 0xe9, 0x80, 0x00)			
Δ_c	(0x80, 0xe9, 0x80, 0x00, 0x89, 0xa4, 0x2b, 0xe9)			
Δ_{mk}	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00)			

Table 14. The 14-round weak related-key characteristic of HIGHT. The round -1 corresponds to the initial key whitening.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
-1	(0x00, 0x00, 0x00, 0x40)	0	(0x62, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	1
0	(0x40, 0x00, 0x00, 0x00)	0.791	(0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	2
1	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0)	0
2	(0x00, 0x00, 0x00, 0x3a)	1.0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x00)	0
3	(0x00, 0x00, 0x00, 0x40)	0.752	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	1
4	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	0
5	(0x00, 0x00, 0x00, 0x40)	0.791	(0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x00, 0x00)	1
6	(0x00, 0x00, 0x2e, 0x00)	4.0	(0x00, 0x00, 0x00, 0x00, 0xc0, 0x00, 0x00, 0x00)	5
7	(0x00, 0x00, 0x40, 0x00)	1.093	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	1
8	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	0
9	(0x00, 0x00, 0xc0, 0x00)	0.046	(0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00)	1
10	(0x00, 0x16, 0x00, 0x00)	4.0	(0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00)	0
11	(0x00, 0x40, 0x00, 0x00)	0.142	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	1
12	(0x00, 0x00, 0x00, 0x00)	0	(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)	0
13	(0x00, 0x00, 0xc0, 0x00)	0.476	(0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00)	1
Total		13.091		14
Δ_p	(0x62, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40)			
Δ_c	(0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00)			
Δ_{mk}	(0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x7a, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00)			

Table 15. The 7-round weak related-key characteristic of LEA.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
0	(0x20000000, 0x00000000, 0x00000000, 0x00000000)	0.408	(0x80000000, 0x40000000, 0xc0000010, 0x4000000c)	14
1	(0x40000000, 0x00000000, 0x00000000, 0x00000000)	0.462	(0x00000000, 0x80000000, 0x80000000, 0x80000000)	7
2	(0x80000000, 0x00000000, 0x00000000, 0x00000000)	0.695	(0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
3	(0x00000001, 0x00000000, 0x00000000, 0x00000000)	0	(0x00000200, 0x00000000, 0x00000000, 0x00000000)	1
4	(0x00000002, 0x00000000, 0x00000000, 0x00000000)	0	(0x00040400, 0x00000000, 0x00000000, 0x00000200)	2
5	(0x00000004, 0x00000000, 0x00000000, 0x00000000)	0	(0x08080800, 0x00000000, 0x00000040, 0x00040400)	4
6	(0x00000008, 0x00000000, 0x00000000, 0x00000000)	1.0	(0x10101010, 0x00000002, 0x00008088, 0x08080800)	8
Total		2.565		36
Δ_p	(0x4000000c, 0x2040000c, 0x20400004, 0x20400082)			
Δ_c	(0x10101010, 0x00000002, 0x00008088, 0x08080800)			
Δ_{mk}	(0x10000000, 0x00000000, 0x00000000, 0x00000000)			

Table 16. The 30-round strong related-key characteristic of SHACAL-1.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
0	0x80000000	0	(0x00000000, 0x00000002, 0x00000000, 0x80100008, 0x00008000)	7
1	0x80000000	0	(0x00008000, 0x00000000, 0x80000000, 0x00000000, 0x80100008)	5
2	0x00000000	0	(0x00000008, 0x00008000, 0x00000000, 0x80000000, 0x00000000)	5
3	0x00000000	0	(0x00000100, 0x00000008, 0x00002000, 0x00000000, 0x80000000)	5
4	0x80000000	0	(0x00000008, 0x00000100, 0x00000002, 0x00002000, 0x00000000)	6
5	0x00000000	0	(0x00000102, 0x00000008, 0x00000040, 0x00000002, 0x00002000)	8
6	0x00000000	0	(0x00000000, 0x00000102, 0x00000002, 0x00000040, 0x00000002)	8
7	0x00000000	0	(0x00000000, 0x00000000, 0x80000040, 0x00000002, 0x00000040)	5
8	0x80000000	0	(0x00000000, 0x00000000, 0x00000000, 0x80000040, 0x00000002)	5
9	0x00000000	0	(0x00000002, 0x00000000, 0x00000000, 0x00000000, 0x80000040)	3
10	0x80000000	0	(0x00000000, 0x00000002, 0x00000000, 0x00000000, 0x00000000)	3
11	0x00000000	0	(0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000)	1
12	0x80000000	0	(0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000)	1
13	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000)	1
14	0x80000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
15	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
16	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
17	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
18	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
19	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
20	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
21	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
22	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
23	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
24	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
25	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
26	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
27	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
28	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
29	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
Total		0		63

Δ_p	(0x00000002, 0x00000000, 0x80100008, 0x00008000, 0x80000040)
Δ_c	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)
Δ_{mk}	(0x80000000, 0x80000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x80000000, 0x00000000, 0x80000000, 0x00000000, 0x80000000, 0x00000000)

Table 17. The 25-round weak related-key characteristic of SHACAL-1.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
0	0x00000000	0	(0x00000000, 0x00000000, 0x80000040, 0x00000002, 0x80000000)	3
1	0x80000000	0	(0x00000000, 0x00000000, 0x00000000, 0x80000040, 0x00000002)	3
2	0x00000000	0	(0x00000002, 0x00000000, 0x00000000, 0x00000000, 0x80000040)	3
3	0x80000000	0	(0x00000000, 0x00000002, 0x00000000, 0x00000000, 0x00000000)	3
4	0x00000000	0	(0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000)	1
5	0x80000000	0	(0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000)	1
6	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000)	1
7	0x80000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
8	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
9	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
10	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
11	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
12	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
13	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
14	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
15	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
16	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
17	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
18	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
19	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
20	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
21	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
22	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
23	0x00000003	1	(0x00000001, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	4
24	0x00000000	0	(0x00000020, 0x00000001, 0x00000000, 0x00000000, 0x00000000)	3
Total		1		22

Δ_p	(0x00000000, 0x00000102, 0x00000002, 0x80000000, 0x80000000)
Δ_c	(0x00000020, 0x00000001, 0x00000000, 0x00000000, 0x00000000)
Δ_{mk}	(0x00000000, 0x80000000, 0x00000000, 0x80000000, 0x00000000, 0x80000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)

Table 18. The 23-round strong related-key characteristic of SHACAL-2.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
0	0x00000000	0	(0x00000000, 0x00000000, 0x221c0240, 0x80000000, 0x00000000, 0x00082200, 0x20040000, 0x80000000)	12
1	0x00000000	0	(0x80000000, 0x00000000, 0x00000000, 0x221c0240, 0x00000000, 0x00000000, 0x00082200, 0x20040000)	26
2	0x00000000	0	(0x00000000, 0x80000000, 0x00000000, 0x00000000, 0x02100040, 0x00000000, 0x00000000, 0x00082200)	7
3	0x00000000	0	(0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000, 0x02100040, 0x00000000, 0x00000000)	4
4	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000, 0x02100040, 0x00000000)	3
5	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000, 0x02100040)	4
6	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000)	1
7	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000)	1
8	0x80000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000)	0
9	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
10	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	3
11	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
12	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
13	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
14	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
15	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
16	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
17	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
18	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
19	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
20	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
21	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
22	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
Total		0		58

Δ_p	(0x00000000, 0x00000000, 0x221c0240, 0x80000000, 0x00000000, 0x00082200, 0x20040000, 0x80000000)
Δ_c	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)
Δ_{mk}	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x80000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)

Table 19. The 22-round weak related-key characteristic of SHACAL-2.

i-th round	Δ_{k_i}	w_{k_i}	Δ_{x_i}	w_{x_i}
0	0x00000000	0	(0x00000000, 0x00020000, 0x00000000, 0x00000000, 0x01000840, 0x00000000, 0x00000000, 0x88000020)	6
1	0x00000000	0	(0x00000000, 0x00000000, 0x00020000, 0x00000000, 0x00000000, 0x01000840, 0x00000000, 0x00000000)	4
2	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00020000, 0x00000000, 0x00000000, 0x01000840, 0x00000000)	4
3	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00020000, 0x00000000, 0x00000000, 0x01000840)	4
4	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00020000, 0x00000000, 0x00000000)	1
5	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00020000, 0x00000000)	1
6	0x00020000	0.405	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00020000)	3
7	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
8	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
9	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
10	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
11	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
12	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	3
13	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
14	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
15	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
16	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
17	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
18	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
19	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
20	0x00000000	0	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	0
21	0x80004400	6.262	(0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)	6
Total		6.67		29

Δ_p (0x00000000, 0x00020000, 0x00000000, 0x00000000, 0x01000840, 0x00000000, 0x00000000, 0x88000020)

Δ_c (0x80004400, 0x00000000, 0x00000000, 0x00000000, 0x80004400, 0x00000000, 0x00000000, 0x00000000)

Δ_{mk} (0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00020000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000)