A Bit-Vector Differential Model for the Modular Addition by a Constant

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, 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, Bergen, Norway

Abstract. ARX algorithms are a class of symmetric-key algorithms constructed by Addition, Rotation, and XOR, which achieve the best software performances in low-end microcontrollers. To evaluate the resistance of an ARX cipher against differential cryptanalysis and its variants, the recent automated methods employ constraint satisfaction solvers, such as SMT solvers, to search for optimal characteristics. The main difficulty to formulate this search as a constraint satisfaction problem is obtaining the differential models of the non-linear operations, that is, the constraints describing the differential probability of each non-linear operation of the cipher. 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 also represent an SMT-based automated method to look for differential characteristics of ARX, including constant additions, and we provide an open-source tool `ArxPy` to find ARX differential characteristics in a fully automated way. To provide some examples, we have searched for related-key differential characteristics of TEA, XTEA, HIGHT, and LEA, obtaining better results than previous works. Our differential model and our automated tool allow cipher designers to select the best constant inputs for modular additions and cryptanalysts to evaluate the resistance of ARX ciphers against differential attacks.

Keywords: Modular addition by a constant · Differential probability · ARX · SMT · Automated search · Bit-vector theory

S. A. Azimi and A. Ranea—These authors contributed equally to this work.

S. Moriai and H. Wang (Eds.): ASIACRYPT 2020, LNCS 12491, pp. 385–414, 2020.
https://doi.org/10.1007/978-3-030-64837-4_13

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 will be more than 25 billion connected devices forming the IoT by 2021 [1]. 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 [2].

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 [3]. There are many noteworthy ARX algorithms, such as the hash function BLAKE [4], the stream cipher Salsa20 [5], the MAC algorithm Chaskey [6] and notable block ciphers like HIGHT [7], LEA [8], SPECK [9] or SPARX [10]. 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 [11], TEA [12], or XTEA [13].

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 [8,14]. These attacks exploit differences in the inputs that propagate through the cipher with high probability. The standard approach to show an ARX cipher is secure against differential attacks is by finding the optimal characteristics (i.e., trails of differences with the highest probabilities) that cover most of the rounds of the cipher and checking that their probabilities are negligible [7,8]. When the best attack in the design stage is a differential attack, the number of rounds of the cipher is determined by the number of rounds that optimal characteristics can cover with non-negligible probability. Thus, searching for optimal characteristics 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 [15,16] based on Matsui's algorithm [17], and the recent automated methods based on constraint satisfaction problems, such as SMT (Satisfiability Modulo Theories) or MILP (Mixed Integer Linear Programming) problems [18,19]. Automated methods formulate the characteristic search problem as a constraint satisfaction problem and delegate the solving task to one of the powerful off-the-shelf constraint satisfaction solvers available nowadays [20,21]. The main difficulty to formulate the 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 [22–24]. 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 $O(\log_2 n)$ [22]. 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 of ARX ciphers [18,24,25].

However, no bit-vector differential model has been proposed for the modular addition with a constant input, preventing from searching for characteristics 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 [26], 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 which formulates the SMT problems from the Single Static Assignment (SSA) form of the cipher. We have implemented our method in an open-source tool `ArxPy`[1], which fully automated the search of ARX characteristics. `ArxPy` offers a simple interface to represent any ARX cipher, different types of characteristics to search, and a complete documentation. To provide some examples, we have applied our characteristic search method to several ARX ciphers containing constant additions. In particular, we have searched for different types of related-key characteristics of TEA, XTEA, HIGHT and LEA. With our automated approach, we have revisited results previously found with manual and ad-hoc techniques, and we have obtained better characteristics in terms of probability and number of rounds.

With our bit-vector model for the constant addition, the SMT-based automated method, 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 attacks. Thus, cipher designers can choose the best constants for the modular additions and optimize the number of rounds to strike a balance between security and efficiency.

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

Outline. The notation and preliminaries are introduced in Sect. 2, and the bit-vector model for the modular addition by a constant is described in Sect. 3. Section 4 illustrates the formulation of the characteristic search as a sequence of SMT problems, SMT-based search method, and the encoding of bit-vector SMT problems for ARX characteristics. Section 5 presents the characteristics found for TEA, XTEA, HIGHT, and LEA using our automated approach and finally Sect. 6 concludes the paper and addresses future works.

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 [27]. 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 [22].

Apart from the basic bit-vector operations listed in Table 1, we will also employ the following well-known bit-vector functions: $\mathsf{Carry}, \mathsf{Rev}, \mathsf{RevCarry}, \mathsf{HW}$ 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 $c[i+1] = (x[i] \wedge y[i]) \oplus (x[i] \wedge c[i]) \oplus (y[i] \wedge c[i])$. Note that the carry has bit-vector complexity $O(1)$, since $\mathsf{Carry}(x, y) = x \oplus y \oplus (x \boxplus y)$.

The bit-reversal function $\mathsf{Rev}(x)$ reverses the order of bits of x, i.e., $\mathsf{Rev}(x) = (x[0], x[1], \ldots, x[n-1])$. We will use this function to define the reverse carry, $\mathsf{RevCarry}(x, y) = \mathsf{Rev}(\mathsf{Carry}(\mathsf{Rev}(x), \mathsf{Rev}(y)))$. The hamming weight $\mathsf{HW}(x)$ returns an n-bit vector denoting the number of non-zero bits of the n-bit input x. Lastly, the leading zeros function $\mathsf{LZ}(x)$ marks the leading zeros of an n-bit input x, that is, for $0 \leq i < n$ we have $\mathsf{LZ}(x)[i] = 1 \iff x[n-1, i] = 0$. Note that the functions $\mathsf{Rev}, \mathsf{RevCarry}, \mathsf{HW}$ and LZ can be computed using a divide and conquer approach with bit-vector complexity $O(\log_2 n)$ [28].

2.2 Differential Cryptanalysis

A block cipher is a family of permutations parameterized 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 powerful attacks, specially to ARX primitives, is differential cryptanalysis [29]. Basically, it exploits 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 [29] 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 [30] 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 $O(2^m + 2^{n-m})$ [31]. 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.

Searching for Differentials. The most difficult step to launch a differential attack is finding a differential with high probability. The main approach is to analyse how differences propagate through the round function and search for a characteristic, that is, a trail of differences

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

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 to compute 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 [32], which allows to compute 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 [18,19,23,25,33,34].

SMT Solvers. A recent approach to search for characteristics of ARX ciphers is by formulating the search problem as an SMT problem in the bit-vector theory [18,23–25,35]. 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 [20]. Most SMT solvers can not only determine the satisfiability of a problem but also 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., $\mathsf{Equals}, \mathsf{NotEquals}, \mathsf{Implies}$, etc.) and with the usual bit-vector operations (e.g., $\oplus, \boxplus, \lll$, etc.).

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 $\mathtt{True}$ 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 et al. [22] 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 `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 [26].

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 $0 \le i < n$ by

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

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 [36, Corollary 2] [37] 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 Eq. (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 [38], that is, the differential properties of the 2-input addition are the same of some quadratic function. In particular, the set of inputs (x_1, x_2) satisfying a differential $((\Delta_{x_1}, \Delta_{x_2}) \rightarrow \Delta_y)$ 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 checked experimentally how accurate was the approximation given by Eq. (1) for 8-bit constants a. For most values of a, validity formulas differ roughly in 2^{13} out of all 2^{16} differentials, and for those differentials where they did not differ, the difference between their weights was significantly high in 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 [26]; 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 $S_i = 11*$, $S_{i-1} \in \{010, 100, 110, 000\}$ and for the first three cases, δ_{i-1} is equal to $a[i-2]$. However, considering the forth case, i.e., $S_{i-1} = 000$, δ_{i-1} depends on δ_{i-2} and this dependency will proceed until we obtain a state $S_{i-\ell_i} \neq 000$ for some positive integer ℓ_i. Thus, δ_{i-1} has the following expression when $S_i = 11*$,

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

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

Definition 1. *Let $S_i = 11*$. 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 $S_{i-\ell_i} \neq 000$ and the remaining states in the chain (if any) are all equal to 000.

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 001, 110 or 111. While $S_i = 001$ 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 $11\mathtt{b}$, 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 $S_i = 11\mathtt{b}$, $\varphi_i = 0$ 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 Eq. (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,*

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

Then, for all states $S_i = \mathtt{11*}$, 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 $s_{00*}[i] = 1$ (resp. $s_{**1}[i] = 1$) if and only if $S_i = \mathtt{00*}$ (resp. $S_i = \mathtt{**1}$).

The first operand of $g[i]$, i.e., $(s_{**1} \oplus a')[i]$, 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

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

Thus, $c[i] = c[i-1] \wedge \neg a'[i-1]$ and $c[i-\ell_i+1] = c[i-\ell_i] \wedge \neg s_{00*}[i-\ell_i-1] = 0$; otherwise, for $0 \leq j \leq i-\ell_i-1$ we will obtain $s_{00*}[j] = 0$ 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 $(s_{**1} \oplus a')[i] = 1$, 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 $S_i = \mathtt{001}$ (respectively $S_i = \mathtt{11*}$). 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 $O(1)$.

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 $1/2$. Since $\varphi_i = 1/2$ when $S_i \in \{01*, 10*\}$, it is easy to see that

$$-\sum_{i\notin\mathcal{I}}\log_2(\varphi_i) = \mathsf{HW}((u \oplus v) \ll 1). \tag{4}$$

Equation 4 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 3. *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 $S_i =$ `11*`. Then, $\varphi_i = \dfrac{p_i}{2^{\ell_i-1}}$.

Proof. Considering the definition of φ_i when $S_i =$ `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 Eq. (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 $0 < \varphi_i \le 1$ and $\ell_i > 1$ results in $0 < p_i \le 2^{\ell_i-1}$. Thus, p_i is always a positive integer. □

Due to Lemma 3, 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 4. *Consider the n-bit vector* $s_{000} = \neg(u \ll 1) \wedge \neg(v \ll 1)$. *Then,*

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

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

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

Therefore, we have $\sum_{i\in\mathcal{I}}(\ell_i - 1) = \#\{S_j | S_j = \mathtt{000} \text{ and } \exists i \in \mathcal{I} \; s.t. \; S_j \in \Gamma_i\}$. When $S_j = \mathtt{000}$, the next state S_{j+1} will be a member of the set $\{\mathtt{000}, \mathtt{11*}\}$. As a result, it is easy to see that for an arbitrary j, if S_j is equal to $\mathtt{000}$, 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

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

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

Since we are assuming that the differential is valid, there are no states $S_j = \mathtt{001}$, and $s_{000}[j] = 1$ if and only if $S_j = \mathtt{000}$. 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 $\mathtt{1}$ if and only if $S_i \in \Gamma'$. Therefore, we obtain

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

□

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 5. *Consider the following n-bit values,*

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

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

Proof. For each $i \in \mathcal{I}$ and $0 \leq j < n$, 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

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

result in $s[i-1, i-\ell_i] = a[i-2, i-\ell_i] + a[i-\ell_i-1]$. In particular, $s[i-1, i-\ell_i] \leq 2^{\ell_i - 1}$ 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

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

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 $i \in \mathcal{I}$

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

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 $s[i-1, i-\ell_i] \leq \texttt{10...0}$. 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 3, we obtain that $w[i-1, i-\ell_i] = p_i$. □

Recall that both LZ and $\mathsf{RevCarry}$ 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,

$$\mathsf{apxlog}_2(x) \triangleq m + \frac{\mathsf{Truncate}(x[m-1,0])}{2^4}\,, \tag{5}$$

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 $\mathsf{Truncate}$ 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 $\mathsf{ParallelLog}$ and $\mathsf{ParallelTrunc}$. 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 $\mathsf{ParallelLog}$ *and* $\mathsf{ParallelTrunc}$ *by*

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

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

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

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 6. *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 5, $w[i-1, i-\ell_i]$ 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,

$$\mathsf{apxweight}_{\boxplus_a}(u, v) \triangleq -\sum_{i \notin \mathcal{I}} \log_2(\varphi_i) - \sum_{i \in \mathcal{I}} \log_2\left(\frac{1}{2^{\ell_i - 1}}\right) - \sum_{i \in \mathcal{I}} \mathsf{apxlog}_2(p_i)\,. \quad (6)$$

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 $\mathsf{BvWeight}(u, v, a)$.

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

Lemma 7. *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 Eq. 4 and Lemmas 4 and 6 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,

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

□

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

Theorem 4 shows that $\mathsf{BvWeight}$ 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 $-0.029 \cdot n \leq E \leq 0$.

The next subsection is devoted to the proof of Theorem 4, where we will also analyse the error caused by our approximated binary logarithm.

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 Eqs. (3) and (6), the gap between $\mathsf{weight}_{\boxplus_a}(u, v)$ and $\mathsf{apxweight}_{\boxplus_a}(u, v)$ is

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

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. While $\mathsf{apxlog}_2(x)$ considers four bits of the input x for the fraction part, we generalize the definition of $\mathsf{apxlog}_2(x)$ to include variable number of bits of x. Given a positive integer x and the corresponding $m = \lfloor \log_2(x) \rfloor$, we define $\mathsf{apxlog}_2^{\kappa}$ 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, and for $\kappa = 4$ we have $\mathsf{apxlog}_2^4(x) = \mathsf{apxlog}_2(x)$, which is defined in Eq. (5).

The following lemma bounds the approximation error of apxlog_2 when $\kappa \geq \lfloor \log_2(x) \rfloor$, with a similar proof as [39] 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 8. *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 $B = 1 - \big(1 + \ln(\ln(2))\big)/\ln(2) \approx 0.086$.*

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 $\log_2(3) - (1 + \frac{1}{2}) \approxeq 0.085$. Similar to $\mathsf{apxlog}_2^{\kappa}$, we generalize $\mathsf{apxweight}_{\boxplus_a}$ as

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

where $\mathsf{apxweight}^{4}_{\boxplus_a}(u, v) = \mathsf{apxweight}_{\boxplus_a}(u, v)$ is defined by Eq. (6).

Finally, we prove Theorem 4 by generalizing the definition of approximated weight error $E_\kappa = \mathsf{weight}_{\boxplus_a}(u, v) - \mathsf{apxweight}^{\kappa}_{\boxplus_a}(u, v)$ and showing that if we dedicate at least 4 bits to the fraction precision κ, the approximation error is always bounded by $-0.086 \cdot (n/3) \leq E_\kappa \leq 0$.

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 $\ell_i \geq 3$, let $p_i = 2^{m_i} + b_i$ such that m_i and b_i are two non-negative integers, $m_i \leq \ell_i - 2$ and $0 \leq b_i < 2^{m_i}$. If $\ell_i \leq \kappa + 2$, we obtain $m_i \leq \kappa$ and $\mathsf{apxlog}_2^{\kappa}(p_i) = m_i + b_i \cdot 2^{-m_i}$. Thus, the resulting error

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

is exactly the same as the error defined in Proposition 8, and $0 \leq e_i \leq B \approx 0.086$.

On the other hand, for $m_i > \kappa$, i.e., $\ell_i \geq \kappa + 3$, 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 $0 \leq t_i < 2^{\kappa}$ 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 Proposition 8, the new error e'_i is bounded by $0 \leq e'_i \leq B + 2^{-\kappa}$. Finally, by defining the conditional index set $\mathcal{I}_\alpha^\beta = \{i \in \mathcal{I} \mid \alpha \leq \ell_i \leq \beta\}$ we obtain

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

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

$$0 \geq E_\kappa \geq -\Big(\frac{B}{3} \sum_{i \in \mathcal{I}_3^n} \ell_i\Big) \geq -\Big(\frac{B}{3} n\Big) \approx -0.029n\,.$$

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 $-0.035n$, 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 of ARX ciphers.

4 SMT-Based Search of Characteristics

In this section, we describe how to formulate the search of an optimal characteristic as a sequence of SMT problems, which can be solved by an off-the-shelf SMT solver such as Boolector [40] or STP [41]. This approach was originally used by Mouha and Preneel to search for single-key characteristics of Salsa20 [18].

To search for characteristics up to probability 2^{-n}, the probability space is decomposed into n intervals $I_w = \big(2^{-w-1}, 2^{-w}\big]$, 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. Sect. 4.1 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. 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 the rectangle or rebound attacks [42,43].

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, this method search by default 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 $\mathsf{weight}(\Omega_E) = 0$.

In some cases, the integer weight computed by the SMT solver is not the exact integer weight of the characteristic, but a bound of the error ϵ is known. For example, for an ARX cipher with constant additions, the weight of the constant additions is computed in the SMT problems using Theorem 4, which introduces an error that can be bounded (Theorem 4). Nonetheless, this method can find the optimal characteristic in this case by finding all the characteristics with integer weights $\{\hat{w}, \hat{w}+1, \ldots, \hat{w}+\lfloor\epsilon\rfloor\}$, where $\hat{w}$ is the integer weight of the first characteristic found by the SMT solver.

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.

4.1 Encoding the SMT Problems

In this section, we explain how to formulate the decision problem of determining whether there exists a characteristic Ω 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 2.

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

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

Table 2. 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

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.

4.2 Implementation

We have developed an open-source[2] tool to find characteristics of ARX ciphers implementing the method described in the previous sections. `ArxPy` provides high-level functions that automate the search of optimal characteristics, a simple interface to represent ARX ciphers, and a complete documentation in HTML format, among other features.

`ArxPy` workflow is represented in Fig. 1. The user first defines the ARX cipher using the interface provided by `ArxPy` and chooses the parameters of the search

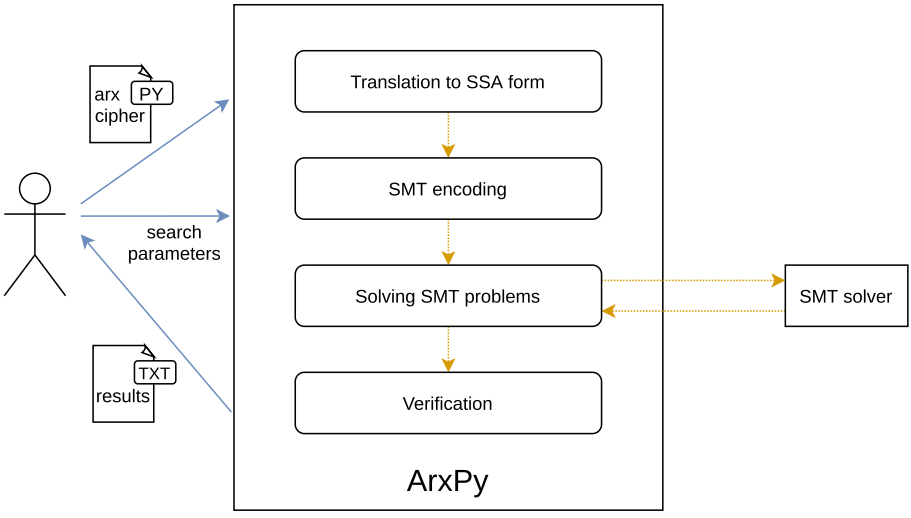

Fig. 1. Workflow of `ArxPy`

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

(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. For each satisfiable SMT problem 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.

Internally, `ArxPy` is implemented in Python 3 and uses the libraries `SymPy` [44] to obtain the SSA representation through symbolic execution and `PySMT` [45] 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 method for finding characteristics to some ARX ciphers that include constant additions. In particular, we have searched for related-key characteristics of TEA, XTEA, HIGHT and LEA.

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 can 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 makes them more suitable to be analysed with other types of differences, such as additive differences in the case of TEA [15]. As a result, we have focused on searching related-key characteristics of some well-known ciphers.

However, the usual assumptions (i.e., round independence and the hypothesis of stochastic equivalence) do not always hold for related-key characteristics, as in this case. 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 $\Omega_i = (\Delta_{x_i} \rightarrow \cdots \rightarrow \Delta_{y_i})$ 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, we provide: (1) the theoretical key schedule and encryption integer weights (w_{KS}, w_E), computed by summing the weight of each ARX operation; (2) the empirical key schedule and encryption integer 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 extended version, we provide the

round weights and round differences for the characteristics covering the most rounds.

For the experiments, we have used `ArxPy` equipped with the SMT solver Boolector [40], winner of the SMT competition SMT-COMP 2019 in the bit-vector track [46]. We run the search for one week on a single core of an Intel Xeon 6244 at 3.60 GHz. Table 3 lists the characteristics we have found and compares them with the previous longest-known characteristics. Note that better characteristics could be found if the round-reduced searches are not stopped after the first valid characteristic or if more time is employed.

Table 3. Best related-key characteristics of XTEA, HIGHT and LEA.

Cipher	Ch. Type	Rounds	$(w_{\mathsf{KS}}, \overline{w_{\mathsf{KS}}})$	$(w_E, \overline{w_E})$	% valid keys	Reference
XTEA	Strongrelated-key	16	0	32	–	[47]
		16	(0,0)	(37, 32)	46%	This paper
		18	(0,0)	(57, 49)	48%	This paper
	Weakrelated-key	18	17	19	–	[47,48]
		18	(4, 3)	(16, 14)	100%	This paper
		27	(6, 5)	(40, 39)	7%	This paper
HIGHT	Strongrelated-key	10	0	12	–	[49]
		10	(0, 0)	(12, 9)	34%	This paper
		15	(0, 0)	(45, 42)	8%	This paper
	Weakrelated-key	12	2	19	–	[50]
		12	(2, 3)	(19, 17)	40%	This paper
		14	(13, 9)	(14, 11)	17%	This paper
LEA	Weakrelated-key	11	–	–	–	[8]
		6	(1, 1)	(24, 22)	100%	This paper
		7	(2, 4)	(36, 34)	100%	This paper

TEA. Designed by Wheeler and Needham, TEA [12] 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. Since the logical shifts propagate XOR differences deterministically, the encoding method presented in Sect. 4.1 can be easily extended to include these operations.

The best related-key characteristics were obtained by Kelsey, Schneier, and Wagner in [51]. 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 [30] that each TEA key has 3 other equivalent keys.

Using `ArxPy`, we revisited the results by Kelsey et al., but in a fully automated way. We found three related-key characteristic 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 [51], there is no need to search for characteristics containing more rounds of TEA and we stop at 60 rounds.

XTEA. To fix the weakness of TEA against related-key attacks, the same designers propose XTEA [13]. This block cipher has a 64-bit block size and a 128-bit key size. The ARX round function includes logical shifts, but the key schedule is composed exclusively of constant additions.

The longest related-key characteristics found so far are the 16-round strong related-key differential with weight 32, manually found by Lu in [47], and the 18-round weak related-key characteristic with weights $(w_{\mathsf{KS}}, w_E) = (19, 19)$, manually found by Lee et al. [48] but later improved to (17, 19) by Lu [47].

The results of our automated search are listed in Table 3. 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.

HIGHT. Adopted as an international standard by ISO/IEC [52], HIGHT [7] is a lightweight cipher with block size of 64 bits and a key size of 128 bits. The encryption function performs an 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 [49], and a 12-round weak characteristic with weights $(w_{\mathsf{KS}}, w_E) = (2, 19)$ found by Koo et al. [50].

In our automated search, we found related-key characteristic up to 15 rounds, listed in Table 3. 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 $13 + 14 = 27$.

LEA. Among the family of ARX ciphers LEA [8], we have 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.

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 [8]. Excluding these characteristics, there are no others examples of related-key characteristics of LEA. In our automated search, we found weak related-key characteristic up to 7 rounds valid for the full key space, listed in Table 3. 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.

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 an SMT-based automated method to search for characteristics of ARX ciphers including constant additions. Our method formulates the search problem as a sequence of SMT problems in the bit-vector theory, which are encoded from the SSA representation of the cipher and the bit-vector differential models of each operation. We have implemented our method in `ArxPy`, an open-source tool to find characteristic of ARX ciphers in a fully automated way. To show some examples, we have applied our automated method to search for equivalent keys and related-key characteristics of TEA, XTEA, HIGHT, and LEA. For TEA, we revisited previous results obtained in a manual approach, whereas for XTEA, HIGHT and LEA we improved the previous best-known related-key characteristics in both the strong-key and weak-key settings.

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 have been proposed so far, such as the modular multiplication. Obtaining differential models for more operations will allow designing ciphers with more flexibility, leading to new designs that potentially are more efficient.

Acknowledgements. Adrián Ranea is supported by a PhD Fellowship from the Research Foundation – Flanders (FWO). The authors would like to thank the anonymous reviewers for their comments and suggestions.

References

1. Omale, G.: Gartner identifies top 10 strategic IoT technologies and trends (2018). https://www.gartner.com/en/newsroom/press-releases/2018-11-07-gartner-identifies-top-10-strategic-iot-technologies-and-trends
2. National Institute of Standards and Technology. Lightweight cryptography project. https://csrc.nist.gov/Projects/Lightweight-Cryptography
3. Dinu, D., Corre, L.Y., Khovratovich, D., Perrin, L., Großschädl, J., Biryukov, A.: Triathlon of lightweight block ciphers for the internet of things. J. Cryptographic Eng. **9**(3), 283–302 (2019)
4. Aumasson, J.-P., Henzen, L., Meier, W., Phan, R.C.-W.: Sha-3 proposal blake. Submission to NIST (round 3), 92 (2008)

5. Bernstein, D.J.: The Salsa20 family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 84–97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_8
6. Mouha, N., Mennink, B., Van Herrewege, A., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: an efficient MAC algorithm for 32-bit microcontrollers. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 306–323. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13051-4_19
7. Hong, D., et al.: HIGHT: a new block cipher suitable for low-resource device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_4
8. Hong, D., Lee, J.-K., Kim, D.-C., Kwon, D., Ryu, K.H., Lee, D.-G.: LEA: a 128-bit block cipher for fast encryption on common processors. In: Kim, Y., Lee, H., Perrig, A. (eds.) WISA 2013. LNCS, vol. 8267, pp. 3–27. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05149-9_1
9. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptology ePrint Archive 2013, 404 (2013)
10. Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., Biryukov, A.: Design strategies for ARX with provable bounds: Sparx and LAX. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 484–513. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_18
11. Lai, X., Massey, J.L.: A proposal for a new block encryption standard. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 389–404. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46877-3_35
12. Wheeler, D.J., Needham, R.M.: TEA, a tiny encryption algorithm. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 363–366. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60590-8_29
13. Needham, R., Wheeler, D.: Tea extensions. Technical report, Computer Laboratory, University of Cambridge (1997)
14. Koo, B., Roh, D., Kim, H., Jung, Y., Lee, D.-G., Kwon, D.: CHAM: a family of lightweight block ciphers for resource-constrained devices. In: Kim, H., Kim, D.-C. (eds.) ICISC 2017. LNCS, vol. 10779, pp. 3–25. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78556-1_1
15. Biryukov, A., Velichkov, V.: Automatic search for differential trails in ARX ciphers. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 227–250. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04852-9_12
16. Biryukov, A., Velichkov, V., Le Corre, Y.: Automatic search for the best trails in ARX: application to block cipher Speck. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 289–310. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_15
17. Matsui, M.: On correlation between the order of S-boxes and the strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 366–375. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0053451
18. Mouha, N., Preneel., B.: Towards finding optimal differential characteristics for ARX: Application to Salsa20. IACR Cryptology ePrint Archive, 2013:328 (2013). http://eprint.iacr.org/2013/328
19. Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: MILP-based automatic search algorithms for differential and linear trails for speck. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 268–288. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_14

20. Barrett, C., Tinelli, C.: Satisfiability modulo theories. Handbook of Model Checking, pp. 305–343. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_11
21. Lodi, A.: Mixed integer programming computation. In: Jünger, M., et al. (eds.) 50 Years of Integer Programming 1958-2008, pp. 619–645. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-540-68279-0_16
22. Lipmaa, H., Moriai, S.: Efficient algorithms for computing differential properties of addition. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 336–350. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45473-X_28
23. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON block cipher family. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 161–185. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_8
24. Liu, Y., De Witte, G., Ranea, A., Ashur, T.: Rotational-XOR cryptanalysis of reduced-round SPECK. IACR Trans. Symmetric Cryptol. **2017**(3), 24–36 (2017). https://doi.org/10.13154/tosc.v2017.i3.24-36
25. Song, L., Huang, Z., Yang, Q.: Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In: Liu, J.K., Steinfeld, R. (eds.) ACISP 2016. LNCS, vol. 9723, pp. 379–394. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40367-0_24
26. Machado, A.W.: Differential probability of modular addition with a constant operand. IACR Cryptology ePrint Archive, 2001:52 (2001). http://eprint.iacr.org/2001/052
27. Kovásznai, G., Fröhlich, A., Biere, A.: Complexity of fixed-size bit-vector logics. Theory Comput. Syst. **59**(2), 323–376 (2016)
28. Warren Jr., H.S.: Hacker's Delight. Addison-Wesley, Boston (2003)
29. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991). https://doi.org/10.1007/BF00630563
30. Kelsey, J., Schneier, B., Wagner, D.: Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and triple-DES. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 237–251. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_19
31. Winternitz, R.S., Hellman, M.E.: Chosen-key attacks on a block cipher. Cryptologia **11**(1), 16–20 (1987)
32. Lai, X., Massey, J.L., Murphy, S.: Markov ciphers and differential cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_2
33. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_9
34. Sun, S., et al.: Analysis of AES, skinny, and others with constraint programming. IACR Trans. Symmetric Cryptol. **2017**(1), 281–306 (2017). https://doi.org/10.13154/tosc.v2017.i1.281-306
35. Aumasson, J.-P., Jovanovic, P., Neves, S.: Analysis of NORX: investigating differential and rotational properties. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014. LNCS, vol. 8895, pp. 306–324. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16295-9_17

36. Lipmaa, H.: On differential properties of pseudo-Hdamard transform and related mappings (extended abstract). In: Menezes, A., Sarkar, P. (eds.) INDOCRYPT 2002. LNCS, vol. 2551, pp. 48–61. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36231-2_5
37. Bagherzadeh, E., Ahmadian, Z.: Milp-based automatic differential searches for LEA and HIGHT. IACR Cryptology ePrint Archive, 2018:948 (2018). https://eprint.iacr.org/2018/948
38. Schulte-Geers, E.: On CCZ-equivalence of addition mod 2^n. Des. Codes Cryptogr. **66**(1–3), 111–127 (2013). https://doi.org/10.1007/s10623-012-9668-4
39. Mitchell, J.N.: Computer multiplication and division using binary logarithms. IRE Trans. Electr. Comput. (4), 512–517 (1962)
40. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0 system description. J. Satisfiab. Boolean Model. Comput **9**, 53–58 (2015)
41. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 519–531. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_52
42. Wagner, D.: The boomerang attack. In: Knudsen, L. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48519-8_12
43. Biham, E., Dunkelman, O., Keller, N.: The rectangle attack — rectangling the serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_21
44. Meurer, A., et al.: Sympy: symbolic computing in python. Peer J. Comput. Sci. **3**:e103 (2017). ISSN 2376–5992. https://doi.org/10.7717/peerj-cs.103
45. Gario, M., Micheli, A.: Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In: SMT Workshop 2015 (2015)
46. Hadarean, L., Hyvarinen, A., Niemetz, A., Reger, G.: 14th International Satisfiability Modulo Theories Competition (smt-comp 2019): Rules and Procedures (2019)
47. Jiqiang, L.: Related-key rectangle attack on 36 rounds of the XTEA block cipher. Int. J. Inf. Sec. **8**(1), 1–11 (2009)
48. Lee, E., Hong, D., Chang, D., Hong, S., Lim, J.: A weak key class of XTEA for a related-key rectangle attack. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 286–297. Springer, Heidelberg (2006). https://doi.org/10.1007/11958239_19
49. Lu, J.: Cryptanalysis of reduced versions of the HIGHT block cipher from CHES 2006. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 11–26. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76788-6_2
50. Koo, B., Hong, D., Kwon, D.: Related-key attack on the full HIGHT. In: Rhee, K.-H., Nyang, D.H. (eds.) ICISC 2010. LNCS, vol. 6829, pp. 49–67. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24209-0_4
51. Kelsey, J., Schneier, B., Wagner, D.: Related-key cryptanalysis of 3-WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA. In: Han, Y., Okamoto, T., Qing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 233–246. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0028479
52. ISO/IEC 18033–3:2010. Information technology - Security techniques - Encryption algorithms - Part 3: Block ciphers. Standard, International Organization for Standardization, March 2010