IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 0, No. 0, pp. 0–0. DOI:10.46586/tosc.v0.i0.0-0

SAT-aided Automatic Search of Boomerang Distinguishers for ARX Ciphers (Long Paper)

Dachao Wang[1], Baocang Wang[1]✉ and Siwei Sun[2,3]

[1] State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, China,
mr.ongor@gmail.com, bcwang79@aliyun.com
[2] School of Cryptology, University of Chinese Academy of Sciences, Beijing, China,
sunsiwei@ucas.ac.cn
[3] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, China

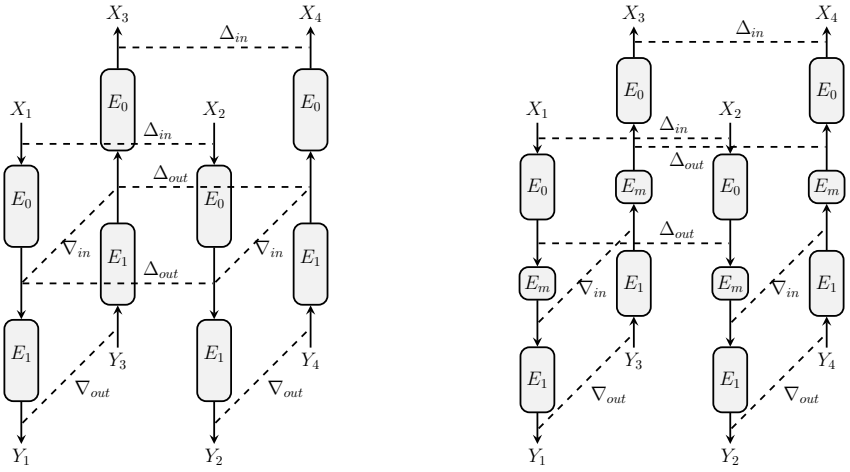
Abstract. In Addition-Rotation-Xor (ARX) ciphers, the large domain size obstructs the application of the boomerang connectivity table. In this paper, we explore the problem of computing this table for a modular addition and the automatic search of boomerang characteristics for ARX ciphers. We provide dynamic programming algorithms to efficiently compute this table and its variants. These algorithms are the most efficient up to now. For the boomerang connectivity table, the execution time is $4^2(n-1)$ simple operations while the previous algorithm costs $8^2(n-1)$ simple operations, which generates a smaller model in the searching phase. After rewriting these algorithms with boolean expressions, we construct the corresponding Boolean Satisfiability Problem models. Two automatic search frameworks are also proposed based on these models. This is the first time bringing the SAT-aided automatic search techniques into finding boomerang attacks on ARX ciphers. Finally, under these frameworks, we find out the first verifiable 10-round boomerang trail for SPECK32/64 with probability $2^{-29.15}$ and a 12-round trail for SPECK48/72 with probability $2^{-44.15}$. These are the best distinguishers for them so far. We also perceive that the previous boomerang attacks on LEA are constructed with an incorrect computation of the boomerang connection probability. The result is then fixed by our frameworks.

Keywords: ARX · Boomerang · Automatic Search · SAT

1 Introduction

Differential cryptanalysis [BS91] is one of the most fundamental cryptanalytic technique in symmetric cryptography. It studies the propagation of difference of the plaintexts, and establishes the relation between the difference of plaintexts and that of ciphertexts. For iterated ciphers based on S-boxes, the propagation through the S-box mainly determines the security against differential cryptanalysis. The propagation for an n-bit S-box is typically depicted by a $2^n \times 2^n$ table T_{ddt}, named the Differential Distribution Table (DDT). For any pair $(\Delta_{in}, \Delta_{out})$, the entry $T_{ddt}(\Delta_{in}, \Delta_{out})$ stores the number of $x \in \mathbb{F}_2^n$ such that $S(x) \oplus S(x \oplus \Delta_{in}) = \Delta_{out}$ holds. The entry $T_{ddt}(\Delta_{in}, \Delta_{out})$ in T_{ddt} means that the input difference Δ_{in} of S propagates to the output difference Δ_{out} with probability $T_{ddt}(\Delta_{in}, \Delta_{out}) \cdot 2^{-n}$. The whole table T_{ddt} is constructed directly by enumerating all possible values of Δ_{in} and x.

In many cases, there is no differential characteristic with a high probability for the entire cipher. The boomerang attack framework [Wag99] could be more suitable to the cipher. This technique aims to concatenate two short differential characteristics to form a distinguisher covering more rounds. In a boomerang attack, the target cipher E is



(a) Original attack. (b) Sandwich attack.

Figure 1: Diagrams of boomerang attacks.

decomposed into E_0 and E_1, i.e., $E = E_1 \circ E_0$, showed in Figure 1a. Suppose that there is a differential characteristic $(\Delta_{in}, \Delta_{out})$ with probability p for E_0, and another one $(\nabla_{in}, \nabla_{out})$ with probability q for E_1. Meanwhile, the characteristics cover r_0 and r_1 rounds respectively. Then, under the assumption that these characteristics are independent, the expected probability of the boomerang distinguisher is

$$\Pr[E^{-1}(E(X_1) \oplus \nabla_{out}) \oplus E^{-1}(E(X_1 \oplus \Delta_{in}) \oplus \nabla_{out}) = \Delta_{in}] = p^2 q^2.$$

The number of rounds covered is $r_0 + r_1$.

However, pointed out by Murphy, this simple concatenation may result in an invalid distinguisher [Mur11]. Several works have studied the compatibility of two characteristics [BDD03, BK09, DKS14, Leu12]. Most of the observations are captured by the sandwich attack framework proposed in [DKS14]. In this framework, the cipher is regarded as a composition of three parts, i.e., $E = E_1 \circ E_m \circ E_0$, where E_m typically contains short transformations, depicted in Figure 1b. Two short differential characteristics are still needed for E_0 and E_1. For E_m, a short boomerang characteristic is applied whose probability is

$$p_m = \Pr\left[E_m^{-1}(E_m(X_1) \oplus \nabla_{in}) \oplus E_m^{-1}(E_m(X_1 \oplus \Delta_{out}) \oplus \nabla_{in}) = \Delta_{out}\right].$$

For ciphers based on S-boxes, the computation of this probability is reduced to that of a single S-box when E_m is a single round. This idea was proposed in [CHP+18], where the switching effect was described by a new table of size $2^n \times 2^n$, called Boomerang Connectivity Table (BCT). It further enables the automatic search of boomerang characteristics. Similar to DDT, BCT can also be constructed by enumerating. Later, Orr Dunkelman provided an improved algorithm to construct the table in $O\left(2^{2n}\right)$ time for an $n \times n$ S-box [Dun18].

When BCT meets the Addition-Rotation-Xor (ARX) ciphers, everything become difficult. ARX ciphers are composed of only three operations: additions modulo 2^n, rotations and XOR operations. The rotation and XOR operation are linear operations, while the modular addition is a nonlinear operation. Following the sandwich attack framework, the simplest case is that E_m consists of a single modular addition. Under this circumstance, the modular addition could be regarded as a $2n \times 2n$ S-box, and again described by the corresponding BCT. In most cases, n is greater than or equal to 16, leading to a large S-box. Even if the value of n is 16 (a 32×32 S-box), the resulting

Table 1: Comparison of our distinguishers with previous ones.

Cipher	Rounds	Type	Prob.	Reference
SPECK32/64	9	Differential	2^{-30}	[SWW21]
	10*	Differential	$2^{-30.39}$	[LKK+18]
	10	**Boomerang**	$\mathbf{2^{-29.15}}$	**Section 5.1**
SPECK48/72	11*	Differential	$2^{-44.31}$	[SHY16]
	11	Differential	2^{-45}	[SWW21]
	12*	Differential	$2^{-46.8}$	[LKK+18]
	12	**Boomerang**	$\mathbf{2^{-44.15}}$	**Section 5.1**

* The key-recovery attacks in [SHY16, LKK+18] cover more rounds, but this paper focuses on the distinguishers on SPECK. Thereby, this table only lists the best distinguishers.

execution time of computing the whole BCT would be 2^{64} simple operations which is infeasible. The BCT of a modular addition was first mentioned in [CHP+18], but no efficient algorithm was given. To the best of our knowledge, only one related work [KKS20] was published. This work proposed the first practical algorithm, following the differential analysis of S-functions [MVDP11]. However, the authors of [KKS20] did not merge the BCT of the modular addition into the Boolean Satisfiability Problem (SAT) models or Mixed Integer Linear Programming (MILP) models. They selected several differential characteristics, trying all combinations of them, and checked the validity by their algorithm. Beyond the idea of the BCT, another tool called ARXtools [Leu12] was proposed based on the theory of S-functions to study ARX constructions. This tool can be used to detect incompatibility or compute the probability of a differential characteristic as well as a boomerang characteristic, but it cannot automate the search of the characteristics.

Our contributions. In this paper, we focus on the computation of BCT, as well as its variants, and the automatic search of boomerang distinguishers on ARX ciphers. First, we notice that the entries of BCT can be computed by a dynamic programming algorithm without using the S-functions. This provides us a new perspective on the computation. We then extend our new algorithm to efficiently compute the variants of BCT, i.e., Upper BCT (UBCT), Lower BCT (LBCT), and Extended BCT (EBCT), defined by Delaune et al. [DDV20]. The time complexities are all linear to the size of the modular addition (bit-length of an addend). Second, we transform our algorithms into new computations so that they can be described by the language of SAT. Nonetheless, the existing modelling methods either target specific ciphers [CHP+17] or require computing the whole BCT and encode it in the models [LS19], which are not workable for the huge BCT of an addition. We borrow the idea from [BV14] to overcome this difficulty. Based on the new computations and their SAT models, we propose two automatic search frameworks to find out boomerang distinguishers with high probabilities. Finally, to verify the power of the proposed technique, we apply it to the ARX-based block ciphers SPECK and LEA. Consequently, we find out several new boomerang distinguishers on SPECK. Table 1 compares our distinguishers with previous ones. For SPECK32/64, ours is the first 10-round distinguisher verified with experiments. The experimental evaluation showed that the probability is about $2^{-27.31}$. We also discuss our results on LEA and those from [KKS20]. Although no improved characteristic was found, a new result is obtained on the previous one. For the same boomerang switch, the estimation of the probability is 0.710802 from our framework, while the experimental evaluation gives about 0.71. It is more precise, compared with 0.661755 in [KKS20]. Furthermore, the comparison of our technique with ARXtools is carried out, showing that ARXtools performs better in computing probabilities while ours is more suitable for searching characteristics.

2 Preliminaries

2.1 Addition and Subtraction Modulo 2^n

In this paper, most of the integer values during the analysis are represented as binary numbers with a fixed bit length which can contain some leading zeros. For such a binary number x, the i-th bit is denoted as $x[i]$ and $x[0]$ is the least significant bit. The one's complement of it is denoted as $\overline{x}$ whose length is the same as x's. For example, the binary number $x = 00101$ has a least significant bit $x[0] = 1$, and $\overline{x}$ is 11010.

We use $\boxplus_n$ to denote an addition modulo 2^n and $\boxminus_n$ to refer to a subtraction modulo 2^n. When an addition or subtraction is applied to two binary strings, it means respectively the addition or subtraction of these two binary numbers. For instance, the addition of 001 and 101 modulo 2^3 is $001 \boxplus_3 101 = 1 \boxplus_3 5 = 6 = 110$. For the convenience of differential analysis, the following property is widely used.

$$x \boxplus_n y = x \oplus y \oplus \mathrm{carry0}_n(x, y)$$

For the subtraction modulo 2^n, a trick is applied to derive a similar property:

$$x \boxminus_n y = x \boxplus_n \bar{y} \boxplus_n 1 = x \oplus \bar{y} \oplus \mathrm{carry1}_n(x, \bar{y}).$$

The functions $\mathrm{carry0}_n$ and $\mathrm{carry1}_n$ are defined in Definition 1 and Definition 2. The only difference lies where i is 0.

Definition 1. The carry $c = \mathrm{carry0}_n(x, y)$ is an n-bit number computed from two n-bit number x and y. It is defined recursively as follows.

$$c[i] = \begin{cases} 0 & i = 0 \\ (x[i-1] \wedge y[i-1]) \oplus (x[i-1] \wedge c[i-1]) \oplus (y[i-1] \wedge c[i-1]) & i > 0 \end{cases}$$

Definition 2. The special carry $b = \mathrm{carry1}_n(x, y)$ is an n-bit number computed from two n-bit number x and y. It is defined recursively as follows.

$$b[i] = \begin{cases} 1 & i = 0 \\ (x[i-1] \wedge y[i-1]) \oplus (x[i-1] \wedge b[i-1]) \oplus (y[i-1] \wedge b[i-1]) & i > 0 \end{cases}$$

We note that the $(i+1)$-th bit of $\mathrm{carry0}_n(x, y)$, resp. $\mathrm{carry1}_n(x, y)$, only depends on three bits—the i-th bits of x, y and $\mathrm{carry0}_n(x, y)$, resp. $\mathrm{carry1}_n(x, y)$. A function is then defined for $\mathrm{carry0}_n$ and $\mathrm{carry1}_n$ which concentrates on the bits.

Definition 3. Define a function $\mathrm{carry} : \mathbb{F}_2 \times \mathbb{F}_2 \times \mathbb{F}_2 \to \mathbb{F}_2$ which is

$$\mathrm{carry}(x, y, c) = (x \wedge y) \oplus (x \wedge c) \oplus (y \wedge c).$$

This function directly comes from Definition 1 and Definition 2. It takes two bits, x and y, as well as a carry bit, and outputs the next carry bit.

2.2 Boomerang Connectivity Table and Its Variants

For an n-bit to n-bit S-box, the diagram of how the difference propagates through the boomerang switch is shown in Figure 2. Let S^{-1} be the inverse of S. The BCT is defined as

$$\mathrm{BCT}(\Delta, \nabla) = \#\left\{ x \in \mathbb{F}_2^n \;\middle|\; S^{-1}(S(x) \oplus \nabla) \oplus S^{-1}(S(x \oplus \Delta) \oplus \nabla) = \Delta \right\}.$$

Given a pair of (Δ, ∇), the probability that a right quartet is generated in S is given by $\mathrm{BCT}(\Delta, \nabla) \cdot 2^{-n}$.

The BCT tool considers the scenario in which the boomerang switch contains only one S-box layer. However, many works [WP19, DDV20, BHL+20] have shown that a boomerang switch with multiple rounds is possible which has more than one S-box layers. Such a switch often results in a better boomerang characteristic. In order to analyse the switching effect, variants of BCT are used in these works. These variants consider more differences besides those required by the BCT in the switch. We use the same definitions from [DDV20] which defines the variants as follows.

Definition 4 ([DDV20])**.** Three variants of BCT are defined respectively as

$$\mathrm{UBCT}(\Delta,\Delta',\nabla) = \#\left\{x\in\mathbb{F}_2^n \;\middle|\; \begin{array}{l} S(x)\oplus S(x\oplus\Delta)=\Delta' \\ S^{-1}(S(x)\oplus\nabla)\oplus S^{-1}(S(x\oplus\Delta)\oplus\nabla)=\Delta \end{array}\right\},$$

$$\mathrm{LBCT}(\Delta,\nabla',\nabla) = \#\left\{x\in\mathbb{F}_2^n \;\middle|\; \begin{array}{l} S(x)\oplus S(x\oplus\nabla')=\nabla \\ S^{-1}(S(x)\oplus\nabla)\oplus S^{-1}(S(x\oplus\Delta)\oplus\nabla)=\Delta \end{array}\right\},$$

$$\mathrm{EBCT}(\Delta,\Delta',\nabla',\nabla) = \#\left\{x\in\mathbb{F}_2^n \;\middle|\; \begin{array}{l} S(x)\oplus S(x\oplus\Delta)=\Delta' \\ S(x)\oplus S(x\oplus\nabla')=\nabla \\ S^{-1}(S(x)\oplus\nabla)\oplus S^{-1}(S(x\oplus\Delta)\oplus\nabla)=\Delta \end{array}\right\}.$$

The BCT table was generalized to the case where S is a modular addition by Cid et al. in [CHP+18]. They considered the modular addition as a $2n$-bit to n-bit mapping, and redefined a BCT table for it. In this paper, we instead regard it as an S-box which maps $2n$-bit input to $2n$-bit output. Illustrated in Figure 3, the input consists of two n-bit values corresponding to two addends, where $\|$ is the concatenation. The left half of the output is the modular addition of the addends, and the right half is still the second addend. Formally, this S-box is defined as $S(L\|R) = (L+R)\|R$. Differences are represented in the same form. For example, the difference between X_1 and X_2 is $\Delta_l\|\Delta_r$, which means that $X_2 = X_1\oplus\Delta = L\oplus\Delta_l\|R\oplus\Delta_r$.

Thus, the definition of BCT for an S-box still works for modular addition. In the definition, we can substitute S with its actual computation, i.e., the modular addition. The result is

$$\begin{aligned} &\mathrm{BCT}_n(\Delta_l,\Delta_r,\nabla_l,\nabla_r) \\ &= \#\Big\{(L,R)\in\mathbb{F}_2^n\times\mathbb{F}_2^n \;\Big|\; \Big(\big((L\boxplus_n R)\oplus\nabla_l\big)\boxminus_n(R\oplus\nabla_r)\Big) \\ &\quad\oplus\Big(\Big(\big((L\oplus\Delta_l)\boxplus_n(R\oplus\Delta_r)\big)\oplus\nabla_l\Big)\boxminus_n(R\oplus\Delta_r\oplus\nabla_r)\Big)=\Delta_l\Big\}. \end{aligned} \tag{1}$$

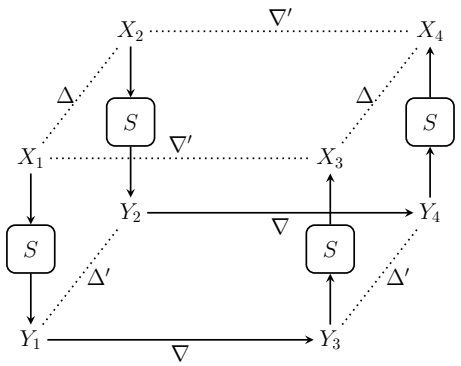


Figure 2: An S-box in the boomerang switch.

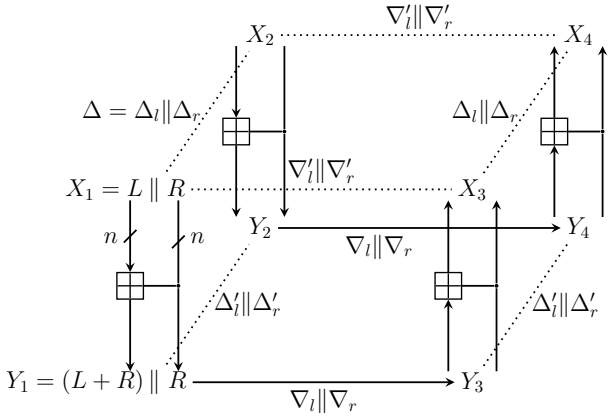


Figure 3: A boomerang switch on a modular addition.

The subscript n denotes that the modulus is 2^n. This equation is the same as the one provided in [CHP+18], but the idea can be easily extended to all variants of BCT.

3 Computing Boomerang Tables for A Modular Addition

In this section, we propose dynamic programming algorithms for computing the entries of BCT and its variants. Dynamic programming is a classical but significant method in algorithms. The word 'programming' refers to a tabular method [CLRS09]. A dynamic programming algorithm starts by partitioning the problem into subproblems. The subproblems can share some subsubproblems. Each subsubproblem is solved just once, and the solution is saved in a table for the next occurrence of the same subsubproblem. The algorithm then combines the solutions in the table to form a solution to each subproblem and finally to the original problem.

In most of the time, to avoid repeating similar contents, we only illustrate our detailed idea on BCT. However, we would like to stress that it is not limited to BCT but applicable to all the variants. The details for the variants are provided in the appendix. Our explanations use the symbols in Figure 3.

3.1 Computation of BCT

3.1.1 Construct the Dynamic Programming Algorithm

The definition of BCT for modular addition is given in Equation (1). To simplify the equation, we use c_1, c_2, b_1 and b_2 to refer to carry0 and carry1 in the equation which are defined as follows.

$$
\begin{aligned}
c_1 &= \mathrm{carry0}_n(L, R) \\
b_1 &= \mathrm{carry1}_n(L \oplus R \oplus c_1 \oplus \nabla_l, \overline{R \oplus \nabla_r}) \\
c_2 &= \mathrm{carry0}_n(L \oplus \Delta_l, R \oplus \Delta_r) \\
b_2 &= \mathrm{carry1}_n(L \oplus \Delta_l \oplus R \oplus \Delta_r \oplus c_2 \oplus \nabla_l, \overline{R \oplus \Delta_r \oplus \nabla_r})
\end{aligned}
$$

Note that, similar to Definition 1 and Definition 2, the values of $c_1[i+1]$, $b_1[i+1]$, $c_2[i+1]$, and $b_2[i+1]$ only depend on $L[i]$, $R[i]$, $\Delta_l[i]$, $\Delta_r[i]$, $\nabla_l[i]$, $\nabla_r[i]$, $c_1[i]$, $b_1[i]$, $c_2[i]$, and $b_2[i]$. By using the properties of the modular addition and subtraction, Equation (1) can be

rewritten to a simpler form:

$$\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r) = \#\Big\{(L, R) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \;\Big|\; c_1 \oplus b_1 \oplus c_2 \oplus b_2 = 0\Big\}$$

or

$$\begin{aligned}&\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r)\\ &= \#\Big\{(L, R) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \;\Big|\; \forall i \in [0, n-1], c_1[i] \oplus b_1[i] \oplus c_2[i] \oplus b_2[i] = 0\Big\}.\end{aligned} \tag{2}$$

That is, given the values of Δ_l, Δ_r, ∇_l, and ∇_r, computing the BCT entry is equivalent to counting the number of (L, R) that satisfies $c_1[i] \oplus b_1[i] \oplus c_2[i] \oplus b_2[i] = 0$ for all $i \in [0, n-1]$. Next, we concentrate on these values of (L, R). Because of the requirement, the tuple $(c_1[i], b_1[i], c_2[i], b_2[i])$ can take only 8 possible values. We use a 4-bit string $c_1[i]\|b_1[i]\|c_2[i]\|b_2[i]$ to represent the value of the tuple. For example, $(c_1[i], b_1[i], c_2[i], b_2[i]) = 1100$ means $c_1[i] = 1$, $b_1[i] = 1$, $c_2[i] = 0$, and $b_2[i] = 0$. Define a set S_{BCT} whose elements are only these 8 values:

$$S_{\mathrm{BCT}} = \{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\}.$$

Note that this is the only restriction on (L, R). It leads to a special property claimed in Lemma 1.

Lemma 1. *The most significant bits (MSB) of L and R, i.e., $L[n-1]$ and $R[n-1]$, can take arbitrary values.*

Lemma 1 is a direct result from Equation (2). For $i = 0$, $c_1[0] \oplus b_1[0] \oplus c_2[0] \oplus b_2[0] = 0$ always holds according to Definition 1 and Definition 2. For $i > 0$, as the aforementioned statement, $c_1[i] \oplus b_1[i] \oplus c_2[i] \oplus b_2[i] = 0$ only restricts the values of $L[i-1]$ and $R[i-1]$. It should be emphasized that $L[n-1]$ and $R[n-1]$ are free, since there is no equation on $c_1[n]$, $b_1[n]$, $c_2[n]$, and $b_2[n]$, which gives the lemma.

Remark 1. This property was first used by the boomerang attacks in [BDD03]. It was observed again in [CHP+18] where it was facilitated in the MSB switch. However, the MSB switch only considered the MSB of L. Our lemma claims that the MSB of R can also be taken into account.

Then, the values of (L, R) are classified by $c_1[n-1]\|b_1[n-1]\|c_2[n-1]\|b_2[n-1]$. Formally and generally, for any bit length i, define a function $f(i, v) = \#S_i(v)$ where

$$S_i(v) = \left\{(L^i, R^i) \in \mathbb{F}_2^i \times \mathbb{F}_2^i \;\middle|\; \begin{array}{l} c_1[i]\|b_1[i]\|c_2[i]\|b_2[i] = v \\ \forall j \in [0, i-1], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \end{array}\right\}.$$

Here, the i-bit carry0 and carry1 are extended by 1 bit following the definitions of them respectively. The superscript i of (L^i, R^i) is just to denote that L^i and R^i are i-bit strings. Thus, we derive a new computation of Equation (2):

$$\begin{aligned}&\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r)\\ &= 4 \times \sum_{v \in S_{\mathrm{BCT}}} \#\left\{\begin{array}{l}(L^{n-1}, R^{n-1})\\ \in \mathbb{F}_2^{n-1} \times \mathbb{F}_2^{n-1}\end{array} \;\middle|\; \begin{array}{l} c_1[n-1]\|b_1[n-1]\|c_2[n-1]\|b_2[n-1] = v \\ \forall j \in [0, n-2], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \end{array}\right\}\\ &= 4 \times \sum_{v \in S_{\mathrm{BCT}}} \#S_{n-1}(v)\\ &= 4 \times \sum_{v \in S_{\mathrm{BCT}}} f(n-1, v).\end{aligned} \tag{3}$$

The constant 4 comes from Lemma 1. Because, for every (L^{n-1}, R^{n-1}), there are 4 difference choices of $(L[n-1], R[n-1])$ to form the values of (L^n, R^n) and every one of them is valid.

In the remaining part of this section, we present a dynamic programming algorithm to compute Equation (3). Based on the idea of dynamic programming [CLRS09], it is necessary to find out a recursive expression for function f. As a base case, it is already known that

$$f(0, v) = \begin{cases} 1 & v = 0101 \\ 0 & \text{otherwise} \end{cases}$$

since $c_1[0] = 0$, $b_1[0] = 1$, $c_2[0] = 0$ and $b_2[0] = 1$, and there is only one $(L^0, R^0) \in \mathbb{F}_2^0 \times \mathbb{F}_2^0$, i.e., L^0 and R^0 are bit strings of length 0.

For all $(L^{i+1}, R^{i+1}) \in S_{i+1}(v)$ with $i \geq 0$, we again partition them into several subsets according to $c_1[i]\|b_1[i]\|c_2[i]\|b_2[i]$. The value $c_1[i]\|b_1[i]\|c_2[i]\|b_2[i]$ can take only 8 values in S_{BCT}, since every $(L^{i+1}, R^{i+1}) \in S_{i+1}(v)$ must satisfy $c_1[i] \oplus b_1[i] \oplus c_2[i] \oplus b_2[i] = 0$. The number of the subsets is 8. Moreover, these subsets are mutually exclusive. This is because the bits $c_1[i]$, $b_1[i]$, $c_2[i]$, and $b_2[i]$ are uniquely determined by each (L^{i+1}, R^{i+1}).

We define a function g to denote the relation among consecutive bits of carries and the corresponding bits of L, R, Δ_l, Δ_r, ∇_l, and ∇_r. This function also shows the dependences among subproblems in dynamic programming.

Definition 5. Define a function $g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r) \in \mathbb{F}_2$, where $u, v \in \mathbb{F}_2^4$, $x, y \in \mathbb{F}_2$, and $\delta_l, \delta_r, \gamma_l, \gamma_r \in \mathbb{F}_2$. Given the forms $u = u_0\|u_1\|u_2\|u_3$ and $v = v_0\|v_1\|v_2\|v_3$, then $g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r) = 1$ if and only if:

$$\begin{aligned} v_0 &= \text{carry}(x, y, u_0); \\ v_1 &= \text{carry}(x \oplus y \oplus v_0 \oplus \gamma_l, \overline{y \oplus \gamma_r}, u_1); \\ v_2 &= \text{carry}(x \oplus \delta_l, y \oplus \delta_r, u_2); \\ v_3 &= \text{carry}(x \oplus \delta_l \oplus y \oplus \delta_r \oplus v_2 \oplus \gamma_l, \overline{y \oplus \delta_r \oplus \gamma_r}, u_3). \end{aligned}$$

Otherwise, $g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r) = 0$.

Then, the recursive expression of function f comes out:

$$\begin{aligned} & f(i+1, v) \\ = & \#S_{i+1}(v) \\ = & \sum_{u \in S_{\text{BCT}}} \# \left\{ \begin{array}{l|l} (L^{i+1}, R^{i+1}) & c_1[i+1]\|b_1[i+1]\|c_2[i+1]\|b_2[i+1] = v \\ \in \mathbb{F}_2^{i+1} \times \mathbb{F}_2^{i+1} & c_1[i]\|b_1[i]\|c_2[i]\|b_2[i] = u \\ & \forall j \in [0, i], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \end{array} \right\} \\ = & \sum_{u \in S_{\text{BCT}}} \sum_{L[i], R[i] \in \mathbb{F}_2} g(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \\ & \qquad \times \# \left\{ \begin{array}{l|l} (L^i, R^i) & c_1[i]\|b_1[i]\|c_2[i]\|b_2[i] = u, \\ \in \mathbb{F}_2^i \times \mathbb{F}_2^i & \forall j \in [0, i-1], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \end{array} \right\} \\ = & \sum_{u \in S_{\text{BCT}}} \sum_{L[i], R[i] \in \mathbb{F}_2} g(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f(i, u) \\ = & \sum_{u \in S_{\text{BCT}}} f(i, u) \sum_{L[i], R[i] \in \mathbb{F}_2} g(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]). \end{aligned}$$

It is feasible to enumerate all possible inputs of the function g and compute its outputs, which can be stored in a table of size $2^{14} \times 1$. The cost to construct such a table is cheap.

To keep simplicity, we define a new function

$$T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) = \sum_{L[i],R[i]\in\mathbb{F}_2} g(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]).$$

It is also computed in advance.

The final expression of f becomes

$$f(i+1, v) = \sum_{u\in S_{\mathrm{BCT}}} T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f(i, u). \tag{4}$$

In the language of dynamic programming, the problem $f(i+1, v)$ is partitioned into 8 subproblems: $f(i, u)$ for all $u \in S_{\mathrm{BCT}}$. There are two ways to construct a dynamic programming algorithm. The direct way is computing $f(i+1, v)$ recursively while storing each value of f in a table for later use. We adopt the other way, starting from the base case and constructing the solution to a 'bigger' problem from its subproblems until reaching $f(i+1, v)$, listed in Algorithm 1.

Algorithm 1 A dynamic programming algorithm to compute f.

```
1:  procedure DP(n, v, Δ_l, Δ_r, ∇_l, ∇_r)        ▷ The value of f(n, v) where v ∈ S_BCT
2:      Initialize a hash table T_dp such that, for all u ∈ S_BCT, T_dp[u] = 0 except that
        T_dp[0101] = 1;
3:      for all i ∈ {0, 1, 2, ..., n − 1} do
4:          Initialize a hash table T'_dp such that T'_dp[u] = 0 for all u ∈ S_BCT;
5:          for all u ∈ S_BCT do
6:              for all u' ∈ S_BCT do
7:                  Look up the value T(u', u, Δ_l[i], Δ_r[i], ∇_l[i], ∇_r[i]). Assume that it is t;
8:                  T'_dp[u] ← T'_dp[u] + t × T_dp[u'];
9:          T_dp ← T'_dp;
10:     return T_dp[v];
```

Theorem 1. *In Algorithm 1, assume Lines 7 to 8 are a simple operation. Then the time complexity is $O(n)$. In particular, the algorithm requires $8^2 n$ simple operations.*

This algorithm contains three loops which loop n, 8, and 8 times respectively. The table look-up in line 7 costs just constant time in modern computers. Thus, the time complexity is shown as Theorem 1. In fact, this algorithm computes all $f(n-1, v)$ for $v \in S_{\mathrm{BCT}}$. While the fourfold sum of them is $\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r)$, we can simply modify line 10 of the algorithm to obtain the algorithm for computing entries of BCT. The execution time of the resulting algorithm is $8^2(n-1)$ simple operations.

3.1.2 Optimize the Algorithm

So far, $\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r)$ is computed bit by bit. Given a bit-length n, the top level `for` loop in Algorithm 1 is fixed. We focus on the inner loops and improve them with another property of the function g.

In the beginning, we provide Lemma 2 revealing an important property of the carry. The correctness is verified directly from Definition 3. This lemma leads to the result that $f(i, v)$ and $f(i, \overline{v})$ can be 'merged', formally stated in Theorem 2.

Lemma 2. *For $x, y, c \in \mathbb{F}_2$, the carry has the following equation.*

$$\mathrm{carry}(x, y, c) = \overline{\mathrm{carry}(\overline{x}, \overline{y}, \overline{c})}$$

Theorem 2. *Let* $S'_{\mathrm{BCT}} = \{0000, 0011, 0101, 0110\}$. *For* $u, v \in S'_{\mathrm{BCT}}$, *denote*

$$f'(i, v) = f(i, v) + f(i, \overline{v})$$

and

$$\begin{aligned} &T'(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \\ &= T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) + T(\overline{u}, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]). \end{aligned}$$

Then, Equation (4) implies

$$f'(i+1, v) = \sum_{u \in S'_{\mathrm{BCT}}} T'(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f'(i, u). \tag{5}$$

Proof. A direct result from Definition 5 and Lemma 2 is

$$g(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) = g(\overline{u}, \overline{v}, \overline{L[i]}, \overline{R[i]}, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]).$$

Then, for function T,

$$\begin{aligned} &T(\overline{u}, \overline{v}, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \\ &= \sum_{L[i], R[i] \in \mathbb{F}_2} g(\overline{u}, \overline{v}, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \\ &= \sum_{L[i], R[i] \in \mathbb{F}_2} g(\overline{u}, \overline{v}, \overline{L[i]}, \overline{R[i]}, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \\ &= \sum_{L[i], R[i] \in \mathbb{F}_2} g(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \\ &= T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]). \end{aligned}$$

Note that S'_{BCT} is a special subset selected from S_{BCT}. On one hand, for every $x \in S'_{\mathrm{BCT}}$, x and $\overline{x}$ are both in S_{BCT}. On the other hand, for every $x \in S_{\mathrm{BCT}}$, either x or $\overline{x}$ is in S'_{BCT}. Therefore, S'_{BCT} is a half of S_{BCT} and its elements are not the one's complement of each other. This gives us the following derivation.

$$\begin{aligned} f(i+1, v) + f(i+1, \overline{v}) &= \sum_{u \in S_{\mathrm{BCT}}} T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f(i, u) \\ &\quad + \sum_{u \in S_{\mathrm{BCT}}} T(u, \overline{v}, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f(i, u) \\ &= \sum_{u \in S_{\mathrm{BCT}}} T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f(i, u) \\ &\quad + \sum_{u \in S_{\mathrm{BCT}}} T(\overline{u}, \overline{v}, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f(i, \overline{u}) \\ &= \sum_{u \in S_{\mathrm{BCT}}} T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f(i, u) \\ &\quad + \sum_{u \in S_{\mathrm{BCT}}} T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f(i, \overline{u}) \\ &= \sum_{u \in S_{\mathrm{BCT}}} T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \big(f(i, u) + f(i, \overline{u})\big) \\ &= \sum_{u \in S'_{\mathrm{BCT}}} \big(T(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \\ &\quad + T(\overline{u}, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i])\big) \big(f(i, u) + f(i, \overline{u})\big). \end{aligned}$$

That is,

$$f'(i+1,v) = \sum_{u \in S'_{\mathrm{BCT}}} T'(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f'(i,u).$$

□

The forms of Equation (4) and (5) are identical. The Algorithm 1 still works for Equation (5) by replacing the set and the functions with the new ones. In addition, Equation (3) turns into

$$\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r) = 4 \times \sum_{v \in S'_{\mathrm{BCT}}} f'(n-1, v). \tag{6}$$

Since the set S'_{BCT} is smaller, the execution time of computing one entry of BCT is reduced to $4^2(n-1)$ simple operations.

3.2 Computation of Other Tables

At first, we provide the definitions of UBCT, LBCT, and EBCT for a modular addition, following the counterparts for an S-box.

$$\begin{aligned}
&\mathrm{UBCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r, \Delta'_l) \\
&= \#\Big\{(L,R) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \;\Big|\; \Big(\big((L \boxplus_n R) \oplus \nabla_l\big) \boxminus_n (R \oplus \nabla_r)\Big) \\
&\qquad \oplus \Big(\Big(\big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) \oplus \nabla_l\Big) \boxminus_n (R \oplus \Delta_r \oplus \nabla_r)\Big) = \Delta_l, \\
&\qquad (L \boxplus_n R) \oplus \big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) = \Delta'_l\Big\}.
\end{aligned} \tag{7}$$

$$\begin{aligned}
&\mathrm{LBCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r, \nabla'_l) \\
&= \#\Big\{(L,R) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \;\Big|\; \Big(\big((L \boxplus_n R) \oplus \nabla_l\big) \boxminus_n (R \oplus \nabla_r)\Big) \\
&\qquad \oplus \Big(\Big(\big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) \oplus \nabla_l\Big) \boxminus_n (R \oplus \Delta_r \oplus \nabla_r)\Big) = \Delta_l, \\
&\qquad \big((L \oplus \nabla'_l) \boxplus_n (R \oplus \nabla_r)\big) \oplus (L \boxplus_n R) = \nabla_l\Big\}.
\end{aligned} \tag{8}$$

$$\begin{aligned}
&\mathrm{EBCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r, \Delta'_l, \nabla'_l) \\
&= \#\Big\{(L,R) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \;\Big|\; \Big(\big((L \boxplus_n R) \oplus \nabla_l\big) \boxminus_n (R \oplus \nabla_r)\Big) \\
&\qquad \oplus \Big(\Big(\big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) \oplus \nabla_l\Big) \boxminus_n (R \oplus \Delta_r \oplus \nabla_r)\Big) = \Delta_l, \\
&\qquad (L \boxplus_n R) \oplus \big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) = \Delta'_l, \\
&\qquad \big((L \oplus \nabla'_l) \boxplus_n (R \oplus \nabla_r)\big) \oplus (L \boxplus_n R) = \nabla_l\Big\}.
\end{aligned}$$

It is clear to see that these tables are based on BCT and have one or two more restrictions on (L, R). Actually, with a little modification on the dynamic programming algorithm for BCT, we hence have similar algorithms for computing the entries of these tables. Due to the limited space, we only take LBCT as an example, showing how to modify the previous algorithm. The algorithms for UBCT and EBCT can be constructed in the same way.

After comparing Equation (8) and (1), the new restriction is $\big((L \oplus \nabla_l') \boxplus_n (R \oplus \nabla_r)\big) \oplus (L \boxplus_n R) = \nabla_l$. Define the carry of $(L \oplus \nabla_l') \boxplus_n (R \oplus \nabla_r)$ as

$$c_3 = \mathrm{carry0}_n(L \oplus \nabla_l', R \oplus \nabla_r).$$

The $(i+1)$-th bit of c_3 still only depends on the previous bits of L, R, ∇_l', and ∇_r. This implies the existence of a recursive expression for LBCT like the one for BCT. Then, the restriction is rewritten to $c_3 \oplus c_1 = \nabla_l \oplus \nabla_l' \oplus \nabla_r$. Together with $c_1 \oplus b_1 \oplus c_2 \oplus b_2 = 0$, there are only two restrictions on the value of (L, R). For all $i \in [0, n-1]$, the tuple $(c_1[i], b_1[i], c_2[i], b_2[i], c_3[i])$ can take only a few values which form a set:

$$\begin{aligned} S_{\mathrm{LBCT}} = \{&00000, 00110, 01010, 01100, 10010, 10100, 11000, 11110,\\ &00001, 00111, 01011, 01101, 10011, 10101, 11001, 11111\}. \end{aligned}$$

This set is constructed by extending every element in S_{BCT} by one bit represented for c_3 and taking any value of this bit.

During the same derivation of a recursive expression like Equation (4), we define a function g_{LBCT}, similar to function g but specific to LBCT.

Definition 6. Define $g_{\mathrm{LBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \gamma_l') \in \mathbb{F}_2$, where $u, v \in \mathbb{F}_2^5$, $x, y \in \mathbb{F}_2$, and $\delta_l, \delta_r, \gamma_l, \gamma_r, \gamma_l' \in \mathbb{F}_2$. Given the forms $u = u_0 \| u_1 \| u_2 \| u_3 \| u_4$ and $v = v_0 \| v_1 \| v_2 \| v_3 \| v_4$, $g_{\mathrm{LBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \gamma_l') = 1$ if and only if:

$$\begin{aligned} &v_0 = \mathrm{carry}(x, y, u_0);\\ &v_1 = \mathrm{carry}(x \oplus y \oplus v_0 \oplus \gamma_l, \overline{y \oplus \gamma_r}, u_1);\\ &v_2 = \mathrm{carry}(x \oplus \delta_l, y \oplus \delta_r, u_2);\\ &v_3 = \mathrm{carry}(x \oplus \delta_l \oplus y \oplus \delta_r \oplus v_2 \oplus \gamma_l, \overline{y \oplus \delta_r \oplus \gamma_r}, u_3);\\ &u_4 \oplus u_0 = \gamma_l \oplus \gamma_l' \oplus \gamma_r;\\ &v_4 = \mathrm{carry}(x \oplus \gamma_l', y \oplus \gamma_r, u_4). \end{aligned}$$

Otherwise, $g_{\mathrm{LBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \gamma_l') = 0$.

As the counterpart of the function T in the previous subsection, a new function $T_{\mathrm{LBCT}}(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i])$ is also defined. Note that we are still able to go through all possible inputs and compute T_{LBCT} in advance. Therefore, following the same idea in Section 3.1.1, the dynamic programming algorithm for LBCT comes out, listed in Algorithm 2.

Since the rest of the idea and the optimization are almost the same as those in Section 3.1, the detailed procedure is not explained again here. We only point out that the size of S_{LBCT} decreases to 8. The execution time of the optimized algorithm is $8^2(n-1)$ simple operations. Moreover, for UBCT and EBCT, the corresponding sets are of size 4 and 8, respectively. The numbers of simple operations that the optimized algorithms spend are $4^2(n-1)$ for UBCT and $8^2(n-1)$ for EBCT. For more details, please see Appendix A.

3.3 Observations on the Computations

This subsection presents three observations on the computations of BCT and its variants. The observations illustrate the relation between previous results and ours as well as basic properties for constructing models in the next chapter. We only explain the observations on BCT. For LBCT, UBCT, and EBCT, they have identical results as well.

The first observation is that Equation (4) and (5) can be written as matrix multiplications. Take Equation (5) as an example here. When all the equations for $f'(i+1, v)$

Algorithm 2 A dynamic programming algorithm to compute entries of LBCT.

```
1:  procedure DP_LBCT(n, Δ_l, Δ_r, ∇_l, ∇_r, ∇'_l)
2:      Initialize a hash table T_dp such that, for all u ∈ S_LBCT, T_dp[u] = 0 except that
        T_dp[01010] = 1;
3:      for all i ∈ {0, 1, 2, ..., n − 2} do
4:          Initialize a hash table T'_dp such that T'_dp[u] = 0 for all u ∈ S_LBCT;
5:          for all u ∈ S_LBCT do
6:              for all u' ∈ S_LBCT do
7:                  Look up the value T_LBCT(u', u, Δ_l[i], Δ_r[i], ∇_l[i], ∇_r[i], ∇'_l[i]). Assume
                    that it is t;
8:                  T'_dp[u] ← T'_dp[u] + t × T_dp[u'];
9:          T_dp ← T'_dp;
10:     sum ← 0;
11:     for all u ∈ S_LBCT do
12:         Let u be u_0‖u_1‖u_2‖u_3‖u_4;
13:         if u_4 ⊕ u_0 = ∇_l[n − 1] ⊕ ∇'_l[n − 1] ⊕ ∇_r[n − 1] then
14:             sum ← sum + T_dp[u];
15:     return 4 × sum;
```

for $v \in S'_{\mathrm{BCT}}$ combine, they can be replaced with a single matrix multiplication as the following equation.

$$\begin{bmatrix} f'(i{+}1{,}0000) \\ f'(i{+}1{,}0011) \\ f'(i{+}1{,}0101) \\ f'(i{+}1{,}0110) \end{bmatrix} = \begin{bmatrix} T'(0000{,}0000) & T'(0011{,}0000) & T'(0101{,}0000) & T'(0110{,}0000) \\ T'(0000{,}0011) & T'(0011{,}0011) & T'(0101{,}0011) & T'(0110{,}0011) \\ T'(0000{,}0101) & T'(0011{,}0101) & T'(0101{,}0101) & T'(0110{,}0101) \\ T'(0000{,}0110) & T'(0011{,}0110) & T'(0101{,}0110) & T'(0110{,}0110) \end{bmatrix}_{w[i]} \begin{bmatrix} f'(i{,}0000) \\ f'(i{,}0011) \\ f'(i{,}0101) \\ f'(i{,}0110) \end{bmatrix} \tag{9}$$

$\Delta_l[i]$, $\Delta_r[i]$, $\nabla_l[i]$, and $\nabla_r[i]$ are ignored here because of the limited space. We should emphasize that each value of function T' depends on these four bits. The 4×4 matrix is determined by a 4-bit string $w[i] = \Delta_l[i]\|\Delta_r[i]\|\nabla_l[i]\|\nabla_r[i]$, which is denoted by $A_{w[i]}$. The total number of different matrices is 16. By rearranging the precomputed table for function T', a table storing these matrices is obtained. Recall that, for $i = 0$, we have

$$\begin{bmatrix} f'(0{,}0000) \\ f'(0{,}0011) \\ f'(0{,}0101) \\ f'(0{,}0110) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

Therefore, the computation of $\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r)$ can use a chain of matrix multiplications, showed as Equation (10).

$$\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r) = 4 \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}^T \left(\prod_{i=0}^{n-2} A_{w[i]} \right) \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{10}$$

In Algorithm 1, Lines 2 to 9 can then use a multiplication of a matrix and a vector instead, while the time complexity remains unchanged. This formula is almost identical to Kim et al.'s result [KKS20], except that their matrices are of size 8×8, larger than ours. Consequently, the execution time of their algorithm is $8^2(n-1)$ simple operations. The smaller matrices not only result in a lower time complexity, but also reduce the size of the SAT models, which is detailed in Section 4.1.

The second observation is stated in Theorem 3. Before proving this theorem, we provide 2 lemmas on the properties of g as well as prove them.

Theorem 3. *Given the boolean values δ_l, δ_r, γ_l, and γ_r, for any $u \in S'_{\mathrm{BCT}}$, the sum $\sum_{v \in S'_{\mathrm{BCT}}} T'(u, v, \delta_l, \delta_r, \gamma_l, \gamma_r)$ is either 0 or 4.*

Lemma 3. *Given a $u \in S_{\mathrm{BCT}}$ and the values of δ_l, δ_r, γ_l, and γ_r, if there exists values $w \in S_{\mathrm{BCT}}$ and $a, b \in \mathbb{F}_2$ such that $g(u, w, a, b, \delta_l, \delta_r, \gamma_l, \gamma_r) = 1$, then, for any boolean values x and y, there exists a $v \in S_{\mathrm{BCT}}$ such that $g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r) = 1$.*

Proof. Let $u = u_0 \| u_1 \| u_2 \| u_3$ and $v = v_0 \| v_1 \| v_2 \| v_3$. Because of the premise that $u, v \in S_{\mathrm{BCT}}$, it means that $u_0 \oplus u_1 \oplus u_2 \oplus u_3 = 0$ and $v_0 \oplus v_1 \oplus v_2 \oplus v_3 = 0$. According to Definition 5, the equation for v can be rewritten as

$$\begin{aligned} &\mathrm{carry}(x, y, u_0) \oplus \mathrm{carry}(x \oplus y \oplus v_0 \oplus \gamma_l, \overline{y \oplus \gamma_r}, u_1) \\ &\oplus \mathrm{carry}(x \oplus \delta_l, y \oplus \delta_r, u_2) \oplus \mathrm{carry}(x \oplus \delta_l \oplus y \oplus \delta_r \oplus v_2 \oplus \gamma_l, \overline{y \oplus \delta_r \oplus \gamma_r}, u_3) \\ &= 0. \end{aligned} \tag{11}$$

It can be further expanded with Definition 3. The resulting equation is very long, so we are not able to put it here. It is only composed of boolean variables and logical operations, which can be analysed by tools of boolean functions.

With the help of the `BooleanFunction` module in the SageMath computer algebra system [Sag20], we simplify the left side of Equation (11) to $x(u_0 + u_1 + u_2 + u_3)$ plus a boolean polynomial without variables x and y. Since $u_0 + u_1 + u_2 + u_3$ is 0, all the x and y are cancelled in the equation. It means that x and y have no impact on the satisfiability of Equation (11). Thus, if there exists $x = a$ and $y = b$ such that Equation (11) satisfies, then, for any boolean values x and y, it remains satisfied. Note that the satisfiability of Equation (11) implies a value of $v \in S_{\mathrm{BCT}}$. Then the lemma is proved. □

Lemma 4. *Given a $u \in S_{\mathrm{BCT}}$ and the values of x, y, δ_l, δ_r, γ_l, and γ_r, for any two values $v, v' \in S_{\mathrm{BCT}}$ such that $v \neq v'$, $g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r)$ and $g(u, v', x, y, \delta_l, \delta_r, \gamma_l, \gamma_r)$ cannot be 1 simultaneously.*

This lemma is trivial. Let u be $u_0 \| u_1 \| u_2 \| u_3$, v be $v_0 \| v_1 \| v_2 \| v_3$ and v' be $v'_0 \| v'_1 \| v'_2 \| v'_3$. Since $v \neq v'$, there must be a different bit. Without loss of generality, assume that $v_0 \neq v'_0$. In contrast, according to Definition 5, we have $v_0 = \mathrm{carry}(x, y, u_0)$ and $v'_0 = \mathrm{carry}(x, y, u_0)$. However, this is impossible.

Based on Lemma 3 and Lemma 4, Theorem 3 is proved.

Proof. (Theorem 3) The proof starts by expanding the sum to an expression about g.

$$\begin{aligned} &\sum_{v \in S'_{\mathrm{BCT}}} T'(u, v, \delta_l, \delta_r, \gamma_l, \gamma_r) \\ =& \sum_{v \in S'_{\mathrm{BCT}}} \big(T(u, v, \delta_l, \delta_r, \gamma_l, \gamma_r) + T(\overline{u}, v, \delta_l, \delta_r, \gamma_l, \gamma_r)\big) \\ =& \sum_{v \in S'_{\mathrm{BCT}}} \big(T(u, v, \delta_l, \delta_r, \gamma_l, \gamma_r) + T(u, \overline{v}, \delta_l, \delta_r, \gamma_l, \gamma_r)\big) \\ =& \sum_{v \in S_{\mathrm{BCT}}} T(u, v, \delta_l, \delta_r, \gamma_l, \gamma_r) \\ =& \sum_{v \in S_{\mathrm{BCT}}} \sum_{x, y \in \mathbb{F}_2} g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r) \\ =& \sum_{x, y \in \mathbb{F}_2} \sum_{v \in S_{\mathrm{BCT}}} g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r) \end{aligned}$$

Due to Lemma 4,

$$\sum_{v \in S_{\mathrm{BCT}}} g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r) \in \mathbb{F}_2. \tag{12}$$

If there exists $x = a$ and $y = b$ such that this sum equals 1, then it means that there must be one and only one $v \in S_{\mathrm{BCT}}$ such that $g(u, v, a, b, \delta_l, \delta_r, \gamma_l, \gamma_r) = 1$. Then, with Lemma 3, we would know that, for any values of x and y,

$$\sum_{v \in S_{\mathrm{BCT}}} g(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r) = 1.$$

Therefore, the sum $\sum_{v \in S'_{\mathrm{BCT}}} T'(u, v, \delta_l, \delta_r, \gamma_l, \gamma_r)$ would be 4. Otherwise, for any values of x and y, if the sum in Equation (12) is always 0, then $\sum_{v \in S'_{\mathrm{BCT}}} T'(u, v, \delta_l, \delta_r, \gamma_l, \gamma_r)$ would be 0. □

The last observation is that, given a $u \in S_{\mathrm{BCT}}$ and the values of δ_l, δ_r, γ_l, and γ_r, if $\sum_{v \in S'_{\mathrm{BCT}}} T'(u, v, \delta_l, \delta_r, \gamma_l, \gamma_r)$ is 4, then $T'(u, u, \delta_l, \delta_r, \gamma_l, \gamma_r)$ must be larger than 0. This is directly observed from the table of T'.

Furthermore, when considering the sum $s_i = \sum_{v \in S'_{\mathrm{BCT}}} f'(i, v)$,

$$\begin{aligned}
s_i &= \sum_{v \in S'_{\mathrm{BCT}}} f'(i, v) \\
&= \sum_{v \in S'_{\mathrm{BCT}}} \sum_{u \in S'_{\mathrm{BCT}}} T'(u, v, \Delta_l[i-1], \Delta_r[i-1], \nabla_l[i-1], \nabla_r[i-1]) f'(i-1, u) \\
&= \sum_{u \in S'_{\mathrm{BCT}}} \sum_{v \in S'_{\mathrm{BCT}}} T'(u, v, \Delta_l[i-1], \Delta_r[i-1], \nabla_l[i-1], \nabla_r[i-1]) f'(i-1, u) \\
&= \sum_{u \in S'_{\mathrm{BCT}}} f'(i-1, u) \sum_{v \in S'_{\mathrm{BCT}}} T'(u, v, \Delta_l[i-1], \Delta_r[i-1], \nabla_l[i-1], \nabla_r[i-1]).
\end{aligned}$$

If, for all $u \in S'_{\mathrm{BCT}}$, we have $T'(u, u, \Delta_l[i-1], \Delta_r[i-1], \nabla_l[i-1], \nabla_r[i-1]) \neq 0$, then

$$\begin{aligned}
s_i &= \sum_{u \in S'_{\mathrm{BCT}}} f'(i-1, u) \times 4 \\
&= 4 \times \sum_{u \in S'_{\mathrm{BCT}}} f'(i-1, u) \\
&= 4 s_{i-1}.
\end{aligned}$$

Otherwise, s_i is less than $4s_{i-1}$. Whether this happens or not depends on the values of $\Delta_l[i-1]$, $\Delta_r[i-1]$, $\nabla_l[i-1]$, and $\nabla_r[i-1]$. For those entries in BCT with the largest value, i.e., 4^n, they must have $s_i = 4s_{i-1}$ for all $i \in [1, n-1]$.

4 Automatic Search of Boomerang for ARX Ciphers

In this section, we propose the automatic search technique for boomerang distinguishers on ARX ciphers. We start from modelling the switch effect on a single modular addition. The key problem is how to construct SAT/SMT models for BCT and its variants.

A natural approach would be using an SMT model describing Algorithm 1. The power of SMT makes this possible. However, the value of an entry can be very large, exceeding the limitation of most of the SMT solvers. If one tries to take the binary logarithm of the value, he/she would encounter that it seems impossible, since the corresponding formula (Equation (10)) is a chain of matrix multiplications. Moreover, the possible values form an extremely large subset of $\{0, 1, 2, \ldots, 4^n\}$. Thus, it is impractical to list all the possible values and encode them like any other models of ciphers based on S-boxes.

As a result, it requires us to develop new models. In the beginning, a model is proposed to describe all non-zero entries in the tables. With a heuristic strategy, this model is

Algorithm 3 A dynamic programming algorithm to check an entry in BCT.

```
1: procedure IsZero(n, Δ_l, Δ_r, ∇_l, ∇_r)
2:     Initialize a hash table T_dp such that, for all u ∈ S_BCT, T_dp[u] = False except that
       T_dp[0101] = True;
3:     for all i ∈ {0, 1, 2, ..., n − 1} do
4:         Initialize a hash table T'_dp such that T'_dp[u] = False for all u ∈ S_BCT;
5:         for all u ∈ S'_BCT do
6:             for all u' ∈ S'_BCT do
7:                 Look up the value T'_b(u', u, Δ_l[i], Δ_r[i], ∇_l[i], ∇_r[i]). Assume that it is t;
8:                 T'_dp[u] ← T'_dp[u] ∨ (t ∧ T_dp[u']);
9:         T_dp ← T'_dp;
10:    sum ← False;
11:    for all u ∈ S'_BCT do
12:        sum ← sum ∨ T_dp[u];
13:    return sum;        ▷ sum is True if the entry is non-zero and False if it is zero.
```

improved. We should emphasize that our new models give us neither the value of the entry nor the probability. They just encode the information: 'non-zero'. That is, they merely restrict a boomerang switch to a valid switch. We then merge these models into our automatic search frameworks to find out boomerang distinguishers.

4.1 SAT Models for Any Non-zero Entries

We still consider the case for BCT, but the idea can also be applied to LBCT, UBCT, and EBCT. According to Equation (6), $\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r) \neq 0$ implies that there exists a $v \in S'_{\mathrm{BCT}}$ such that $f'(n-1, v) \neq 0$. On account of the recursive expression of function f', it further requires that, for any $i \in [0, n-1]$, there exists a $v \in S'_{\mathrm{BCT}}$ such that $f'(i, v) \neq 0$. Formally, the propagation of this property is computed by

$$f'_b(i+1, v) = \max_{u \in S'_{\mathrm{BCT}}} T'_b(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) f'_b(i, u),$$

where $f'_b(i, u), T'_b(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \in \mathbb{F}_2$ and $f'_b(i, u)$ equals 1 if and only if $f'(i, u)$ is not zero, and $T'_b(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i])$ is equal to 1 if and only if $T'(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i])$ is not zero as well. The algorithm to check whether an entry in BCT is zero or not is listed in Algorithm 3.

Since Algorithm 3 only contains boolean variables and bit operations, it is convenient to construct the corresponding SAT model. Lines 4 to 9 are transformed into four expressions as follows.

$$\forall u \in S'_{\mathrm{BCT}}, T'_{dp}[u] = \bigvee_{u' \in S'_{\mathrm{BCT}}} \left(T'_b(u', u, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i]) \wedge T_{dp}[u'] \right)$$

Note that, the bigger the set S'_{BCT} is, the more expressions are created in the model. Meanwhile, the size of S'_{BCT} matches the size of the square matrix in Equation (9), and thereby the smaller matrices are better. Finally, the model is supposed to include the non-zero entries, so Lines 10 to 13 are modelled by

$$f'_b(n-2, 0000) \vee f'_b(n-2, 0011) \vee f'_b(n-2, 0101) \vee f'_b(n-2, 0110) = True.$$

4.2 A Heuristic Strategy

One of the disadvantages of the model in Section 4.1 is that it contains too many entries with small values. In cryptanalysis, an attack with high probability is expected. We then

Table 2: Initial values of k_{lb}.

bit length (n)	BCT	LBCT, UBCT, EBCT
16	8	6
24	16	12
32	20	16
48	34	30
64	48	44

decide to make the model to describe only the entries with sufficiently large values. This idea comes from the concept of a *partial differential distribution table* in [BV14]. Here, our model in this subsection can be regarded as a SAT model for a *partial boomerang connectivity table*.

As we have already explained in Section 3.3, the largest entries in BCT require that $s_i = 4s_{i-1}$ for all $i \in [1, n-1]$. Conversely, if we restrict k out of $n-1$ these equations to be satisfied, the resulting entry would be 4^{k+1} multiplying some value q. In other words, the value of the entry would have the form of $4^{k+1}q$ where q is unknown. Currently, we do not have enough knowledge about q. However, our experiments show that this restriction always forced the model to return an entry with a value around 4^{k+1}. The research on q is left to future work.

Note that, based on the second and third observations in Section 3.3, $s_i = 4s_{i-1}$ is equivalent to $T'(u, u, \Delta_l[i-1], \Delta_r[i-1], \nabla_l[i-1], \nabla_r[i-1]) \neq 0$ for all $u \in S'_{\text{BCT}}$. Therefore, based on the model in Section 4.1, we add $4(n-1)$ more boolean variables to indicate whether $T'_b(u, u, \Delta_l[i-1], \Delta_r[i-1], \nabla_l[i-1], \nabla_r[i-1])$ is 0 or not for all $u \in S'_{\text{BCT}}$ and $i \in [1, n-1]$. For a bit-position $i \in [0, n-2]$, $s_{i+1} = 4s_i$ is satisfied if and only if the four corresponding boolean variables are set to $True$. The parameter k is configurable. It depends on which cipher one is analysing.

During the automatic search, we do not directly set the value of k but restrict its lower bound. That is, an integer variable is created for k and a new constraint $k \geq k_{lb}$ is added. The larger k_{lb} is, the smaller part of BCT is contained in the model. However, if k_{lb} is too small, the model often returns a small entry in BCT, which is observed in our experiments. While k_{lb} is too large, the model may not contain an appropriate entry. It then implies that there exists an optimal k_{lb}. k_{lb} now replaces k, becoming the tweakable parameter. It is set to be a small value and increases steadily until a desired BCT entry is found. For LBCT, UBCT, and EBCT, the above idea still works. Our suggestion on initial values of the lower bounds are listed in Table 2.

4.3 Our Automatic Search Frameworks

Recall that, in boomerang attacks, a cipher E is decomposed as $E_1 \circ E_m \circ E_0$. The models for E_0 and E_1 are identical to the differential models. In ARX ciphers, E_m is made up of only modular additions, rotations and XOR operations. Each modular addition is regarded as a special S-box. Constructing the model of E_m is similar to constructing that for ciphers based on S-boxes. It is feasible to handle dependences between modular additions with our models of BCT and its variants. We do not elaborate the model of E_m here, since it relies on the specification of the cipher. The details are left to the next chapter. In this subsection, our main concentration is how to utilize the models in Section 4.2, although they lack the exact probabilities. We propose two frameworks.

The first one is to search for a boomerang characteristic with a high probability. For ciphers based on S-boxes, the objective function can be the product of the probabilities

of E_0, E_1, and E_m (or the sum of their binary logarithms). By optimizing it, the model returns a boomerang characteristic with the optimal probability. However, our model for E_m only tells us that the probability is probably high according to Section 4.2. To make use of this information, we propose a two-step framework:

1. Set the product of the probabilities of E_0 and E_1 as the objective function and maximize it;

2. Compute the probability of E_m according to the information provided by the boomerang characteristic and then obtain the total probability.

The parameter k_{lb} is tweaked to find a better boomerang characteristic. For every different k_{lb}, the above two steps are repeated. Specifically, $k_{lb} = n-1$ gives us an E_m with probability 1. If $k_{lb} = 0$, the model returns any valid boomerang characteristic. Note that, when k_{lb} is approaching to $n-1$, the limited choices of E_m could cause E_0 and E_1 away from the optimal ones. This is another reason why we recommend to start from a small k_{lb} in the previous subsection. This framework is suitable for searching long boomerang characteristics or for ciphers with a complicated E_m. In this paper, we show an example on LEA [HLK+14].

The second framework aims to take advantage of the clustering effect. The complete model is the same as the one in the first framework. It has three stages, listed below.

1. The objective function is still the product of the probabilities of E_0 and E_1. Set k_{lb} to 0 and maximize the objective function to obtain a boomerang characteristic. Let the optimized value of the objective function be $p_{\max}$. It is an upper bound on the probabilities of boomerang characteristics in this framework.

2. Set k_{lb} to a proper value and the value of the objective function to be $p_{\max} - t$. Let the SAT solver to enumerate all solutions in this model. With the help of a hash table, it is easy to count the solutions by the input difference of E_0, Δ_{in}, and the output difference of E_1, ∇_{out}. In other words, the index of the hash table is $(\Delta_{in}, \nabla_{out})$, and the value is the number of boomerang characteristics in accordance with these differential values.

3. Take the record with the largest value in the hash table. This record means that, given k_{lb} and t, the largest cluster has input difference Δ_{in} for E_0 and output difference ∇_{out} for E_1. Set k_{lb} to 0 or a smaller value than the one at Stage 2. Set the value of the objective function to be in $[p_{\max} - t_p, p_{\max}]$, where t_p is a value less than or equal to the t at Stage 2. Then, let the SAT solver to enumerate all solutions again under the given Δ_{in} and ∇_{out}. This will give us the probability of the cluster.

The parameters k_{lb} and t are supposed to be tweaked to obtain a better cluster, while the parameter t_p only affects the precision of the probability. At Stage 2, they affect the total number of solutions to be enumerated. More solutions lead to a more accurate result. Nonetheless, more solutions also cause a higher cost of time. Thus, they should be configured with proper values which are suitable to the hardware and the target cipher. At Stage 3, the model is given the exact values, Δ_{in} and ∇_{out}. This offers more information about the model to the SAT solver and allows the solver to enumerate much more solutions. Therefore, we are able to use a smaller k_{lb} and t_p, which leads to a more precise estimation of the probability. The precision of the estimation is the advantage over the first framework.

Because of the enumerations, this framework requires more time. Thus, it is applicable to short boomerang trails or ciphers with a simple E_m. In this paper, we apply it to SPECK [BSS+13].

4.4 Comparing the Technique with S-functions

This subsection compares our technique with those based on S-functions since they are very similar. The concept of S-functions was proposed by Mouha et al. [MVDP11] to describe several operations in ARX constructions, which is the core part of their general framework for analysing the ARX-based ciphers.

The boomerang characteristics of ARX-based ciphers can indeed be analysed by the original technique of S-functions, as Kim et al. have already shown [KKS20]. This technique requires graph theory, representing the S-functions as a graph, and counting the paths by matrix multiplications. Appendix C shows an example of this process. In contrast, the idea in Section 3.1 directly constructs the algorithm from the perspective of dynamic programming, although it starts from the same property used by S-functions—the $(i+1)$-th state only relies on the i-th state. Our idea is a new view of the property and can be easily extended to the variants of BCT. Furthermore, this approach is natural, because the property perfectly matches the key ingredients in dynamic programming, i.e., optimal substructure and overlapping subproblems [CLRS09]. This dynamic programming algorithm also has a graph representation [CLRS09] similar to the S-functions' where the function g (Definition 5) is the same as the transition function in S-functions. But our function g presents how subproblems depend on one another in the algorithm. Thus, these two approaches are from two different perspectives while reaching the same result. Moreover, to reduce the sizes of the matrices, Mouha et al. proposed a new algorithm to find out equivalent states, while ours reveals mathematical properties behind the matrices and then generates smaller matrices. These properties were not published before.

In addition, it should be pointed out that the property stated in Lemma 2 can explain more observations about the modular addition. We take the result of the Finite State Machine (FSM) reduction algorithm from Mouha et al. [MVDP11] as an example. The key equations in the differential analysis are

$$\begin{aligned} c_1 \oplus c_2 &= \Delta_x \oplus \Delta_y \oplus \Delta_z, \\ c_1 &= \mathrm{carry0}(x, y), \\ c_2 &= \mathrm{carry0}(x \oplus \Delta_x, y \oplus \Delta_y), \end{aligned}$$

where all the variables are represented in n-bit strings. Then, it can be derived that

$$\begin{aligned} &\mathrm{carry}(x[i], y[i], c_1[i]) \oplus \mathrm{carry}(x[i] \oplus \Delta_x[i], y[i] \oplus \Delta_y[i], c_2[i]) \\ =&\Delta_x[i+1] \oplus \Delta_y[i+1] \oplus \Delta_z[i+1] \end{aligned} \tag{13}$$

for all $i \in [0, n-1)$. Following S-functions technique, the state is defined as $S = (c_1[i], c_2[i])$ and the FSM reduction algorithm outputs two equivalence classes, i.e., $\{(0,0),(1,1)\}$ and $\{(0,1),(1,0)\}$. This fact is explainable with Lemma 2. For any values $x[i]$ and $y[i]$ satisfying Equation 13, Lemma 2 implies that

$$\begin{aligned} &\mathrm{carry}(\overline{x[i]}, \overline{y[i]}, \overline{c_1[i]}) \oplus \mathrm{carry}(\overline{x[i]} \oplus \Delta_x[i], \overline{y[i]} \oplus \Delta_y[i], \overline{c_2[i]}) \\ =&\mathrm{carry}(\overline{x[i]}, \overline{y[i]}, \overline{c_1[i]}) \oplus \mathrm{carry}(\overline{x[i] \oplus \Delta_x[i]}, \overline{y[i] \oplus \Delta_y[i]}, \overline{c_2[i]}) \\ =&\overline{\mathrm{carry}(\overline{x[i]}, \overline{y[i]}, \overline{c_1[i]}) \oplus \mathrm{carry}(\overline{x[i] \oplus \Delta_x[i]}, \overline{y[i] \oplus \Delta_y[i]}, \overline{c_2[i]})} \\ =&\mathrm{carry}(x[i], y[i], c_1[i]) \oplus \mathrm{carry}(x[i] \oplus \Delta_x[i], y[i] \oplus \Delta_y[i], c_2[i]) \\ =&\Delta_x[i+1] \oplus \Delta_y[i+1] \oplus \Delta_z[i+1]. \end{aligned}$$

Thus, the number of pairs of $x[i]$ and $y[i]$ connecting states $(c_1[i], c_2[i])$ and $(c_1[i+1], c_2[i+1])$ is equal to that connecting states $(\overline{c_1[i]}, \overline{c_2[i]})$ and $(\overline{c_1[i+1]}, \overline{c_2[i+1]})$. The indistinguishable states are $(c_1[i], c_2[i])$ and $(\overline{c_1[i]}, \overline{c_2[i]})$, which forms the equivalence classes.

Rooted in S-functions, ARXtools is an improved and automatic tool to study the differential properties in ARX constructions. The previous work [Leu12] demonstrates

the power of this tool in detecting incompatibility and computing the probability of a characteristic but also points out that it cannot automate the search of the characteristics. Specifically, several incompatible boomerang attacks are detected automatically, while the search of boomerang characteristics with high probabilities is absent. Later, another work by Leurent [Leu13] presents a rebound-like approach to construct differential characteristics with the help of the ARXtools. However, to the best of our knowledge, this approach cannot be adapted for boomerang characteristics. On the other hand, when computing the probability of a complicated boomerang switch, ARXtools has some advantages over ours. For example, according to [Leu12], the time of solving an S-function is proportional to the word length. Nonetheless, solving with SAT solvers could be much slower, since SAT is NP-complete. ARXtools is able to count all the solutions while ours have to discard a lot of possible solutions following the partial tables due to the speed of SAT solvers. Moreover, S-functions and ARXtools can be generalized to more cases, but currently our modelling technique is specific to BCT and its variants.

The automatic search technique in this paper creates SAT models in the benefits of the hidden properties within the matrices. Although the S-functions analysis gives the same result as Equation (10) and ARXtools is able to generate the corresponding finite automaton, they both ignore the properties in Section 3.3. In Appendix D, we provide the results generated by ARXtools to demonstrate this claim. Therefore, at this moment, we have not found a way to combine these techniques, which is left to future work.

5 Applications

In this section, we apply our technique in Section 4 to SPECK [BSS+13] and LEA [HLK+14]. SPECK is the simplest ARX cipher but still secure up until now. LEA is an ARX cipher approved by the Korean Cryptographic Module Validation Program, and has a more complicated structure. Boomerang attacks have been considered by the designers of LEA. Thereby LEA is a proper target for showing the advantage of our technique.

We briefly describe the specifications of the ciphers. Since we do not facilitate properties of the key schedules, their details are omitted. Then, we explain the SAT models of them and provide the results of the automatic search. In all our experiments, the CPU is Intel® Xeon® Silver 4110 CPU @ 2.10GHz, the SAT solver is OR-Tools [Goo21], and the programming language is C++. OR-Tools has ranked first in most of the categories in the MiniZinc Challenge[1] since 2018. Its C++ library is easy to use. We can conveniently control the number of threads, parameters, and various conditions in the code. Besides, it is not necessary to transform logical expressions into their algebraic normal forms. All the source codes are available at https://github.com/0NG/boomerang_search.

5.1 Applications on SPECK

SPECK is a family of lightweight block ciphers which are all ARX ciphers. For word size $n \in \{16, 24, 32, 48, 64\}$, each variant is identified by SPECK$2n/mn$, where $2n$ is its block size and mn is the key size. The value m depends on n. The round function of SPECK is shown in Figure 4. The rotation constants are $\alpha = 7$ and $\beta = 2$ for SPECK32/64, while $\alpha = 8$ and $\beta = 3$ for the others.

The application on SPECK is straightforward. SPECK is decomposed into three parts, such that the middle part, E_m, only contains a modular addition. The other parts are described by the differential models. The model of E_m corresponds to the model for the BCT in Section 4.2. Nevertheless, we notice that it is feasible to add one more round to E_m, that is, a 2-round switch, see Figure 5. It gives us a much better result than the 1-round switch. For ciphers based on S-boxes, the probability for switches with multiple

[1] https://www.minizinc.org/challenge.html

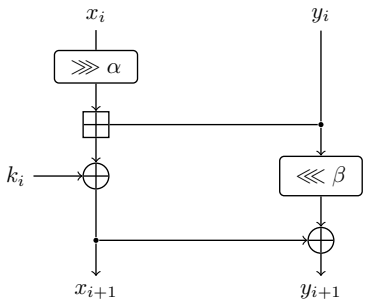
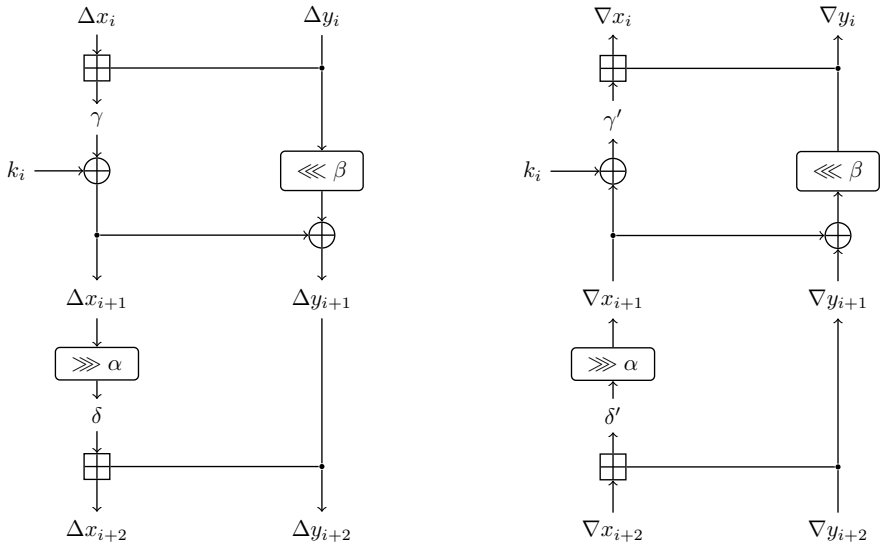


Figure 4: SPECK round function.



(a) Forward direction. (b) Backward direction.

Figure 5: A 2-round switch of SPECK.

rounds is fully studied in [WP19, BHL+20]. Since the modular addition is viewed as an S-box and each round of SPECK has only one addition, the results on S-boxes are still correct here. Namely, the probability of the 2-round switch in Figure 5 is

$$\sum_{\substack{\Delta x_{i+1}\in\mathbb{F}_2^n \\ \nabla x_{i+1}\in\mathbb{F}_2^n}} 2^{-2n}\mathrm{UBCT}(\Delta x_i, \Delta y_i, \gamma', \nabla y_i, \gamma) \cdot 2^{-2n}\mathrm{LBCT}(\delta, \Delta y_{i+2}, \nabla x_{i+2}, \nabla y_{i+2}, \delta'),$$

where γ, δ, γ', and δ' are determined by Δx_{i+1} and ∇x_{i+1}. In other words, the property of the first modular addition is described by the UBCT, and the second one is described by LBCT, since they perfectly match Equation (7) and Equation (8). When the SAT model is constructed, variables are created for Δx_{i+1} and ∇x_{i+1} without fixing their values. This allows the SAT solver to enumerate valid values of them, i.e., to do the summation during searching. Later, the complete model is simply the concatenation of two differential models and the models for the UBCT and LBCT with dealing the linear operations among them. We then apply the second framework in Section 4.3 to search for boomerang distinguishers using the parameters in Table 3.

Specifically, in the first step, SPECK32/64 has $p_{\max} = 2^{-20}$, and SPECK48/72 has $p_{\max} = 2^{-40}$. This step is very fast with 10 threads, costing 4 seconds and 34 seconds

Table 3: Parameters for SPECK.

Cipher	Rounds of E_0	Rounds of E_1	k_{lb} (Step 2)	k_{lb} (Step 3)
SPECK32/64	4	4	8	0
SPECK48/72	5	5	16	0

Table 4: Boomerang distinguishers on SPECK.

Cipher	Rounds	$-\log_2$ Prob. EST.	$-\log_2$ Prob. EXP.	Input difference of E_0 (hex)	Output difference of E_1 (hex)	Time (Step 3)
SPECK32/64	10	29.15	27.34	2800 ‖ 0010	8102 ‖ 8108	260s
		29.17	27.39	0014 ‖ 0800	8000 ‖ 840a	77s
		29.78	28.43	2800 ‖ 0010	0040 ‖ 0542	498s
SPECK48/72	12	44.15	-	020082 ‖ 120200	0080a0 ‖ 2085a4	494s
		46.41	-	820200 ‖ 001202	800084 ‖ 8400a0	816s
		47.93	-	820200 ‖ 001202	008400 ‖ 00a084	416s

EST.: The estimation of the probability from our automatic search.
EXP.: The probability observed in our experiments.

for SPECK32/64 and SPECK48/72 respectively. In the second step, we find out that $p_{\max} - t = p_{\max}$ is not sufficient to provide good trails in both ciphers. Then, the values of $p_{\max} - t$ are set to 2^{-22} for SPECK32/64 and 2^{-42} for SPECK48/72. The running time in this step is much longer, since OR-Tools does not allow enumerating with multiple threads. For SPECK32/64, it costs about 1417 seconds. Meanwhile, one more trick is applied to SPECK48/72. After 10 hours, the solver cannot find out more characteristics, and in the next 24 hours, no new characteristic is output. Thus, we just stop the solver and take the current hash table. During the last step, we notice that there are several large clusters in the hash table and the time of computing one probability is short, so we decide to compute and compare the top three of them. For SPECK32/64, $p_{\max} - t_p$ is 2^{-36}, and, for SPECK48/72, it is 2^{-46}. The results are listed in Table 4. Two examples of the boomerang characteristics are also listed in Appendix B. Finally, we choose the best ones of them as our main results. However, these improved distinguishers could not result in better key-recovery attacks than previous results. It is because that the 2-round attack from Dinur [Din14] can turn an n-round differential distinguisher into a $(n+2)$-round key-recovery attack while, to the best of our knowledge, a powerful attack like this is absent for boomerang distinguishers. Thus, the differential distinguishers in Table 1 still lead to better key-recovery attacks than ours.

The distinguisher on SPECK32/64 is practical, so we implement it to verify the correctness. The program randomly selects a pair of plaintexts, following the boomerang trail to obtain a new pair of plaintexts. Then, it checks the difference of the new pair. The key of SPECK is also randomly selected. We ran the experiments 2000 times for each characteristic. As a result, it reported that all the probabilities are slightly larger than the estimations from our automatic search.

5.2 Applications on LEA

LEA is a block cipher with a 128-bit block, whose key size is 128, 192 or 256. It is an ARX cipher with a Feistel structure. The states of LEA are separated into four 32-bit words. Meanwhile, the operations are 32-bit operations. Figure 6 is the diagram of the round function of LEA.

Let E_m be 1-round LEA, see Figure 7. Since the XOR of the round keys has no impact

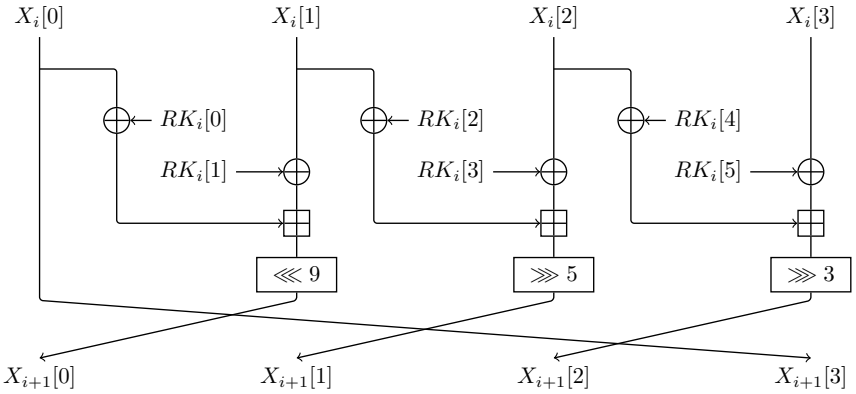


Figure 6: LEA round function.

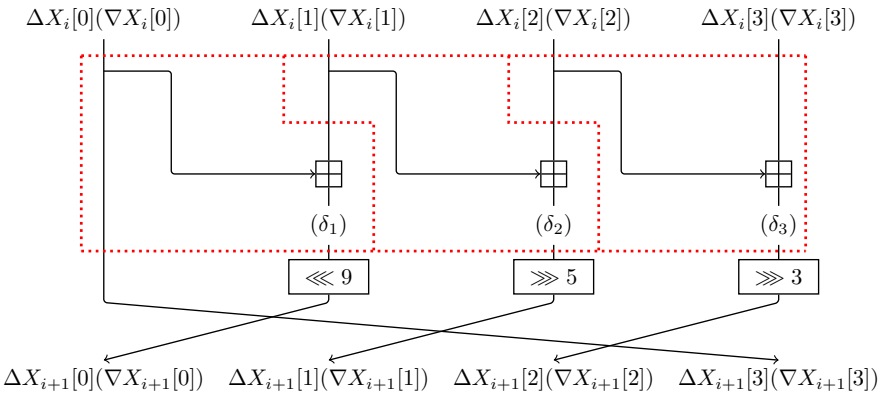


Figure 7: A 1-round switch of LEA. (The symbols for the backward direction are given in the parentheses.)

on the differences, the round keys are omitted to simplify the diagram. Meanwhile, the linear operations are simple transformations on the differences, so the focus is how to model the modular additions. We regard the additions as three parts separated by the red lines. It is clear that the modular additions share some addends. Thus, we should carefully construct the model.

Firstly, given the input difference and output difference of the switch, $\Delta X_i[0]$, $\Delta X_i[1]$, $\Delta X_i[2]$, $\Delta X_i[3]$, $\nabla X_{i+1}[0]$, $\nabla X_{i+1}[1]$, $\nabla X_{i+1}[2]$, and $\nabla X_{i+1}[3]$ are known, which also determines δ_1, δ_2, and δ_3. Assume that $\nabla X_i[1]$ and $\nabla X_i[2]$ are also fixed. Then, the right most addition conforms to what the BCT describes (Equation (1)). The probability is $2^{-64}\mathrm{BCT}(\Delta X_i[3], \Delta X_i[2], \delta_3, \nabla X_i[2])$. For the middle one, since the difference $\nabla X_i[2]$ in the backward direction is given besides the other values needed by the BCT, the LBCT is suitable for it, which has probability $2^{-64}\mathrm{LBCT}(\Delta X_i[2], \Delta X_i[1], \delta_2, \nabla X_i[1], \nabla X_i[2])$. For the same reason, the addition on the left is also described by the LBCT with probability $2^{-64}\mathrm{LBCT}(\Delta X_i[1], \Delta X_i[0], \delta_1, \nabla X_{i+1}[3], \nabla X_i[1])$. Finally, the model for the switch comprises the models for the BCT and the LBCT. If we regard the values of $\nabla X_i[1]$ and $\nabla X_i[2]$ are independent for the additions, the probability is the sum over all possible values of $\nabla X_i[1]$ and $\nabla X_i[2]$ of the product of the above three probabilities. Similar to the 2-round switch of SPECK in Section 5.1, variables are created for $\nabla X_i[1]$ and $\nabla X_i[2]$. Later, The SAT solver will select valid values for them. When we set the solver to the enumeration mode, it will go through all possible values of them and give us the probability.

Table 5: The Estimated Probabilities of the Switch in LEA

Method	Prob.	Time*
[KKS20]	0.661755	221 milliseconds
ARXtools	0.710850	34 seconds
Our SAT-aided Method	0.708521	17 hours
Experimental Evaluation	0.71	17 hours

* All the execution time contain the precomputation phase, e.g., computing some tables.

In [KKS20], the authors claimed that the probability can be computed from three BCTs. However, as our diagram shows, it is impossible to compute such a switch with three BCTs, due to the missing information about $\nabla X_i[1]$ and $\nabla X_i[2]$. To demonstrate our computation, we set up a SAT model to compute the probability following our idea above. Note that it is infeasible to directly enumerate all possible values of $\nabla X_i[1]$ and $\nabla X_i[2]$. Thanks to our heuristic strategy in Section 4.2, the parameter k_{lb} can be set to some value larger than 0, which can eliminate a lot of negligible probabilities to speed up the computation.

The target switch in the comparison is a boomerang switch reported in their work. The input difference of the switch is $\Delta = 28000200\|0002a000\|00080000\|00000001$, and the output difference is $\nabla = 80000004\|80400004\|80400014\|80400010$. By setting $k_{lb} = 16$, the SAT solver returned the probability 0.708521 in about 17 hours. We then tried $k_{lb} = 8$, and received the result 0.710802, which is only a slight improvement but costs around 33 hours, so $k_{lb} = 16$ is enough. To compare the results, we also implemented a program to get the approximating probability of the same switch by randomly selecting 2^{30} plaintexts. The program showed that the probability is about 0.71. Thereby, we believe our model and computation are correct. In contrast, the estimation in [KKS20] is 0.661755. Thus, the boomerang distinguishers reported in this work need to be checked carefully. The security of LEA against boomerang attacks is supposed to be further studied.

In fact, the 1-round switch of LEA is an S-function, which can also be analysed by ARXtools. The detailed analysis is explained in Appendix C. The results of the experiments are listed in Table 5. As expected, ARXtools returned the result in a much shorter time than the SAT solver, which is consistent with the discussion in Section 4.4. This is the main advantage of ARXtools over our techniques in practical analysis. However, when it comes to the search of a characteristic, our techniques still work, but ARXtools cannot. The last but important observation is that our technique can reach a good estimation approaching the result of ARXtools in a practical time, which indicates the reliability of the technique.

To search for a boomerang distinguisher for LEA, the first framework in Section 4.3 is chosen. We have also tried the second framework, but the complexity of LEA made it infeasible to enumerate all the characteristics. The best 15-round boomerang distinguisher found is exactly the one reported in [KKS20]. However, for 16-round LEA, no boomerang trail with probability higher than 2^{-128} was found. When the model takes the differences given by [KKS20], the upper bound $p_{\max}$ is only 2^{-132}.

6 Conclusion

In this paper, we concentrate on the automatic search of boomerang distinguishers on ARX ciphers. We study the computation of the boomerang connectivity table and its variants from the perspective of dynamic programming. Based on the algorithms, we use boolean variables and logical expressions to describe the tables. Then several models for the propagations in boomerang switches of ARX ciphers are provided. We further propose

two frameworks powered by SAT solvers, facilitating our models of the tables. These frameworks for the first time provide the methods to search for boomerang distinguishers on ARX ciphers. We take SPECK and LEA as the targets in our applications. Several boomerang distinguishers for SPECK are found, which have not been reported before. Although no new boomerang distinguisher on LEA is found, we compare our result with the previous result, showing that our method is more precise.

However, there are some disadvantages of our techniques. First, the computations of BCT and its variants are not convenient to be modelled. Second, the model for a switch with more than 2 rounds are too large, even for SPECK. It is infeasible to handle this huge model in SAT solvers. Third, we can only find out distinguishers with large probability by our heuristic strategy. The optimal distinguishers are still unknown. These are left to future work. Finally, computing the probability of a boomerang switch is much slower than ARXtools. How to combine both of the these tools/techniques is a challenging work.

7 Acknowledgement

We would like to thank the reviewers for their constructive comments and suggestions, which helped us improve the quality of the paper. In particular, one of the anonymous reviewers reminded us that ARXtools can compute the probability of the 1-round switch in LEA very efficiently. This not only drove us to improve the experiments but also helped us better understand the differences between these techniques. Meanwhile, another researcher Zhongfeng Niu noticed the wider applications of Lemma 2 beyond BCT. We want to thank him for helping us improve the comparison in Section 4.4 and the editorial quality of the paper. This work is supported by the National Key Research and Development Program of China (2022YFB2701900), the Natural Science Foundation of China (62272362, U19B2021, 62032014), and the Fundamental Research Funds for the Central Universities.

References

[BDD03] Alex Biryukov, Christophe De Cannière, and Gustaf Dellkrantz. Cryptanalysis of SAFER++. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 195–211. Springer, Heidelberg, August 2003.

[BHL+20] Hamid Boukerrou, Paul Huynh, Virginie Lallemand, Bimal Mandal, and Marine Minier. On the Feistel counterpart of the boomerang connectivity table (long paper). *IACR Trans. Symm. Cryptol.*, 2020(1):331–362, 2020.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 1–18. Springer, Heidelberg, December 2009.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 2–21. Springer, Heidelberg, August 1991.

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

[BV14] Alex Biryukov and Vesselin Velichkov. Automatic search for differential trails in ARX ciphers. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 227–250. Springer, Heidelberg, February 2014.

[CHP+17] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. A security analysis of Deoxys and its internal tweakable block ciphers. *IACR Trans. Symm. Cryptol.*, 2017(3):73–107, 2017.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 683–714. Springer, Heidelberg, April / May 2018.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[DDV20] Stéphanie Delaune, Patrick Derbez, and Mathieu Vavrille. Catching the fastest boomerangs application to SKINNY. *IACR Trans. Symm. Cryptol.*, 2020(4):104–129, 2020.

[Din14] Itai Dinur. Improved differential cryptanalysis of round-reduced Speck. In Antoine Joux and Amr M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 147–164. Springer, Heidelberg, August 2014.

[DKS14] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. *Journal of Cryptology*, 27(4):824–849, October 2014.

[Dun18] Orr Dunkelman. Efficient construction of the boomerang connection table. Cryptology ePrint Archive, Report 2018/631, 2018. https://eprint.iacr.org/2018/631.

[Goo21] Google. *OR-Tools (Version 9.1)*, 2021. https://developers.google.com/optimization/.

[HLK+14] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In Yongdae Kim, Heejo Lee, and Adrian Perrig, editors, *WISA 13*, volume 8267 of *LNCS*, pages 3–27. Springer, Heidelberg, August 2014.

[KKS20] Dongyeong Kim, Dawoon Kwon, and Junghwan Song. Efficient computation of boomerang connection probability for ARX-based block ciphers with application to SPECK and LEA. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 103(4):677–685, 2020.

[Leu12] Gaëtan Leurent. Analysis of differential attacks in ARX constructions. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 226–243. Springer, Heidelberg, December 2012.

[Leu13] Gaëtan Leurent. Construction of differential characteristics in ARX designs application to Skein. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 241–258. Springer, Heidelberg, August 2013.

[LKK+18] HoChang Lee, Seojin Kim, HyungChul Kang, Deukjo Hong, Jaechul Sung, and Seokhie Hong. Calculating the approximate probability of differentials for arx-based cipher using sat solver. *Journal of the Korea Institute of Information Security & Cryptology*, 28(1):15–24, 2018.

[LS19] Yunwen Liu and Yu Sasaki. Related-key boomerang attacks on GIFT with automated trail search including BCT effect. In Julian Jang-Jaccard and Fuchun Guo, editors, *ACISP 19*, volume 11547 of *LNCS*, pages 555–572. Springer, Heidelberg, July 2019.

[Mur11] Sean Murphy. The return of the cryptographic boomerang. *IEEE Transactions on Information Theory*, 57(4):2517–2521, 2011.

[MVDP11] Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The differential analysis of S-functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 36–56. Springer, Heidelberg, August 2011.

[Sag20] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.2)*, 2020. https://www.sagemath.org.

[SHY16] Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In Joseph K. Liu and Ron Steinfeld, editors, *ACISP 16, Part II*, volume 9723 of *LNCS*, pages 379–394. Springer, Heidelberg, July 2016.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Trans. Symm. Cryptol.*, 2021(1):269–315, 2021.

[Wag99] David Wagner. The boomerang attack. In Lars R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 156–170. Springer, Heidelberg, March 1999.

[WP19] Haoyang Wang and Thomas Peyrin. Boomerang switch in multiple rounds. *IACR Trans. Symm. Cryptol.*, 2019(1):142–169, 2019.

A Dynamic Programming Algorithms for The Variants of BCT

In this section, we provide the details about the dynamic programming algorithms for the variants of BCT and their optimization.

A.1 LBCT

The definition of LBCT of a modular addition is

$$\begin{aligned}
&\mathrm{LBCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r, \nabla_l') \\
=&\#\Big\{(L,R) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \;\Big|\; \big((L \boxplus_n R) \oplus \nabla_l \boxminus_n (R \oplus \nabla_r)\big) \\
&\quad \oplus \Big(\big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) \oplus \nabla_l \boxminus_n (R \oplus \Delta_r \oplus \nabla_r)\Big) = \Delta_l, \\
&\quad \big((L \oplus \nabla_l') \boxplus_n (R \oplus \nabla_r)\big) \oplus (L \boxplus_n R) = \nabla_l\Big\}.
\end{aligned}$$

Compare with the definition of BCT, the additional restriction is $\big((L \oplus \nabla_l') \boxplus_n (R \oplus \nabla_r)\big) \oplus (L \boxplus_n R) = \nabla_l$. Define the carry of $(L \oplus \nabla_l') \boxplus_n (R \oplus \nabla_r)$ as $c_3 = \mathrm{carry0}_n(L \oplus \nabla_l', R \oplus \nabla_r)$. It can also be rewritten as $c_3 \oplus c_1 = \nabla_l \oplus \nabla_l' \oplus \nabla_r$. Therefore, for all $i \in [0, n-1]$, the tuple $(c_1[i], b_1[i], c_2[i], b_2[i], c_3[i])$ can only take the values in

$$\begin{aligned}
S_{\mathrm{LBCT}} = \{&00000, 00110, 01010, 01100, 10010, 10100, 11000, 11110, \\
&00001, 00111, 01011, 01101, 10011, 10101, 11001, 11111\}.
\end{aligned}$$

Denote the counterparts of functions g, T, and f for LBCT as g_{LBCT}, T_{LBCT}, and f_{LBCT}, respectively. Their formal definitions are listed below. Note that definition of g_{LBCT} is already given in Definition 6.

Definition 7. Define a function T_{LBCT} as

$$\begin{aligned}
&T_{\mathrm{LBCT}}(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]) \\
=&\sum_{L[i],R[i] \in \mathbb{F}_2} g_{\mathrm{LBCT}}(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]).
\end{aligned}$$

Definition 8. Define a function f_{LBCT} as

$$\begin{aligned}
&f_{\mathrm{LBCT}}(i, v) \\
=&\#\left\{(L^i, R^i) \in \mathbb{F}_2^i \times \mathbb{F}_2^i \;\middle|\; \begin{array}{l} c_1[i] \| b_1[i] \| c_2[i] \| b_2[i] \| c_3[i] = v \\ \forall j \in [0, i-1], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \\ \forall j \in [0, i-1], c_3[j] \oplus c_1[j] = \nabla_l[j] \oplus \nabla_l'[j] \oplus \nabla_r[j] \end{array}\right\}.
\end{aligned}$$

Note that some elements in S_{LBCT} may not satisfy $c_3[i] \oplus c_1[i] = \nabla_l[i] \oplus \nabla_l'[i] \oplus \nabla_r[i]$ for $i \in [0, n-1]$. We define a new set Q_{LBCT} for the given values of $\nabla_l[n-1]$, $\nabla_r[n-1]$, and $\nabla_l'[n-1]$.

$$Q_{\mathrm{LBCT}} = \big\{v_0 \| v_1 \| v_2 \| v_3 \| v_4 \in S_{\mathrm{LBCT}} \;\big|\; v_0 \oplus v_4 = \nabla_l[n-1] \oplus \nabla_l'[n-1] \oplus \nabla_r[n-1]\big\}$$

Then, since Lemma 1 still holds here, we have

$$\begin{aligned}
&\mathrm{LBCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r, \nabla_l') \\
=&4 \\
&\times \sum_{v \in Q_{\mathrm{LBCT}}} \# \left\{ \begin{array}{l|l} (L^{n-1}, R^{n-1}) & c_1[n-1] \| b_1[n-1] \| c_2[n-1] \| b_2[n-1] \| c_3[n-1] = v \\ \in \mathbb{F}_2^{n-1} \times \mathbb{F}_2^{n-1} & \forall j \in [0, n-2], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \\ & \forall j \in [0, n-2], c_3[j] \oplus c_1[j] = \nabla_l[j] \oplus \nabla_l'[j] \oplus \nabla_r[j] \end{array} \right\} \\
=&4 \times \sum_{v \in Q_{\mathrm{LBCT}}} f_{\mathrm{LBCT}}(n-1, v).
\end{aligned}$$

Similar to the recursive expression of the function f, the expression of f_{LBCT} is

$$\begin{aligned}
&f_{\mathrm{LBCT}}(i+1, v) \\
&= \sum_{u \in S_{\mathrm{LBCT}}} \# \left\{ \begin{array}{l|l} & c_1[i+1] \| b_1[i+1] \| c_2[i+1] \| b_2[i+1] \| c_3[i+1] = v \\ (L^{i+1}, R^{i+1}) & c_1[i] \| b_1[i] \| c_2[i] \| b_2[i] \| c_3[i] = u \\ \in \mathbb{F}_2^{i+1} \times \mathbb{F}_2^{i+1} & \forall j \in [0, i], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \\ & \forall j \in [0, i], c_3[j] \oplus c_1[j] = \nabla_l[j] \oplus \nabla_l'[j] \oplus \nabla_r[j] \end{array} \right\} \\
&= \sum_{u \in S_{\mathrm{LBCT}}} \sum_{L[i], R[i] \in \mathbb{F}_2} g_{\mathrm{LBCT}}(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]) \\
&\qquad \times \# \left\{ \begin{array}{l|l} (L^i, R^i) & c_1[i] \| b_1[i] \| c_2[i] \| b_2[i] \| c_3[i] = u, \\ \in \mathbb{F}_2^i \times \mathbb{F}_2^i & \forall j \in [0, i-1], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \\ & \forall j \in [0, i-1], c_3[j] \oplus c_1[j] = \nabla_l[j] \oplus \nabla_l'[j] \oplus \nabla_r[j] \end{array} \right\} \\
&= \sum_{u \in S_{\mathrm{LBCT}}} \sum_{L[i], R[i] \in \mathbb{F}_2} g_{\mathrm{LBCT}}(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]) f_{\mathrm{LBCT}}(i, u) \\
&= \sum_{u \in S_{\mathrm{LBCT}}} f_{\mathrm{LBCT}}(i, u) \sum_{L[i], R[i] \in \mathbb{F}_2} g_{\mathrm{LBCT}}(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]).
\end{aligned}$$

This result has the same form as the one for BCT, but the functions are replaced with the new ones. In fact, during this derivation, the only impact of the new restrictions is changing the symbols of the functions and the set. This allows us quickly writing down the expressions for UBCT and EBCT.

Next, with Lemma 2, we also have $g_{\mathrm{LBCT}}(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]) = g_{\mathrm{LBCT}}(\overline{u}, \overline{v}, \overline{L[i]}, \overline{R[i]}, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i])$. It leads to

$$f'_{\mathrm{LBCT}}(i+1, v) = \sum_{u \in S'_{\mathrm{LBCT}}} T'_{\mathrm{LBCT}}(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]) f'_{\mathrm{LBCT}}(i, u)$$

where $S'_{\mathrm{LBCT}} = \{00000, 00110, 01010, 01100, 00001, 00111, 01011, 01101\}$, $f'_{\mathrm{LBCT}}(i, v) = f_{\mathrm{LBCT}}(i, v) + f_{\mathrm{LBCT}}(i, \overline{v})$, and

$$\begin{aligned}
&T'_{\mathrm{LBCT}}(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]) \\
=&T_{\mathrm{LBCT}}(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]) + T_{\mathrm{LBCT}}(\overline{u}, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla_l'[i]).
\end{aligned}$$

This is because that Theorem 2 only relies on the invariant of the function g under the one's complement of u, v, $L[i]$, and $R[i]$, and this invariant still holds for g_{LBCT}. Again, it enables us to quickly obtain the optimization for UBCT and EBCT.

Finally, the dynamic programming algorithm for LBCT is listed below. The difference between this algorithm and the one for BCT is the last code about summing up $T_{dp}[u]$. It is due to the special subset Q_{LBCT}. For UBCT and EBCT, this code conforms to the corresponding special subset and the others remain almost unchanged.

Algorithm 4 An optimized dynamic programming algorithm to compute entries of LBCT.

procedure $\mathrm{DP}_{\mathrm{LBCT}}(n, \Delta_l, \Delta_r, \nabla_l, \nabla_r, \nabla'_l)$
  Initialize a hash table T_{dp} such that, for all $u \in S'_{\mathrm{LBCT}}$, $T_{dp}[u] = 0$ except that $T_{dp}[01010] = 1$;
  for all $i \in \{0, 1, 2, \ldots, n-2\}$ **do**
    Initialize a hash table T'_{dp} such that $T'_{dp}[u] = 0$ for all $u \in S'_{\mathrm{LBCT}}$;
    for all $u \in S'_{\mathrm{LBCT}}$ **do**
      for all $u' \in S'_{\mathrm{LBCT}}$ **do**
        Look up the value $T'_{\mathrm{LBCT}}(u', u, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \nabla'_l[i])$. Assume that it is t;
        $T'_{dp}[u] \leftarrow T'_{dp}[u] + t \times T_{dp}[u']$;
    $T_{dp} \leftarrow T'_{dp}$;
  $sum \leftarrow 0$;
  for all $u \in S'_{\mathrm{LBCT}}$ **do**
    Let u be $u_0 \| u_1 \| u_2 \| u_3 \| u_4$;
    if $u_4 \oplus u_0 = \nabla_l[n-1] \oplus \nabla'_l[n-1] \oplus \nabla_r[n-1]$ **then**
      $sum \leftarrow sum + T_{dp}[u]$
  return $4 \times sum$;

A.2 UBCT

The definition of UBCT of a modular addition is

$$\begin{aligned}
&\mathrm{UBCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r, \Delta'_l) \\
=\#&\Big\{(L, R) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \;\Big|\; \big((L \boxplus_n R) \oplus \nabla_l \boxminus_n (R \oplus \nabla_r)\big) \\
&\quad \oplus \Big(\big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) \oplus \nabla_l \boxminus_n (R \oplus \Delta_r \oplus \nabla_r)\Big) = \Delta_l, \\
&\quad (L \boxplus_n R) \oplus \big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) = \Delta'_l\Big\}.
\end{aligned}$$

Compare with the definition of BCT, the additional restriction is $(L \boxplus_n R) \oplus \big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) = \Delta'_l$. The carries in this equation are already defined, so no new definition is needed. For all $i \in [0, n-1]$, the tuple $(c_1[i], b_1[i], c_2[i], b_2[i])$ can still take the values in S_{BCT}. In order to keep the consistency of the symbols, we still defined a new set S_{UBCT} which is identical to S_{BCT}. Like LBCT, a special subset Q_{UBCT} is defined as follows.

$$Q_{\mathrm{UBCT}} = \big\{v_0 \| v_1 \| v_2 \| v_3 \in S_{\mathrm{UBCT}} \;\big|\; v_0 \oplus v_2 = \Delta[n-1] \oplus \Delta'_l[n-1] \oplus \Delta_r[n-1]\big\}$$

Denote the counterparts of functions g, T, and f for UBCT as g_{UBCT}, T_{UBCT}, and f_{UBCT}, respectively. Their formal definitions are listed below.

Definition 9. Define $g_{\mathrm{UBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \delta'_l) \in \mathbb{F}_2$, where $u, v \in \mathbb{F}_2^4$, $x, y \in \mathbb{F}_2$, and $\delta_l, \delta_r, \gamma_l, \gamma_r, \delta'_l \in \mathbb{F}_2$. Given the forms $u = u_0 \| u_1 \| u_2 \| u_3$ and $v = v_0 \| v_1 \| v_2 \| v_3$, $g_{\mathrm{UBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \delta'_l) = 1$ if and only if:

$$\begin{aligned}
&v_0 = \mathrm{carry}(x, y, u_0); \\
&v_1 = \mathrm{carry}(x \oplus y \oplus v_0 \oplus \gamma_l, \overline{y \oplus \gamma_r}, u_1); \\
&v_2 = \mathrm{carry}(x \oplus \delta_l, y \oplus \delta_r, u_2); \\
&v_3 = \mathrm{carry}(x \oplus \delta_l \oplus y \oplus \delta_r \oplus v_2 \oplus \gamma_l, \overline{y \oplus \delta_r \oplus \gamma_r}, u_3); \\
&u_2 \oplus u_0 = \delta_l \oplus \delta'_l \oplus \delta_r.
\end{aligned}$$

Otherwise, $g_{\mathrm{UBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \delta'_l) = 0$.

Definition 10. Define a function T_{UBCT} as

$$\begin{aligned}&T_{\mathrm{UBCT}}(u,v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i])\\=&\sum_{L[i],R[i]\in\mathbb{F}_2} g_{\mathrm{UBCT}}(u,v,L[i],R[i],\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i]).\end{aligned}$$

Definition 11. Define a function f_{UBCT} as

$$\begin{aligned}&f_{\mathrm{UBCT}}(i,v)\\=&\#\left\{(L^i,R^i)\in\mathbb{F}_2^i\times\mathbb{F}_2^i \;\middle|\; \begin{array}{l}c_1[i]\|b_1[i]\|c_2[i]\|b_2[i]=v\\ \forall j\in[0,i-1], c_1[j]\oplus b_1[j]\oplus c_2[j]\oplus b_2[j]=0\\ \forall j\in[0,i-1], c_2[j]\oplus c_1[j]=\Delta_l[j]\oplus\Delta'_l[j]\oplus\Delta_r[j]\end{array}\right\}.\end{aligned}$$

Then, according to Lemma 1, we have

$$\mathrm{UBCT}_n(\Delta_l,\Delta_r,\nabla_l,\nabla_r,\Delta'_l)=4\times\sum_{v\in Q_{\mathrm{UBCT}}} f_{\mathrm{UBCT}}(n-1,v).$$

Similar to the recursive expression of the function f, the expression of f_{UBCT} is

$$\begin{aligned}&f_{\mathrm{UBCT}}(i+1,v)\\=&\sum_{u\in S_{\mathrm{UBCT}}} f_{\mathrm{UBCT}}(i,u)\sum_{L[i],R[i]\in\mathbb{F}_2} g_{\mathrm{UBCT}}(u,v,L[i],R[i],\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i]).\end{aligned}$$

Next, with Lemma 2, we have $g_{\mathrm{UBCT}}(u,v,L[i],R[i],\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i]) = g_{\mathrm{UBCT}}(\overline{u},\overline{v},\overline{L[i]},\overline{R[i]},\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i])$. It leads to

$$f'_{\mathrm{UBCT}}(i+1,v)=\sum_{u\in S'_{\mathrm{UBCT}}} T'_{\mathrm{UBCT}}(u,v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i])f'_{\mathrm{UBCT}}(i,u)$$

where $S'_{\mathrm{UBCT}}=S'_{\mathrm{BCT}}$, $f'_{\mathrm{UBCT}}(i,v)=f_{\mathrm{UBCT}}(i,v)+f_{\mathrm{UBCT}}(i,\overline{v})$, and

$$\begin{aligned}&T'_{\mathrm{UBCT}}(u,v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i])\\=&T_{\mathrm{UBCT}}(u,v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i])+T_{\mathrm{UBCT}}(\overline{u},v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i]).\end{aligned}$$

Finally, the dynamic programming algorithm for UBCT is listed in Algorithm 5.

A.3 EBCT

For EBCT, all the definitions and the algorithm can be viewed as the combination of those for LBCT and UBCT. They are listed below.

The definition of EBCT of a modular addition is

$$\begin{aligned}&\mathrm{EBCT}_n(\Delta_l,\Delta_r,\nabla_l,\nabla_r,\Delta'_l,\nabla'_l)\\=&\#\Big\{(L,R)\in\mathbb{F}_2^n\times\mathbb{F}_2^n \;\Big|\; \big((L\boxplus_n R)\oplus\nabla_l\boxminus_n(R\oplus\nabla_r)\big)\\&\quad\oplus\Big(\big((L\oplus\Delta_l)\boxplus_n(R\oplus\Delta_r)\big)\oplus\nabla_l\boxminus_n(R\oplus\Delta_r\oplus\nabla_r)\Big)=\Delta_l,\\&\quad(L\boxplus_n R)\oplus\big((L\oplus\Delta_l)\boxplus_n(R\oplus\Delta_r)\big)=\Delta'_l,\\&\quad\big((L\oplus\nabla'_l)\boxplus_n(R\oplus\nabla_r)\big)\oplus(L\boxplus_n R)=\nabla_l\Big\}.\end{aligned}$$

The set S_{EBCT} is identical to S_{LBCT}. The subset Q_{EBCT} is defined as follows.

$$Q_{\mathrm{EBCT}}=\left\{v_0\|v_1\|v_2\|v_3\|v_4\in S_{\mathrm{EBCT}} \;\middle|\; \begin{array}{l}v_0\oplus v_4=\nabla_l[n-1]\oplus\nabla'_l[n-1]\oplus\nabla_r[n-1]\\ v_0\oplus v_2=\Delta[n-1]\oplus\Delta'_l[n-1]\oplus\Delta_r[n-1]\end{array}\right\}$$

The functions g_{EBCT}, T_{EBCT}, and f_{EBCT} are respectively defined as follows.

Algorithm 5 An optimized dynamic programming algorithm to compute entries of UBCT.

procedure $\mathrm{DP}_{\mathrm{UBCT}}(n, \Delta_l, \Delta_r, \nabla_l, \nabla_r, \Delta'_l)$
  Initialize a hash table T_{dp} such that, for all $u \in S'_{\mathrm{UBCT}}$, $T_{dp}[u] = 0$ except that $T_{dp}[0101] = 1$;
  for all $i \in \{0, 1, 2, \ldots, n-2\}$ **do**
    Initialize a hash table T'_{dp} such that $T'_{dp}[u] = 0$ for all $u \in S'_{\mathrm{UBCT}}$;
    for all $u \in S'_{\mathrm{UBCT}}$ **do**
      for all $u' \in S'_{\mathrm{UBCT}}$ **do**
        Look up the value $T'_{\mathrm{UBCT}}(u', u, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \Delta'_l[i])$. Assume that it is t;
        $T'_{dp}[u] \leftarrow T'_{dp}[u] + t \times T_{dp}[u']$;
    $T_{dp} \leftarrow T'_{dp}$;
  $sum \leftarrow 0$;
  for all $u \in S'_{\mathrm{UBCT}}$ **do**
    Let u be $u_0 \| u_1 \| u_2 \| u_3$;
    if $u_2 \oplus u_0 = \Delta[n-1] \oplus \Delta'_l[n-1] \oplus \Delta_r[n-1]$ **then**
      $sum \leftarrow sum + T_{dp}[u]$
  return $4 \times sum$;

Definition 12. Define $g_{\mathrm{EBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \delta'_l, \gamma'_l) \in \mathbb{F}_2$, where $u, v \in \mathbb{F}_2^5$, $x, y \in \mathbb{F}_2$, and $\delta_l, \delta_r, \gamma_l, \gamma_r, \delta'_l, \gamma'_l \in \mathbb{F}_2$. Given the forms $u = u_0 \| u_1 \| u_2 \| u_3 \| u_4$ and $v = v_0 \| v_1 \| v_2 \| v_3 \| v_4$, $g_{\mathrm{EBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \delta'_l, \gamma'_l) = 1$ if and only if:

$$
\begin{aligned}
&v_0 = \mathrm{carry}(x, y, u_0);\\
&v_1 = \mathrm{carry}(x \oplus y \oplus v_0 \oplus \gamma_l, \overline{y \oplus \gamma_r}, u_1);\\
&v_2 = \mathrm{carry}(x \oplus \delta_l, y \oplus \delta_r, u_2);\\
&v_3 = \mathrm{carry}(x \oplus \delta_l \oplus y \oplus \delta_r \oplus v_2 \oplus \gamma_l, \overline{y \oplus \delta_r \oplus \gamma_r}, u_3);\\
&u_4 \oplus u_0 = \gamma_l \oplus \gamma'_l \oplus \gamma_r;\\
&v_4 = \mathrm{carry}(x \oplus \gamma'_l, y \oplus \gamma_r, u_4);\\
&u_2 \oplus u_0 = \delta_l \oplus \delta'_l \oplus \delta_r.
\end{aligned}
$$

Otherwise, $g_{\mathrm{EBCT}}(u, v, x, y, \delta_l, \delta_r, \gamma_l, \gamma_r, \delta'_l, \gamma'_l) = 0$.

Definition 13. Define a function T_{EBCT} as

$$
\begin{aligned}
&T_{\mathrm{EBCT}}(u, v, \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \Delta'_l[i], \nabla'_l[i])\\
=&\sum_{L[i], R[i] \in \mathbb{F}_2} g_{\mathrm{EBCT}}(u, v, L[i], R[i], \Delta_l[i], \Delta_r[i], \nabla_l[i], \nabla_r[i], \Delta'_l[i], \nabla'_l).
\end{aligned}
$$

Definition 14. Define a function f_{EBCT} as

$$
\begin{aligned}
&f_{\mathrm{EBCT}}(i, v)\\
=\#&\left\{ (L^i, R^i) \in \mathbb{F}_2^i \times \mathbb{F}_2^i \;\middle|\; \begin{array}{r} c_1[i] \| b_1[i] \| c_2[i] \| b_2[i] \| c_3[i] = v \\ \forall j \in [0, i-1], c_1[j] \oplus b_1[j] \oplus c_2[j] \oplus b_2[j] = 0 \\ \forall j \in [0, i-1], c_3[j] \oplus c_1[j] = \nabla_l[j] \oplus \nabla'_l[j] \oplus \nabla_r[j] \\ \forall j \in [0, i-1], c_2[j] \oplus c_1[j] = \Delta_l[j] \oplus \Delta'_l[j] \oplus \Delta_r[j] \end{array} \right\}.
\end{aligned}
$$

The formula for $\mathrm{EBCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r, \Delta'_l, \nabla'_l)$ is

$$
\mathrm{EBCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r, \Delta'_l, \nabla'_l) = 4 \times \sum_{v \in Q_{\mathrm{EBCT}}} f_{\mathrm{EBCT}}(n-1, v).
$$

We still have the following optimization.

$$f'_{\mathrm{EBCT}}(i+1,v)=\sum_{u\in S'_{\mathrm{EBCT}}} T'_{\mathrm{EBCT}}(u,v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i],\nabla'_l[i])f'_{\mathrm{EBCT}}(i,u)$$

where $S'_{\mathrm{EBCT}}=S'_{\mathrm{LBCT}}$, $f'_{\mathrm{EBCT}}(i,v)=f_{\mathrm{EBCT}}(i,v)+f_{\mathrm{EBCT}}(i,\overline{v})$, and

$$\begin{aligned}&T'_{\mathrm{EBCT}}(u,v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i],\nabla'_l[i])\\=&T_{\mathrm{EBCT}}(u,v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i],\nabla'_l[i])\\&+T_{\mathrm{EBCT}}(\overline{u},v,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i],\nabla'_l[i]).\end{aligned}$$

The recursive expression of f_{EBCT} is

$$\begin{aligned}&f_{\mathrm{EBCT}}(i+1,v)\\=&\sum_{u\in S_{\mathrm{EBCT}}} f_{\mathrm{EBCT}}(i,u)\sum_{L[i],R[i]\in\mathbb{F}_2} g_{\mathrm{EBCT}}(u,v,L[i],R[i],\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i],\nabla'_l[i]).\end{aligned}$$

The dynamic programming algorithm for UBCT is listed in Algorithm 6.

Algorithm 6 An optimized dynamic programming algorithm to compute entries of EBCT.

procedure $\mathrm{DP}_{\mathrm{EBCT}}(n,\Delta_l,\Delta_r,\nabla_l,\nabla_r,\Delta'_l,\nabla'_l)$
  Initialize a hash table T_{dp} such that, for all $u\in S'_{\mathrm{EBCT}}$, $T_{dp}[u]=0$ except that $T_{dp}[01010]=1$;
  for all $i\in\{0,1,2,\ldots,n-2\}$ **do**
    Initialize a hash table T'_{dp} such that $T'_{dp}[u]=0$ for all $u\in S'_{\mathrm{EBCT}}$;
    for all $u\in S'_{\mathrm{EBCT}}$ **do**
      for all $u'\in S'_{\mathrm{EBCT}}$ **do**
        Look up the value $T'_{\mathrm{EBCT}}(u',u,\Delta_l[i],\Delta_r[i],\nabla_l[i],\nabla_r[i],\Delta'_l[i],\nabla'_l[i])$. Assume that it is t;
        $T'_{dp}[u]\leftarrow T'_{dp}[u]+t\times T_{dp}[u']$;
    $T_{dp}\leftarrow T'_{dp}$;
  $sum\leftarrow 0$;
  for all $u\in S'_{\mathrm{EBCT}}$ **do**
    Let u be $u_0\|u_1\|u_2\|u_3\|u_4$;
    if $u_2\oplus u_0=\Delta[n-1]\oplus\Delta'_l[n-1]\oplus\Delta_r[n-1]$ **then**
      if $u_4\oplus u_0=\nabla_l[n-1]\oplus\nabla'_l[n-1]\oplus\nabla_r[n-1]$ **then**
        $sum\leftarrow sum+T_{dp}[u]$
  return $4\times sum$;

B Examples of Boomerang Characteristics in SPECK

Two characteristics output by the SAT solver are listed below for SPECK32/64 and SPECK48/72, respectively.

Table 6: A Boomerang Characteristic for SPECK32/64.

Round	Part	Left Difference (hex)	Right Difference (hex)
1	E_0	2800	0010
2		0040	0000
3		8000	8000
4		8100	8102
5	E_m	8000	840a
6		-	-
7	E_1	0a04	0804
8		0010	2000
9		0000	8000
10		8000	8002
11		8102	8108

Table 7: A Boomerang Characteristic for SPECK48/72.

Round	Part	Left Difference (hex)	Right Difference (hex)
1	E_0	020082	120200
2		900000	001000
3		008000	000000
4		000080	000080
5		800080	800480
6	E_m	008480	00a084
7		-	-
8	E_1	009000	000010
9		000080	000000
10		800000	800000
11		808000	808004
12		800084	8400a0
13		0080a0	2085a4

C Analysis by S-function

In this section, we present the S-functions analysis of the 1-round switch of LEA in Section 5.2. To simplify the explanation, the linear operations are ignored as Figure 8 shows.

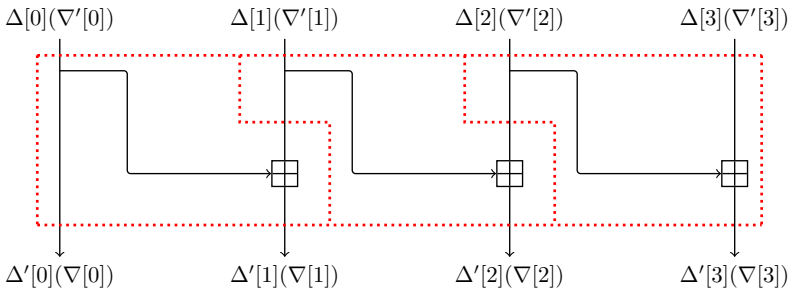


Figure 8: A simplified 1-round switch of LEA. (The symbols for the backward direction are given in the parentheses.)

C.1 Definition of the Probability

Let E_m be the short cipher in Figure 8. The probability of this switch is

$$p_m = \Pr\left[E_m^{-1}(E_m(X_1) \oplus \nabla) \oplus E_m^{-1}(E_m(X_1 \oplus \Delta) \oplus \nabla) = \Delta\right],$$

where $X_1 = X_1[0]\|X_1[1]\|X_1[2]\|X_1[3]$, $\Delta = \Delta[0]\|\Delta[1]\|\Delta[2]\|\Delta[3]$, and $\nabla = \nabla[0]\|\nabla[1]\|\nabla[2]\|\nabla[3]$. To use the S-functions technique, this formula should be rewritten in detail. Given X_1, Δ, and ∇, we calculate ΔZ using

$$\begin{aligned}
X_2 &\leftarrow X_1 \oplus \Delta, && (14)\\
T_1 &\leftarrow X_1[0]\|(X_1[1] + X_1[0])\|(X_1[2] + X_1[1])\|(X_1[3] + X_1[2]), && (15)\\
T_2 &\leftarrow X_2[0]\|(X_2[1] + X_2[0])\|(X_2[2] + X_2[1])\|(X_2[3] + X_2[2]), && (16)\\
U_1 &\leftarrow T_1 \oplus \nabla, && (17)\\
U_2 &\leftarrow T_2 \oplus \nabla, && (18)\\
Z_1 &\leftarrow U_1[0]\|(U_1[1] - U_1[0])\| && (19)\\
&\quad (U_1[2] - (U_1[1] - U_1[0]))\|(U_1[3] - (U_1[2] - (U_1[1] - U_1[0]))), && (20)\\
Z_2 &\leftarrow U_2[0]\|(U_2[1] - U_2[0])\| && (21)\\
&\quad (U_2[2] - (U_2[1] - U_2[0]))\|(U_2[3] - (U_2[2] - (U_2[1] - U_2[0]))), && (22)\\
\Delta Z &\leftarrow Z_1 \oplus Z_2. && (23)
\end{aligned}$$

Then, the probability of the switch is redefined as

$$p_m = 2^{-128} \cdot \#\left\{X_1 \in \mathbb{F}_2^{128} \mid \Delta Z = \Delta\right\}.$$

C.2 Constructing the S-Function

Equations (14)-(23) are rewritten on a bit level:

$$\begin{aligned}
X_2[j][i] &\leftarrow X_1[j][i] \oplus \Delta[j][i], \text{for } j \in [0,3], && (24)\\
T_1[0][i] &\leftarrow X_1[0][i], && (25)\\
T_1[1][i] &\leftarrow X_1[1][i] \oplus X_1[0][i] \oplus c_1[0][i], && (26)
\end{aligned}$$

$$T_1[2][i] \leftarrow X_1[2][i] \oplus X_1[1][i] \oplus c_1[1][i], \tag{27}$$
$$T_1[3][i] \leftarrow X_1[3][i] \oplus X_1[2][i] \oplus c_1[2][i], \tag{28}$$
$$c_1[0][i+1] \leftarrow (X_1[1][i] + X_1[0][i] + c_1[0][i]) \gg 1, \tag{29}$$
$$c_1[1][i+1] \leftarrow (X_1[2][i] + X_1[1][i] + c_1[1][i]) \gg 1, \tag{30}$$
$$c_1[2][i+1] \leftarrow (X_1[3][i] + X_1[2][i] + c_1[2][i]) \gg 1, \tag{31}$$
$$T_2[0][i] \leftarrow X_2[0][i], \tag{32}$$
$$T_2[1][i] \leftarrow X_2[1][i] \oplus X_2[0][i] \oplus c_2[0][i], \tag{33}$$
$$T_2[2][i] \leftarrow X_2[2][i] \oplus X_2[1][i] \oplus c_2[1][i], \tag{34}$$
$$T_2[3][i] \leftarrow X_2[3][i] \oplus X_2[2][i] \oplus c_2[2][i], \tag{35}$$
$$c_2[0][i+1] \leftarrow (X_2[1][i] + X_2[0][i] + c_2[0][i]) \gg 1, \tag{36}$$
$$c_2[1][i+1] \leftarrow (X_2[2][i] + X_2[1][i] + c_2[1][i]) \gg 1, \tag{37}$$
$$c_2[2][i+1] \leftarrow (X_2[3][i] + X_2[2][i] + c_2[2][i]) \gg 1, \tag{38}$$
$$U_1[0][i] \leftarrow T_1[0][i] \oplus \nabla[0][i], \tag{39}$$
$$U_1[1][i] \leftarrow T_1[1][i] \oplus \nabla[1][i], \tag{40}$$
$$U_1[2][i] \leftarrow T_1[2][i] \oplus \nabla[2][i], \tag{41}$$
$$U_1[3][i] \leftarrow T_1[3][i] \oplus \nabla[3][i], \tag{42}$$
$$U_2[0][i] \leftarrow T_2[0][i] \oplus \nabla[0][i], \tag{43}$$
$$U_2[1][i] \leftarrow T_2[1][i] \oplus \nabla[1][i], \tag{44}$$
$$U_2[2][i] \leftarrow T_2[2][i] \oplus \nabla[2][i], \tag{45}$$
$$U_2[3][i] \leftarrow T_2[3][i] \oplus \nabla[3][i], \tag{46}$$
$$Z_1[0][i] \leftarrow U_1[0][i], \tag{47}$$
$$Z_1[1][i] \leftarrow U_1[1][i] \oplus \overline{Z_1[0][i]} \oplus b_1[0][i], \tag{48}$$
$$Z_1[2][i] \leftarrow U_1[2][i] \oplus \overline{Z_1[1][i]} \oplus b_1[1][i], \tag{49}$$
$$Z_1[3][i] \leftarrow U_1[3][i] \oplus \overline{Z_1[2][i]} \oplus b_1[2][i], \tag{50}$$
$$b_1[0][i+1] \leftarrow (U_1[1][i] + \overline{Z_1[0][i]} + b_1[0][i]) \gg 1, \tag{51}$$
$$b_1[1][i+1] \leftarrow (U_1[2][i] + \overline{Z_1[1][i]} + b_1[1][i]) \gg 1, \tag{52}$$
$$b_1[2][i+1] \leftarrow (U_1[3][i] + \overline{Z_1[2][i]} + b_1[2][i]) \gg 1, \tag{53}$$
$$Z_2[0][i] \leftarrow U_2[0][i], \tag{54}$$
$$Z_2[1][i] \leftarrow U_2[1][i] \oplus Z_2[0][i] \oplus b_2[0][i], \tag{55}$$
$$Z_2[2][i] \leftarrow U_2[2][i] \oplus Z_2[1][i] \oplus b_2[1][i], \tag{56}$$
$$Z_2[3][i] \leftarrow U_2[3][i] \oplus Z_2[2][i] \oplus b_2[2][i], \tag{57}$$
$$b_2[0][i+1] \leftarrow (U_2[1][i] + \overline{Z_2[0][i]} + b_2[0][i]) \gg 1, \tag{58}$$
$$b_2[1][i+1] \leftarrow (U_2[2][i] + \overline{Z_2[1][i]} + b_2[1][i]) \gg 1, \tag{59}$$
$$b_2[2][i+1] \leftarrow (U_2[3][i] + \overline{Z_2[2][i]} + b_2[2][i]) \gg 1, \tag{60}$$
$$\Delta Z[j][i] \leftarrow Z_1[j][i] \oplus Z_2[j][i], \text{for } j \in [0,3]. \tag{61}$$

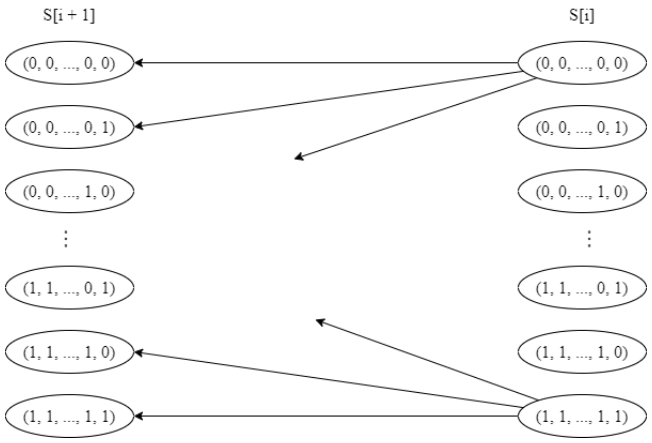


Figure 9: A bipartite graph from the S-function.

where carries $c_1[j][0] = c_2[j][0] = 0$ and $b_1[j][0] = b_2[j][0] = 1$ for $j \in [0, 2]$. Let the states be

$$\begin{aligned} S[i] \leftarrow (&c_1[0][i], c_1[1][i], c_1[2][i],\\ &b_1[0][i], b_1[1][i], b_1[2][i],\\ &c_2[0][i], c_2[1][i], c_2[2][i],\\ &b_2[0][i], b_2[1][i], b_2[2][i]),\\ S[i+1] \leftarrow (&c_1[0][i+1], c_1[1][i+1], c_1[2][i+1],\\ &b_1[0][i+1], b_1[1][i+1], b_1[2][i+1],\\ &c_2[0][i+1], c_2[1][i+1], c_2[2][i+1],\\ &b_2[0][i+1], b_2[1][i+1], b_2[2][i+1]). \end{aligned}$$

Then, Equations (24)-(61) corresponds to the S-function for the probability of the 1-round switch, as Equation 62.

$$(\Delta Z[0][i], \Delta Z[1][i], \Delta Z[2][i], \Delta Z[3][i], S[i+1]) = f(X_1, \Delta, \nabla, S[i]), 0 \le i < 32 \qquad (62)$$

C.3 Computing the Probability

In order to derive the formula for computing the probability, a graph is generated from the S-function (Equation (62)). For $0 \le i \le 32$, every state $S[i]$ is represented as a vertex in the graph. Each edge is drawn according to Equation (62). Specifically, for every pair of states, $S[i]$ and $S[i+1]$, if there are values of $X_1[0][i]$, $X_1[1][i]$, $X_1[2][i]$, $X_1[3][i]$, $\Delta[0][i]$, $\Delta[1][i]$, $\Delta[2][i]$, $\Delta[3][i]$, $\nabla[0][i]$, $\nabla[1][i]$, $\nabla[2][i]$, and $\nabla[3][i]$ such that $\Delta Z[j][i]$ is equal to $\Delta[j][i]$ for $j \in [0, 3]$, then an edge is drawn between vertices $S[i]$ and $S[i+1]$. The resulting graph consists of bipartite graphs each of which contains the vertices $S[i]$ and $S[i+1]$. Figure 9 shows an example of a bipartite graph. Note that, for any bit position i, there are 2^{12} values for $S[i]$, which generates 2^{12} different vertices.

According to [MVDP11], the probability is equal to 2^{-128} times the number of the paths from a state whose values are all zero to any of the 2^{12} vertices $S[32]$. Let $w[i]$ be

$$\Delta[0][i] \| \Delta[1][i] \| \Delta[2][i] \| \Delta[3][i] \| \nabla[0][i] \| \nabla[1][i] \| \nabla[2][i] \| \nabla[3][i].$$

Thanks to the nice properties of a bipartite graph, we can use a matrix $A_{w[i]} = [x_{kj}]$ to describe it, where x_{kj} is the number of edges that connect vertices $j = S[i]$ and $k = S[i+1]$.

Define a 1×2^{12} matrix $L = [1\ 1\ \dots\ 1]$ and a $2^{12} \times 1$ matrix $C = [1\ 0\ \dots\ 0]^T$. Then, the formula of the probability is

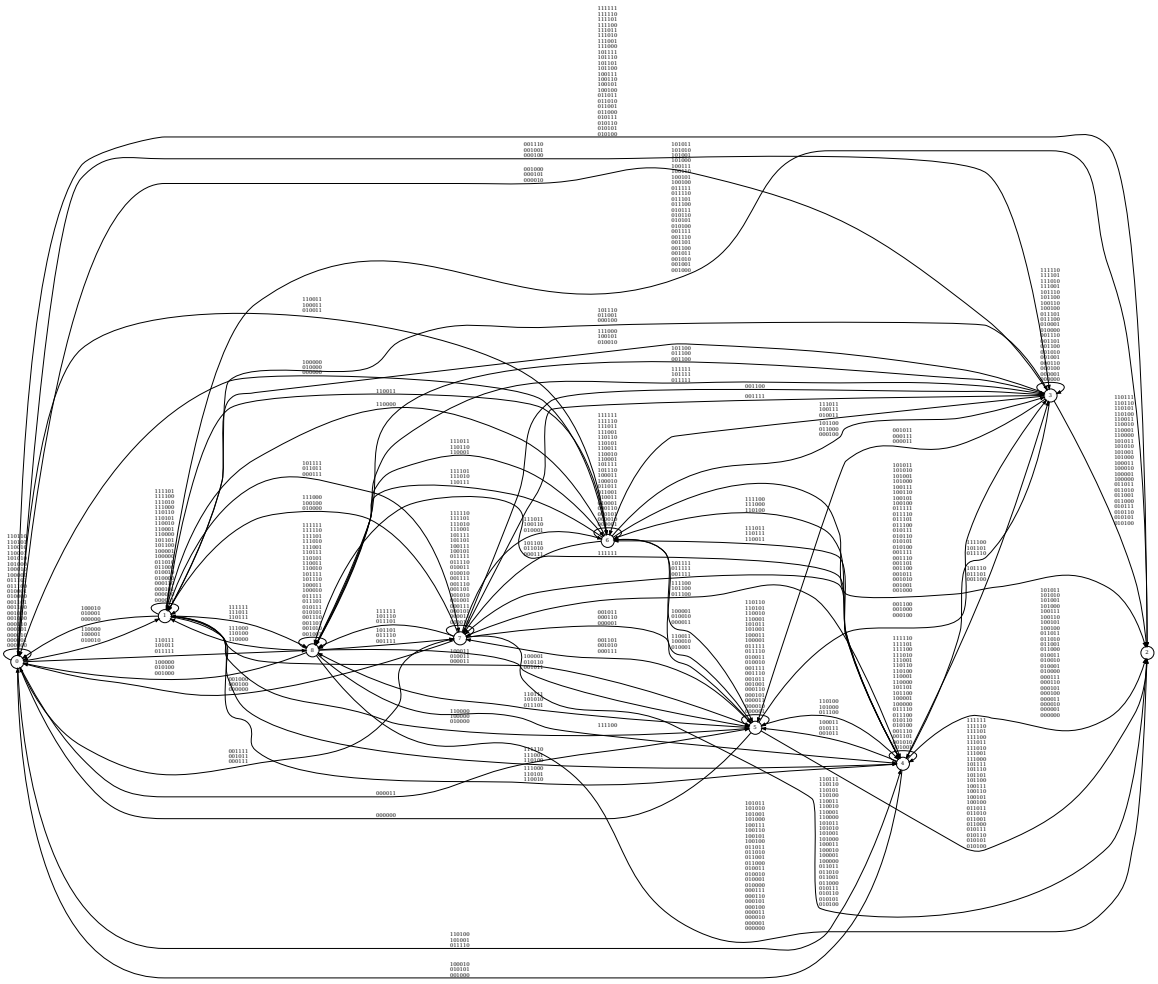
$$p_m = 2^{-128} \cdot LA_{w[31]}A_{w[30]} \cdots A_{w[0]}C. \tag{63}$$

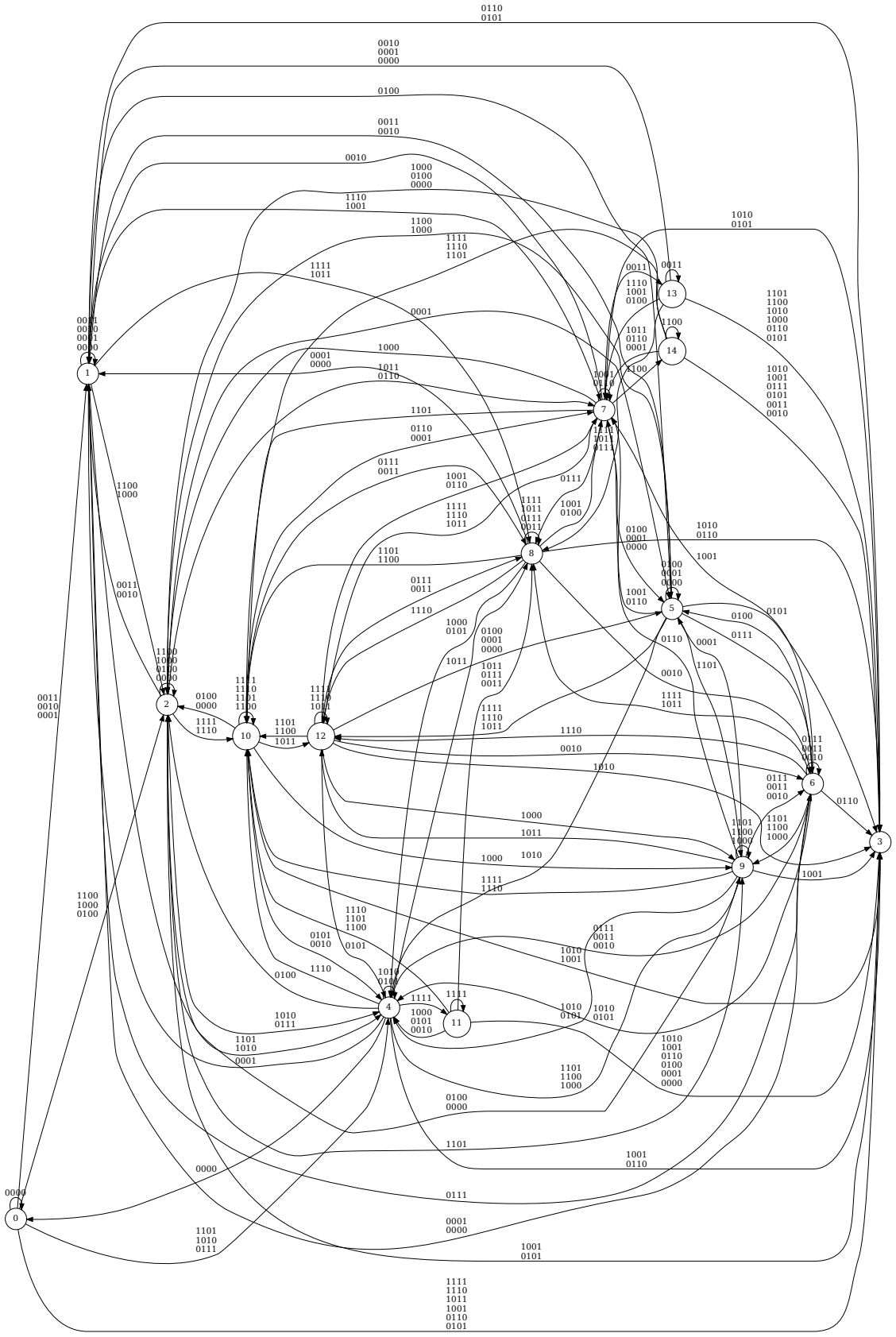
Finally, the FSM reduction algorithm in [MVDP11] helps to reduce the number of states.

ARXtools can automate the above process following [Leu12] and the instructions on its webpage (https://who.rocq.inria.fr/Gaetan.Leurent/arxtools.html). Our code is available at https://github.com/ONG/boomerang_search. The program successfully obtained the probability of the switch in Section 5.2 in 34s, and reported that the exact probability is about 0.710850.

D Results from the ARXTools

In this section, we present the results from ARXtools to show that it cannot provide the necessary information needed by our technique. The first graph is generated from the command: "build_fsm -e '(((V0+V1)^P2)-(V1^P3))^((((V0^P0)+(V1^P1))^P2)-(V1^P1^P3))==P0' -t -g". The second graph is generated from the command: "build_fsm -e '(((V0+V1)^P2)-(V1^P3))^((((V0^P0)+(V1^P1))^P2)-(V1^P1^P3))==P0' -d -g". Please see the next two pages for the results.




0
1
2
3
4
5
6
7
8
9
10
11
12
13
14