Approximate Modeling of Signed Difference and Digraph based Bit Condition Deduction

New Boomerang Attacks on BLAKE

Yonglin Hao[1], Qingju Wang[2], Lin Jiao[1], and Xinxin Gong[1]

[1] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China,
haoyonglin@yeah.net, jiaolin_jl@126.com, xinxgong@126.com
[2] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg,
qjuwang@gmail.com

Abstract. The signed difference is a powerful tool for analyzing the Addition, XOR, Rotation (ARX) cryptographic primitives. Currently, solving the accurate model for the signed difference propagation is infeasible. We propose an approximate MILP modeling method capturing the propagation rules of signed differences. Unlike the accurate signed difference model, the approximate model only focuses on active bits and ignores the possible bit conditions on inactive bits. To overcome the negative effect of a lower accuracy arising from ignoring bit conditions on inactive bits, we propose an additional tool for deducing all bit conditions automatically. Such a tool is based on a directed-graph capturing the whole computation process of ARX primitives by drawing links among intermediate words and operations. The digraph is also applicable in the MILP model construction process: it enables us to identify the parameters upper bounding the number of bit conditions so as to define the objective function; it is further used to connect the boomerang top and bottom signed differential paths by introducing proper constraints to avoid incompatible intersections. Benefiting from the approximate model and the directed-graph based tool, the solving time of the new MILP model is significantly reduced, enabling us to deduce signed differential paths efficiently and accurately.

To show the utility of our method, we propose boomerang attacks on the keyed permutations of three ARX hash functions of BLAKE. For the first time we mount an attack on the full 7 rounds of BLAKE3, with the complexity as low as 2^{180}. Our best attack on BLAKE2s can improve the previously best result by 0.5 rounds but with lower complexity. The attacks on BLAKE-256 cover the same 8 rounds with the previous best result but with complexity 2^{16} times lower. All our results are verified practically with round-reduced boomerang quartets.

Keywords: Signed Difference, Boomerang Attack, ARX, MILP modeling

1 Introduction

The boomerang attack originally proposed by Wagner [1] is an adaptive chosen plaintext and ciphertext attack derived from differential cryptanalysis. Soon afterwards, Kelsey et al. [2] developed the original version into a chosen plaintext

attack called the amplified boomerang attack. Developments were also made by Biham et al. in [3,4,5] making the boomerang attack an efficient tool for analyzing block ciphers such as AES [6,7], ARIA [8], SHACAL [9] etc (just name some as [10,9,11,8,12,13,14]). The idea of the boomerang attack is also applied to hash functions for constructing distinguishers of the underlying keyed permutations or compression functions [15]. Such boomerang distinguishers are specifically efficient on ARX hash functions such as BLAKE [16,17,18], SHA-2 [19,20,21,22], SIMD-512 [23], HAVAL [24], RIPEMD-128/160 [25], HAS-160 [26], Skein [27,28], SM3[29,30] etc.

The goal of boomerang attacks are to find the state (or state-message) quartets satisfying particular input-output differences within the generic complexity bounds: there are 3 main types of boomerang input-output differences namely **Type I**, **II** and **III** with different complexity bounds. The feasibility of a boomerang attack is based on the 2 differential paths, referred as the top path and the bottom path hereafter, intersecting at some intermediate state in the middle and propagating in backward and forward directions. The right quartet should satisfy both top and bottom paths simultaneously in the boomerang manner. Therefore, the top and bottom paths should not only have high differential propagation probabilities but be compatible in the intersecting state as well, so as to guarantee the existence of right quartets and the complexities below generic bounds.

For both S-box-oriented and ARX-like cryptographic primitives, deducing high-probabilistic and compatible top-bottom paths for boomerang attacks has always been a challenging task. But in recent years, significant progress has been made in the boomerang attacks on S-box oriented block ciphers. Firstly, Cid et al. proposed the Boomerang Connective Table (BCT) [31]: a lookup table capturing the differential propagation rule of quartets through S-box operations. In this way, for S-box oriented primitives, the top and bottom paths can be connected with a S-box layer and the compatible intersection of the 2 paths can be verified by referring to the BCTs of the corresponding S-boxes. Ever since its proposal, the BCT technique is further improved in all aspects including efficiency, accuracy etc [32,33,34,35]. There are also works for searching high probabilistic top-bottom paths and key-guessing strategies automatically based on MILP and CP models [36,37,38,39,40,41,40,42,43,44]. In fact, most of the current best boomerang attacks on block ciphers like SKINNY are using top and bottom paths deduced automatically through MILP/CP models.

On the contrary, many current best boomerang attacks on ARX hash functions are still using differential paths deduced by hand and many results have not been improved for quite many years. In fact, the cryptanalysis of ARX hash functions has quite long history because previous cryptographic hash function standards like SHA-1 and MD-5 are all ARX-like hash functions. The ground breaking works are the differential attack given by Wang *et al.* in 2005 resulting in the 1st theoretically collision attack on full SHA-1 [45] and MD-5 [46]. The most important concepts in Wang *et al.* 's work are the signed difference, the bit condition and the message modification technique. The signed difference cap-

tures the accurate differential propagation rules of modular adds including the hard-to-predict carry effects. The bit conditions are simply linear equations of bits but Wang *et al.* 's success further proved that a particular differential propagation can only happen when the corresponding bit conditions are satisfied. The message modification technique is used during the collision search process: some unsatisfied bit conditions can be directly fixed by manipulating particular message block bits so that the probability of finding collision pairs can be improved. The probability of Wang et al.'s collision attacks is determined by the number of unfixed bit conditions. Wang et al.'s technique has long been the dominating tools for analyzing all kinds of ARX-like primitives [47,48,49,50,51]. The bit condition and message modification technique are also applied to non-ARX bit-oriented primitives such as Keccak for finding collisions [52,53] or even deducing cube distinguishers [54]. Derived from differential attacks, the boomerang attacks on ARX hash functions also employ Wang et al.'s technique and its general procedure can be summarized as follows.

1. **Path Deduction:** Deduce top and bottom paths.
2. **Bit Condition Deduction:** Deduce the bit conditions corresponding to top and bottom paths.
3. **Message Modification Strategy:** Determine a strategy for fixing the bit conditions around the intersecting state so as to lower the complexities.
4. **Quartet Search:** Starting from intersecting state, compute backward and forward for quartets satisfying the boomerang differential constraints.

Motivations. Signed differential paths are either deduced by hand [45,46] or using heuristic (semi-)automatic tools dedicated to specific primitives such as SHA-1, SHA-2, RIPEMD [47,51,55,56,57,58,59,60,61]: profound experience and great efforts are required in both cases. There is an urgent need of an automatic modeling methods for deducing signed differential paths of all ARX primitives uniformly. Mixed integer linear programming (MILP), which has efficiently accomplished arduous tasks such as searching differentials for many symmetric primitives automatically [62], becomes our first choice. Furthermore, a proper definition of MILP objective functions maximizing the probability can potentially prove the optimality of signed differential paths.

Currently, many best boomerang attacks on hash functions are using top and bottom XOR differential paths deduced linearly by regarding the modular add operations as XORs [17,18]. For treating the intersecting part, a "0-AND constraint" is utilized: in order to avoid active bits in both paths resulting in contradictions [27], the bitwise AND of the top and bottom XOR differences at the intersecting state is set to 0. Apparently, the linear propagation of XOR differences completely ignores the effect of carries in modular adds, which is a huge loss of accuracy. However, carries in modular adds are not always bad: there are even cases where carries are stimulated on purpose so as to enhance the overall probability of the differential path [63], whether the 0-AND constraint is sufficient to guarantee compatible top-bottom paths remains as suspicious. On the other hand, the existing boomerang automatic tools [64,27] are more suitable for checking the compatibility of existing paths rather than constructing

compatible paths directly. Lastly, since signed difference can carry the carry effects by its nature, the boomerang attack also calls for automatic tools for deducing compatible signed differential top and bottom paths efficiently.

It is noticeable that the Bit Condition Deduction phase of existing boomerang attacks are accomplished by hand which can be quite time consuming even infeasible due to excessively complex constructions. An automatic bit condition deducing framework for ARX primitives should be useful for future research.

Our Contributions. We propose a MILP modeling method for the signed difference. The signed difference of each bit is encoded as two binary variables. The modular 2^ω add operation is decomposed into a half-adder at the least significant bit (LSB) followed by $\omega-1$ full-adder at the remaining bits. The signed difference propagation rules of half- and full-adders can be described as linear constraints in MILP models. It is noticeable that the MILP model description of accurate signed difference propagations is equivalent to finding the right pairs following the differential paths making the whole MILP model extremely hard to be solved. So we further propose an approximate signed difference modeling method that focuses on the accuracy of active bits. We deduce the MILP model capturing the approximate signed differential propagation rules for modular add and XOR operations. This modeling technique can be applied to efficiently searching for signed differential paths of any ARX structures, which is of independent interest.

As aforementioned, the approximate signed differential models focus on the active bits for high efficiency which obviously results in a loss of accuracy. According to an approximate signed differential path given by our model, we further propose a general framework for deducing bit conditions automatically based on a given signed differential path. The framework is based on a directed graph. The digraph vertices can be divided into two categories namely the "word nodes" corresponding to all state/message words and the "operator nodes" corresponding all operations. The directed edges of the digraph simply reflect the computational process from the starting state to the ending state. In this way, the digraph captures the whole computation process of ARX primitives and the approximate signed differential path given by the MILP model can be represented as a lookup table mapping all the word nodes to their signed differences. Based on the digraph and the lookup table, all bit conditions can be deduced automatically by traversing all operator nodes.

The digraph not only works for predefined signed differential paths, but also is useful in MILP model constructing process for deducing new paths. In this case, the lookup table maps word nodes to MILP model variable vectors representing their signed differences. For each operator nodes representing modular add operations, we introduce into the MILP model additional parameters tracing the newly generated bit conditions. With such parameters, a proper MILP objective function can be defined to find optimal signed differential paths. The digraph can also be used to introduce particular boomerang constraints in the intersection of top and bottom signed differential paths so as to guarantee a compatible connection.

With the approximate modeling and the digraph based techniques, we propose new boomerang attacks on BLAKE3, BLAKE2s and BLAKE-256 which are all members of BLAKE [65]: an ARX hash function family that enters the finalist of the SHA-3 competition [66], acts as the building block of the Password Hashing Competition winner Argon2 [67], and is widely implemented in standard software libraries such as OpenSSL[3], GNU Coreutils[4] etc. For BLAKE3, we find the 1st third-party boomerang attacks mounting to full 7-round BLAKE3 with complexities as low as 2^{180}. For BLAKE-256 and BLAKE2s, we are able to find new top and bottom paths with higher probabilities resulting in new boomerang results mounting to 8-round BLAKE2s and BLAKE-256 with the same complexity of 2^{182}. As can be seen from Table 1, we improve the previous best BLAKE2s boomerang result by 0.5 rounds and lower the complexities of BLAKE-256 results. All such results are **Type I** boomerang distinguishers (we will explain in Section 2.2) targeting at the underlying keyed permutations and we guarantee the compatibility of our top-bottom paths by providing practically found quartets. They do not threat the security of the hash functions in practice.

Table 1: Summary of boomerang results on BLAKE-256, BLAKE2s and BLAKE3.

Hash Function	#Full Rounds	Target	#Rounds	Time	Source
BLAKE-256	14	CF	6	2^{102}	[16]
		CF	6.5*	2^{184}	[16]
		CF	7*	2^{232}	[16]
		KP	6	$2^{11.75}$	[16]
		KP	7*	2^{122}	[16]
		KP	8*	2^{242}	[16]
		KP	7	$2^{37\star}$	[17]
		KP	8	2^{200}	[17]
		KP	8	2^{198}	[18]
		KP	**8**	$\mathbf{2^{182}}$	**Section 5.2**
BLAKE2s	10	KP	7.5	2^{184}	[18]
		KP	**8**†	$\mathbf{2^{230}}$	**Section 5.2**
		KP	**8**	$\mathbf{2^{182}}$	**Section 5.2**
BLAKE3	7	**KP**	**7**	$\mathbf{2^{180}}$	**Section 5.1**

KP: Keyed Permutation. CF: Compression Function.
∗: There are some incompatible problems in their attacks according to [27]
⋆: The complexity is of Type III boomerang while others are of Type I.
†: Using the same local-collision-construction strategy as [18].

Outline. Section 2 provides all the background knowledge for understanding this paper. Section 3 describes our approximate MILP modeling technique capturing the propagation rules of signed difference. Section 4 describes the digraph capturing the whole computation process of ARX primitives along with its applications in bit condition deductions (Section 4.3), objective function definitions (Section 4.4) and boomerang constraint introductions (Section 4.5). In Section 5,

[3] https://www.openssl.org/
[4] https://www.gnu.org/software/coreutils/manual/coreutils.html

we apply of our techniques to BLAKE3, BLAKE2s and BLAKE-256 for new boomerang attack results. Section 6 conclude the whole paper.

2 Preliminary

In this part, we provide necessary background knowledge. The following notations have to be introduced first.

ω	the word length of ARX primitives which is usually set to 32 or 64.
$\leftarrow$	variable assignment.
$+$	modular 2^ω addition (according to the word length).
$-$	modular 2^ω subtraction (according to the word length).
$\oplus$	bitwise exclusive or.
$\lll n$	cyclic shift n bits towards the most significant bit.
$\ggg n$	cyclic shift n bits towards the least significant bit.
$\wedge$	bitwise AND operation for words.

The state blocks of multiple ω-bit words are represented with capital letters and each ω-bit word entry is denoted as the corresponding small letters. For example, a n-word state can be represented $V = (v_0, \ldots, v_{n-1})$ where $v_0, \ldots, v_{n-1}$ are ω-bit words. The i-th bit of word v can further be represented as $v[i]$ $(i = 0, \ldots, \omega - 1)$ where v_0 is the LSB and v_{w-1} is the MSB. Since both the ordinary XOR difference and the signed difference are used in this paper, we explicitly represent the two with different notations: for a word pair (x, x'), its XOR difference is denoted as $\Delta x = x \oplus x'$ while the signed difference is denoted as ∇x and detailed in Section 2.1.

2.1 Signed Differences and Bit Conditions

Given a pair of words (x, x'), the XOR difference $\Delta x = x \oplus x'$ can be represented numerically as a subset of $[0, \omega - 1]$ containing the indices of active bits. For example, when $(x, x') = (\texttt{0x1}, \texttt{0x4})$, the XOR difference is commonly defined as $\Delta x = x \oplus x' = \texttt{0x5}$ and can be represented as $\Delta x : \{0, 2\}$ numerically. The signed difference ∇x in Wang *et al.* 's work [45,46] is simply adding an additional sign value "+" or "-" to each active bit of Δx corresponding to the active bit pair values $(0, 1)$ and $(1, 0)$ respectively. So the (x, x') above has signed difference $\nabla x : \{-0, 5\}$. Wang *et al.* also propose the concept of bit conditions which are simply introduced to guarantee particular signed differential propagations in non-linear operations which is simply modular add in ARX primitives. For example, for the $f : (\mathbb{F}_2^\omega)^3 \to \mathbb{F}_2^\omega$ function defined in Eq. (1)

$$z = f(w_1, w_2, x_2) = (w_1 \oplus w_2) + x_2 : \begin{cases} x_1 = w_1 \oplus w_2 \\ z = x_1 + x_2 \end{cases} \tag{1}$$

the propagation $\nabla_{in} \xrightarrow{f} \nabla_{out}$ in Eq. (2)

$$\nabla_{in} = \begin{cases} \nabla w_1 : \{1\} \\ \nabla w_2 : \{1, -3\} \\ \nabla x_2 : \{3, 5\} \end{cases} \xrightarrow{f} \nabla_{out} = \{\nabla z : \{5\} \tag{2}$$

$$w_1[3] = 0, z[5] = x_2[5] (= 0) \tag{3}$$

can only happen when the two bit conditions in Eq. (3) are satisfied so the probability of Eq. (2) is evaluated as 2^{-2}. The conditions in Eq. (3) are introduced due to the modular add operations. ∇w_1 and ∇w_2 guarantee the XOR difference $\Delta x_1 : \{3\}$ but the modular add operation further restrict the signed difference to $\nabla x_1 : \{-3\}$ resulting in the bit condition on $w_1[3]$; the output signed difference further add restriction to the bit $z[5]$. One may also find that $x_1[1] = x_1'[1] = 0$ result from the XOR operation but such a bit condition is satisfied by its nature. Therefore, when evaluating the signed differential path probabilities, we only need to focus on the non-MSB[5] bit conditions generated by modular add operations. Wang *et al.* also propose the modification technique for further improve the signed differential path probability. When the condition $z[5] = x_2[5]$ does not hold, the message modification technique [45,46] might be used to flip $w_1[5]$ (or $w_2[5]$) so as to get the bit condition $z[5] = x_2[5]$ fixed and increase the propagation probability in Eq. (2) to 2^{-1}.

According to Eq. (3), the bit condition $w_1[3] = 0$ is imposed on an inactive bit while $z[5] = x_2[5]$ is on an active bit. In order to reflect the bit condition and activity simultaneously, a 1-bit condition symbolic system was introduced [47]. The signed difference entry $\nabla x[i]$ is assigned to symbols such as $\{\mathtt{u}, \mathtt{n}, \mathtt{0}, \mathtt{1}, =\}$ in Table 2 according to the $(x[i], x'[i])$ values. $\nabla x[i]$ is also encoded as 2 binary variables $\nabla x[i].\mathtt{sign}, \nabla x[i].\mathtt{xdiff}$ by Marc Stevens in his HashClash[6]: for the bit pair $(x[i], x'[i])$, the binary variables of $\nabla x[i]$ are defined as:

$$\nabla x[i]. \begin{cases} \mathtt{xdiff} = x[i] \oplus x'[i] \\ \mathtt{sign} = x'[i] \end{cases} i = 0, \ldots, \omega - 1 \tag{4}$$

We use both the 1-bit symbolic system and the the binary-variable encoding in this paper so we list them uniformly in Table 2.

Table 2: The binary-variable encoding and symbolic representation of $\nabla x[i]$

$\nabla x[i]$	$(\mathtt{sign}, \mathtt{xdiff})$	$(x[i], x'[i])$
$\mathtt{n}$	$(1, 1)$	$(0, 1)$
$\mathtt{u}$	$(0, 1)$	$(1, 0)$
$\mathtt{0}$	$(0, 0)$	$(0, 0)$
$\mathtt{1}$	$(1, 0)$	$(1, 1)$
$=$	$\{(1, 0), (0, 0)\}$	$\{(1, 1), (0, 0)\}$

2.2 Boomerang Attacks

The keyed permutation of hash functions can be regarded as $E : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \to \mathbb{F}_2^n$ as $C = E(P, M)$ and is the building block of the compression function (CF) of MD-structural hash functions as $CF(P, M) = E(P, M) + P$: P is the initial internal state and M is the message block. The boomerang attacks on the keyed permutation of hash functions are often applied in the known-related-key setting [19] where the adversary can start from arbitrary intermediate state X since X

[5] MSB differences $\pm(\omega - 1)$ are equivalent for modular adds and cannot cause carries so no MSB bit condition is required for the modular add [45,46].

[6] https://marc-stevens.nl/p/hashclash/

and M can be chosen randomly [19,27]. After decomposing the target function E into two parts $E = E_0 \circ E_1$, the top and bottom paths are defined as:

$$\Delta^t P \xleftarrow{E_0^{-1}} (\Delta^t X, \Delta^t M) \xleftrightarrow{\text{Compatible}} (\Delta^b X, \Delta^b M) \xrightarrow{E_1} \Delta^b C, \tag{5}$$

where the top path $(\Delta^t X, \Delta^t M) \xrightarrow{E_0^{-1}} \Delta^t P$ holds with probability p, and the bottom path $(\Delta^b X, \Delta^b M) \xrightarrow{E_1} \Delta^b C$ with probability q. The 0-AND constraint for compatible intersection can be represented as $\Delta^t X \bigwedge \Delta^b X = 0$. Finally, we can launch the known-related-key boomerang attack with these top-bottom paths as follows:

1. Choose randomly an intermediate state (X_1, M_1) and compute $(X_i, K_i), i = 2, 3, 4$ by $X_3 = X_1 \oplus \Delta^t X$, $X_2 = X_1 \oplus \Delta^b X$, $X_4 = X_2 \oplus \Delta^t X$, and $M_3 = M_1 \oplus \Delta^t M$, $M_2 = M_1 \oplus \Delta^b M$, $M_4 = M_2 \oplus \Delta^t M$.
2. Compute backward from (X_i, M_i) and obtain P_i by $P_i = E_0^{-1}(X_i, M_i)$ $(i = 1, 2, 3, 4)$.
3. Compute forward from (X_i, M_i) and obtain C_i by $C_i = E_1(X_i, M_i)$ $(i = 1, 2, 3, 4)$.
4. Check whether $P_1 \oplus P_3 = P_2 \oplus P_4 = \Delta^t P$ and $C_1 \oplus C_2 = C_3 \oplus C_4 = \Delta^b C$.

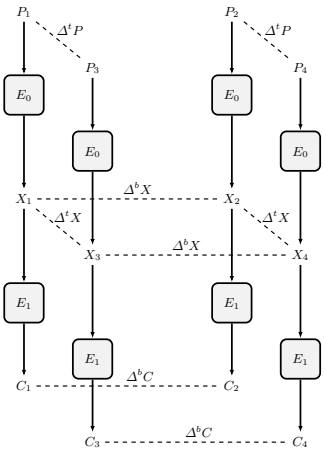

Fig. 1: Boomerang distinguisher

It can be deduced that $P_1 \oplus P_3 = P_2 \oplus P_4 = \Delta^t P$ and $C_1 \oplus C_2 = C_3 \oplus C_4 = \Delta^b C$ hold with probability at least p^2 in the E_0^{-1} direction and q^2 in the E_1 direction. Therefore, the attack succeeds with probability $p^2 q^2$ when assuming that the differential characteristics are independent. According to [28], for an n-bit random permutation, three types of boomerang distinguishers are:

Type I: A quartet satisfies $P_1 \oplus P_3 = P_2 \oplus P_4 = \Delta^t P$ and $C_1 \oplus C_2 = C_3 \oplus C_4 = \Delta^b C$ for fixed differences $\Delta^t P$ and $\Delta^b C$. In this case, the generic complexity is 2^n.

Type II: Only $C_1 \oplus C_2 = C_3 \oplus C_4$ is satisfied (This property is also called zero-sum or second-order differential collision). In this case, the complexity for obtaining such a quartet is $2^{n/3}$ [68].

Type III: A quartet satisfies $P_1 \oplus P_2 = P_3 \oplus P_4$ and $C_1 \oplus C_3 = C_2 \oplus C_4$. In this case, the best known still takes time $2^{n/2}$.

We only study **Type I** boomerang distinguisher in this paper.

2.3 MILP Modeling Technique

The MILP modeling technique has long been used in the realm of cryptanalysis. It has good performance in fields such as finding differential/linear/integral characteristics of block ciphers, giving cube attacks on stream ciphers, and constructing all kinds of distinguishers on hash functions etc.

The MILP modeling technique constructs an MILP model $\mathcal{M}$ consisting of binary variables $\mathcal{M}.\texttt{var}$, linear constraints $\mathcal{M}.\texttt{con}$ and an objective function $\mathcal{M}.\texttt{obj}$. When deducing differential characteristics, the difference on each bit $x[i]$ (or the truncated difference on some word) are represented as a binary variable as $\mathcal{M}.\texttt{var} \leftarrow x[i]$ as binary.

The MILP model of widely used operations has already been defined and widely used in existing MILP-aided cryptanalysis results. For example, for $x, y, z \in \mathcal{F}_2$, the relationship $z = x \oplus y$ can be captured with the MILP model $\mathcal{M}$ generated by Algorithm 26 in Supp. Mat. E as $(\mathcal{M}, y) \leftarrow \texttt{xorModel}(\mathcal{M}, \{x_1, \ldots, x_n\})$. Another example is the active symbol: for a set of 0-1 variable vector $\{x_1, \ldots, x_n\}$, the 0-1 variable y is of value 0 when $x_1 = \ldots = x_n = 0$ and 1 otherwise. The relationship between y and $\{x, \ldots, x_n\}$ can be described as $y = x_1 \vee \ldots \vee x_n$ captured with the MILP model $(\mathcal{M}, y) \leftarrow \texttt{orModel}(\mathcal{M}, \{x_1, \ldots, x_n\})$ defined in Algorithm 27 in Supp. Mat. E.

2.4 Keyed Permutations of BLAKE3, BLAKE2s and BLAKE-256

BLAKE3, BLAKE2s and BLAKE-256 share many similarities. The word length are all set to $\omega = 32$. They all process 16-word message blocks $M = (m_0, \ldots, m_{15})$. The internal states also contains 16 words represented by a 4×4 matrix as follows:

$$V = \begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix}.$$

Once the state V is initialized, it is processed by a sequence of a round function (7, 10, 14 for BLAKE3, BLAKE2s, BLAKE-256 respectively), where one round of G does the following:

$$G_0(v_0, v_4, v_8, v_{12}), G_1(v_1, v_5, v_9, v_{13}), G_2(v_2, v_6, v_{10}, v_{14}), G_3(v_3, v_7, v_{11}, v_{15})$$
$$G_4(v_0, v_5, v_{10}, v_{15}), G_5(v_1, v_6, v_{11}, v_{12}), G_6(v_2, v_7, v_8, v_{13}), G_7(v_3, v_4, v_9, v_{14}).$$

In the r-th round function, $G_i(a, b, c, d), i = 0, \cdots, 7$ differs slightly among BLAKE3, BLAKE2s and BLAKE-256 as listed in Table 3. BLAKE2s and BLAKE-256 share the same word permutation $\sigma_r(\cdot)$ while BLAKE3 uses a different one. The word rc's used in BLAKE-256 are round constants selected from a 16-word

Table 3: The G_i Functions of BLAKE-256, BLAKE2s and BLAKE3

Step	BLAKE-256	BLAKE3/2s
1	$a = a + b + (m_{\sigma_r(2i)} \oplus rc_{\sigma_r(2i+1)})$	$a = a + b + m_{\sigma_r(2i)}$
2	$d = (d \oplus a) \ggg 16$	$d = (d \oplus a) \ggg 16$
3	$c = c + d$	$c = c + d$
4	$b = (b \oplus c) \ggg 12$	$b = (b \oplus c) \ggg 12$
5	$a = a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)})$	$a = a + b + m_{\sigma_r(2i+1)}$
6	$d = (d \oplus a) \ggg 8$	$d = (d \oplus a) \ggg 8$
7	$c = c + d$	$c = c + d$
8	$b = (b \oplus c) \ggg 7$	$b = (b \oplus c) \ggg 7$

constant state $RC = (rc_0, \ldots, rc_{15})$. Further details can be seen in the specifications [65].

Since we need detailed analysis of the intermediate states, we further breakdown the round functions. We denote the state after r rounds of iterations by V^r $(r = 0, 1, \cdots)$. Then, TV^r is acquired after the first 4 steps of $G_{0,\cdots,3}$ and $V^{r+0.5}$ is computed after $G_{0,\cdots,3}$ are completed. Similarly, we can compute $TV^{r+0.5}$ from $V^{r+0.5}$ by applying steps 1,2,3,4 of $G_{4,\cdots,7}$ and further compute V^{r+1} by finishing $G_{4,\cdots,7}$. This representation is illustrated as Eq. (6) and Eq. (7).

$$G_{0,\cdots,3} : V^r \xrightarrow{\text{Step } 1,\cdots,4} TV^r \xrightarrow{\text{Step } 5,\cdots,8} V^{r+0.5} \tag{6}$$

$$G_{4,\cdots,7} : V^{r+0.5} \xrightarrow{\text{Step } 1,\cdots,4} TV^{r+0.5} \xrightarrow{\text{Step } 5,\cdots,8} V^{r+1} \tag{7}$$

In this way, we can refer to any intermediate state word of any round easily.

To further simplify the interpretation, we further define the following function $hG : (\mathbb{F}_2^\omega)^5 \times (\mathbb{Z}_+)^2 \to (\mathbb{F}_2^\omega)^4$:

$$(x_0, \ldots, x_3, m, \alpha, \beta) \xrightarrow{hG} (y_0, \ldots, y_3) \text{ where } \begin{cases} y_0 = x_0 + x_1 + m \\ y_3 = (y_0 \oplus x_3) \ggg \alpha \\ y_2 = x_2 + y_3 \\ y_1 = (x_1 \oplus y_2) \ggg \beta \end{cases} \tag{8}$$

In this way, each G_i call is equivalent to 2 consecutive calls of $hG(\star, 16, 12)$ and $hG(\star, 8, 7)$; each $V^r \to TV^r$ $(TV^r \to V^{r+0.5})$ transformation can be decomposed to 4 parallel $hG(\star, 16, 12)$ $(hG(\star, 8, 7))$ calls in parallel.

3 MILP Modeling the Signed Difference Propagation in ARX Primitives

The model $\mathcal{M}$ can be constructed in an **accurate** or an **approximate** manner. In an accurate model, the signed difference is $\nabla x[i] \in \{\mathtt{u}, \mathtt{n}, \mathtt{0}, \mathtt{1}\}$ while in an approximate model, $\nabla x[i] \in \{\mathtt{u}, \mathtt{n}, =\}$. As can be seen, the $\nabla x[i]$ values in the accurate model is of 1-1 correspondence to the (x, x') pairs. Therefore, solving an accurate model is equivalent to finding the right pair corresponding to the signed difference which is computationally infeasible in most cases. Therefore, in this section, we focus on the approximate signed difference propagation rules for

modular add (Section 3.1) and XOR operations (Section 3.2) from both theoretic and MILP modeling aspects. Details of the accurate model are moved to Supp. Mat. C.

In the approximate model, there is $\nabla x[i] \in \{\mathtt{u}, \mathtt{n}, =\}$. With the binary-variable encoding in Table 2, we can let $\nabla x[i].\mathtt{xdiff}, \nabla x[i].\mathtt{sign} \in \mathcal{M}.\mathtt{var}$ and represent the bit conditions with MILP model constraints as Eq. (9).

$$\begin{aligned}
&\mathcal{M}.\mathtt{con} \leftarrow \nabla x[i] \leftarrow \mathtt{n} \Leftrightarrow \nabla x[i].(\mathtt{sign}, \mathtt{xdiff}) = (1, 1) \\
&\mathcal{M}.\mathtt{con} \leftarrow \nabla x[i] \leftarrow \mathtt{u} \Leftrightarrow \nabla x[i].(\mathtt{sign}, \mathtt{xdiff}) = (0, 1) \\
&\mathcal{M}.\mathtt{con} \leftarrow \nabla x[i] \leftarrow = \Leftrightarrow \nabla x[i].(\mathtt{sign}, \mathtt{xdiff}) = (0, 0)
\end{aligned} \tag{9}$$

Algorithm 1: `halfAdderApprox`

Input : MILP model $\mathcal{M}$, encoded signed differences $\nabla x[0], \nabla y[0]$ whose `sign` and `xdiff`'s are binary variables in $\mathcal{M}.\mathtt{var}$

Output: The updated MILP model $\mathcal{M}$ and the encoded $\nabla z[0], \nabla c[0]$ whose `sign` and `xdiff`'s are binary variables in $\mathcal{M}.\mathtt{var}$

// Declare variables

1 $\mathcal{M}.\mathtt{var} \leftarrow \nabla z[0].\mathtt{sign}, \nabla z[0].\mathtt{xdiff}, \nabla c[0].\mathtt{sign}, \nabla c[0].\mathtt{xdiff}$ as binaries

// Define the column vector

2 $\boldsymbol{x} = (\nabla x[0].\mathtt{sign}, \nabla x[0].\mathtt{xdiff}, \nabla y[0].\mathtt{sign}, \nabla y[0].\mathtt{xdiff}, \nabla z[0].\mathtt{sign}, \nabla z[0].\mathtt{xdiff}, \nabla c[0].\mathtt{sign}, \nabla c[0].\mathtt{xdiff})^T$

// Add constraints.

// Matrix A_h is defined as Eq. (11)

3 $\mathcal{M}.\mathtt{con} \leftarrow A_h \boldsymbol{x} \geq \mathbf{0}$

4 **Return** $(\mathcal{M}, \nabla z[0], \nabla c[0])$

Algorithm 2: `fullAdderApprox`

Input : MILP model $\mathcal{M}$, encoded signed differences $\nabla x[i], \nabla y[i], \nabla c[i-1]$ whose `sign` and `xdiff`'s are binary variables in $\mathcal{M}.\mathtt{var}$

Output: The updated MILP model $\mathcal{M}$ and the encoded $\nabla z[i], \nabla c[i]$ whose `sign` and `xdiff`'s are binary variables in $\mathcal{M}.\mathtt{var}$

// Declare variables

1 $\mathcal{M}.\mathtt{var} \leftarrow \nabla z[i].\mathtt{sign}, \nabla z[i].\mathtt{xdiff}, \nabla c[i].\mathtt{sign}, \nabla c[i].\mathtt{xdiff}$ as binaries

// Define the column vector

2 $\boldsymbol{x} = (\nabla x[i].\mathtt{sign}, \nabla x[i].\mathtt{xdiff}, \nabla y[i].\mathtt{sign}, \nabla y[i].\mathtt{xdiff}, \nabla c[i-1].\mathtt{sign}, \nabla c[i-1].\mathtt{xdiff}, \nabla z[i].\mathtt{sign}, \nabla z[i].\mathtt{xdiff}, \nabla c[i].\mathtt{sign}, \nabla c[i].\mathtt{xdiff})^T$

// Add constraints.

// Matrix A_f and column vector $\boldsymbol{b}$ are defined as Eq. (13)

3 $\mathcal{M}.\mathtt{con} \leftarrow A_f \boldsymbol{x} + \boldsymbol{b} \geq \mathbf{0}$

4 **Return** $(\mathcal{M}, \nabla z[i], \nabla c[i])$

3.1 The Modular Add Operation

Consider the modular add operation $z = x + y$. We analyze the cases at bit position $i = 0, \ldots, \omega - 1$.

For the LSB $i = 0$, the input bits $(x[0], y[0])$ generate the LSB of output $z[0]$ along with a carry bit $c[0]$ where:

$$\begin{cases} z[0] = x[0] \oplus y[0] \\ c[0] = x[0] \wedge y[0] \end{cases} \tag{10}$$

The procedure in Eq. (10) corresponds to functionality of the half-adder which is a basic hardware circuit. We traverse all possible values of $(x[0], x'[0], y[0], y'[0])$ and acquire all the available combinations of $(\nabla x[0], \nabla y[0], \nabla z[0], \nabla c[0])$ represented in both accurate and approximate manners in Table 4.

Table 4: The signed differences $(\nabla x[0], \nabla y[0], \nabla z[0], \nabla c[0])$ of the half-adder

Model	$(\nabla x[0], \nabla y[0], \nabla z[0], \nabla c[0])$
Accurate	`0000`, `n0n0`, `u0u0`, `1010`, `0nn0`, `0uu0`, `0110`, `nn0n`, `un10`, `1nun`, `nu10`, `uu0u`, `1unu`, `n1un`, `u1nu`, `1101`
Approximate	`====`, `n=n=`, `u=u=`, `=nn=`, `=uu=`, `nn=n`, `un==`, `=nun`, `nu==`, `uu=u`, `=unu`, `n=un`, `u=nu`

With the signed difference encoding technique in Section 2.1, we are able to describe the approximate signed differential propagation rule of the half-adder in Eq. (10) with MILP models by calling $(\mathcal{M}, \nabla z[0], \nabla c[0]) \leftarrow$ `halfAdderApprox`$(\mathcal{M}, \nabla x[0], \nabla y[0])$ in Algorithm 1. With the help of H-representation method [62], the matrix A_h is deduced as:

$$A_h = \begin{pmatrix} 2 & -1 & 2 & -1 & -2 & 1 & -4 & 2 \\ -2 & 1 & -2 & 1 & 2 & -1 & 4 & -2 \\ 0 & 1 & 0 & 1 & -2 & 1 & -2 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & -1 & -2 & 1 & -4 & 2 \end{pmatrix}. \tag{11}$$

For $i = 1, \ldots, \omega-1$, the input bits $(x[i], y[i], c[i-1])$ and the outputs $(z[i], c[i])$, corresponding to the full-adder circuit, are computed as:

$$\begin{cases} z[i] = x[i] \oplus y[i] \oplus c[i-1] \\ c[i] = (x[i] \wedge y[i]) \vee (x[i] \wedge c[i-1]) \vee (y[i] \wedge c[i-1]) \end{cases} \tag{12}$$

Traversing all possible values of $(x[i], x'[i], y[i], y'[i], c[i-1], c'[i-1])$, we acquire all available combinations of $(\nabla x[i], \nabla y[i], \nabla c[i-1], \nabla z[i], \nabla c[i])$ represented in both accurate and approximate manners in Table 5.

Table 5: The signed differences $(\nabla x[0], \nabla y[0], \nabla z[0], \nabla c[0])$ of the full-adder

Model	$(\nabla x[i], \nabla y[i], \nabla c[i-1], \nabla z[i], \nabla c[i])$
Accurate	`00000`, `n00n0`, `u00u0`, `10010`, `0n0n0`, `nn00n`, `un010`, `1n0un`, `0u0u0`, `nu010`, `uu00u`, `1u0nu`, `01010`, `n10un`, `u10nu`, `11001`, `00nn0`, `n0n0n`, `u0n10`, `10nun`, `0nn0n`, `nnnnn`, `unnun`, `1nn1n`, `0un10`, `nunun`, `uunnu`, `1un01`, `01nun`, `n1n1n`, `u1n01`, `11nn1`, `00uu0`, `n0u10`, `u0u0u`, `10unu`, `0nu10`, `nnuun`, `ununu`, `1nu01`, `0uu0u`, `nuunu`, `uuuuu`, `1uu1u`, `01unu`, `n1u01`, `u1u1u`, `11uu1`, `00110`, `n01un`, `u01nu`, `10101`, `0n1un`, `nn11n`, `un101`, `1n1n1`, `0u1nu`, `nu101`, `uu11u`, `1u1u1`, `01101`, `n11n1`, `u11u1`, `11111`
Approximate	`=====`, `n==n=`, `u==u=`, `=n=n=`, `nn==n`, `un===`, `=n=un`, `=u=u=`, `nu===`, `uu==u`, `=u=nu`, `n==un`, `u==nu`, `==nn=`, `n=n=n`, `u=n==`, `==nun`, `=nn=n`, `nnnnn`, `unnun`, `=un==`, `nunun`, `uunnu`, `==uu=`, `n=u==`, `u=u=u`, `==unu`, `=nu==`, `nnuun`, `ununu`, `=uu=u`, `nuunu`, `uuuuu`

Similar to the half-adder case, the approximate signed differential propagation rule of the full-adder in Eq. (12) can also be described with MILP models by calling $(\mathcal{M}, \nabla z[i], \nabla c[i]) \leftarrow \texttt{fullAdderApprox}(\mathcal{M}, \nabla x[i], \nabla y[i], \nabla c[i-1])$ in Algorithm 2, where the matrix A_f and the column vector $\boldsymbol{b}$ are defined as

$$A_f = \begin{pmatrix} 2 & -1 & 2 & -1 & 2 & -1 & -2 & 1 & -4 & 2 \\ -2 & 1 & -2 & 1 & -2 & 1 & 2 & -1 & 4 & -2 \\ 0 & 1 & 0 & 1 & 0 & 1 & -2 & 1 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & -1 & 2 & -1 & -2 & 1 & -4 & 2 \\ 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 2 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \end{pmatrix}. \tag{13}$$

The approximate signed differential propagation rule of modular 2^ω add can be described with MILP models by calling $(\mathcal{M}, \nabla z, \nabla c) \leftarrow \texttt{modAddApprox}(\mathcal{M}, \nabla x, \nabla y)$ defined as Algorithm 3.

Algorithm 3: `modAddApprox`

Input : MILP model $\mathcal{M}$, the word signed differences ∇x and ∇y.
Output: The updated MILP model $\mathcal{M}$, the signed differences $\nabla z, \nabla c$.

$(\mathcal{M}, \nabla z[0], \nabla c[0]) \leftarrow \texttt{halfAdderApprox}(\mathcal{M}, \nabla x[0], \nabla y[0])$
for $i = 1, \ldots, \omega - 1$ **do**
    $(\mathcal{M}, \nabla z[i], \nabla c[i]) \leftarrow \texttt{fullAdderApprox}(\mathcal{M}, \nabla x[i], \nabla y[i], \nabla c[i-1])$
Return $(\mathcal{M}, \nabla z, \nabla c)$

3.2 The XOR Operation

We consider the XOR operation as $z = x \oplus y$. In the accurate model, it can constantly be modeled as

$$\begin{cases} \nabla z[i].\texttt{sign} = \nabla x[i].\texttt{sign} \oplus \nabla y[i].\texttt{sign} \\ \nabla z[i].\texttt{xdiff} = \nabla x[i].\texttt{xdiff} \oplus \nabla y[i].\texttt{xdiff} \end{cases} \text{for } i = 0, \ldots, \omega - 1.$$

In the approximate model, the linear constraint on the `xdiff` still holds while the `sign` only satisfies that $\texttt{sign} \leq \texttt{xdiff}$. Therefore, the approximate signed difference propagation rule can be described as $(\mathcal{M}, \nabla z) \leftarrow \texttt{xorApprox}(\mathcal{M}, \nabla x, \nabla y)$ defined in Algorithm 4. Here, `xorModel` is the MILP description of XOR operation of two words, and is provided as Algorithm 26 in Supp. Mat. E.

For the special case of $z = x \oplus c$ where c is a known constant word, we may find in approximate model that $\nabla z[i] = \nabla x[i]$ when $c[i] = 0$ and

$$\begin{cases} \nabla z[i].\texttt{xdiff} = \nabla x[i].\texttt{xdiff} \\ \nabla z[i].\texttt{sign} = \nabla x[i].\texttt{xdiff} \oplus \nabla x[i].\texttt{sign} \end{cases}$$

Algorithm 4: `xorApprox`

Input : MILP model $\mathcal{M}$, the word signed differences ∇x and ∇y
Output: The updated MILP model $\mathcal{M}$, the signed difference ∇z

for $i = 0, \ldots, \omega - 1$ **do**
  $(\mathcal{M}, \nabla z[i].\texttt{xdiff}) \leftarrow \texttt{xorModel}(\mathcal{M}, \nabla x[i].\texttt{xdiff}, \nabla y[i].\texttt{xdiff})$
  Declare variable $\mathcal{M}.\texttt{var} \leftarrow z[i].\texttt{sign}$
  Add constraint $\mathcal{M}.\texttt{con} \leftarrow z[i].\texttt{sign} \leq z[i].\texttt{xdiff}$.
Return $(\mathcal{M}, \nabla z)$

when $c[i] = 1$. Therefore, the approximate signed difference propagation rule for XORing a constant can be described as $(\mathcal{M}, \nabla z) \leftarrow \texttt{xorConstApprox}(\mathcal{M}, \nabla x, c)$ defined in Algorithm 5.

Algorithm 5: `xorConstApprox`

Input : MILP model $\mathcal{M}$, the word signed difference ∇x and constant word c
Output: The updated MILP model $\mathcal{M}$, the signed difference ∇z

for $i = 0, \ldots, \omega - 1$ **do**
  if $c[i] = 0$ **then**
    Assign $\nabla z[i] \leftarrow \nabla x[i]$
  else
    $(\mathcal{M}, \nabla z[i].\texttt{sign}) \leftarrow \texttt{xorModel}(\mathcal{M}, \nabla x[i].\texttt{xdiff}, \nabla x[i].\texttt{sign})$
    Assign $z[i].\texttt{xdiff} \leftarrow x[i].\texttt{xdiff}$
Return $(\mathcal{M}, \nabla z)$

4 Digraph Capturing the ARX Computations and Representing Bit Conditions

The signed differential path deduced from the approximate model only contains bit conditions $\{\texttt{u}, \texttt{n}, =\}$ so the word difference ∇x deduced with such a model can only have accurate information on active bits. In some cases, the approximate signed differential paths are actually infeasible because of contradicted bit conditions on inactive bits. Therefore, we should deduce the bit conditions based on the approximate signed differential path, which, according to Section 2.1, requires not only the knowledge of signed differences of all intermediate state words but the operations among words as well.

As our solution, a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is constructed to capture the relationships of words and operations for the ARX primitive computations, where $\mathcal{V}$ is the vertex set and $\mathcal{E}$ is the set of all directed edges. The vertex set $\mathcal{V}$ contains two kind of vertices:

1. **word nodes**: denoted as $\mathcal{V}_w$, correspond to all intermediate state words;
2. **operator nodes**: denoted as $\mathcal{V}_o$, correspond to all operations.

4.1 Structure of Operator Node $\mathcal{V}_o$

In ARX primitives, there are three kinds operations namely modular add, XOR and rotations. Among them, rotations merely change the positions of bit rather than changing any bit conditions or differences, therefore the main focus remains as two operator node types, namely ADD corresponding to modular add operations, and XOR corresponding to XOR-based operations. The effect of the rotation is captured with a integer `rotBit` value capturing the rotation after the implementation of modular adds or XORs: `rotBit` > 0 for rotating from MSB to LSB and `rotBit` < 0 otherwise. Besides, the operator node also has a ω-bit word named `const` setting to the constant parameter involved in the modular addition or XORs. When no constant is involved, the `const` word is set to 0 by default. The in-edges of each operator node is connected to all word nodes representing all the intermediate state words involved in the operation. The word nodes connect with in-edges are stored in a set `iWords` $\subseteq \mathcal{V}_w$. It has a unique out-edge connecting to the word node corresponding to the output intermediate state word of the operation. The word node connected with the out-edge is therefore defined as `oWord` $\in \mathcal{V}_w$. To sum up, an operator node $\boldsymbol{op} \in \mathcal{V}_o$ is of the following data structures:

`type` $\in \{ADD, XOR\}$**:** the type of the operations.
`rotBit` $\in \mathbb{Z}$**:** the rotation after the implementation.
`const` $\in \mathbb{F}_2^\omega$**:** the constant parameter involved in the operation (default is 0).
`iWords` $\subseteq \mathcal{V}_w$**:** the word nodes connected with in-edges corresponding to all the words involved in the operation.
`oWord` $\in \mathcal{V}_w$**:** the word node connect with the out-edge corresponding to the output word of the operation.

For example, the three principal operations in the round function of BLAKE are

$$y = (x_1 \oplus x_2) \ggg \alpha \, (\alpha > 0), \quad z = w_1 + w_2, \quad m' = m \oplus rc \; (rc \in \mathbb{F}_2^\omega).$$

They can be represented by the defined operator nodes as in Table 6.

Table 6: Operator node representation for the three principal operations in BLAKE round functions.

Structure \ Operation	$y = (x_1 \oplus x_2) \ggg \alpha$	$z = w_1 + w_2$	$m' = m \oplus rc$
`type`	XOR	ADD	XOR
`rotBit`	α	0	0
`const`	0	0	rc
`iWords`	$\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$	$\{\boldsymbol{w}_1, \boldsymbol{w}_2\}$	$\{\boldsymbol{m}\}$
`oWord`	$\boldsymbol{y}$	$\boldsymbol{z}$	$\boldsymbol{m'}$

4.2 Structure of Word Nodes $\mathcal{V}_w$

Each ω-bit word x in the ARX primitive computation corresponds to exactly one word node $\boldsymbol{x} \in \mathcal{V}_w$. The word node $\boldsymbol{x} \in \mathcal{V}_w$ has (at most) one in-edge

connecting to the operator node for computing x and such an operator node is stored in a set $\boldsymbol{x}.\texttt{parent} \subseteq \mathcal{V}_o$. When x is a word in the initial state, we naturally set $\boldsymbol{x}.\texttt{parent} = \phi$. The out-edges of $\boldsymbol{x}$ connect $\boldsymbol{x}$ to all the operator nodes corresponding to all the operations x involved. Since x is an ω-bit word, the word node $\boldsymbol{x}$ also contains ω symbolic variables namely $\boldsymbol{x}.\texttt{b}_i$ for $i = 0, \ldots, \omega - 1$ corresponding to bit $w[i]$. Such $\boldsymbol{x}.b_i$ is simply referred as the "bit node". To sum up, the word node $\boldsymbol{x} \in \mathcal{V}_w$ are classified in the following categories:

1. $\texttt{parent} \subseteq \mathcal{V}_o$: the operator node for computing the intermediate word x.
2. $\texttt{b}_0, \ldots, \texttt{b}_{\omega-1}$: the bit nodes corresponding to the bits $x[0, \ldots, \omega - 1]$.

4.3 Bit Condition Deduction

Since bit conditions are simply linear equations of bits and all intermediate state bits are represented as bit nodes, arbitrary bit equation $\boldsymbol{E}$ can now be represented as a equation with bit nodes on the left hand side and a 0-1 constant on the right hand side. So the bit equation $\boldsymbol{E}$ can be represented as the following data structure:

1. `lhs`: the set of bit nodes corresponding to the bits involved in the bit condition.
2. `rhs`: a 0-1 constant.

As can be seen in Section 2.1, the bit conditions on inactive bits are to be imposed when active and inactive bits are XORed. This can happen in two situations in ARX primitives:

1. When multiple words are XORed, all ω bits in all words take part in XOR operations.
2. When multiple words are modular added, the LSBs take part in XOR operations.

With the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the signed differential path acquired from the approximate model in Section 3 can be regarded as a lookup table $\mathcal{T}$ projecting all word nodes $\boldsymbol{w} \in \mathcal{V}_w$ to its approximate signed difference ∇w, which means $\mathcal{T}[\boldsymbol{w}] = \nabla w$ for all $\boldsymbol{w} \in \mathcal{V}_w$ and $\nabla w[i] \in \{\texttt{u}, \texttt{n}, =\}$ for $i = 0, \ldots, \omega - 1$. For an operator node $\boldsymbol{op} \in \mathcal{V}_o$ and the signed differential path $\mathcal{T}$, we are able to deduce a set of bit conditions ξ by calling Algorithm 6 as $\xi \leftarrow \texttt{optBC}(\mathcal{G}, \mathcal{T}, \boldsymbol{op})$. The ξ set containing all bit conditions can be acquired by calling Algorithm 7 as $\xi \leftarrow \texttt{allBC}(\mathcal{G}, \mathcal{T})$ which is simply calling `optBC` in Algorithm 6 for all $\boldsymbol{op} \in \mathcal{V}_o$.

4.4 Digraph based Objective Function Definition

According to the analysis in Section 2.1, the bit conditions determining the signed differential probability are all generated by the modular add operations. With the digraph $\mathcal{G}$ and the approximate differential path $\mathcal{T}$, the modular-add-generated bit conditions can be determined by simply referring to all the modular add operator nodes denoted as $\mathcal{V}_+ \subseteq \mathcal{V}_o$. We first prove the following Lemma 1.

Algorithm 6: `optBC`

Input : The digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the approximate signed differential path $\mathcal{T}$, the operator node $\boldsymbol{op} \in \mathcal{V}_o$

Output: The set ξ containing the bit conditions imposed to the inactive bits of input-output words

Let $\boldsymbol{op}.\texttt{iWords} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and $\boldsymbol{op}.\texttt{iWord} = \boldsymbol{y}$, refer to $\mathcal{T}$ for the signed differences corresponding to the input-output words namely $\nabla x_1, \ldots, \nabla x_n$ and ∇y.
Initialize the bit condition set $\xi = \phi$
Set $N \leftarrow \omega$ if $\boldsymbol{op}.\texttt{type}$ = XOR and $N \leftarrow 1$ otherwise
for $i = 0, \ldots, N-1$ **do**
    Compute offset value $\alpha_i \equiv i + \boldsymbol{op}.\texttt{rotBit} \mod \omega$.
    if $\nabla y[i] = \nabla x_1[\alpha_i] = \ldots = \nabla x_n[\alpha_i] \in \{=\}$ **then**
        continue
    else
        Initialize a bit equation $\boldsymbol{E}$ with $\boldsymbol{E}.\texttt{lhs} = \phi$ and $\boldsymbol{E}.\texttt{rhs} = \boldsymbol{op}.\texttt{const}[\alpha_i]$
        for $j = 1, \ldots, n$ **do**
            if $\nabla x_j[\alpha_i] \in \{\texttt{u}, \texttt{n}\}$ **then**
                $\boldsymbol{E}.\texttt{rhs} \leftarrow \boldsymbol{E}.\texttt{rhs} \oplus \nabla x_j[\alpha_i].\texttt{sign}$
            else
                $\boldsymbol{E}.\texttt{lhs} \leftarrow \boldsymbol{E}.\texttt{lhs} \bigcup \{\boldsymbol{x_j}.\texttt{b}_{\alpha_i}\}$
        if $\nabla y[i] \in \{\texttt{u}, \texttt{n}\}$ **then**
            $\boldsymbol{E}.\texttt{rhs} \leftarrow \boldsymbol{E}.\texttt{rhs} \oplus \nabla y[i].\texttt{sign}$
            If $\boldsymbol{E}.\texttt{lhs}$ is not empty, update $\xi \leftarrow \xi \bigcup \{\boldsymbol{E}\}$
        else
            $\boldsymbol{E}.\texttt{lhs} \leftarrow \boldsymbol{E}.\texttt{lhs} \bigcup \{\boldsymbol{y}.\texttt{b}_i\}$
            Update $\xi \leftarrow \xi \bigcup \{\boldsymbol{E}\}$
Return ξ

Algorithm 7: `allBC`

Input : The digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the approximate signed differential path $\mathcal{T}$.

Output: The set ξ containing all bit conditions imposed to the inactive bits

Initialize the bit condition set $\xi = \phi$
for $\boldsymbol{op} \in \mathcal{V}_o$ **do**
    Call Algorithm 6 as $\xi' \leftarrow \texttt{optBC}(\mathcal{G}, \mathcal{T}, \boldsymbol{op})$.
    Update $\xi \leftarrow \xi \bigcup \xi'$.
Return ξ.

Lemma 1. *For a modular add $z = x_1 + \ldots + x_n$ with input-output signed differences $\nabla x_1, \ldots, \nabla x_n, \nabla z$, we define the 0-1 variables $\theta_0, \ldots, \theta_{\omega-2}$ as*

$$\theta_i = \left(\bigvee_{j=1}^{n} \nabla x_j[i].\texttt{xdiff} \right) \vee \nabla z[i].\texttt{xdiff} \tag{14}$$

The number of newly generated bit conditions is no higher than $\beta = \sum_{i=0}^{\omega-2} \theta_i$.

Proof. We only need to consider $\nabla z[i]$ $(i = 0, \ldots, \omega - 2)$:

1. The active $z[i]$ is naturally a newly generated bit condition as $\nabla z[i] = \texttt{u/n}$ so the number new condition at the i-th bit equal to θ_i.
2. If $z[i]$ is inactive and $x_1[i], \ldots, x_n[i]$ are also inactive, no new bit condition is generated which is equal to θ_i as well.
3. If $z[i]$ is inactive and some of $x_1[i], \ldots, x_n[i]$ are active, the number of newly generated bit condition is 0 which is smaller than the $\theta_i = 1$ in this case which completes the proof. □

Lemma 1 defines a β parameter upper bounding the number of modular-add-generated bit conditions. For $z = f(w_1, w_2, x_2)$ in Eq. (1), we can further prove that the number of newly generated bit conditions in f actually equals to β.

Proposition 1. *For $z = f(w_1, w_2, x_2)$ in Eq. (1) with signed differences ∇w_1, ∇w_2, ∇x_2 and ∇z, we naturally define 0-1 variables $\theta_0, \ldots, \theta_{\omega-2}$ as Eq. (14). The number of newly generated bit conditions is equal to the parameter β defined in Lemma 1.*

Proof. The knowledge of $\nabla w_1, \nabla w_2$ directly result in the the XOR difference of x_1. Consider $\nabla z[i]$ $(i = 0, \ldots, \omega - 2)$:

1. The active $z[i]$ is naturally a newly generated bit condition, so the number new condition at the i-th bit equal to θ_i.
2. If $z[i]$ is inactive and $x_1[i], x_2[i]$ are also inactive, no new bit condition is generated which is equal to θ_i as well.
3. If $z[i]$ is inactive, $x_1[i], x_2[i]$ are both active, this indicates one newly generated bit condition on the inactive member of $(\nabla w_1[i], \nabla w_2[i])$ so as to make $\nabla x_1[i].\texttt{sign} \neq \nabla x_2[i].\texttt{sign}$ for cancellation or $\nabla x_1[i].\texttt{sign} = \nabla x_2[i].\texttt{sign}$ for carry. So the number of newly generated bit condition equals to $\theta_i = 1$ which completes the proof.

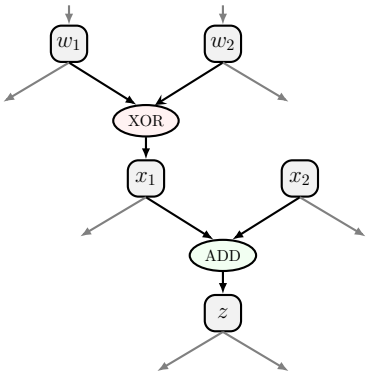

Fig. 2: The digraph for f function in Eq. (1)

Note although Proposition 1 prove that the

The Lemma 1 and Proposition 1 inspire us to set the objective function of the MILP model to minimize the parameter β for all $\boldsymbol{op} \in \mathcal{V}_+$ as Eq. (15) so as to find the signed differential paths with the fewest bit conditions.

$$\mathcal{M}.\texttt{obj} \leftarrow \min \sum_{\boldsymbol{op} \in \mathcal{V}_+} \beta \tag{15}$$

Remark. Although it is proved in Proposition 1 that the number of newly generated bit conditions are equal to β, there are special situations where some bit conditions are satisfied naturally without affecting the overall probability: since $z[0] = w_1[0] \oplus w_2[0] \oplus x_2[0]$, when all $\nabla w_1[0], \nabla w_2[0], \nabla x_2[0]$ are active, $\nabla z[0]$ is satisfied naturally. So even for f function in Eq. (1), β should still be regarded as the upper bound rather than the exact number of bit conditions.

4.5 Boomerang Intersection Constraints

For the signed differential top and bottom paths $\nabla^t X \xrightarrow{E_0^{-1}} \nabla^t P$ and $\nabla^b X \xrightarrow{E_1} \nabla^b C$ deduced separately with the approximate model in Section 3, we may construct 2 digraphs $\mathcal{G}^t$ and $\mathcal{G}^b$, and deduce the bit conditions independently. After the bit condition deduction, we may find that at some intersecting $x[i]$ bits, there are active bit condition in one side and inactive bit condition on the other. For example, there may be $\nabla^b x[i] = \texttt{n}$ and $\nabla^t x[i] = \texttt{0}$ which is obviously incompatible: since $\nabla^b x[i] = \texttt{n}$ should be satisfied in both (X_1, X_3) and (X_2, X_4) sides of the quartets, there are obviously $x_1[i] = x_3[i] = 0$ and $x_2[i] = x_4[i] = 1$ violating the $\nabla^t x[i] = \texttt{0}$ bit condition in the top. This is also a proof that the 0-AND constraint $\Delta^t X \wedge \Delta^b X = 0$ is insufficient for compatible top-bottom paths. In order to avoid such incompatibilities, we must add additional boomerang intersection constraints to the MILP model.

As can be seen in previous analysis, the bit conditions on inactive bits are to be generated in XORed bits, including all ω bits related to type-XOR operator nodes and the LSBs of type-ADD operators. For a word node $\boldsymbol{w} \in \mathcal{V}_w$ of digraph $\mathcal{G}$, its related operator nodes, corresponding to the operations taking the word w as either input or output, can be extracted by calling $\mathcal{R}_{\boldsymbol{w}} \leftarrow \texttt{relatedOpt}(\mathcal{G}, \boldsymbol{w})$ in Algorithm 8. Then, for $\mathcal{G}^t$ and $\mathcal{G}^b$ describing the computations of E_0^{-1} and E_1, and for each intersecting state word node $\boldsymbol{x}$ in either $\mathcal{G}^t$ or $\mathcal{G}^b$, we can acquire $\mathcal{R}^t_{\boldsymbol{x}} \leftarrow \texttt{relatedOpt}(\mathcal{G}^t, \boldsymbol{x})$ and $\mathcal{R}^b_{\boldsymbol{x}} \leftarrow \texttt{relatedOpt}(\mathcal{G}^b, \boldsymbol{x})$ containing all the $\boldsymbol{x}$-related operator nodes in top and bottom paths. For an operator in $\boldsymbol{op} \in \mathcal{R}^t_{\boldsymbol{x}}$, we can deduce that some $x[i]$ bit is XORed with other bits denoted as $w_1[i_1], \ldots, w_n[i_n]$ by the operation corresponding to $\boldsymbol{op}$ in E_0^{-1}. In order to avoid incompatibilities, we can add the constraints in Eq. (16) to the MILP model $\mathcal{M}$ when deducing signed differential paths:

$$\begin{cases} (\mathcal{M}, a) \leftarrow \texttt{orModel}(\mathcal{M}, \{\nabla^t x[i].\texttt{xdiff}, \nabla^t w_1[i_1].\texttt{xdiff}, \ldots, \nabla^t w_n[i_n].\texttt{xdiff}\}) \\ \mathcal{M}.\texttt{con} \leftarrow a + \nabla^b x[i].\texttt{xdiff} \leq 1 \end{cases} \tag{16}$$

In this way, the incompatibility caused by $\nabla^b x[i] \in \{\texttt{u}, \texttt{n}\} \bigwedge \nabla^t x[i] \in \{\texttt{0}, \texttt{1}\}$ can be eliminated. Similarly, for $\boldsymbol{op} \in \mathcal{R}^b_{\boldsymbol{x}}$, the constraints of similar form as Eq. (16)

can be deduced as Eq. (17).

$$\begin{cases} (\mathcal{M}, a) \leftarrow \texttt{orModel}(\mathcal{M}, \{\nabla^b x[i].\texttt{xdiff}, \nabla^b w_1[i_1].\texttt{xdiff}, \ldots, \nabla^b w_n[i_n].\texttt{xdiff}\}) \\ \mathcal{M}.\texttt{con} \leftarrow a + \nabla^t x[i].\texttt{xdiff} \leq 1 \end{cases} \tag{17}$$

By traversing all XORed bits caused by $\mathcal{R}^t_{\boldsymbol{x}}$- and $\mathcal{R}^b_{\boldsymbol{x}}$-operators, we can acquire an updated model $\mathcal{M}$ that can deduce top-bottom paths without the afore mentioned compatibility issues. The constraints in Eq. (16) and Eq. (17) can therefore be referred along with the 0-AND constraint as the boomerang intersection constraints. All such constraints can be added to the MILP model by calling Algorithm 29 in Supp. Mat. E.2.

Algorithm 8: `relatedOpt`

Input : The digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the word node $\boldsymbol{x}$
Output: The set $\mathcal{R}_{\boldsymbol{x}}$ containing all operator nodes taking the word x as input or output words

Initialize the operator node set $\mathcal{R}_{\boldsymbol{x}} = \phi$
for $\boldsymbol{op} \in \mathcal{V}_o$ **do**
    If $\boldsymbol{op}.\texttt{oWord} = \boldsymbol{x}$, update $\mathcal{R}_{\boldsymbol{x}} \leftarrow \mathcal{R}_{\boldsymbol{x}} \bigcup \{\boldsymbol{op}\}$
    If $\boldsymbol{x} \in \boldsymbol{op}.\texttt{iWord}$, update $\mathcal{R}_{\boldsymbol{x}} \leftarrow \mathcal{R}_{\boldsymbol{x}} \bigcup \{\boldsymbol{op}\}$
Return $\mathcal{R}_{\boldsymbol{x}}$.

5 Application to BLAKE Hash Functions

We apply our approximate signed differential modeling and digraph techniques to conduct boomerang attacks on the keyed permutation of BLAKE3, BLAKE2s and BLAKE-256. The general ideas of BLAKE top-bottom path deduction are similar: for both top and bottom paths, a local collision at round r caused by 1-bit difference in M and V^r is constructed so as to make several consecutive intermediate states have no active bits at all. We demonstrate the top and bottom path deduction procedure in detail for BLAKE3 and briefly for BLAKE2s and BLAKE256.

5.1 Boomerang Attack on BLAKE3

Since BLAKE3 has seven rounds, we divide the whole keyed permutation into E_0 and E_1 intersecting at state $V^{3.5}$ as follows:

$$V^0 \xrightarrow{E_0} V^{3.5} \xrightarrow{E_1} V^7.$$

In the top path, a local collision can be constructed by setting $\nabla^t v_0^{1.5}[i].\texttt{xdiff} = \nabla^t m_1[i].\texttt{xdiff} = 1$ $(i = 0, \ldots, 31)$: if we further add $\nabla^t v_0^{1.5}[i].\texttt{sign} \oplus \nabla^t m_1[i].\texttt{sign} = 1$ $(i = 0, \ldots, 30$, unnecessary for MSB $i = 31)$, there should be five consecutive

states $TV^{1.5}, V^2, TV^2, V^{2.5}, TV^{2.5}$ with no active bits at all. Similarly, in the bottom path, let $\nabla^b v_0^5[j].\texttt{xdiff} = \nabla^b m_9[j].\texttt{xdiff} = 1$ $(j = 0, \ldots, 31)$ and further add $\nabla^b v_0^5[j].\texttt{sign} \oplus \nabla^b m_9[j].\texttt{sign} = 1$ $(j = 0, \ldots, 30$, unnecessary for MSB $j = 31)$, there should be five consecutive states $TV^5, V^{5.5}, TV^{5.5}, V^6, TV^6$ with no active bits.

Three different MILP models namely $\mathcal{M}^t$, $\mathcal{M}^b$, $\mathcal{M}^c$ are constructed:

- $\mathcal{M}^t$: deduce the top path $\mathcal{T}_i^t : \nabla^t V^{1.5} \to \nabla^t V^0$ where $i = 0, \ldots, 31$ corresponds to the active bit selection on $v_0^{1.5}$ and m_1.
- $\mathcal{M}^b$: deduce the bottom path $\mathcal{T}_j^b : \nabla^b V^6 \to \nabla^b V^7$ where $j = 0, \ldots, 31$ corresponds to the active bit selection on v_0^5 and m_9.
- $\mathcal{M}^c$: For each top-bottom path combination $\mathcal{T}_i^t, \mathcal{T}_j^b$ $(i, j \in \{0, \ldots, 31\}$, model $\mathcal{M}^c$ is constructed to deduce the top and bottom path around the intersecting part represented as $\mathcal{T}_{i,j} : \nabla^t V^{2.5} \leftarrow \nabla^t V^{3.5}, \nabla^b V^{3.5} \to \nabla^b V^5$.

Since for BLAKE3, as well as BLAKE2s and BLAKE-256, round functions can be decomposed into the hG function calls defined in Eq. (8), the MILP model capturing the approximate signed differential propagation rules of hG can be generated by calling $(\mathcal{M}, \nabla y_0, \ldots, \nabla y_3) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla x_0, \ldots, \nabla x_3, \nabla m, \alpha, \beta)$ in Algorithm 9 acting as the building block for constructing the three models. Besides modeling hG as Algorithm 9, the objective functions of all 3 models are

Algorithm 9: `hGModel`

Input : The initial MILP model $\mathcal{M}$, the encoded input signed differences $\nabla x_0, \ldots, \nabla x_3, \nabla m$, $\alpha, \beta \in \mathbb{Z}_+$.

Output: The updated MILP model $\mathcal{M}$, the encoded output signed differences $\nabla y_0, \ldots, \nabla y_3$

1 Call Algorithm 3 as $(\mathcal{M}, \nabla w_0) \leftarrow \texttt{modAdd}(\mathcal{M}, \nabla x_0, \nabla x_1)$
2 Call Algorithm 3 as $(\mathcal{M}, \nabla y_0) \leftarrow \texttt{modAdd}(\mathcal{M}, \nabla w_0, \nabla m)$
3 Call Algorithm 4 as $(\mathcal{M}, \nabla w_1) \leftarrow \texttt{xorApprox}(\mathcal{M}, \nabla y_0, \nabla x_3)$
4 Define $\nabla y_3 \leftarrow \nabla w_1 \ggg \alpha$
5 Call Algorithm 3 as $(\mathcal{M}, \nabla y_2) \leftarrow \texttt{modAdd}(\mathcal{M}, \nabla y_3, \nabla x_3)$
6 Call Algorithm 4 as $(\mathcal{M}, \nabla w_2) \leftarrow \texttt{xorApprox}(\mathcal{M}, \nabla y_2, \nabla x_1)$
7 Define $\nabla y_1 \leftarrow \nabla w_2 \ggg \beta$
8 **Return** $(\mathcal{M}, \nabla y_0, \ldots, \nabla y_3)$.

set as Eq. (15) and the boomerang constraints in Section 4.5 are added to $\mathcal{M}_c$. In this way, the solution of each model returns an objective value corresponding to the upper bound of bit conditions. In fact, in BLAKE3 (as well as all other BLAKE hash function family members), Proposition 1 is applicable for all modular add operations making the bit condition number bound extremely tight. Let $\eta^t = \mathcal{M}^t.\texttt{obj}$, $\eta^b = \mathcal{M}^b.\texttt{obj}$ and $\eta^c = \mathcal{M}^c.\texttt{obj}$. Since all bit conditions in $\mathcal{T}_i^t$ and $\mathcal{T}_j^b$ are to be satisfied randomly, the probability for finding quartets satisfying $\mathcal{T}_i^t$ and $\mathcal{T}_j^b$ can be evaluated as $2^{-2(\eta^t + \eta^b)}$. As to the $\mathcal{T}_{i,j}$, the message modification can fix one side of the bit conditions in $\nabla^t V^3 \to \nabla^t V^{3.5}$ and $\nabla^b V^{3.5} \to \nabla^b V^4$. Supposing that there are η_f^c bit conditions fixed, according to

the existing results [16,17,18], the complexity of finding the boomerang quartets $(V_s^{3.5}, M_s)$ $(s = 1, \ldots, 4)$ can therefore be evaluated as

$$Comp = 2^{\eta_f^c} + 2^{2(\eta^t + \eta^b + \eta^c - \eta_f^c)} \tag{18}$$

The $2^{\eta_f^c}$ term corresponds to the process of finding $(V^{3.5}, M)$ quartets following the top-bottom paths $\nabla^t V^{3.5} \to \nabla^t V^3$ and $\nabla^b V^{3.5} \to \nabla^b V^4$: the message modifications applied to $(V_1^{3.5}, M_1)$ guarantee that the signed differential propagations in $(V_1^{3.5}, V_3^{3.5})$ and $(V_1^{3.5}, V_2^{3.5})$ follow the paths $\nabla^t V^{3.5} \to \nabla^t V^3$ and $\nabla^b V^{3.5} \to \nabla^b V^4$ respectively while all η_f^c bit conditions are satisfied randomly for $(V_2^{3.5}, V_4^{3.5})$ and $(V_2^{3.5}, V_3^{3.5})$ with a theoretic probability of $2^{-\eta_f^c}$. After acquiring a $(V^{3.5}, M)$ quartet satisfying $\nabla^t V^{3.5} \to \nabla^t V^3$ and $\nabla^b V^{3.5} \to \nabla^b V^4$ simultaneously, the remaining $(\eta^t + \eta^b + \eta^c - \eta_f^c)$ bit conditions are satisfied randomly utilizing the free degrees in message blocks, which dominates the overall complexities.

We give the η^t, η^b, η^c and η_f^c values along with the complexities corresponding to different i, j combinations in Table 7. Note that the top path $\mathcal{T}_{24}^t$ deduced the with the approximate model is further detected infeasible due to contradictory bit conditions on inactive bits indicating that the approximate signed differential paths should always be checked using digraph based the bit condition deduction technique in Section 4.3.

Table 7: The η^t, η^b, η^c and η_f^c values corresponding to different i, j combinations for BLAKE3. The $(i, j) = (24, 31)$ setting marked with * is infeasible because the top path has contradicted bit conditions on inactive bits.

i	j	η^t	η^b	η^c	η_f^c	$\log_2 Comp$	i	j	η^t	η^b	η^c	η_f^c	$\log_2 Comp$
31	2	47	28	101	85	182	0	**31**	77	25	76	72	212
31	6	47	28	101	85	182	4	**31**	78	25	76	72	214
31	10	47	28	101	85	182	8	**31**	76	25	76	72	210
31	14	47	27	101	85	180	12	**31**	73	25	76	72	204
31	18	47	28	101	85	182	16	**31**	76	25	76	72	210
31	22	47	28	101	85	182	20	**31**	76	25	76	72	210
31	26	47	28	101	85	182	24*	**31**	72	25	76	72	202
31	30	47	27	99	83	180	28	**31**	75	25	76	72	208

By analyzing the parameters in Table 7, we may find that the η^b, η^c and $(\eta^c - \eta_f^c)$ values for $i = 31 \wedge j \neq 31$ are all slightly larger than those for $i \neq 31 \wedge j = 31$. On the contrary, the η^t values for $i = 31 \wedge j \neq 31$ are much smaller than those for $i \neq 31 \wedge j = 31$. As a result, the advantage of a much smaller η^t makes up with the disadvantage of slightly larger η^b, η^c and $(\eta^c - \eta_f^c)$ making $i = 31 \wedge j \neq 31$ a better choice for lower attacking complexities: as can be seen in Table 7, $(i, j) = (31, 14)$ makes the lowest complexity of 2^{180}.

It is noticeable that the overall complexity is usually dominated by the η^t, η^b parameters: a larger η^c is not the main factor since most of the bit conditions are to be fixed with the message modification technique. The practical experiments in [17,18] indicate that such $(V^{3.5}, M)$ quartets can be found with complexities

much lower than $2^{\eta_f^c}$. This phenomenon is can also be seen in BLAKE2s and BLAKE-256 which makes our (i, j)-selection strategy different from that in previous works: we prefer to use $i = 31$ so as to lower η^t, η^b while previous works use $j = 31$ so as to acquire a lower η^c. We have to admit that a lower η^c can make it easier to find practically $(V^{3.5}, M)$ quartets covering short rounds around $V^{3.5}$. But it is also true that $j = 31$ is more likely to have a higher overall complexities for full-round attacks than the $i = 31$ counterparts. Details for model and digraph constructions of BLAKE3 can be found in Supp. Mat. E.3 and Supp. Mat. D.1 respectively.

Practical Verifications. The theoretic complexity for finding $(V^{3.5}, M)$ quartets satisfying $\nabla^t V^{3.5} \to \nabla^t V^3$ and $\nabla^b V^{3.5} \to \nabla^b V^4$ is $2^{\eta_f^c}$ which is computationally infeasible. However, in practice, the complexity is lower enabling us to construct quartets satisfying round-reduce top-bottom paths practically: as can be seen, a practical quartets for 7-round BLAKE-256 is given in [17] and those for 6.5-round BLAKE2s and BLAKE-256 are given in [18] so as to verify the compatibility of their paths. We also provide practical quartets for 3-round BLAKE3 using both $i = 31 \wedge j \neq 31$ and $i \neq 31 \wedge j = 31$ settings covering V^1 to V^4: two quartets following $(i, j) = (31, 14)$ and $(i, j) = (12, 31)$ respectively are given in Table 9 indicating the compatibility of our paths.

5.2 Boomerang Attacks on BLAKE2s and BLAKE-256

For both BLAKE2s and BLAKE-256, the boomerang attack targets are the keyed permutation $E : V^{2.5} \to V^{10.5}$ which is decomposed into E_0 and E_1 intersecting at state $V^{6.5}$ as follows:

$$V^{2.5} \xrightarrow{E_0} V^{6.5} \xrightarrow{E_1} V^{10.5}.$$

For local collisions, the cancelable 1-bit signed differences should be imposed to $(\nabla^t v_1^4[i], \nabla m_5[i])$ in the top path and $(\nabla^b v_2^8[j], \nabla^b m_{11}[j])$ in the bottom path. In this way, both top and bottom paths have seven consecutive zero-difference intermediate states. We also construct the three models as in Section 5.1 and deduce the η^t, η^b, η^c and η_f^c values along with the complexities corresponding to different i, j combinations. We find that the best 8-round attack complexities for both BLAKE-256 and BLAKE2s are all acquired with $i = 31$ and the good j selections as well as the corresponding complexity parameters are all listed in Table 10. In comparison with previous boomerang results, our method improve the best result on BLAKE2s by 0.5 rounds and lower the complexity of BLAKE-256 result by 2^{16}. As can be seen, Table 10 includes not only our new $i = 31$ setting attacks but the (i, j) settings of previous best attacks as well: for BLAKE-256, we evaluate the $(i, j) = (28, 31), (20, 31)$ setting paths used in [17,18] and acquire the same complexities proving the accuracy of our digraph based bit condition deduction method. Based on the $(i, j) = (28, 31)$ setting used originally in [18] for attacking 7.5-round BLAKE2s, we extend the path by 0.5 rounds acquiring an 8-round attack with complexity 2^{230}. It is also noticeable from Table 10, the "small η^t for $i = 31$" phenomenon for BLAKE3 also exists in

Table 8: A signed differential path with all bit conditions settled using the digraph-based method with the $(i, j) = (31, 14)$ setting.

$\nabla^t M$	================================	u===============================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
$\nabla^t V^0$	========u=========u===n=======	================u===============	==========u===u==========u=====	uu==u========u===u===n====u==n===
	u=======n==========n===u=======	u===============================	u============n=====u=========n==	=n==n===u===nu=======u===n===u===
	u========u========u========u=====	================n========n======	n====================n==========	================================
	1===============================	1=======u=======0===============	u===u====u==========u=====u======	u=======0====u===u========uu======
$\nabla^t V^{0.5}$	u==============0===============	u================u=======1======	====================u===========	========0==============0=======
	=======1===============================	================================	n=======1========n===============	u======0=======1====n==1=======0
	u===============================	================================	u========0==============0======	u=======u====0==u========u1======
	========n================n=======	n=======0==============1=======	================================0===	u================u================
$\nabla^t V^1$	u===============================	========0==============0=======	================================	================================
	================================	================================	================================	=======1========================
	u===============================	================================	================================	================================
	n==============n================	================================	================================	========1=======================
$\nabla^t V^{1.5}$	n===============================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
	========0=======================	================================	================================	================================
$\nabla^t V^2$	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
$\nabla^t V^{2.5}$	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
$\nabla^t V^3$	================================	================================	================================	u===============================
	==============u===============0	========0=======================	============================0=======	================0================
	================================	========u=======================	================================	================================
	==============0================	================================	========u=======================	0===============================
$\nabla^t V^{3.5}$	==========u===u==========u===0	====================u==========	====u==================0=======	u==============0==========u===
	===u====u====u====u====u====u=	===u==========u==========u======	========u===u========u========u	===u=======u===u=======u=======u
	===u===u===0=======u===u===0===u	========u==========0=======u===	u===0=======u==========u=======	====u===u========u========u===0===
	===u===u==========u===u========0	================================u===	u==========u====================	=====u===u====================u=======
$\nabla^b M$	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================u===============	================================	================================
	================================	================================	================================	================================
$\nabla^b V^{3.5}$	=====n===n========n========n======	=u===n=====u==n=====n==n=======n==	==========u===n==========u===n=	=u================================
	==============u===============	=====u===u========u=======u======	=====u===u====u==u==u===uu=	=====u========u==u============u=
	=====u==========u===============	=n========n==================	=n========u========u========u======	================u===============
	=1=======0u==============0====u=	=====u====u=======u===u====u======	=0==============0=======u======	================u===============
$\nabla^b V^4$	=1==============n==============	=========0=======n=======0======	=====u============================	=u=======1========u===============
	================================	========================1=======	1====n==1=======1u======1=======	=n==============n=======1=======
	=========0======u========0=====	================================	================u==============	=u========u1======u========u====0=
	=u==============u=======0======	============1===================	==========0=======n=======0======	==========n==============n=======
$\nabla^b V^{4.5}$	================n===============	================================	================================	=========0==============0======
	================================	================================	========================1=======	================================
	================================	================================	================u===============	================================
	======================0=========	================================	================================	=n==============n===============
$\nabla^b V^5$	================n===============	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	========================0=======
$\nabla^b V^{5.5}$	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
$\nabla^b V^6$	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
	================================	================================	================================	================================
$\nabla^b V^{6.5}$	================================	================================	================u===============	================================
	========0=======================	========================0=======	u================0===============	=0===============================
	================================	================================	================================u=======	================================
	0===============================	================0===============	================================u======	================================
$\nabla^b V^7$	=====u==========================	u==========u===0==========u===	=0==========u===u==============	========0==========u==========
	=====u==========u==========u====	u==========u========u==========	====u========u===u===u===u====u	u========u========u========u====
	=u=======u===0=======u===u======	==========u========u===0=======u==	======0=======u============u======	====u===u===0===u===u===u===0===
	=====u===u==========u===u=======	==========u==========u===u======	================u============u==	============u==================

Table 9: $(V^{3.5}, M)$ quartet for BLAKE3 from V^1 to V^4 satisfying the $(i,j) = (31,14)$ and $(i,j) = (12,31)$ settings.

	$(i,j) = (31,14)$
M_1	0xee80884a, 0xc43832f9, 0x3d7d500e, 0x8e624dbe, 0x96c82faf, 0x73936843, 0x23bb25ab, 0xb4cc6d83, 0x31cd7903, 0xa511602f, 0x1cb0e950, 0x591526e2, 0x9b11f9c6, 0xa4ad1fb5, 0xe6da4589, 0x10f8033
M_2	0xee80884a, 0xc43832f9, 0x3d7d500e, 0x8e624dbe, 0x96c82faf, 0x73936843, 0x23bb25ab, 0xb4cc6d83, 0x31cd7903, 0xa511202f, 0x1cb0e950, 0x591526e2, 0x9b11f9c6, 0xa4ad1fb5, 0xe6da4589, 0x10f8033
M_3	0xee80884a, 0x443832f9, 0x3d7d500e, 0x8e624dbe, 0x96c82faf, 0x73936843, 0x23bb25ab, 0xb4cc6d83, 0x31cd7903, 0xa511602f, 0x1cb0e950, 0x591526e2, 0x9b11f9c6, 0xa4ad1fb5, 0xe6da4589, 0x10f8033
M_4	0xee80884a, 0x443832f9, 0x3d7d500e, 0x8e624dbe, 0x96c82faf, 0x73936843, 0x23bb25ab, 0xb4cc6d83, 0x31cd7903, 0xa511202f, 0x1cb0e950, 0x591526e2, 0x9b11f9c6, 0xa4ad1fb5, 0xe6da4589, 0x10f8033
$V_1^{3.5}$	0xba33879a, 0xeb695a7a, 0x8cb4b42c, 0xfdbe455d, 0x2337f6e2, 0x164b4ffa, 0x157e3de7, 0xbd77e373, 0xf5ab732d, 0x90bd040a, 0x8559dad7, 0xe9a8e080, 0x57231596, 0x3f2df6b9, 0xb8fc1152, 0xfac5e6d2
$V_2^{3.5}$	0xbe73c7da, 0xaf4d7e7e, 0x8c96b40e, 0xbdbe455d, 0x2337b6e2, 0x120b0fba, 0x113a19a1, 0xb975a371, 0xf1ab332d, 0xd0fd040a, 0xc5199a97, 0xe9a8a080, 0x57031594, 0x3b0db299, 0xb8fc1112, 0xfac5a6d2
$V_3^{3.5}$	0xba22878a, 0xeb69527a, 0x84b4b42c, 0x7dbe4555, 0x115d6c0, 0x64a4fea, 0x146e2de6, 0xad66e272, 0xe4ab622c, 0x903d0402, 0x551da57, 0xe1286000, 0x46230496, 0x3f2df6b1, 0x38f41152, 0xf245e652
$V_4^{3.5}$	0xbe62c7ca, 0xaf4d767e, 0x8496b40e, 0x3dbe4555, 0x11596c0, 0x20a0faa, 0x102a09a0, 0xa964a270, 0xe0ab222c, 0xd07d0402, 0x45119a17, 0xe1282000, 0x46030494, 0x3b0db291, 0x38f41112, 0xf245a652
	$(i,j) = (31,14)$
M_1	0x591dd3bd, 0x90071f7a, 0x72eee34f, 0x7e6116bb, 0xd0527ca1, 0xb40c585a, 0x4e12c108, 0x730e2f3e, 0x133279e9, 0x32a000cf, 0xcf9cca60, 0xf42a63ca, 0x8a660309, 0x9d31a602, 0xe739a151, 0xb443037c
M_2	0x591dd3bd, 0x90071f7a, 0x72eee34f, 0x7e6116bb, 0xd0527ca1, 0xb40c585a, 0x4e12c108, 0x730e2f3e, 0x133279e9, 0xb2a000cf, 0xcf9cca60, 0xf42a63ca, 0x8a660309, 0x9d31a602, 0xe739a151, 0xb443037c
M_3	0x591dd3bd, 0x90070f7a, 0x72eee34f, 0x7e6116bb, 0xd0527ca1, 0xb40c585a, 0x4e12c108, 0x730e2f3e, 0x133279e9, 0x32a000cf, 0xcf9cca60, 0xf42a63ca, 0x8a660309, 0x9d31a602, 0xe739a151, 0xb443037c
M_4	0x591dd3bd, 0x90070f7a, 0x72eee34f, 0x7e6116bb, 0xd0527ca1, 0xb40c585a, 0x4e12c108, 0x730e2f3e, 0x133279e9, 0xb2a000cf, 0xcf9cca60, 0xf42a63ca, 0x8a660309, 0x9d31a602, 0xe739a151, 0xb443037c
$V_1^{3.5}$	0xacb69906, 0xef9b9ecb, 0x5ae44d23, 0x8931df34, 0xd7cffcdf, 0x6846678a, 0x97e6e726, 0xe178ba1e, 0x367da271, 0xfe1db232, 0xb3be96af, 0xb414cbf9, 0x6b34b665, 0xc8ed187b, 0x838b3c93, 0x9c773fd1
$V_2^{3.5}$	0x2c369186, 0xa7931603, 0x5aa04d67, 0x89315f34, 0x57cffcdf, 0xe8c66f0a, 0xdf6aefae, 0x617cb21a, 0xb67daa71, 0xfe1d32b2, 0x333e162f, 0x3414cbf9, 0x6b30b625, 0x40ad103b, 0x830b3c93, 0x1c773fd1
$V_3^{3.5}$	0x8cb49904, 0xee9b9ecb, 0x5ae44c23, 0x8930cf34, 0x93cbb89b, 0x4844658a, 0x95e6c704, 0xc158981c, 0x345d8051, 0xfe1cb222, 0xb3ae86ae, 0xa404cae9, 0x6914b445, 0xc8ec187b, 0x838b2c92, 0x9c673ec1
$V_4^{3.5}$	0xc349184, 0xa6931603, 0x5aa04c67, 0x89304f34, 0x13cbb89b, 0xc8c46d0a, 0xdd6acf8c, 0x415c9018, 0xb45d8851, 0xfe1c32a2, 0x332e062e, 0x2404cae9, 0x6910b405, 0x40ac103b, 0x830b2c92, 0x1c673ec1

BLAKE2s and BLAKE-256 while previous results [17,18] commonly apply the $j = 31$ setting so as to minimize η^c which, according to our analysis, is not the optimal choice.

Practical Verification. For $(i,j) = (31,30)$, we are able to find practical $(V^{6.5}, M)$ quartets satisfying the top-bottom paths from $V^{3.5}$ to V^7 for both BLAKE2s and BLAKE-256. The quartets are shown in Table 11.

6 Discussions and Conclusions

In this paper, we propose the approximate MILP modeling technique for automatic signed differential path deduction along with a digraph based framework for bit condition deduction, objective function definition and compatible boomerang intersections. The two techniques form a thorough cryptanalysis tool for ARX primitives enabling us to launch new boomerang attacks on BLAKE3, BLAKE2s and BLAKE-256 hash functions.

According to our experiments, the approximate signed differential model is efficient only for short rounds. For longer rounds, the model solving process can be extremely time consuming which is an obvious direction for further improvements. Besides, the signed difference is by its nature applicable for collision

Table 10: The η^t, η^b, η^c and η_f^c values correspond to different i, j combinations for BLAKE-256 and BLAKE2s. Such parameters are almost the same for both primitives except in setting of $(i, j) = \{(31, 6), (28, 31)\}$ where the corresponding values are listed in bold and the ones of BLAKE2s are given in "()".

i	j	η^t	η^b	η^c	η_f^c	$\log_2 Comp$	Source
31	2	48	28	99	83	184	This Paper
31	6	48	**29(28)**	99	83	**186(184)**	This Paper
31	10	48	28	99	83	184	This Paper
31	14	48	27	99	83	182	This Paper
31	18	48	28	99	83	184	This Paper
31	22	48	28	99	83	184	This Paper
31	26	48	28	99	83	184	This Paper
31	30	48	27	97	81	182	This Paper
28	31	**70(86)**	25	77	73	**198(230)**	[18](This Paper)
20	31	71(-)	25(-)	77(-)	73(-)	200(-)	[17]

Table 11: $(V^{6.5}, M)$ quartet for BLAKE-256 and BLAKE2s from $V^{3.5}$ to V^7 satisfying the $(i, j) = (31, 30)$ setting.

	BLAKE-256
M_1	0x216b4119, 0xcdbee602, 0x31dfd1c9, 0xea32e3c9, 0xaa8dcdad, 0x61c5da01, 0x816d0dd, 0xc31c67e2, 0x59860f6d, 0xe2baaacd, 0xfe012bf2, 0x56c71ddd, 0x4de39df7, 0x8b993c13, 0xdfadcabe, 0xf1ea633c
M_2	0x216b4119, 0xcdbee602, 0x31dfd1c9, 0xea32e3c9, 0xaa8dcdad, 0x61c5da01, 0x816d0dd, 0xc31c67e2, 0x59860f6d, 0xe2baaacd, 0xfe012bf2, 0x16c71ddd, 0x4de39df7, 0x8b993c13, 0xdfadcabe, 0xf1ea633c
M_3	0x216b4119, 0xcdbee602, 0x31dfd1c9, 0xea32e3c9, 0xaa8dcdad, 0xe1c5da01, 0x816d0dd, 0xc31c67e2, 0x59860f6d, 0xe2baaacd, 0xfe012bf2, 0x56c71ddd, 0x4de39df7, 0x8b993c13, 0xdfadcabe, 0xf1ea633c
M_4	0x216b4119, 0xcdbee602, 0x31dfd1c9, 0xea32e3c9, 0xaa8dcdad, 0xe1c5da01, 0x816d0dd, 0xc31c67e2, 0x59860f6d, 0xe2baaacd, 0xfe012bf2, 0x16c71ddd, 0x4de39df7, 0x8b993c13, 0xdfadcabe, 0xf1ea633c
$V_1^{6.5}$	0xc9237803, 0xe84246aa, 0xfd7dfc5e, 0x6f68d8fc, 0x51fb7f99, 0x13f10371, 0x7bb37c33, 0x9017231e, 0x81ddd2fe, 0x4cf1ab85, 0x352a9103, 0xeeae25de, 0x92faee1c, 0xf9cedfc4, 0x1307d5f2, 0xeee76f2c
$V_2^{6.5}$	0x89017821, 0xa84206aa, 0xbd3df81e, 0x4b689cd8, 0x75b97bdd, 0x53f30773, 0x7bb37c33, 0xd057275e, 0xc19d92be, 0x4cf1ab85, 0x352a9503, 0xaeae65de, 0x92faee1c, 0xb9cedfc4, 0x13059592, 0xeac72b0c
$V_3^{6.5}$	0x41237803, 0x684246a2, 0xfd6cfc4e, 0x6f68d0fc, 0x50ea6f98, 0x3e00270, 0x59915c11, 0x8016230e, 0x155d27e, 0x44712b05, 0x242a8002, 0xee2e25d6, 0x1272ee1c, 0xf14edf44, 0x207c4f2, 0xeee76f24
$V_4^{6.5}$	0x1017821, 0x284206a2, 0xbd2cf80e, 0x4b6894d8, 0x74a86bdc, 0x43e20672, 0x59915c11, 0xc056274e, 0x4115923e, 0x44712b05, 0x242a8402, 0xae2e65d6, 0x1272ee1c, 0xb14edf44, 0x2058492, 0xeac72b04
	BLAKE2s
M_1	0xc77266d6, 0x620ce970, 0xb9a5573c, 0xb434a731, 0xa8996f30, 0xa10339da, 0x47cc583a, 0x2a3f20eb, 0x2b93a60f, 0x1b9d6bc4, 0xa43023a1, 0x7d135ea9, 0xa6cc4cdd, 0x9531106e, 0x596823aa, 0x43b8c256
M_2	0xc77266d6, 0x620ce970, 0xb9a5573c, 0xb434a731, 0xa8996f30, 0xa10339da, 0x47cc583a, 0x2a3f20eb, 0x2b93a60f, 0x1b9d6bc4, 0xa43023a1, 0x3d135ea9, 0xa6cc4cdd, 0x9531106e, 0x596823aa, 0x43b8c256
M_3	0xc77266d6, 0x620ce970, 0xb9a5573c, 0xb434a731, 0xa8996f30, 0x210339da, 0x47cc583a, 0x2a3f20eb, 0x2b93a60f, 0x1b9d6bc4, 0xa43023a1, 0x7d135ea9, 0xa6cc4cdd, 0x9531106e, 0x596823aa, 0x43b8c256
M_4	0xc77266d6, 0x620ce970, 0xb9a5573c, 0xb434a731, 0xa8996f30, 0x210339da, 0x47cc583a, 0x2a3f20eb, 0x2b93a60f, 0x1b9d6bc4, 0xa43023a1, 0x3d135ea9, 0xa6cc4cdd, 0x9531106e, 0x596823aa, 0x43b8c256
$V_1^{6.5}$	0xe8a29f6f, 0xe51f66d9, 0x7f7f15bc, 0x9c50eb73, 0x317f5417, 0x391d97ed, 0x3b7227bb, 0xbcb98072, 0x93fe42f1, 0x5ffec280, 0x15899b2f, 0xc0b2358d, 0xa59e4022, 0x6ab31bb8, 0xdfa6fd6a, 0x572bddbd
$V_2^{6.5}$	0xa8809f4d, 0xa51f26d9, 0x3f3f11fc, 0x3850af57, 0x153d5053, 0x791f93ef, 0x3b7227bb, 0xfcf98432, 0xd3be02b1, 0x5ffec280, 0x15899f2f, 0x80b2758d, 0xa59e4022, 0x2ab31bb8, 0x5fa4bd0a, 0x530b999d
$V_3^{6.5}$	0x60a29f6f, 0x651f66d1, 0x7f6e15ac, 0x9c50e373, 0x306e4416, 0x290c96ec, 0x19500799, 0xacb88062, 0x13764271, 0x577e4200, 0x4898a2e, 0xc0323585, 0x25164022, 0x62331b38, 0xcea6ec6a, 0x572bddb5
$V_4^{6.5}$	0x20809f4d, 0x251f26d1, 0x3f2e11ec, 0x3850a757, 0x142c4052, 0x690e92ee, 0x19500799, 0xecf88422, 0x53360231, 0x577e4200, 0x4898e2e, 0x80327585, 0x25164022, 0x22331b38, 0x4ea4ac0a, 0x530b9995

attacks so we expect our approximate signed differential model be applied to collision attacks on ARX hash functions in the future.

References

1. Wagner, D.: The boomerang attack. In Knudsen, L.R., ed.: FSE'99. Volume 1636 of LNCS., Springer, Heidelberg (March 1999) 156–170
2. Kelsey, J., Kohno, T., Schneier, B.: Amplified boomerang attacks against reduced-round MARS and Serpent. In Schneier, B., ed.: FSE 2000. Volume 1978 of LNCS., Springer, Heidelberg (April 2001) 75–93
3. Biham, E., Dunkelman, O., Keller, N.: The rectangle attack - rectangling the Serpent. In Pfitzmann, B., ed.: EUROCRYPT 2001. Volume 2045 of LNCS., Springer, Heidelberg (May 2001) 340–357
4. Biham, E., Dunkelman, O., Keller, N.: New results on boomerang and rectangle attacks. In Daemen, J., Rijmen, V., eds.: FSE 2002. Volume 2365 of LNCS., Springer, Heidelberg (February 2002) 1–16
5. Biham, E., Dunkelman, O., Keller, N.: Related-key boomerang and rectangle attacks. In Cramer, R., ed.: EUROCRYPT 2005. Volume 3494 of LNCS., Springer, Heidelberg (May 2005) 507–525
6. Gorski, M., Lucks, S.: New related-key boomerang attacks on AES. In Chowdhury, D.R., Rijmen, V., Das, A., eds.: INDOCRYPT 2008. Volume 5365 of LNCS., Springer, Heidelberg (December 2008) 266–278
7. Bariant, A., Leurent, G.: Truncated boomerang attacks and application to AES-based ciphers. Cryptology ePrint Archive, Paper 2022/701 (2022)
8. Fleischmann, E., Forler, C., Gorski, M., Lucks, S.: New boomerang attacks on ARIA. In Gong, G., Gupta, K.C., eds.: INDOCRYPT 2010. Volume 6498 of LNCS., Springer, Heidelberg (December 2010) 163–175
9. Kim, J., Moon, D., Lee, W., Hong, S., Lee, S., Jung, S.: Amplified boomerang attack against reduced-round SHACAL. In Zheng, Y., ed.: ASIACRYPT 2002. Volume 2501 of LNCS., Springer, Heidelberg (December 2002) 243–253
10. Sasaki, Y.: Improved related-tweakey boomerang attacks on deoxys-BC. In Joux, A., Nitaj, A., Rachidi, T., eds.: AFRICACRYPT 18. Volume 10831 of LNCS., Springer, Heidelberg (May 2018) 87–106
11. Jeong, K., Lee, C., Sung, J., Hong, S., Lim, J.: Related-key amplified boomerang attacks on the full-round Eagle-64 and Eagle-128. In Pieprzyk, J., Ghodosi, H., Dawson, E., eds.: ACISP 07. Volume 4586 of LNCS., Springer, Heidelberg (July 2007) 143–157
12. Dobraunig, C., List, E.: Impossible-differential and boomerang cryptanalysis of round-reduced kiasu-BC. In Handschuh, H., ed.: CT-RSA 2017. Volume 10159 of LNCS., Springer, Heidelberg (February 2017) 207–222
13. Ashur, T., Dunkelman, O.: A practical related-key boomerang attack for the full MMB block cipher. In Abdalla, M., Nita-Rotaru, C., Dahab, R., eds.: CANS 13. Volume 8257 of LNCS., Springer, Heidelberg (November 2013) 271–290
14. Isobe, T., Sasaki, Y., Chen, J.: Related-key boomerang attacks on KATAN32/48/64. In Boyd, C., Simpson, L., eds.: ACISP 13. Volume 7959 of LNCS., Springer, Heidelberg (July 2013) 268–285
15. Joux, A., Peyrin, T.: Hash functions and the (amplified) boomerang attack. In Menezes, A., ed.: CRYPTO 2007. Volume 4622 of LNCS., Springer, Heidelberg (August 2007) 244–263
16. Biryukov, A., Nikolic, I., Roy, A.: Boomerang attacks on BLAKE-32. In Joux, A., ed.: FSE 2011. Volume 6733 of LNCS., Springer, Heidelberg (February 2011) 218–237

17. Bai, D., Yu, H., Wang, G., Wang, X.: Improved boomerang attacks on round-reduced SM3 and BLAKE-256. Cryptology ePrint Archive, Report 2013/852 (2013)
18. Hao, Y.: The boomerang attacks on BLAKE and BLAKE2. In Lin, D., Yung, M., Zhou, J., eds.: Inscrypt 2014. Volume 8957 of LNCS., Springer (2014) 286–310
19. Biryukov, A., Lamberger, M., Mendel, F., Nikolic, I.: Second-order differential collisions for reduced SHA-256. In Lee, D.H., Wang, X., eds.: ASIACRYPT 2011. Volume 7073 of LNCS., Springer, Heidelberg (December 2011) 270–287
20. Lamberger, M., Mendel, F.: Higher-order differential attack on reduced SHA-256. Cryptology ePrint Archive, Report 2011/037 (2011)
21. Yu, H., Bai, D.: Boomerang attack on step-reduced SHA-512. Cryptology ePrint Archive, Report 2014/945 (2014)
22. Yu, H., Hao, Y., Bai, D.: Evaluate the security margins of SHA-512, SHA-256 and DHA-256 against the boomerang attack. Sci. China Inf. Sci. **59**(5) (2016) 052110:1–052110:14
23. Mendel, F., Nad, T.: Boomerang distinguisher for the SIMD-512 compression function. In Bernstein, D.J., Chatterjee, S., eds.: INDOCRYPT 2011. Volume 7107 of LNCS., Springer, Heidelberg (December 2011) 255–269
24. Sasaki, Y.: Boomerang distinguishers on MD4-family: First practical results on full 5-pass HAVAL. In Miri, A., Vaudenay, S., eds.: SAC 2011. Volume 7118 of LNCS., Springer, Heidelberg (August 2012) 1–18
25. Sasaki, Y., Wang, L.: Distinguishers beyond three rounds of the RIPEMD-128/-160 compression functions. In Bao, F., Samarati, P., Zhou, J., eds.: ACNS 12. Volume 7341 of LNCS., Springer, Heidelberg (June 2012) 275–292
26. Sasaki, Y., Wang, L., Takasaki, Y., Sakiyama, K., Ohta, K.: Boomerang distinguishers for full HAS-160 compression function. In Hanaoka, G., Yamauchi, T., eds.: IWSEC 12. Volume 7631 of LNCS., Springer, Heidelberg (November 2012) 156–169
27. Leurent, G., Roy, A.: Boomerang attacks on hash function using auxiliary differentials. In Dunkelman, O., ed.: CT-RSA 2012. Volume 7178 of LNCS., Springer, Heidelberg (February / March 2012) 215–230
28. Yu, H., Chen, J., Wang, X.: The boomerang attacks on the round-reduced Skein-512. In Knudsen, L.R., Wu, H., eds.: SAC 2012. Volume 7707 of LNCS., Springer, Heidelberg (August 2013) 287–303
29. Kircanski, A., Shen, Y., Wang, G., Youssef, A.M.: Boomerang and slide-rotational analysis of the SM3 hash function. In Knudsen, L.R., Wu, H., eds.: SAC 2012. Volume 7707 of LNCS., Springer, Heidelberg (August 2013) 304–320
30. Bai, D., Yu, H., Wang, G., Wang, X.: Improved boomerang attacks on SM3. In Boyd, C., Simpson, L., eds.: ACISP 13. Volume 7959 of LNCS., Springer, Heidelberg (July 2013) 251–266
31. Cid, C., Huang, T., Peyrin, T., Sasaki, Y., Song, L.: Boomerang connectivity table: A new cryptanalysis tool. In Nielsen, J.B., Rijmen, V., eds.: EUROCRYPT 2018, Part II. Volume 10821 of LNCS., Springer, Heidelberg (April / May 2018) 683–714
32. Song, L., Qin, X., Hu, L.: Boomerang connectivity table revisited. IACR Trans. Symm. Cryptol. **2019**(1) (2019) 118–141
33. Dunkelman, O.: Efficient construction of the boomerang connection table. Cryptology ePrint Archive, Report 2018/631 (2018)
34. Boukerrou, H., Huynh, P., Lallemand, V., Mandal, B., Minier, M.: On the Feistel counterpart of the boomerang connectivity table (long paper). IACR Trans. Symm. Cryptol. **2020**(1) (2020) 331–362

35. Boura, C., Canteaut, A.: On the boomerang uniformity of cryptographic sboxes. IACR Trans. Symm. Cryptol. **2018**(3) (2018) 290–310
36. Hadipour, H., Nageler, M., Eichlseder, M.: Throwing boomerangs into feistel structures: Application to CLEFIA, WARP, LBlock, LBlock-s and TWINE. IACR Transactions on Symmetric Cryptology (2022) 271–302
37. Rahman, M., Saha, D., Paul, G.: Boomeyong: Embedding yoyo within boomerang and its applications to key recovery attacks on AES and Pholkos. IACR Trans. Symm. Cryptol. **2021**(3) (2021) 137–169
38. Wang, H., Peyrin, T.: Boomerang switch in multiple rounds. IACR Trans. Symm. Cryptol. **2019**(1) (2019) 142–169
39. Dunkelman, O., Keller, N., Ronen, E., Shamir, A.: The retracing boomerang attack. In Canteaut, A., Ishai, Y., eds.: EUROCRYPT 2020, Part I. Volume 12105 of LNCS., Springer, Heidelberg (May 2020) 280–309
40. Delaune, S., Derbez, P., Vavrille, M.: Catching the fastest boomerangs application to SKINNY. IACR Trans. Symm. Cryptol. **2020**(4) (2020) 104–129
41. Frixons, P., Naya-Plasencia, M., Schrottenloher, A.: Quantum boomerang attacks and some applications. Cryptology ePrint Archive, Report 2022/060 (2022)
42. Qin, L., Dong, X., Wang, X., Jia, K., Liu, Y.: Automated search oriented to key recovery on ciphers with linear key schedule. IACR Trans. Symm. Cryptol. **2021**(2) (2021) 249–291
43. Zhao, B., Dong, X., Jia, K.: New related-tweakey boomerang and rectangle attacks on deoxys-bc including BDT effect. IACR Trans. Symm. Cryptol. **2019**(3) (2019) 121–151
44. Liu, Y., Sasaki, Y.: Related-key boomerang attacks on GIFT with automated trail search including BCT effect. In Jang-Jaccard, J., Guo, F., eds.: ACISP 19. Volume 11547 of LNCS., Springer, Heidelberg (July 2019) 555–572
45. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In Shoup, V., ed.: CRYPTO 2005. Volume 3621 of LNCS., Springer, Heidelberg (August 2005) 17–36
46. Wang, X., Yu, H.: How to break MD5 and other hash functions. In Cramer, R., ed.: EUROCRYPT 2005. Volume 3494 of LNCS., Springer, Heidelberg (May 2005) 19–35
47. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: General results and applications. In Lai, X., Chen, K., eds.: ASIACRYPT 2006. Volume 4284 of LNCS., Springer, Heidelberg (December 2006) 1–20
48. Mendel, F., Nad, T., Schläffer, M.: Finding collisions for round-reduced SM3. In Dawson, E., ed.: CT-RSA 2013. Volume 7779 of LNCS., Springer, Heidelberg (February / March 2013) 174–188
49. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A.K., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In Halevi, S., ed.: CRYPTO 2009. Volume 5677 of LNCS., Springer, Heidelberg (August 2009) 55–69
50. Karpman, P., Peyrin, T., Stevens, M.: Practical free-start collision attacks on 76-step SHA-1. In Gennaro, R., Robshaw, M.J.B., eds.: CRYPTO 2015, Part I. Volume 9215 of LNCS., Springer, Heidelberg (August 2015) 623–642
51. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In Katz, J., Shacham, H., eds.: CRYPTO 2017, Part I. Volume 10401 of LNCS., Springer, Heidelberg (August 2017) 570–596
52. Qiao, K., Song, L., Liu, M., Guo, J.: New collision attacks on round-reduced Keccak. In Coron, J.S., Nielsen, J.B., eds.: EUROCRYPT 2017, Part III. Volume 10212 of LNCS., Springer, Heidelberg (April / May 2017) 216–243

53. Song, L., Liao, G., Guo, J.: Non-full sbox linearization: Applications to collision attacks on round-reduced Keccak. In Katz, J., Shacham, H., eds.: CRYPTO 2017, Part II. Volume 10402 of LNCS., Springer, Heidelberg (August 2017) 428–451
54. Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional cube attack on reduced-round Keccak sponge function. In Coron, J.S., Nielsen, J.B., eds.: EUROCRYPT 2017, Part II. Volume 10211 of LNCS., Springer, Heidelberg (April / May 2017) 259–288
55. Liu, F., Mendel, F., Wang, G.: Collisions and semi-free-start collisions for round-reduced RIPEMD-160. In Takagi, T., Peyrin, T., eds.: ASIACRYPT 2017, Part I. Volume 10624 of LNCS., Springer, Heidelberg (December 2017) 158–186
56. Dobraunig, C., Eichlseder, M., Mendel, F.: Analysis of SHA-512/224 and SHA-512/256. In Iwata, T., Cheon, J.H., eds.: ASIACRYPT 2015, Part II. Volume 9453 of LNCS., Springer, Heidelberg (November / December 2015) 612–630
57. Liu, F., Dobraunig, C., Mendel, F., Isobe, T., Wang, G., Cao, Z.: Efficient collision attack frameworks for RIPEMD-160. In Boldyreva, A., Micciancio, D., eds.: CRYPTO 2019, Part II. Volume 11693 of LNCS., Springer, Heidelberg (August 2019) 117–149
58. Mendel, F., Peyrin, T., Schläffer, M., Wang, L., Wu, S.: Improved cryptanalysis of reduced RIPEMD-160. In Sako, K., Sarkar, P., eds.: ASIACRYPT 2013, Part II. Volume 8270 of LNCS., Springer, Heidelberg (December 2013) 484–503
59. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 characteristics: Searching through a minefield of contradictions. In Lee, D.H., Wang, X., eds.: ASIACRYPT 2011. Volume 7073 of LNCS., Springer, Heidelberg (December 2011) 288–307
60. Leurent, G.: Analysis of differential attacks in ARX constructions. In Wang, X., Sako, K., eds.: ASIACRYPT 2012. Volume 7658 of LNCS., Springer, Heidelberg (December 2012) 226–243
61. Leurent, G.: Construction of differential characteristics in ARX designs application to Skein. In Canetti, R., Garay, J.A., eds.: CRYPTO 2013, Part I. Volume 8042 of LNCS., Springer, Heidelberg (August 2013) 241–258
62. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Sarkar, P., Iwata, T., eds.: ASIACRYPT 2014, Part I. Volume 8873 of LNCS., Springer, Heidelberg (December 2014) 158–178
63. Eichlseder, M., Mendel, F., Schläffer, M.: Branching heuristics in differential collision search with applications to SHA-512. In Cid, C., Rechberger, C., eds.: FSE 2014. Volume 8540 of LNCS., Springer, Heidelberg (March 2015) 473–488
64. Kircanski, A.: Analysis of boomerang differential trails via a SAT-based constraint solver URSA. In Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M., eds.: ACNS 15. Volume 9092 of LNCS., Springer, Heidelberg (June 2015) 331–349
65. Aumasson, J.P., Neves, S., Wilcox-O'Hearn, Z., Winnerlein, C.: BLAKE2: simpler, smaller, fast as MD5. Cryptology ePrint Archive, Report 2013/322 (2013)
66. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: Sha-3 proposal blake. Submission to NIST (2008)
67. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: New generation of memory-hard functions for password hashing and other applications. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). (2016) 292–302
68. Wagner, D.: A generalized birthday problem. In Yung, M., ed.: CRYPTO 2002. Volume 2442 of LNCS., Springer, Heidelberg (August 2002) 288–303

SUPPLEMENTARY MATERIAL

A Bit Conditions for Boomerang Top-Bottom Paths

We attach the bit conditions used in searching the boomerang distinguishers for three members of BLAKE.

- For BLAKE3, the bit conditions for two top-bottom paths with active selection combination $(i, j) = (12, 31)$ and $(31, 14)$ are give respectively in:

  `Blake3Top12Bottom31BitConditions.txt`
  `Blake3Top31Bottom14BitConditions.txt`

- For BLAKE2s, the bit conditions for the top-bottom path with active selection combination $(i, j) = (31, 30)$ are give in:

  `Blake2sTop31Bottom30BitConditions.txt`

- For BLAKE-256, the bit conditions for the top-bottom path with active selection combination $(i, j) = (31, 30)$ are give in:

  `Blake256Top31Bottom30BitConditions.txt`

Moreover, all the source code of our tool will be publicly available when this paper is accepted.

B Details of the BLAKE3, BLAKE2s and BLAKE-256 Keyed Permutations

BLAKE-256 and BLAKE2s use the same permutation σ_r's $(r = 0, \ldots, 9)$ defined in Table 12. BLAKE3 use a different σ_r $(r = 0, \ldots, 6)$ defined in Table 13. The round constants of BLAKE-256 forms a 16-word state $RC = (rc_0, \ldots, rc_{15})$ defined as Table 14.

Table 12: The permutation σ_r $(r = 0, \ldots, 9)$ used by BLAKE2 and BLAKE-256 round functions.

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

Table 13: The permutation σ_r $(r = 0, \ldots, 6)$ used by BLAKE3 round functions.

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	2	6	3	10	7	0	4	13	1	11	12	5	9	14	15	8
σ_2	3	4	10	12	13	2	7	14	6	5	9	0	11	15	8	1
σ_3	10	7	12	9	14	3	13	15	4	0	11	2	5	8	1	6
σ_4	12	13	9	11	15	10	14	8	7	2	5	3	0	1	6	4
σ_5	9	14	11	5	8	12	15	1	13	3	0	10	2	6	4	7
σ_6	11	15	5	0	1	9	8	6	14	10	2	12	3	4	7	13

Table 14: The round constants $RC = (rc_0, \ldots, rc_{15})$ used in the BLAKE-256 round functions.

RC	`0x243f6a88`, `0x85a308d3`, `0x13198a2e`, `0x3707344`, `0xa4093822`, `0x299f31d0`, `0x82efa98`, `0xec4e6c89`, `0x452821e6`, `0x38d01377`, `0xbe5466cf`, `0x34e90c6c`, `0xc0ac29b7`, `0xc97c50dd`, `0x3f84d5b5`, `0xb5470917`

C Accurate Signed Difference Modeling

In the accurate model, there is $\nabla x[i] \in \{\mathtt{u}, \mathtt{n}, \mathtt{0}, \mathtt{1}\}$. With the binary-variable encoding in Table 2, we are can let $\nabla x[i].\mathtt{xdiff}, \nabla x[i].\mathtt{sign} \in \mathcal{M}.\mathtt{var}$ and represent the bit conditions with MILP model constraints as Eq. (19).

$$\begin{aligned}
\mathcal{M}.\mathtt{con} \leftarrow \nabla x[i] \leftarrow \mathtt{n} &\Leftrightarrow \nabla x[i].(\mathtt{sign}, \mathtt{xdiff}) = (1, 1) \\
\mathcal{M}.\mathtt{con} \leftarrow \nabla x[i] \leftarrow \mathtt{u} &\Leftrightarrow \nabla x[i].(\mathtt{sign}, \mathtt{xdiff}) = (0, 1) \\
\mathcal{M}.\mathtt{con} \leftarrow \nabla x[i] \leftarrow \mathtt{1} &\Leftrightarrow \nabla x[i].(\mathtt{sign}, \mathtt{xdiff}) = (1, 0) \\
\mathcal{M}.\mathtt{con} \leftarrow \nabla x[i] \leftarrow \mathtt{0} &\Leftrightarrow \nabla x[i].(\mathtt{sign}, \mathtt{xdiff}) = (0, 0)
\end{aligned} \tag{19}$$

Similar to the approximate model, the accurate signed difference propagation of modular adds can be decomposed into half-adders and full-adders. The MILP model capturing the accurate signed difference of modular adds can be described as Algorithm 10 whose underlying models describing half-adders and full-adders are described as Algorithm 11 and Algorithm 12 respectively.

In XOR operations, both `sign` and `xdiff` are propagate linearly and can be described as Algorithm 13.

Algorithm 10: `modAddAcc`

Input : MILP model $\mathcal{M}$, the word signed differences ∇x and ∇y.
Output: The updated MILP model $\mathcal{M}$, the signed difference $\nabla z, \nabla c$.

$(\mathcal{M}, \nabla z[0], \nabla c[0]) \leftarrow$ `halfAdderAcc`$(\mathcal{M}, \nabla x[0], \nabla y[0])$
for $i = 1, \ldots, \omega - 1$ **do**
    $(\mathcal{M}, \nabla z[i], \nabla c[i]) \leftarrow$ `fullAdderAcc`$(\mathcal{M}, \nabla x[i], \nabla y[i], \nabla c[i-1])$
Return $(\mathcal{M}, \nabla z, \nabla c)$

Algorithm 11: `halfAdderAcc`

Input : MILP model $\mathcal{M}$, encoded signed differences $\nabla x[0], \nabla y[0]$ whose `sign` and `xdiff`'s are binary variables in $\mathcal{M}$.`var`.

Output: The updated MILP model $\mathcal{M}$ and the encoded $\nabla z[0], \nabla c[0]$ whose `sign` and `xdiff`'s are binary variables in $\mathcal{M}$.`var`.

// Declare variables

$\mathcal{M}$.`var` $\leftarrow \nabla z[0]$.`sign`, $\nabla z[0]$.`xdiff`, $\nabla c[0]$.`sign`, $\nabla c[0]$.`xdiff` as binaries.

// Define the column vector

$\boldsymbol{x} = (\nabla x[0]$.`sign`, $\nabla x[0]$.`xdiff`, $\nabla y[0]$.`sign`, $\nabla y[0]$.`xdiff`, $\nabla z[0]$.`sign`, $\nabla z[0]$.`xdiff`, $\nabla c[0]$.`sign`, $\nabla c[0]$.`xdiff`$)^T$

// Add constraints. Matrix $\hat{A}_h$ and column vector $\hat{\boldsymbol{b}}_h$ are defined as Eq. (20)

$\mathcal{M}$.`con` $\leftarrow \hat{A}_h \boldsymbol{x} + \hat{\boldsymbol{b}}_h \geq \boldsymbol{0}$

Return $(\mathcal{M}, \nabla z[0], \nabla c[0])$

Algorithm 12: `fullAdderAcc`

Input : MILP model $\mathcal{M}$, encoded signed differences $\nabla x[i], \nabla y[i], \nabla c[i-1]$ whose `sign` and `xdiff`'s are binary variables in $\mathcal{M}$.`var`.

Output: The updated MILP model $\mathcal{M}$ and the encoded $\nabla z[i], \nabla c[i]$ whose `sign` and `xdiff`'s are binary variables in $\mathcal{M}$.`var`.

// Declare variables

$\mathcal{M}$.`var` $\leftarrow \nabla z[i]$.`sign`, $\nabla z[i]$.`xdiff`, $\nabla c[i]$.`sign`, $\nabla c[i]$.`xdiff` as binaries.

// Define the column vector

$\boldsymbol{x} = (\nabla x[i]$.`sign`, $\nabla x[i]$.`xdiff`, $\nabla y[i]$.`sign`, $\nabla y[i]$.`xdiff`, $\nabla c[i-1]$.`sign`, $\nabla c[i-1]$.`xdiff`, $\nabla z[i]$.`sign`, $\nabla z[i]$.`xdiff`, $\nabla c[i]$.`sign`, $\nabla c[i]$.`xdiff`$)^T$

// Add constraints. Matrix $\hat{A}_f$ and column vector $\hat{\boldsymbol{b}}_f$ are defined as Eq. (21)

$\mathcal{M}$.`con` $\leftarrow \hat{A}_f \boldsymbol{x} + \hat{\boldsymbol{b}}_f \geq \boldsymbol{0}$

Return $(\mathcal{M}, \nabla z[i], \nabla c[i])$

$$\hat{A}_h = \begin{pmatrix} 1 & 0 & 1 & 0 & -1 & 0 & -2 & 0 \\ -1 & 0 & -1 & 0 & 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & -2 \\ 0 & -1 & 0 & -1 & 0 & -1 & -2 & 2 \\ 2 & 1 & 2 & 1 & 0 & -1 & -2 & -2 \\ 0 & 1 & -2 & -1 & 0 & 1 & 2 & -2 \\ -2 & -1 & 0 & 1 & 0 & 1 & 2 & -2 \\ 0 & -1 & -2 & 1 & 0 & -1 & 0 & 2 \\ -2 & 1 & 0 & -1 & 0 & -1 & 0 & 2 \\ 2 & -1 & 2 & -1 & 0 & 1 & -4 & 2 \end{pmatrix} \quad \hat{\boldsymbol{b}}_h = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 2 \\ 2 \\ 2 \\ 2 \\ 0 \end{pmatrix} \tag{20}$$

$$\hat{A}_f = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -2 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -2 \\ 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 2 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 & 2 & 1 & 2 & 2 \\ 0 & 1 & 2 & -1 & 2 & -1 & -2 & -1 & -2 & 2 \\ 0 & -1 & -2 & 1 & 0 & -1 & 0 & -1 & 2 & 2 \\ 0 & -1 & 0 & -1 & -2 & 1 & 0 & -1 & 2 & 2 \\ 0 & 1 & 0 & 1 & 0 & 1 & 2 & -1 & 2 & -2 \\ 0 & -1 & 2 & 1 & 0 & -1 & 0 & -1 & -2 & 2 \\ 0 & -1 & -2 & 1 & -2 & 1 & 2 & 1 & 2 & -2 \\ 0 & 1 & 0 & 1 & -2 & -1 & 0 & 1 & 2 & -2 \\ 0 & 1 & -2 & -1 & -2 & -1 & 2 & -1 & 2 & 2 \\ 0 & 1 & 0 & 1 & 2 & -1 & 0 & 1 & -2 & -2 \\ 0 & -1 & 0 & -1 & 2 & 1 & 0 & -1 & -2 & 2 \\ 0 & 1 & 2 & -1 & 0 & 1 & 0 & 1 & -2 & -2 \\ 0 & 1 & -2 & -1 & 0 & 1 & 0 & 1 & 2 & -2 \\ 0 & -1 & 2 & 1 & 2 & 1 & -2 & 1 & -2 & -2 \\ 0 & 1 & 0 & 1 & 0 & 1 & -2 & -1 & -2 & -2 \\ 0 & -1 & 0 & -1 & 0 & -1 & -2 & 1 & -2 & 2 \end{pmatrix} \quad \hat{\boldsymbol{b}}_f = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 2 \\ 2 \\ 2 \\ 0 \\ 2 \\ 2 \\ 2 \\ 0 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 4 \\ 4 \end{pmatrix} \tag{21}$$

Algorithm 13: `xorAcc`

Input : MILP model $\mathcal{M}$, the word signed differences ∇x and ∇y
Output: The updated MILP model $\mathcal{M}$, the signed difference ∇z

for $i = 0, \ldots, \omega - 1$ **do**
    $(\mathcal{M}, \nabla z[i].\texttt{xdiff}) \leftarrow \texttt{xorModel}(\mathcal{M}, \nabla x[i].\texttt{xdiff}, \nabla y[i].\texttt{xdiff})$
    $(\mathcal{M}, \nabla z[i].\texttt{sign}) \leftarrow \texttt{xorModel}(\mathcal{M}, \nabla x[i].\texttt{sign}, \nabla y[i].\texttt{sign})$
Return $(\mathcal{M}, \nabla z)$

D Digraph Construction

The computation process of the hG function in Eq. (8) can be captured with a digraph with 5 input word nodes $(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_3, \boldsymbol{m})$, 4 output word nodes $(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_3)$ and 4 operator nodes $(\boldsymbol{op}_0, \ldots, \boldsymbol{op}_3)$. The operator nodes can be defined as Ta-

ble 15 and the digraph can be computed as $(\mathcal{G}, \boldsymbol{y}_0, \ldots, \boldsymbol{y}_3) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{x}_0, \ldots, \boldsymbol{x}_3, \alpha, \beta)$ defined in Algorithm 14.

Table 15: The operator nodes for hG in Eq. (8)

Structure \ Operation	$\boldsymbol{op}_0$	$\boldsymbol{op}_3$	$\boldsymbol{op}_2$	$\boldsymbol{op}_1$
`type`	ADD	XOR	ADD	XOR
`rotBit`	0	α	0	β
`const`	0	0	0	0
`iWords`	$\{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{m}\}$	$\{\boldsymbol{y}_0, \boldsymbol{x}_3\}$	$\{\boldsymbol{y}_3, \boldsymbol{x}_2\}$	$\{\boldsymbol{y}_2, \boldsymbol{x}_1\}$
`oWord`	$\boldsymbol{y}_0$	$\boldsymbol{y}_3$	$\boldsymbol{y}_2$	$\boldsymbol{y}_1$

Algorithm 14: `hGDigraph`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, input word nodes $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_3, \boldsymbol{m} \in \mathcal{V}_w$
Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the output word nodes $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_3 \in \mathcal{V}_w$

1 Declare new word node $\boldsymbol{y}_0, \ldots \boldsymbol{y}_3$
2 Declare new operator nodes $\boldsymbol{op}_0, \ldots \boldsymbol{op}_3$ as in Table 15
3 Set the $\boldsymbol{y}_i$.`parent` $\leftarrow \{\boldsymbol{op}_i\}$ for $i = 0, \ldots, 3$
4 Update the word node set of $\mathcal{V}$ as $\mathcal{V}_w \leftarrow \mathcal{V}_w \bigcup \{\boldsymbol{y}_0, \ldots, \boldsymbol{y}_3\}$
5 Update the operator node set of $\mathcal{V}$ as $\mathcal{V}_o \leftarrow \mathcal{V}_o \bigcup \{\boldsymbol{op}_0, \ldots, \boldsymbol{op}_3\}$
6 Update the edge set

$$\mathcal{E} \leftarrow \mathcal{E} \bigcup \left\{ \begin{array}{l} \overrightarrow{(\boldsymbol{x}_0, \boldsymbol{op}_0)}, \overrightarrow{(\boldsymbol{x}_1, \boldsymbol{op}_0)}, \overrightarrow{(\boldsymbol{m}, \boldsymbol{op}_0)}, \overrightarrow{(\boldsymbol{op}_0, \boldsymbol{y}_0)} \\ \overrightarrow{(\boldsymbol{y}_0, \boldsymbol{op}_3)}, \overrightarrow{(\boldsymbol{x}_3, \boldsymbol{op}_3)}, \overrightarrow{(\boldsymbol{op}_3, \boldsymbol{y}_3)} \\ \overrightarrow{(\boldsymbol{y}_3, \boldsymbol{op}_2)}, \overrightarrow{(\boldsymbol{x}_2, \boldsymbol{op}_2)}, \overrightarrow{(\boldsymbol{op}_2, \boldsymbol{y}_2)} \\ \overrightarrow{(\boldsymbol{y}_2, \boldsymbol{op}_1)}, \overrightarrow{(\boldsymbol{x}_1, \boldsymbol{op}_1)}, \overrightarrow{(\boldsymbol{op}_1, \boldsymbol{y}_1)} \end{array} \right\}$$

7 **Return** $(\mathcal{G}(\mathcal{V}, \mathcal{E}), \boldsymbol{y}_0, \ldots, \boldsymbol{y}_3)$.

D.1 The Digraph Construction for BLAKE3

With the word nodes $\boldsymbol{V}^r = (\boldsymbol{v}_0^r, \ldots, \boldsymbol{v}_{15}^r)$ and $\boldsymbol{M} = (\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{15})$ corresponding to the V^r and M words, the update digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ capturing the computation $V^r \rightarrow TV^r$ can be generated by calling Algorithm 15 as $(\mathcal{G}, \boldsymbol{TV}^r) \leftarrow \texttt{b3DigUpdateTv}(\mathcal{G}, \boldsymbol{V}^r, \boldsymbol{M}, r)$. Similarly, the digraph from $TV^{r-0.5}$ to V^r can be captured with the digraph generated by Algorithm 16 as $(\mathcal{G}, \boldsymbol{V}^r) \leftarrow \texttt{b3DigUpdateV}(\mathcal{G}, \boldsymbol{TV}^{r-0.5}, \boldsymbol{M}, r)$. So the whole digraph from V^{r_0} to V^{r_1} can be constructed by calling $(\mathcal{G}, \boldsymbol{V}^{r_0}, \boldsymbol{V}^{r_1}) \leftarrow \texttt{b3Digraph}(r_0, r_1)$ in Algorithm 17.

D.2 The Digraph Construction for BLAKE2s

Same with BLAKE3, the digraph from V^{r_0} to V^{r_1} $(0 \le r_0 < r_1 \le 10)$ can be constructed by calling $(\mathcal{G}, \boldsymbol{V}^{r_0}, \boldsymbol{V}^{r_1}) \leftarrow \texttt{b2sDigraph}(r_0, r_1)$ in Algorithm 18.

Algorithm 15: `b3DigUpdateTv`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vectors $\boldsymbol{V}^r, \boldsymbol{M} \in \mathcal{V}_w$ corresponding to the state words in V^r and the message words in M respectively, the round number r

Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vector $\boldsymbol{TV}^r$ corresponding to the state words in TV^r

if $r \in \mathbb{Z}$ **then**
- $(\mathcal{G}, \boldsymbol{tv}^r_{0,4,8,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{0,4,8,12}, \boldsymbol{m}_{\sigma_r(0)}, 16, 12)$.
- $(\mathcal{G}, \boldsymbol{tv}^r_{1,5,9,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{1,5,9,13}, \boldsymbol{m}_{\sigma_r(2)}, 16, 12)$.
- $(\mathcal{G}, \boldsymbol{tv}^r_{2,6,10,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{2,6,10,14}, \boldsymbol{m}_{\sigma_r(4)}, 16, 12)$.
- $(\mathcal{G}, \boldsymbol{tv}^r_{3,7,11,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{3,7,11,15}, \boldsymbol{m}_{\sigma_r(6)}, 16, 12)$.

else
- $(\mathcal{G}, \boldsymbol{tv}^r_{0,5,10,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{0,5,10,15}, \boldsymbol{m}_{\sigma_{\lfloor r \rfloor}(8)}, 16, 12)$.
- $(\mathcal{G}, \boldsymbol{tv}^r_{1,6,11,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{1,6,11,12}, \boldsymbol{m}_{\sigma_{\lfloor r \rfloor}(10)}, 16, 12)$.
- $(\mathcal{G}, \boldsymbol{tv}^r_{2,7,8,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{2,7,8,13}, \boldsymbol{m}_{\sigma_{\lfloor r \rfloor}(12)}, 16, 12)$.
- $(\mathcal{G}, \boldsymbol{tv}^r_{3,4,9,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{3,4,9,14}, \boldsymbol{m}_{\sigma_{\lfloor r \rfloor}(14)}, 16, 12)$.

Assign $\boldsymbol{TV}^r \leftarrow (\boldsymbol{tv}^r_0, \ldots, \boldsymbol{tv}^r_{15})$

Return $(\mathcal{G}, \boldsymbol{TV}^r)$.

Algorithm 16: `b3DigUpdateV`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vectors $\boldsymbol{TV}^{r-0.5}, \boldsymbol{M} \in \mathcal{V}_w$ corresponding to the state words in $TV^{r-0.5}$ and the message words in M respectively, the round number r

Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vector $\boldsymbol{V}^r$ corresponding to the state words in V^r

if $r \notin \mathbb{Z}$ **then**
- $(\mathcal{G}, \boldsymbol{v}^r_{0,4,8,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{0,4,8,12}, \boldsymbol{m}_{\sigma_{\lfloor r \rfloor}(1)}, 8, 7)$.
- $(\mathcal{G}, \boldsymbol{v}^r_{1,5,9,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{1,5,9,13}, \boldsymbol{m}_{\sigma_{\lfloor r \rfloor}(3)}, 8, 7)$.
- $(\mathcal{G}, \boldsymbol{v}^r_{2,6,10,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{2,6,10,14}, \boldsymbol{m}_{\sigma_{\lfloor r \rfloor}(5)}, 8, 7)$.
- $(\mathcal{G}, \boldsymbol{v}^r_{3,7,11,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{3,7,11,15}, \boldsymbol{m}_{\sigma_{\lfloor r \rfloor}(7)}, 8, 7)$.

else
- $(\mathcal{G}, \boldsymbol{v}^r_{0,5,10,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{0,5,10,15}, \boldsymbol{m}_{\sigma_{r-1}(9)}, 8, 7)$.
- $(\mathcal{G}, \boldsymbol{v}^r_{1,6,11,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{1,6,11,12}, \boldsymbol{m}_{\sigma_{r-1}(11)}, 8, 7)$.
- $(\mathcal{G}, \boldsymbol{v}^r_{2,7,8,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{2,7,8,13}, \boldsymbol{m}_{\sigma_{r-1}(13)}, 8, 7)$.
- $(\mathcal{G}, \boldsymbol{v}^r_{3,4,9,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{3,4,9,14}, \boldsymbol{m}_{\sigma_{r-1}(15)}, 8, 7)$.

Assign $\boldsymbol{V}^r \leftarrow (\boldsymbol{v}^r_0, \ldots, \boldsymbol{v}^r_{15})$

Return $(\mathcal{G}, \boldsymbol{V}^r)$.

Algorithm 17: `b3Digraph`

Input : The starting round r_0 and the ending round r_1 where $0 \leq r_0 < r_1 \leq 7$the round number r

Output: The digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ capturing the computation from V^{r_0} to V^{r_1}, the word node vectors $\boldsymbol{V}^{r_0}, \boldsymbol{V}^{r_1}$ corresponding to the words in starting and ending states

Declare a word node vector $\boldsymbol{V}^{r_0} = (\boldsymbol{v}_0^{r_0}, \ldots, \boldsymbol{v}_{15}^{r_0})$ with $\boldsymbol{v}_i^{r_0}.\texttt{parent} = \phi$ for $i = 0, \ldots, 15$

Declare a word node vector $\boldsymbol{M} = (\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{15})$ with $\boldsymbol{m}_i.\texttt{parent} = \phi$ for $i = 0, \ldots, 15$

Initialize digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with the word node set of $\mathcal{V}$ defined as $\mathcal{V}_w \leftarrow \{\boldsymbol{v}_0^{r_0}, \ldots, \boldsymbol{v}_{15}^{r_0}, \boldsymbol{m}_0, \ldots, \boldsymbol{m}_{15}\}$

for $r = r_0 + 0.5, r_0 + 1, \ldots, r_1$ **do**

    Call Algorithm 15 as $(\mathcal{G}, \boldsymbol{TV}^{r-0.5}) \leftarrow \texttt{b3DigUpdateTv}(\mathcal{G}, \boldsymbol{V}^{r-0.5}, r - 0.5)$

    Call Algorithm 16 as $(\mathcal{G}, \boldsymbol{V}^{r}) \leftarrow \texttt{b3DigUpdateV}(\mathcal{G}, \boldsymbol{TV}^{r-0.5}, r)$

Assign $\boldsymbol{V}^{r_1} \leftarrow (\boldsymbol{v}_0^{r_1}, \ldots, \boldsymbol{v}_{15}^{r_1})$

Return $(\mathcal{G}(\mathcal{V}, \mathcal{E}), \boldsymbol{V}^{r_0}, \boldsymbol{V}^{r_1})$.

Algorithm 18 only differs with Algorithm 17 in the round function definitions: the `b2DigUpdateTv` and `b2DigUpdateV` used in BLAKE2s round functions are defined in Algorithm 19 and Algorithm 20 respectively.

D.3 The Digraph Construction for BLAKE-256

BLAKE-256 and BLAKE2s round functions share the same σ_r. The only difference is that, in BLAKE-256, the message word is XORed with a round constant word before taking part in the hG computations. The computation $V^r \rightarrow TV^r$ takes 4 constant-XORed message words denoted as $MV^r = (mv_0^r, \ldots, mv_3^r)$. For $TV^{r-0.5} \rightarrow V^r$, the 4 words are denoted as $MT^r = (mt_0^{r-0.5}, \ldots, mt_3^{r-0.5})$. So we define $(\mathcal{G}, \boldsymbol{MT}^r) \leftarrow \texttt{mtDigraph}(\mathcal{G}, \boldsymbol{M}, r)$ and $(\mathcal{G}, \boldsymbol{MV}^r) \leftarrow \texttt{mvDigraph}(\mathcal{G}, \boldsymbol{M}, r)$ as Algorithm 25 and Algorithm 24 respectively to capture the digraph changes. Then, the computations $V^r \rightarrow TV^r$ and $TV^{r-0.5} \rightarrow V^r$ can be captured by Algorithm 22 and Algorithm 23 respectively so the digraph from V^{r_0} to V^{r_1} can be acquired by calling Algorithm 21.

E MILP Models Used in the Paper

The commonly used MILP models capturing the XOR operation of two bits and the OR operation of n bits are already defined and used in almost all existing MILP-aided cryptanalysis results. We simply present them as Algorithm 26 and Algorithm 27.

E.1 The Model Capturing Newly Generated Bit Conditions

According to Proposition 1, given the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and the corresponding lookup table $\mathcal{T}$ mapping all word nodes to their signed differences, the bit con-

Algorithm 18: `b2sDigraph`

Input : The starting round r_0 and the ending round r_1 where $0 \leq r_0 < r_1 \leq 7$the round number r

Output: The digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ capturing the computation from V^{r_0} to V^{r_1}, the word node vectors $\boldsymbol{V}^{r_0}, \boldsymbol{V}^{r_1}$ corresponding to the words in starting and ending states

Declare a word node vector $\boldsymbol{V}^{r_0} = (\boldsymbol{v}_0^{r_0}, \ldots, \boldsymbol{v}_{15}^{r_0})$ with $\boldsymbol{v}_i^{r_0}.\texttt{parent} = \phi$ for $i = 0, \ldots, 15$

Declare a word node vector $\boldsymbol{M} = (\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{15})$ with $\boldsymbol{m}_i.\texttt{parent} = \phi$ for $i = 0, \ldots, 15$

Initialize digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with the word node set of $\mathcal{V}$ defined as $\mathcal{V}_w \leftarrow \{\boldsymbol{v}_0^{r_0}, \ldots, \boldsymbol{v}_{15}^{r_0}, \boldsymbol{m}_0, \ldots, \boldsymbol{m}_{15}\}$

for $r = r_0 + 0.5, r_0 + 1, \ldots, r_1$ **do**

    Call Algorithm 19 as $(\mathcal{G}, \boldsymbol{TV}^{r-0.5}) \leftarrow \texttt{b2sDigUpdateTv}(\mathcal{G}, \boldsymbol{V}^{r-0.5}, r - 0.5)$

    Call Algorithm 20 as $(\mathcal{G}, \boldsymbol{V}^{r}) \leftarrow \texttt{b2sDigUpdateV}(\mathcal{G}, \boldsymbol{TV}^{r-0.5}, r)$

Assign $\boldsymbol{V}^{r_1} \leftarrow (\boldsymbol{v}_0^{r_1}, \ldots, \boldsymbol{v}_{15}^{r_1})$

Return $(\mathcal{G}(\mathcal{V}, \mathcal{E}), \boldsymbol{V}^{r_0}, \boldsymbol{V}^{r_1})$.

Algorithm 19: `b2sDigUpdateTv`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vectors $\boldsymbol{V}^r, \boldsymbol{M} \in \mathcal{V}_w$ corresponding to the state words in V^r and the message words in M respectively, the round number r

Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vector $\boldsymbol{TV}^r$ corresponding to the state words in TV^r

Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$

if $r \in \mathbb{Z}$ **then**

    $(\mathcal{G}, \boldsymbol{tv}_{0,4,8,12}^r) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}_{0,4,8,12}^r, \boldsymbol{m}_{\sigma_{\underline{r}}(0)}, 16, 12)$.

    $(\mathcal{G}, \boldsymbol{tv}_{1,5,9,13}^r) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}_{1,5,9,13}^r, \boldsymbol{m}_{\sigma_{\underline{r}}(2)}, 16, 12)$.

    $(\mathcal{G}, \boldsymbol{tv}_{2,6,10,14}^r) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}_{2,6,10,14}^r, \boldsymbol{m}_{\sigma_{\underline{r}}(4)}, 16, 12)$.

    $(\mathcal{G}, \boldsymbol{tv}_{3,7,11,15}^r) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}_{3,7,11,15}^r, \boldsymbol{m}_{\sigma_{\underline{r}}(6)}, 16, 12)$.

else

    $(\mathcal{G}, \boldsymbol{tv}_{0,5,10,15}^r) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}_{0,5,10,15}^r, \boldsymbol{m}_{\sigma_{\underline{r}}(8)}, 16, 12)$.

    $(\mathcal{G}, \boldsymbol{tv}_{1,6,11,12}^r) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}_{1,6,11,12}^r, \boldsymbol{m}_{\sigma_{\underline{r}}(10)}, 16, 12)$.

    $(\mathcal{G}, \boldsymbol{tv}_{2,7,8,13}^r) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}_{2,7,8,13}^r, \boldsymbol{m}_{\sigma_{\underline{r}}(12)}, 16, 12)$.

    $(\mathcal{G}, \boldsymbol{tv}_{3,4,9,14}^r) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}_{3,4,9,14}^r, \boldsymbol{m}_{\sigma_{\underline{r}}(14)}, 16, 12)$.

Assign $\boldsymbol{TV}^r \leftarrow (\boldsymbol{tv}_0^r, \ldots, \boldsymbol{tv}_{15}^r)$

Return $(\mathcal{G}, \boldsymbol{TV}^r)$.

Algorithm 20: `b2sDigUpdateV`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vectors $\boldsymbol{TV}^{r-0.5}, \boldsymbol{M} \in \mathcal{V}_w$ corresponding to the state words in $TV^{r-0.5}$ and the message words in M respectively, the round number r

Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vector $\boldsymbol{V}^r$ corresponding to the state words in V^r

if $r \notin \mathbb{Z}$ **then**
  Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$
  $(\mathcal{G}, \boldsymbol{v}^r_{0,4,8,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{0,4,8,12}, \boldsymbol{m}_{\sigma_{\underline{r}}(1)}, 8, 7)$.
  $(\mathcal{G}, \boldsymbol{v}^r_{1,5,9,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{1,5,9,13}, \boldsymbol{m}_{\sigma_{\underline{r}}(3)}, 8, 7)$.
  $(\mathcal{G}, \boldsymbol{v}^r_{2,6,10,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{2,6,10,14}, \boldsymbol{m}_{\sigma_{\underline{r}}(5)}, 8, 7)$.
  $(\mathcal{G}, \boldsymbol{v}^r_{3,7,11,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{3,7,11,15}, \boldsymbol{m}_{\sigma_{\underline{r}}(7)}, 8, 7)$.
else
  Define $\underline{r} \leftarrow (r-1) \mod 10$
  $(\mathcal{G}, \boldsymbol{v}^r_{0,5,10,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{0,5,10,15}, \boldsymbol{m}_{\sigma_{\underline{r}}(9)}, 8, 7)$.
  $(\mathcal{G}, \boldsymbol{v}^r_{1,6,11,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{1,6,11,12}, \boldsymbol{m}_{\sigma_{\underline{r}}(11)}, 8, 7)$.
  $(\mathcal{G}, \boldsymbol{v}^r_{2,7,8,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{2,7,8,13}, \boldsymbol{m}_{\sigma_{\underline{r}}(13)}, 8, 7)$.
  $(\mathcal{G}, \boldsymbol{v}^r_{3,4,9,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{3,4,9,14}, \boldsymbol{m}_{\sigma_{\underline{r}}(15)}, 8, 7)$.
Assign $\boldsymbol{V}^r \leftarrow (\boldsymbol{v}^r_0, \ldots, \boldsymbol{v}^r_{15})$
Return $(\mathcal{G}, \boldsymbol{V}^r)$.

Algorithm 21: `b256Digraph`

Input : The starting round r_0 and the ending round r_1 where $0 \le r_0 < r_1 \le 7$the round number r

Output: The digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ capturing the computation from V^{r_0} to V^{r_1}, the word node vectors $\boldsymbol{V}^{r_0}, \boldsymbol{V}^{r_1}$ corresponding to the words in starting and ending states

Declare a word node vector $\boldsymbol{V}^{r_0} = (\boldsymbol{v}^{r_0}_0, \ldots, \boldsymbol{v}^{r_0}_{15})$ with $\boldsymbol{v}^{r_0}_i.\texttt{parent} = \phi$ for $i = 0, \ldots, 15$
Declare a word node vector $\boldsymbol{M} = (\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{15})$ with $\boldsymbol{m}_i.\texttt{parent} = \phi$ for $i = 0, \ldots, 15$
Initialize digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with the word node set of $\mathcal{V}$ defined as $\mathcal{V}_w \leftarrow \{\boldsymbol{v}^{r_0}_0, \ldots, \boldsymbol{v}^{r_0}_{15}, \boldsymbol{m}_0, \ldots, \boldsymbol{m}_{15}\}$
for $r = r_0 + 0.5, r_0 + 1, \ldots, r_1$ **do**
  Call Algorithm 22 as $(\mathcal{G}, \boldsymbol{TV}^{r-0.5}) \leftarrow \texttt{b256DigUpdateTv}(\mathcal{G}, \boldsymbol{V}^{r-0.5}, r-0.5)$
  Call Algorithm 23 as $(\mathcal{G}, \boldsymbol{V}^r) \leftarrow \texttt{b256DigUpdateV}(\mathcal{G}, \boldsymbol{TV}^{r-0.5}, r)$
Assign $\boldsymbol{V}^{r_1} \leftarrow (\boldsymbol{v}^{r_1}_0, \ldots, \boldsymbol{v}^{r_1}_{15})$
Return $(\mathcal{G}(\mathcal{V}, \mathcal{E}), \boldsymbol{V}^{r_0}, \boldsymbol{V}^{r_1})$.

Algorithm 22: `b256DigUpdateTv`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vectors $\boldsymbol{V}^r, \boldsymbol{M} \in \mathcal{V}_w$ corresponding to the state words in V^r and the message words in M respectively, the round number r

Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vector $\boldsymbol{TV}^r$ corresponding to the state words in TV^r

$(\mathcal{G}, \boldsymbol{MV}^r) \leftarrow \texttt{mvDigraph}(\mathcal{G}, \boldsymbol{M}, r)$ Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$

if $r \in \mathbb{Z}$ **then**

  $(\mathcal{G}, \boldsymbol{tv}^r_{0,4,8,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{0,4,8,12}, \boldsymbol{mv}_0, 16, 12)$.

  $(\mathcal{G}, \boldsymbol{tv}^r_{1,5,9,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{1,5,9,13}, \boldsymbol{mv}_1, 16, 12)$.

  $(\mathcal{G}, \boldsymbol{tv}^r_{2,6,10,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{2,6,10,14}, \boldsymbol{mv}_2, 16, 12)$.

  $(\mathcal{G}, \boldsymbol{tv}^r_{3,7,11,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{3,7,11,15}, \boldsymbol{mv}_3, 16, 12)$.

else

  $(\mathcal{G}, \boldsymbol{tv}^r_{0,5,10,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{0,5,10,15}, \boldsymbol{mv}_0, 16, 12)$.

  $(\mathcal{G}, \boldsymbol{tv}^r_{1,6,11,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{1,6,11,12}, \boldsymbol{mv}_1, 16, 12)$.

  $(\mathcal{G}, \boldsymbol{tv}^r_{2,7,8,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{2,7,8,13}, \boldsymbol{mv}_2, 16, 12)$.

  $(\mathcal{G}, \boldsymbol{tv}^r_{3,4,9,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{v}^r_{3,4,9,14}, \boldsymbol{mv}_3, 16, 12)$.

Assign $\boldsymbol{TV}^r \leftarrow (\boldsymbol{tv}^r_0, \ldots, \boldsymbol{tv}^r_{15})$

Return $(\mathcal{G}, \boldsymbol{TV}^r)$.

Algorithm 23: `b256DigUpdateV`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vectors $\boldsymbol{TV}^{r-0.5}, \boldsymbol{M} \in \mathcal{V}_w$ corresponding to the state words in $TV^{r-0.5}$ and the message words in M respectively, the round number r

Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vector $\boldsymbol{V}^r$ corresponding to the state words in V^r

$(\mathcal{G}, \boldsymbol{MT}^{r-0.5}) \leftarrow \texttt{mvDigraph}(\mathcal{G}, \boldsymbol{M}, r - 0.5)$

if $r \notin \mathbb{Z}$ **then**

  $(\mathcal{G}, \boldsymbol{v}^r_{0,4,8,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{0,4,8,12}, \boldsymbol{mt}^{r-0.5}_0, 8, 7)$.

  $(\mathcal{G}, \boldsymbol{v}^r_{1,5,9,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{1,5,9,13}, \boldsymbol{mt}^{r-0.5}_1, 8, 7)$.

  $(\mathcal{G}, \boldsymbol{v}^r_{2,6,10,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{2,6,10,14}, \boldsymbol{mt}^{r-0.5}_2, 8, 7)$.

  $(\mathcal{G}, \boldsymbol{v}^r_{3,7,11,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{3,7,11,15}, \boldsymbol{mt}^{r-0.5}_3, 8, 7)$.

else

  $(\mathcal{G}, \boldsymbol{v}^r_{0,5,10,15}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{0,5,10,15}, \boldsymbol{mt}^{r-0.5}_0, 8, 7)$.

  $(\mathcal{G}, \boldsymbol{v}^r_{1,6,11,12}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{1,6,11,12}, \boldsymbol{mt}^{r-0.5}_1, 8, 7)$.

  $(\mathcal{G}, \boldsymbol{v}^r_{2,7,8,13}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{2,7,8,13}, \boldsymbol{mt}^{r-0.5}_2, 8, 7)$.

  $(\mathcal{G}, \boldsymbol{v}^r_{3,4,9,14}) \leftarrow \texttt{hGDigraph}(\mathcal{G}, \boldsymbol{tv}^{r-0.5}_{3,4,9,14}, \boldsymbol{mt}^{r-0.5}_3, 8, 7)$.

Assign $\boldsymbol{V}^r \leftarrow (\boldsymbol{v}^r_0, \ldots, \boldsymbol{v}^r_{15})$

Return $(\mathcal{G}, \boldsymbol{V}^r)$.

Algorithm 24: `mtDigraph`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vector $\boldsymbol{M} \in \mathcal{V}_w$ corresponding to the the message words in M, the round number r

Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 4-word-node vector $\boldsymbol{MT}^r$ message words XORed with constants

for $i = 0, \ldots, 3$ **do**
  Declare a word node $\boldsymbol{mt}_i^r$
  if $r \in \mathbb{Z}$ **then**
    Define $\underline{r} \leftarrow r \mod 10$
    Declare an operator node $\boldsymbol{op}_i$ satisfying

$$\boldsymbol{op}_i.\begin{Bmatrix} \texttt{type} = \text{XOR}, \texttt{rotBit} = 0, \texttt{const} = rc_{\sigma_{\underline{r}}(2i)} \\ \texttt{iWords} = \{\boldsymbol{m}_{\sigma_{\underline{r}}(2i+1)}\}, \texttt{oWord} = \boldsymbol{mt}_i^r \end{Bmatrix}$$

    Assign $\boldsymbol{mt}_i^r.\texttt{parent} \leftarrow \{\boldsymbol{op}_i\}$
    Update the word node set of $\mathcal{V}$ as $\mathcal{V}_w \leftarrow \mathcal{V}_w \bigcup \{\boldsymbol{mt}_i^r\}$
    Update the operator node set of $\mathcal{V}$ as $\mathcal{V}_o \leftarrow \mathcal{V}_o \bigcup \{\boldsymbol{op}_i\}$
    Update the edges set as

$$\mathcal{E} \leftarrow \mathcal{E} \bigcup \left\{ \overrightarrow{(\boldsymbol{m}_{\sigma_{\underline{r}}(2i+1)}, \boldsymbol{op}_i)}, \overrightarrow{(\boldsymbol{op}_i, \boldsymbol{mt}_i^r)} \right\}$$

  else
    Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$
    Declare an operator node $\boldsymbol{op}_i$ satisfying:

$$\boldsymbol{op}_i.\begin{Bmatrix} \texttt{type} = \text{XOR}, \texttt{rotBit} = 0, \texttt{const} = rc_{\sigma_{\underline{r}}(2i+8)}, \\ \texttt{iWords} = \{\boldsymbol{m}_{\sigma_{\underline{r}}(2i+1+8)}\}, \texttt{oWord} = \boldsymbol{mt}_i^r \end{Bmatrix}$$

    Assign $\boldsymbol{mt}_i^r.\texttt{parent} \leftarrow \{\boldsymbol{op}_i\}$
    Update the word node set of $\mathcal{V}$ as $\mathcal{V}_w \leftarrow \mathcal{V}_w \bigcup \{\boldsymbol{mt}_i^r\}$
    Update the operator node set of $\mathcal{V}$ as $\mathcal{V}_o \leftarrow \mathcal{V}_o \bigcup \{\boldsymbol{op}_i\}$
    Update the edges set as

$$\mathcal{E} \leftarrow \mathcal{E} \bigcup \left\{ \overrightarrow{(\boldsymbol{m}_{\sigma_{\underline{r}}(2i+1+8)}, \boldsymbol{op}_i)}, \overrightarrow{(\boldsymbol{op}_i, \boldsymbol{mt}_i^r)} \right\}$$

Define $\boldsymbol{MT}^r \leftarrow (\boldsymbol{mt}_0^r, \ldots, \boldsymbol{mt}_3^r)$
Return $(\mathcal{G}, \boldsymbol{MT}^r)$.

Algorithm 25: `mvDigraph`

Input : The initial digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 16-word-node vector $\boldsymbol{M} \in \mathcal{V}_w$ corresponding to the the message words in M, the round number r

Output: The updated digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the 4-word-node vector $\boldsymbol{MV}^r$ message words XORed with constants

for $i = 0, \ldots, 3$ **do**
  Declare a word node $\boldsymbol{mv}_i^r$
  if $r \in \mathbb{Z}$ **then**
    Define $\underline{r} \leftarrow r \mod 10$
    Declare a operator node $\boldsymbol{op}_i$ satisfying

$$\boldsymbol{op}_i.\begin{Bmatrix} \texttt{type} = \text{XOR}, \texttt{rotBit} = 0, \texttt{const} = rc_{\sigma_{\underline{r}}(2i+1)} \\ \texttt{iWords} = \{\boldsymbol{m}_{\sigma_{\underline{r}}(2i)}\}, \texttt{oWord} = \boldsymbol{mv}_i^r \end{Bmatrix}$$

    Assign $\boldsymbol{mv}_i^r.\texttt{parent} \leftarrow \{\boldsymbol{op}_i\}$
    Update the word node set of $\mathcal{V}$ as $\mathcal{V}_w \leftarrow \mathcal{V}_w \bigcup \{\boldsymbol{mv}_i^r\}$
    Update the operator node set of $\mathcal{V}$ as $\mathcal{V}_o \leftarrow \mathcal{V}_o \bigcup \{\boldsymbol{op}_i\}$
    Update the edges set as

$$\mathcal{E} \leftarrow \mathcal{E} \bigcup \left\{ \overrightarrow{(\boldsymbol{m}_{\sigma_{\underline{r}}(2i)}, \boldsymbol{op}_i)}, \overrightarrow{(\boldsymbol{op}_i, \boldsymbol{mv}_i^r)} \right\}$$

  else
    Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$
    Declare a operator node $\boldsymbol{op}_i$ satisfying:

$$\boldsymbol{op}_i.\begin{Bmatrix} \texttt{type} = \text{XOR}, \texttt{rotBit} = 0, \texttt{const} = rc_{\sigma_{\underline{r}}(2i+1+8)} \\ \texttt{iWords} = \{\boldsymbol{m}_{\sigma_{\underline{r}}(2i+8)}\}, \texttt{oWord} = \boldsymbol{mv}_i^r \end{Bmatrix}$$

    Assign $\boldsymbol{mv}_i^r.\texttt{parent} \leftarrow \{\boldsymbol{op}_i\}$
    Update the word node set of $\mathcal{V}$ as $\mathcal{V}_w \leftarrow \mathcal{V}_w \bigcup \{\boldsymbol{mv}_i^r\}$
    Update the operator node set of $\mathcal{V}$ as $\mathcal{V}_o \leftarrow \mathcal{V}_o \bigcup \{\boldsymbol{op}_i\}$
    Update the edges set as

$$\mathcal{E} \leftarrow \mathcal{E} \bigcup \left\{ \overrightarrow{(\boldsymbol{m}_{\sigma_{\underline{r}}(2i+8)}, \boldsymbol{op}_i)}, \overrightarrow{(\boldsymbol{op}_i, \boldsymbol{mv}_i^r)} \right\}$$

Define $\boldsymbol{MV}^r \leftarrow (\boldsymbol{mv}_0^r, \ldots, \boldsymbol{mv}_3^r)$
Return $(\mathcal{G}, \boldsymbol{MV}^r)$.

Algorithm 26: `xorModel`

Input : The initial MILP model $\mathcal{M}$, the binary variables $x, y \in \mathcal{M}.\texttt{var}$
Output: The updated MILP model $\mathcal{M}$, the binary variable $z \in \mathcal{M}.\texttt{var}$ satirfying $z = x \oplus y$

1 Declare the variable $\mathcal{M}.\texttt{var} \leftarrow z$ as binary.
2 Update $\mathcal{M}$ by adding the constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} x + y - z \geq 0 \\ x - y + z \geq 0 \\ -x + y + z \geq 0 \\ x + y + z \leq 2 \end{cases}$$

Return $(\mathcal{M}, z)$.

Algorithm 27: `orModel`

Input : The initial MILP model $\mathcal{M}$, the binary variables $x_1, \ldots, x_n \in \mathcal{M}.\texttt{var}$
Output: The updated MILP model $\mathcal{M}$, the binary variable $y \in \mathcal{M}.\texttt{var}$

1 Declare the variable $\mathcal{M}.\texttt{var} \leftarrow y$ as binary.
2 Update $\mathcal{M}$ by adding the constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} y \geq x_i, i = 1, \ldots, n \\ y \leq \sum_{i=1}^{n} x_i \end{cases}$$

Return $(\mathcal{M}, y)$.

ditions generated by each modular add operation can be captured with the summations of the binary variables $\theta_0, \ldots, \theta_{\omega-2}$ in Eq. (14). Such θ's can be stored in a set $\mathcal{S}$ generated by Algorithm 28 as $(\mathcal{M}, \mathcal{S}) \leftarrow \texttt{betaSet}(\mathcal{M}, \mathcal{G}, \mathcal{T})$. In this way, the objective function in Eq. (15) can be equivalently represented as

$$\mathcal{M}.\texttt{obj} \leftarrow \min \sum_{\theta \in \mathcal{S}} \theta \tag{22}$$

Algorithm 28: `betaSet`

Input : The initial MILP model $\mathcal{M}$, the digraph $\mathcal{G}$ and the lookup table $\mathcal{T}$
Output: The updated MILP model $\mathcal{M}$, the set of binary variables $\mathcal{B} \subseteq \mathcal{M}.\texttt{var}$

1 Initialize an empty set $\mathcal{S} = \phi$
2 **for** $\boldsymbol{op} \in \mathcal{V}_+$ **do**
3 Let $\boldsymbol{op}.\texttt{iWords} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, $\boldsymbol{op}.\texttt{oWord} = \boldsymbol{y}$, refer to $\mathcal{T}$ and acquire the corresponding signed differences $\nabla x_1, \ldots, \nabla x_n, \nabla y$
4 **for** $i = 0, \ldots, 30$ **do**
5 $(\mathcal{M}, \theta_i) \leftarrow \texttt{orModel}(\mathcal{M}, \{\nabla y[i].\texttt{xdiff}, x_1[i].\texttt{xdiff}, \ldots, \nabla x_n[i].\texttt{xdiff}\})$
6 Update $\mathcal{S} \leftarrow \mathcal{S} \bigcup \{\theta_i\}$
7 **Return** $(\mathcal{M}, \mathcal{S})$.

E.2 The MILP Model Constraints for Boomerang Intersection

According to Section 4.5, in order to make compatible boomerang intersections, additional constraints should be added for all XORed bits related to the intersecting state bits in the form of Eq. (16) and Eq. (17) respectively. Adding the 0-AND constraint, the constraints imposed to the boomerang intersecting state can be added to the MILP model by calling Algorithm 29. As can be seen, the constraints in line 4 reflects the traditional 0-AND constraint; the constraints in line 10-13 are the constraints in Eq. (16) and those in line 19-22 are constraints in Eq. (17).

E.3 MILP Models for BLAKE3

The approximate signed difference propagation $\nabla V^r \rightarrow \nabla TV^r$ can be described as the MILP model $(\mathcal{M}, \nabla TV^r) \leftarrow \texttt{b3UpdateTv}(\mathcal{M}, \nabla V^r, r)$ in Algorithm 34. $\nabla TV^{r-0.5} \rightarrow \nabla V^r$, the MILP can be constructed in Algorithm 35 as $(\mathcal{M}, \nabla V^r) \leftarrow \texttt{b3UpdateV}(\mathcal{M}, \nabla TV^{r-0.5}, r)$. For $0 \leq r_0 < r_1 \leq 7$, the differential propagation $\nabla V^{r_0} \rightarrow \nabla V^{r_1}$ can be captured with the common model generated by Algorithm 33 as $\mathcal{M} \leftarrow \texttt{b3CommonModel}(r_0, r_1)$.

The $\mathcal{M}^t$ model for $\mathcal{T}_i^t$ can be constructed by calling Algorithm 30. The $\mathcal{M}^b$ model for $\mathcal{T}_j^b$ can be constructed by calling Algorithm 31. For each (i, j) setting, after solving the corresponding $\mathcal{M}^t$ and $\mathcal{M}^b$, we have acquired a concrete

Algorithm 29: `InterSecConstr`

Input : The initial MILP model $\mathcal{M}$, the top and bottom digraphs $\mathcal{G}^t(\mathcal{V}^t, \mathcal{E}^t), \mathcal{G}^b(\mathcal{V}^b, \mathcal{E}^b)$, the lookup tables for top and bottom paths $\mathcal{T}^t, \mathcal{T}^b$, the intersecting word nodes $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\} = \mathcal{V}^t \bigcap \mathcal{V}^b$

Output: The updated MILP model $\mathcal{M}$

for $\ell = 1, \ldots, n$ **do**
  Identify the differences $\nabla^t v_\ell$ and $\nabla^b v_\ell$ by referring to $\mathcal{T}^t$ and $\mathcal{T}^b$ respectively
  for $i = 0, \ldots, \omega - 1$ **do**
    Add the constraint $\mathcal{M}.\texttt{con} \leftarrow \nabla^t v_\ell[i].\texttt{xorDiff} + \nabla^b v_\ell[i].\texttt{xorDiff} \leq 1$
  Call Algorithm 8 as $\mathcal{R}^t_{\boldsymbol{v_\ell}} \leftarrow \texttt{relatedOpt}(\mathcal{G}^t, \boldsymbol{v}_\ell)$
  for $\boldsymbol{op} \in \mathcal{R}^t_{\boldsymbol{v_\ell}}$ **do**
    For $\boldsymbol{op}.\texttt{iWords} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_s\}$ and $\boldsymbol{op}.\texttt{oWord} = \boldsymbol{y}$, identify the signed differences $\nabla^t x_1, \ldots, \nabla^t x_s, \nabla^t y$ by referring to $\mathcal{T}^t$
    Define $\gamma = 1$ if $\boldsymbol{op}.\texttt{type} =$ ADD; or $\gamma = \omega$ if $\boldsymbol{op}.\texttt{type} =$ XOR
    for $i = 0, \ldots, \gamma - 1$ **do**
      Compute the offset $\underline{\texttt{i}} \leftarrow (i + \boldsymbol{op}.\texttt{rotBit}) \mod \omega$
      Define the binary variable set $\mathcal{S}^t \leftarrow \{\nabla^t x_1[\underline{\texttt{i}}].\texttt{xdiff}, \ldots, \nabla^t x_\ell[\underline{\texttt{i}}].\texttt{xdiff}, \nabla^t y[i].\texttt{xdiff}\}$
      Call Algorithm 27 as $(\mathcal{M}, a^t) \leftarrow \texttt{orModel}(\mathcal{M}, \mathcal{S}^t)$
      Add the constraint $\mathcal{M}.\texttt{con} \leftarrow a^t + \nabla^b v_\ell[i].\texttt{xdiff} \leq 1$
  Call Algorithm 8 as $\mathcal{R}^b_{\boldsymbol{v_\ell}} \leftarrow \texttt{relatedOpt}(\mathcal{G}^b, \boldsymbol{v}_\ell)$
  for $\boldsymbol{op} \in \mathcal{R}^b_{\boldsymbol{v_\ell}}$ **do**
    For $\boldsymbol{op}.\texttt{iWords} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_s\}$ and $\boldsymbol{op}.\texttt{oWord} = \boldsymbol{y}$, identify the signed differences $\nabla^b x_1, \ldots, \nabla^b x_s, \nabla^b y$ by referring to $\mathcal{T}^b$
    Define $\gamma = 1$ if $\boldsymbol{op}.\texttt{type} =$ ADD; or $\gamma = \omega$ if $\boldsymbol{op}.\texttt{type} =$ XOR
    for $i = 0, \ldots, \gamma - 1$ **do**
      Compute the offset $\underline{\texttt{i}} \leftarrow (i + \boldsymbol{op}.\texttt{rotBit}) \mod \omega$
      Define the binary variable set $\mathcal{S}^b \leftarrow \{\nabla^b x_1[\underline{\texttt{i}}].\texttt{xdiff}, \ldots, \nabla^b x_\ell[\underline{\texttt{i}}].\texttt{xdiff}, \nabla^b y[i].\texttt{xdiff}\}$
      Call Algorithm 27 as $(\mathcal{M}, a^b) \leftarrow \texttt{orModel}(\mathcal{M}, \mathcal{S}^b)$
      Add the constraint $\mathcal{M}.\texttt{con} \leftarrow a^b + \nabla^t v_\ell[i].\texttt{xdiff} \leq 1$
Return $(\mathcal{M}, \mathcal{G}, \mathcal{T}, \mathcal{B})$.

approximate signed difference of $\nabla^t M$ and $\nabla^b M$, denoted as $\Theta^t, \Theta^b \in \{\mathtt{u}, \mathtt{n}, =\}^{32}$ respectively. The $\mathcal{M}^c$ can then be constructed as Algorithm 32. The additional constraints in Section 4.5 are added by calling Algorithm 29. The objective functions of all 3 models are defined as Eq. (22) with the aid of Algorithm 34.

Algorithm 30: `b3TopModel`

Input : The active bit position i
Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the lookup table $\mathcal{T}$, the binary variable set $\mathcal{S}$

1 Initialize a MILP model $\mathcal{M}$
2 Call Algorithm 33 as $(\mathcal{M}, \mathcal{G}, \mathcal{T}) \leftarrow$ `b3CommonModel`$(\mathcal{M}, 0, 1.5)$
3 Update the model by adding constraints:

$$\mathcal{M}.\mathtt{con} \leftarrow \begin{cases} \nabla v_\ell^{1.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \bigwedge (\ell, k) \neq (0, i) \\ \nabla m_\ell[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \bigwedge (\ell, k) \neq (1, i) \\ \nabla m_1[i].\mathtt{xdiff} = \nabla v_0^{1.5}[i].\mathtt{xdiff} = 1 \\ \nabla v_0^{1.5}[i].\mathtt{sign} + \nabla m_1[i].\mathtt{sign} = 1 \end{cases}$$

4 Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}) \leftarrow$ `betaSet`$(\mathcal{M}, \mathcal{G}, \mathcal{T})$
5 Set the objective function of $\mathcal{M}$ as Eq. (22)
6 **Return** $(\mathcal{M}, \mathcal{G}, \mathcal{T}, \mathcal{S})$.

E.4 MILP Models for BLAKE2s

Similar to BLAKE3, we define the common model for BLAKE2s as Algorithm 39 where the underlying `b2sUpdateTv` and `b2sUpdateV` are defined as Algorithm 40 and Algorithm 41 respectively. $\mathcal{M}^t$ covering $V^{2.5} \to V^{4.5}$ with $\Delta^t V^{4.5} = 0$ is constructed as Algorithm 36. $\mathcal{M}^b$ covering $V^{9.5} \to V^{10.5}$ with $\Delta^t V^{9.5} = 0$ is constructed as Algorithm 37. The solution of $\mathcal{M}^t$ and $\mathcal{M}^b$ gives $\nabla M^t = \Theta^t$ and $\nabla M^b = \Theta^b$ with which the connect model $\mathcal{M}^c$ can be constructed with Algorithm 38 covering $V^{5.5} \to V^{8.5}$ with $(\Delta^t V^{5.5}, \Delta^b V^{8.5}) = (0, 0)$.

E.5 MILP Models for BLAKE-256

Different from BLAKE2s, the intermediate message blocks MT^r and MV^r are computed by XORing message words with constants and are used in the computation of $V^{r+0.5}$ and TV^r respectively. Therefore, the ∇MT^r and ∇MV^r are captured with the MILP models in Algorithm 48 and Algorithm 49 respectively. The common model can then be defined as Algorithm 45 where `b256UpdateTv` and `b256UpdateV` are defined as Algorithm 46 and Algorithm 47 respectively. $\mathcal{M}^t$ covering $V^{2.5} \to V^{4.5}$ with $\Delta^t V^{4.5} = 0$ is constructed as Algorithm 42. $\mathcal{M}^b$

Algorithm 31: `b3BottomModel`

Input : The active bit position j

Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the lookup table $\mathcal{T}$, the binary variable set $\mathcal{S}$

Initialize a MILP model $\mathcal{M}$

Call Algorithm 33 as $(\mathcal{M}, \mathcal{G}, \mathcal{T}) \leftarrow \texttt{b3CommonModel}(\mathcal{M}, 6, 7)$

Update the model by adding constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \nabla v_\ell^6[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \\ \nabla m_\ell[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \bigwedge (\ell, k) \neq (9, j) \\ \nabla m_9[j].\texttt{xdiff} = 1 \end{cases}$$

Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}) \leftarrow \texttt{betaSet}(\mathcal{M}, \mathcal{G}, \mathcal{T})$

Set the objective function of $\mathcal{M}$ as Eq. (22)

Return $(\mathcal{M}, \mathcal{G}, \mathcal{T}, \mathcal{S})$.

Algorithm 32: `b3ConnectModel`

Input : The message block differences $\Theta^t, \Theta^b \in \{\texttt{u}, \texttt{n}, =\}^{32}$

Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}^t(\mathcal{V}^t, \mathcal{E}^t)$ and $\mathcal{G}^b(\mathcal{V}^b, \mathcal{E}^b)$, the lookup table $\mathcal{T}^t$ and $\mathcal{T}^b$, the binary variable set $\mathcal{S}$

Initialize a MILP model $\mathcal{M}$

Call Algorithm 33 as $(\mathcal{M}, \mathcal{G}^t, \mathcal{T}^t) \leftarrow \texttt{b3CommonModel}(\mathcal{M}, 2.5, 3.5)$

Call Algorithm 33 as $(\mathcal{M}, \mathcal{G}^b, \mathcal{T}^b) \leftarrow \texttt{b3CommonModel}(\mathcal{M}, 3.5, 5)$

Update the model by adding constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \nabla^t v_\ell^{2.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \\ \nabla^b v_\ell^5[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \bigwedge (\ell, k) \neq (0, j) \\ \nabla^t M \leftarrow \Theta^t \\ \nabla^b M \leftarrow \Theta^b \\ \nabla^b v_0^5[j].\texttt{xdiff} = 1 \\ \nabla^b v_0^5[j].\texttt{sign} + \nabla m_9[j].\texttt{sign} = 1 \end{cases}$$

Call Algorithm 29 as $\mathcal{M} \leftarrow \texttt{InterSecConstr}(\mathcal{M}, \mathcal{G}^t, \mathcal{G}^b, \mathcal{T}^t, \mathcal{T}^b, \boldsymbol{V}^{3.5})$

Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}^t) \leftarrow \texttt{betaSet}(\mathcal{M}, \mathcal{G}^t, \mathcal{T}^t)$

Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}^b) \leftarrow \texttt{betaSet}(\mathcal{M}, \mathcal{G}^b, \mathcal{T}^b)$

Define $\mathcal{S} \leftarrow \mathcal{S}^t \bigcup \mathcal{S}^b$

Set the objective function of $\mathcal{M}$ as Eq. (22)

Return $(\mathcal{M}, \mathcal{G}^t, \mathcal{G}^b, \mathcal{T}^t, \mathcal{T}^b, \mathcal{S})$.

Algorithm 33: `b3CommonModel`

Input : The initial MILP model $\mathcal{M}$, the starting r_0 and the ending round r_1 with $r_0 < r_1$

Output: The updated MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ capturing the computation from V^{r_0} to V^{r_1}, the lookup table $\mathcal{T}$ mapping word nodes to the corresponding signed differences

Call Algorithm 17 and acquire the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E}) \leftarrow \texttt{b3Digraph}(r_0, r_1)$.

Initialize an empty lookup table $\mathcal{T}$

Initialize V^{r_0}, M as

$$\mathcal{M}.\texttt{var} \leftarrow \begin{cases} \nabla v_i^{r_0}[j].\texttt{sign}, \nabla v_i^{r_0}[j].\texttt{xdiff} \text{ as binary} \\ \nabla m_i[j].\texttt{sign}, \nabla m_i[j].\texttt{xdiff} \text{ as binary} \end{cases} i = 0, \ldots, 15;\ j = 0, \ldots, 31$$

Assign $\mathcal{T}[\boldsymbol{v}_i^{r_0}] \leftarrow \nabla v_i^{r_0}$ for $i = 0, \ldots, 15$

Assign $\mathcal{T}[\boldsymbol{m}_i] \leftarrow \nabla m_i$ for $i = 0, \ldots, 15$

for $r = r_0 + 0.5, r_0 + 1, \ldots, r_1$ **do**

  $(\mathcal{M}, \nabla TV^{r-0.5}) \leftarrow \texttt{b3UpdateTv}(\mathcal{M}, \nabla V^{r-0.5}, \nabla M, r - 0.5)$

  Assign $\mathcal{T}[\boldsymbol{tv}_i^{r-0.5}] \leftarrow \nabla tv_i^{r-0.5}$ for $i = 0, \ldots, 15$

  $(\mathcal{M}, \nabla V^r) \leftarrow \texttt{b3UpdateV}(\mathcal{M}, \nabla TV^{r-0.5}, \nabla M, r)$

  Assign $\mathcal{T}[\boldsymbol{v}_i^r] \leftarrow \nabla v_i^r$ for $i = 0, \ldots, 15$

Return $(\mathcal{M}, \mathcal{G}, \mathcal{T})$.

Algorithm 34: `b3UpdateTv`

Input : The initial MILP model $\mathcal{M}$, the signed differences of intermediate state ∇V^r and message block ∇M, the round number r

Output: The updated MILP model $\mathcal{M}$, the signed differences of intermediate state ∇TV^r

if $r \in \mathbb{Z}$ **then**

  $(\mathcal{M}, \nabla tv_{0,4,8,12}^r) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v_{0,4,8,12}^r, \nabla m_{\sigma_r(0)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{1,5,9,13}^r) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v_{1,5,9,13}^r, \nabla m_{\sigma_r(2)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{2,6,10,14}^r) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v_{2,6,10,14}^r, \nabla m_{\sigma_r(4)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{3,7,11,15}^r) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v_{3,7,11,15}^r, \nabla m_{\sigma_r(6)}, 16, 12)$.

else

  $(\mathcal{M}, \nabla tv_{0,5,10,15}^r) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v_{0,5,10,15}^r, \nabla m_{\sigma_{\lfloor r \rfloor}(8)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{1,6,11,12}^r) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v_{1,6,11,12}^r, \nabla m_{\sigma_{\lfloor r \rfloor}(10)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{2,7,8,13}^r) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v_{2,7,8,13}^r, \nabla m_{\sigma_{\lfloor r \rfloor}(12)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{3,4,9,14}^r) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v_{3,4,9,14}^r, \nabla m_{\sigma_{\lfloor r \rfloor}(14)}, 16, 12)$.

Assign $\nabla TV^r \leftarrow (\nabla tv_0^r, \ldots, \nabla tv_{15}^r)$

Return $(\mathcal{M}, \nabla TV^r)$.

Algorithm 35: `b3UpdateV`

Input : The initial MILP model $\mathcal{M}$, the signed differences of intermediate state $\nabla TV^{r-0.5}$ and message block ∇M, the round number r

Output: The updated MILP model $\mathcal{M}$, the signed differences of intermediate state ∇V^r

if $r \notin \mathbb{Z}$ **then**

  $(\mathcal{M}, \nabla v^r_{0,4,8,12}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{0,4,8,12}, \nabla m_{\sigma_{\lfloor r \rfloor}(1)}, 8, 7)$.

  $(\mathcal{M}, \nabla v^r_{1,5,9,13}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{1,5,9,13}, \nabla m_{\sigma_{\lfloor r \rfloor}(3)}, 8, 7)$.

  $(\mathcal{M}, \nabla v^r_{2,6,10,14}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{2,6,10,14}, \nabla m_{\sigma_{\lfloor r \rfloor}(5)}, 8, 7)$.

  $(\mathcal{M}, \nabla v^r_{3,7,11,15}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{3,7,11,15}, \nabla m_{\sigma_{\lfloor r \rfloor}(7)}, 8, 7)$.

else

  $(\mathcal{M}, \nabla v^r_{0,5,10,15}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{0,5,10,15}, \nabla m_{\sigma_{r-1}(9)}, 8, 7)$.

  $(\mathcal{M}, \nabla v^r_{1,6,11,12}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{1,6,11,12}, \nabla m_{\sigma_{r-1}(11)}, 8, 7)$.

  $(\mathcal{M}, \nabla v^r_{2,7,8,13}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{2,7,8,13}, \nabla m_{\sigma_{r-1}(13)}, 8, 7)$.

  $(\mathcal{M}, \nabla v^r_{3,4,9,14}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{3,4,9,14}, \nabla m_{\sigma_{r-1}(15)}, 8, 7)$.

Assign $\nabla V^r \leftarrow (\nabla v^r_0, \ldots, \nabla v^r_{15})$

Return $(\mathcal{M}, \nabla V^r)$.

Algorithm 36: `b2sTopModel`

Input : The active bit position i

Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the lookup table $\mathcal{T}$, the binary variable set $\mathcal{S}$

Initialize a MILP model $\mathcal{M}$

Call Algorithm 39 as $(\mathcal{M}, \mathcal{G}, \mathcal{T}) \leftarrow \texttt{b2sCommonModel}(\mathcal{M}, 2.5, 4.5)$

Update the model by adding constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \nabla v^{4.5}_{\ell}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \\ \nabla m_{\ell}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \bigwedge (\ell, k) \neq (5, i) \\ \nabla m_5[i].\texttt{xdiff} = 1 \end{cases}$$

Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}) \leftarrow \texttt{betaSet}(\mathcal{M}, \mathcal{G}, \mathcal{T})$

Set the objective function of $\mathcal{M}$ as Eq. (22)

Return $(\mathcal{M}, \mathcal{G}, \mathcal{T}, \mathcal{S})$.

Algorithm 37: `b2sBottomModel`

Input : The active bit position j
Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the lookup table $\mathcal{T}$, the binary variable set $\mathcal{S}$

Initialize a MILP model $\mathcal{M}$
Call Algorithm 39 as $(\mathcal{M}, \mathcal{G}, \mathcal{T}) \leftarrow$ `b2sCommonModel`$(\mathcal{M}, 9.5, 10.5)$
Update the model by adding constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \nabla v_\ell^{9.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \\ \nabla m_\ell[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \bigwedge (\ell, k) \neq (10, j) \\ \nabla m_{10}[j].\texttt{xdiff} = 1 \end{cases}$$

Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}) \leftarrow$ `betaSet`$(\mathcal{M}, \mathcal{G}, \mathcal{T})$
Set the objective function of $\mathcal{M}$ as Eq. (22)
Return $(\mathcal{M}, \mathcal{G}, \mathcal{T}, \mathcal{S})$.

Algorithm 38: `b2sConnectModel`

Input : The message block differences $\Theta^t, \Theta^b \in \{\mathtt{u}, \mathtt{n}, =\}^{32}$
Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}^t(\mathcal{V}^t, \mathcal{E}^t)$ and $\mathcal{G}^b(\mathcal{V}^b, \mathcal{E}^b)$, the lookup table $\mathcal{T}^t$ and $\mathcal{T}^b$, the binary variable set $\mathcal{S}$

Initialize a MILP model $\mathcal{M}$
Call Algorithm 39 as $(\mathcal{M}, \mathcal{G}^t, \mathcal{T}^t) \leftarrow$ `b2sCommonModel`$(\mathcal{M}, 5.5, 6.5)$
Call Algorithm 39 as $(\mathcal{M}, \mathcal{G}^b, \mathcal{T}^b) \leftarrow$ `b2sCommonModel`$(\mathcal{M}, 6.5, 8.5)$
Update the model by adding constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \nabla^t v_\ell^{5.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \\ \nabla^b v_\ell^{8.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \\ \nabla^t M \leftarrow \Theta^t \\ \nabla^b M \leftarrow \Theta^b \end{cases}$$

Call Algorithm 29 as $\mathcal{M} \leftarrow$ `InterSecConstr`$(\mathcal{M}, \mathcal{G}^t, \mathcal{G}^b, \mathcal{T}^t, \mathcal{T}^b, \boldsymbol{V}^{3.5})$
Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}^t) \leftarrow$ `betaSet`$(\mathcal{M}, \mathcal{G}^t, \mathcal{T}^t)$
Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}^b) \leftarrow$ `betaSet`$(\mathcal{M}, \mathcal{G}^b, \mathcal{T}^b)$
Define $\mathcal{S} \leftarrow \mathcal{S}^t \bigcup \mathcal{S}^b$
Set the objective function of $\mathcal{M}$ as Eq. (22)
Return $(\mathcal{M}, \mathcal{G}^t, \mathcal{G}^b, \mathcal{T}^t, \mathcal{T}^b, \mathcal{S})$.

Algorithm 39: `b2sCommonModel`

Input : The initial MILP model $\mathcal{M}$, the starting r_0 and the ending round r_1 with $r_0 < r_1$

Output: The updated MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ capturing the computation from V^{r_0} to V^{r_1}, the lookup table $\mathcal{T}$ mapping word nodes to the corresponding signed differences

Call Algorithm 18 and acquire the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E}) \leftarrow$ `b2sDigraph`(r_0, r_1).

Initialize an empty lookup table $\mathcal{T}$ Initialize V^{r_0}, M as

$$\mathcal{M}.\texttt{var} \leftarrow \begin{cases} \nabla v_i^{r_0}[j].\texttt{sign}, \nabla v_i^{r_0}[j].\texttt{xdiff} \text{ as binary} \\ \nabla m_i[j].\texttt{sign}, \nabla m_i[j].\texttt{xdiff} \text{ as binary} \end{cases} \quad i = 0, \ldots, 15;\ j = 0, \ldots, 31$$

Assign $\mathcal{T}[\boldsymbol{v}_i^{r_0}] \leftarrow \nabla v_i^{r_0}$ for $i = 0, \ldots, 15$

Assign $\mathcal{T}[\boldsymbol{m}_i] \leftarrow \nabla m_i$ for $i = 0, \ldots, 15$

for $r = r_0 + 0.5, r_0 + 1, \ldots, r_1$ **do**

  $(\mathcal{M}, \nabla TV^{r-0.5}) \leftarrow$ `b2sUpdateTv`$(\mathcal{M}, \nabla V^{r-0.5}, \nabla M, r - 0.5)$

  Assign $\mathcal{T}[\boldsymbol{tv}_i^{r-0.5}] \leftarrow \nabla tv_i^{r-0.5}$ for $i = 0, \ldots, 15$

  $(\mathcal{M}, \nabla V^r) \leftarrow$ `b2sUpdateV`$(\mathcal{M}, \nabla TV^{r-0.5}, \nabla M, r)$

  Assign $\mathcal{T}[\boldsymbol{v}_i^r] \leftarrow \nabla v_i^r$ for $i = 0, \ldots, 15$

Return $(\mathcal{M}, \mathcal{G}, \mathcal{T})$.

Algorithm 40: `b2sUpdateTv`

Input : The initial MILP model $\mathcal{M}$, the signed differences of intermediate state ∇V^r and message block ∇M, the round number r

Output: The updated MILP model $\mathcal{M}$, the signed differences of intermediate state ∇TV^r

if $r \in \mathbb{Z}$ **then**

  Define $\underline{r} \leftarrow r \mod 10$

  $(\mathcal{M}, \nabla tv_{0,4,8,12}^r) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla v_{0,4,8,12}^r, \nabla m_{\sigma_{\underline{r}}(0)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{1,5,9,13}^r) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla v_{1,5,9,13}^r, \nabla m_{\sigma_{\underline{r}}(2)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{2,6,10,14}^r) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla v_{2,6,10,14}^r, \nabla m_{\sigma_{\underline{r}}(4)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{3,7,11,15}^r) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla v_{3,7,11,15}^r, \nabla m_{\sigma_{\underline{r}}(6)}, 16, 12)$.

else

  Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$

  $(\mathcal{M}, \nabla tv_{0,5,10,15}^r) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla v_{0,5,10,15}^r, \nabla m_{\sigma_{\underline{r}}(8)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{1,6,11,12}^r) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla v_{1,6,11,12}^r, \nabla m_{\sigma_{\underline{r}}(10)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{2,7,8,13}^r) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla v_{2,7,8,13}^r, \nabla m_{\sigma_{\underline{r}}(12)}, 16, 12)$.

  $(\mathcal{M}, \nabla tv_{3,4,9,14}^r) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla v_{3,4,9,14}^r, \nabla m_{\sigma_{\underline{r}}(14)}, 16, 12)$.

Assign $\nabla TV^r \leftarrow (\nabla tv_0^r, \ldots, \nabla tv_{15}^r)$

Return $(\mathcal{M}, \nabla TV^r)$.

Algorithm 41: `b2sUpdateV`

Input : The initial MILP model $\mathcal{M}$, the signed differences of intermediate state $\nabla TV^{r-0.5}$ and message block ∇M, the round number r

Output: The updated MILP model $\mathcal{M}$, the signed differences of intermediate state ∇V^r

if $r \notin \mathbb{Z}$ **then**
  Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$
  $(\mathcal{M}, \nabla v^r_{0,4,8,12}) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla tv^{r-0.5}_{0,4,8,12}, \nabla m_{\sigma_{\underline{r}}(1)}, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{1,5,9,13}) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla tv^{r-0.5}_{1,5,9,13}, \nabla m_{\sigma_{\underline{r}}(3)}, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{2,6,10,14}) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla tv^{r-0.5}_{2,6,10,14}, \nabla m_{\sigma_{\underline{r}}(4)}, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{3,7,11,15}) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla tv^{r-0.5}_{3,7,11,15}, \nabla m_{\sigma_{\underline{r}}(7)}, 8, 7)$.
else
  Define $\underline{r} \leftarrow (r-1) \mod 10$
  $(\mathcal{M}, \nabla v^r_{0,5,10,15}) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla tv^{r-0.5}_{0,5,10,15}, \nabla m_{\sigma_{\underline{r}}(8)}, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{1,6,11,12}) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla tv^{r-0.5}_{1,6,11,12}, \nabla m_{\sigma_{\underline{r}}(10)}, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{2,7,8,13}) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla tv^{r-0.5}_{2,7,8,13}, \nabla m_{\sigma_{\underline{r}}(12)}, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{3,4,9,14}) \leftarrow$ `hGModel`$(\mathcal{M}, \nabla tv^{r-0.5}_{3,4,9,14}, \nabla m_{\sigma_{\underline{r}}(14)}, 8, 7)$.
Assign $\nabla V^r \leftarrow (\nabla v^r_0, \ldots, \nabla v^r_{15})$
Return $(\mathcal{M}, \nabla V^r)$.

covering $V^{9.5} \rightarrow V^{10.5}$ with $\Delta^t V^{9.5} = 0$ is constructed as Algorithm 43. The solution of $\mathcal{M}^t$ and $\mathcal{M}^b$ gives $\nabla M^t = \Theta^t$ and $\nabla M^b = \Theta^b$ with which the connect model $\mathcal{M}^c$ can be constructed with Algorithm 44 covering $V^{5.5} \rightarrow V^{8.5}$ with $(\Delta^t V^{5.5}, \Delta^b V^{8.5}) = (0, 0)$.

F The Signed Differential Paths for BLAKE2s and BLAKE-256

We give 2 boomerang attacks on 8-round BLAKE2s. using and $(i, j) = (31, 30)$ settings. The signed differential path for $(i, j) = (28, 31)$ is given in Table 16 and that for $(i, j) = (31, 30)$ is given in Table 17.

For BLAKE-256, our boomerang attack is based on the 8-round signed differential path in Table 18.

Algorithm 42: `b256TopModel`

Input : The active bit position i

Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the lookup table $\mathcal{T}$, the binary variable set $\mathcal{S}$

1 Initialize a MILP model $\mathcal{M}$

2 Call Algorithm 45 as $(\mathcal{M}, \mathcal{G}, \mathcal{T}) \leftarrow$ `b256CommonModel`$(\mathcal{M}, 2.5, 4.5)$

3 Update the model by adding constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \nabla v_\ell^{4.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \\ \nabla m_\ell[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \bigwedge (\ell, k) \neq (5, i) \\ \nabla m_5[i].\texttt{xdiff} = 1 \end{cases}$$

4 Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}) \leftarrow$ `betaSet`$(\mathcal{M}, \mathcal{G}, \mathcal{T})$

5 Set the objective function of $\mathcal{M}$ as Eq. (22)

6 **Return** $(\mathcal{M}, \mathcal{G}, \mathcal{T}, \mathcal{S})$.

Algorithm 43: `b256BottomModel`

Input : The active bit position j

Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the lookup table $\mathcal{T}$, the binary variable set $\mathcal{S}$

1 Initialize a MILP model $\mathcal{M}$

2 Call Algorithm 45 as $(\mathcal{M}, \mathcal{G}, \mathcal{T}) \leftarrow$ `b256CommonModel`$(\mathcal{M}, 9.5, 10.5)$

3 Update the model by adding constraints:

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \nabla v_\ell^{9.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \\ \nabla m_\ell[k] \leftarrow = \text{ for } (\ell, k) \in [0, 15] \times [0, 31] \bigwedge (\ell, k) \neq (10, j) \\ \nabla m_{10}[j].\texttt{xdiff} = 1 \end{cases}$$

4 Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}) \leftarrow$ `betaSet`$(\mathcal{M}, \mathcal{G}, \mathcal{T})$

5 Set the objective function of $\mathcal{M}$ as Eq. (22)

6 **Return** $(\mathcal{M}, \mathcal{G}, \mathcal{T}, \mathcal{S})$.

Algorithm 44: `b256ConnectModel`

Input : The message block differences $\Theta^t, \Theta^b \in \{\mathtt{u}, \mathtt{n}, =\}^{32}$
Output: The MILP model $\mathcal{M}$, the digraph $\mathcal{G}^t(\mathcal{V}^t, \mathcal{E}^t)$ and $\mathcal{G}^b(\mathcal{V}^b, \mathcal{E}^b)$, the lookup table $\mathcal{T}^t$ and $\mathcal{T}^b$, the binary variable set $\mathcal{S}$

Initialize a MILP model $\mathcal{M}$
Call Algorithm 45 as $(\mathcal{M}, \mathcal{G}^t, \mathcal{T}^t) \leftarrow \mathtt{b256CommonModel}(\mathcal{M}, 5.5, 6.5)$
Call Algorithm 45 as $(\mathcal{M}, \mathcal{G}^b, \mathcal{T}^b) \leftarrow \mathtt{b256CommonModel}(\mathcal{M}, 6.5, 8.5)$
Update the model by adding constraints:

$$\mathcal{M}.\mathtt{con} \leftarrow \begin{cases} \nabla^t v_\ell^{5.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0,15] \times [0,31] \\ \nabla^b v_\ell^{8.5}[k] \leftarrow = \text{ for } (\ell, k) \in [0,15] \times [0,31] \\ \nabla^t M \leftarrow \Theta^t \\ \nabla^b M \leftarrow \Theta^b \end{cases}$$

Call Algorithm 29 as $\mathcal{M} \leftarrow \mathtt{InterSecConstr}(\mathcal{M}, \mathcal{G}^t, \mathcal{G}^b, \mathcal{T}^t, \mathcal{T}^b, \boldsymbol{V}^{3.5})$
Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}^t) \leftarrow \mathtt{betaSet}(\mathcal{M}, \mathcal{G}^t, \mathcal{T}^t)$
Call Algorithm 28 as $(\mathcal{M}, \mathcal{S}^b) \leftarrow \mathtt{betaSet}(\mathcal{M}, \mathcal{G}^b, \mathcal{T}^b)$
Define $\mathcal{S} \leftarrow \mathcal{S}^t \bigcup \mathcal{S}^b$
Set the objective function of $\mathcal{M}$ as Eq. (22)
Return $(\mathcal{M}, \mathcal{G}^t, \mathcal{G}^b, \mathcal{T}^t, \mathcal{T}^b, \mathcal{S})$.

Algorithm 45: `b256CommonModel`

Input : The initial MILP model $\mathcal{M}$, the starting r_0 and the ending round r_1 with $r_0 < r_1$
Output: The updated MILP model $\mathcal{M}$, the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ capturing the computation from V^{r_0} to V^{r_1}, the lookup table $\mathcal{T}$ mapping word nodes to the corresponding signed differences

Call Algorithm 21 and acquire the digraph $\mathcal{G}(\mathcal{V}, \mathcal{E}) \leftarrow \mathtt{b256Digraph}(r_0, r_1)$.
Initialize an empty lookup table $\mathcal{T}$ Initialize V^{r_0}, M as

$$\mathcal{M}.\mathtt{var} \leftarrow \begin{cases} \nabla v_i^{r_0}[j].\mathtt{sign}, \nabla v_i^{r_0}[j].\mathtt{xdiff} \text{ as binary} \\ \nabla m_i[j].\mathtt{sign}, \nabla m_i[j].\mathtt{xdiff} \text{ as binary} \end{cases} i = 0, \ldots, 15;\ j = 0, \ldots, 31$$

Assign $\mathcal{T}[\boldsymbol{v}_i^{r_0}] \leftarrow \nabla v_i^{r_0}$ for $i = 0, \ldots, 15$
Assign $\mathcal{T}[\boldsymbol{m}_i] \leftarrow \nabla m_i$ for $i = 0, \ldots, 15$
for $r = r_0 + 0.5, r_0 + 1, \ldots, r_1$ **do**
    $(\mathcal{M}, \nabla TV^{r-0.5}, \nabla MV^{r-0.5}) \leftarrow \mathtt{b256UpdateTv}(\mathcal{M}, \nabla V^{r-0.5}, \nabla M, r - 0.5)$
    Assign $\mathcal{T}[\boldsymbol{tv}_i^{r-0.5}] \leftarrow \nabla tv_i^{r-0.5}$ for $i = 0, \ldots, 15$
    Assign $\mathcal{T}[\boldsymbol{mv}_i^{r-0.5}] \leftarrow \nabla mv_i^{r-0.5}$ for $i = 0, \ldots, 3$
    $(\mathcal{M}, \nabla V^r, \nabla MT^{r-0.5}) \leftarrow \mathtt{b256UpdateV}(\mathcal{M}, \nabla TV^{r-0.5}, \nabla M, r)$
    Assign $\mathcal{T}[\boldsymbol{v}_i^r] \leftarrow \nabla v_i^r$ for $i = 0, \ldots, 15$
    Assign $\mathcal{T}[\boldsymbol{mt}_i^{r-0.5}] \leftarrow \nabla mt_i^{r-0.5}$ for $i = 0, \ldots, 3$
Return $(\mathcal{M}, \mathcal{G}, \mathcal{T})$.

Algorithm 46: `b256UpdateTv`

Input : The initial MILP model $\mathcal{M}$, the signed differences of intermediate state ∇V^r and message block ∇M, the round number r

Output: The updated MILP model $\mathcal{M}$, the signed differences of intermediate state ∇TV^r

Call Algorithm 49 as $(\mathcal{M}, \nabla MV^r) \leftarrow \texttt{mvModel}(\mathcal{M}, \nabla M, r)$
if $r \in \mathbb{Z}$ **then**
  Define $\underline{r} \leftarrow r \mod 10$
  $(\mathcal{M}, \nabla tv^r_{0,4,8,12}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v^{\underline{r}}_{0,4,8,12}, \nabla mv^r_0, 16, 12)$.
  $(\mathcal{M}, \nabla tv^r_{1,5,9,13}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v^{\underline{r}}_{1,5,9,13}, \nabla mv^r_1, 16, 12)$.
  $(\mathcal{M}, \nabla tv^r_{2,6,10,14}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v^{\underline{r}}_{2,6,10,14}, \nabla mv^r_2, 16, 12)$.
  $(\mathcal{M}, \nabla tv^r_{3,7,11,15}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v^{\underline{r}}_{3,7,11,15}, \nabla mv^r_3, 16, 12)$.
else
  Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$
  $(\mathcal{M}, \nabla tv^r_{0,5,10,15}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v^{\underline{r}}_{0,5,10,15}, \nabla mv^r_0, 16, 12)$.
  $(\mathcal{M}, \nabla tv^r_{1,6,11,12}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v^{\underline{r}}_{1,6,11,12}, \nabla mv^r_1, 16, 12)$.
  $(\mathcal{M}, \nabla tv^r_{2,7,8,13}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v^{\underline{r}}_{2,7,8,13}, \nabla mv^r_2, 16, 12)$.
  $(\mathcal{M}, \nabla tv^r_{3,4,9,14}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla v^{\underline{r}}_{3,4,9,14}, \nabla mv^r_3, 16, 12)$.
Assign $\nabla TV^r \leftarrow (\nabla tv^r_0, \ldots, \nabla tv^r_{15})$
Return $(\mathcal{M}, \nabla TV^r, \nabla MV^r)$.

Algorithm 47: `b256UpdateV`

Input : The initial MILP model $\mathcal{M}$, the signed differences of intermediate state $\nabla TV^{r-0.5}$ and message block ∇M, the round number r

Output: The updated MILP model $\mathcal{M}$, the signed differences of intermediate states ∇V^r and $\nabla MT^{r-0.5}$

Call Algorithm 48 as $(\mathcal{M}, \nabla MT^{r-0.5}) \leftarrow \texttt{mtModel}(\mathcal{M}, \nabla M, r-0.5)$
if $r \notin \mathbb{Z}$ **then**
  Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$
  $(\mathcal{M}, \nabla v^r_{0,4,8,12}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{0,4,8,12}, \nabla mt^{r-0.5}_0, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{1,5,9,13}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{1,5,9,13}, \nabla mt^{r-0.5}_1, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{2,6,10,14}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{2,6,10,14}, \nabla mt^{r-0.5}_2, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{3,7,11,15}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{3,7,11,15}, \nabla mt^{r-0.5}_3, 8, 7)$.
else
  Define $\underline{r} \leftarrow (r-1) \mod 10$
  $(\mathcal{M}, \nabla v^r_{0,5,10,15}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{0,5,10,15}, \nabla mt^{r-0.5}_0, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{1,6,11,12}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{1,6,11,12}, \nabla mt^{r-0.5}_1, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{2,7,8,13}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{2,7,8,13}, \nabla mt^{r-0.5}_2, 8, 7)$.
  $(\mathcal{M}, \nabla v^r_{3,4,9,14}) \leftarrow \texttt{hGModel}(\mathcal{M}, \nabla tv^{r-0.5}_{3,4,9,14}, \nabla mt^{r-0.5}_3, 8, 7)$.
Assign $\nabla V^r \leftarrow (\nabla v^r_0, \ldots, \nabla v^r_{15})$
Return $(\mathcal{M}, \nabla V^r, \nabla MT^{r-0.5})$.

Algorithm 48: `mtModel`

Input : The initial model $\mathcal{M}$, the 16-word signed difference ∇M, the round number r

Output: The updated model $\mathcal{M}$, the 4-word signed difference ∇MT^r

for $i = 0, \ldots, 3$ **do**
  if $r \in \mathbb{Z}$ **then**
    Define $\underline{r} \leftarrow r \mod 10$
    $(\mathcal{M}, \nabla mt_i^r) \leftarrow \texttt{xorConstApprox}(\mathcal{M}, \nabla m_{\sigma_{\underline{r}}(2i+1)}, rc_{\sigma_{\underline{r}}(2i)})$
  else
    Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$
    $(\mathcal{M}, \nabla mt_i^r) \leftarrow \texttt{xorConstApprox}(\mathcal{M}, \nabla m_{\sigma_{\underline{r}}(2i+1+8)}, rc_{\sigma_{\underline{r}}(2i+8)})$
Define $\nabla MT^r \leftarrow (\nabla mt_0^r, \ldots, \nabla mt_3^r)$
Return $(\mathcal{M}, \nabla MT^r)$.

Algorithm 49: `mvModel`

Input : The initial model $\mathcal{M}$, the 16-word signed difference ∇M, the round number r

Output: The updated model $\mathcal{M}$, the 4-word signed difference ∇MV^r

for $i = 0, \ldots, 3$ **do**
  if $r \in \mathbb{Z}$ **then**
    Define $\underline{r} \leftarrow r \mod 10$
    Call Algorithm 5 as
    $(\mathcal{M}, \nabla mv_i^r) \leftarrow \texttt{xorConstApprox}(\mathcal{M}, \nabla m_{\sigma_{\underline{r}}(2i)}, rc_{\sigma_{\underline{r}}(2i+1)})$
  else
    Define $\underline{r} \leftarrow \lfloor r \rfloor \mod 10$
    Call Algorithm 5 as
    $(\mathcal{M}, \nabla mv_i^r) \leftarrow \texttt{xorConstApprox}(\mathcal{M}, \nabla m_{\sigma_{\underline{r}}(2i+8)}, rc_{\sigma_{\underline{r}}(2i+1+8)})$
Define $\nabla MV^r \leftarrow (\nabla mv_0^r, \ldots, \nabla mv_3^r)$
Return $(\mathcal{M}, \nabla MV^r)$.

Table 16: The signed differential path for 8-round BLAKE2s with the $(i, j) = (28, 31)$ setting.

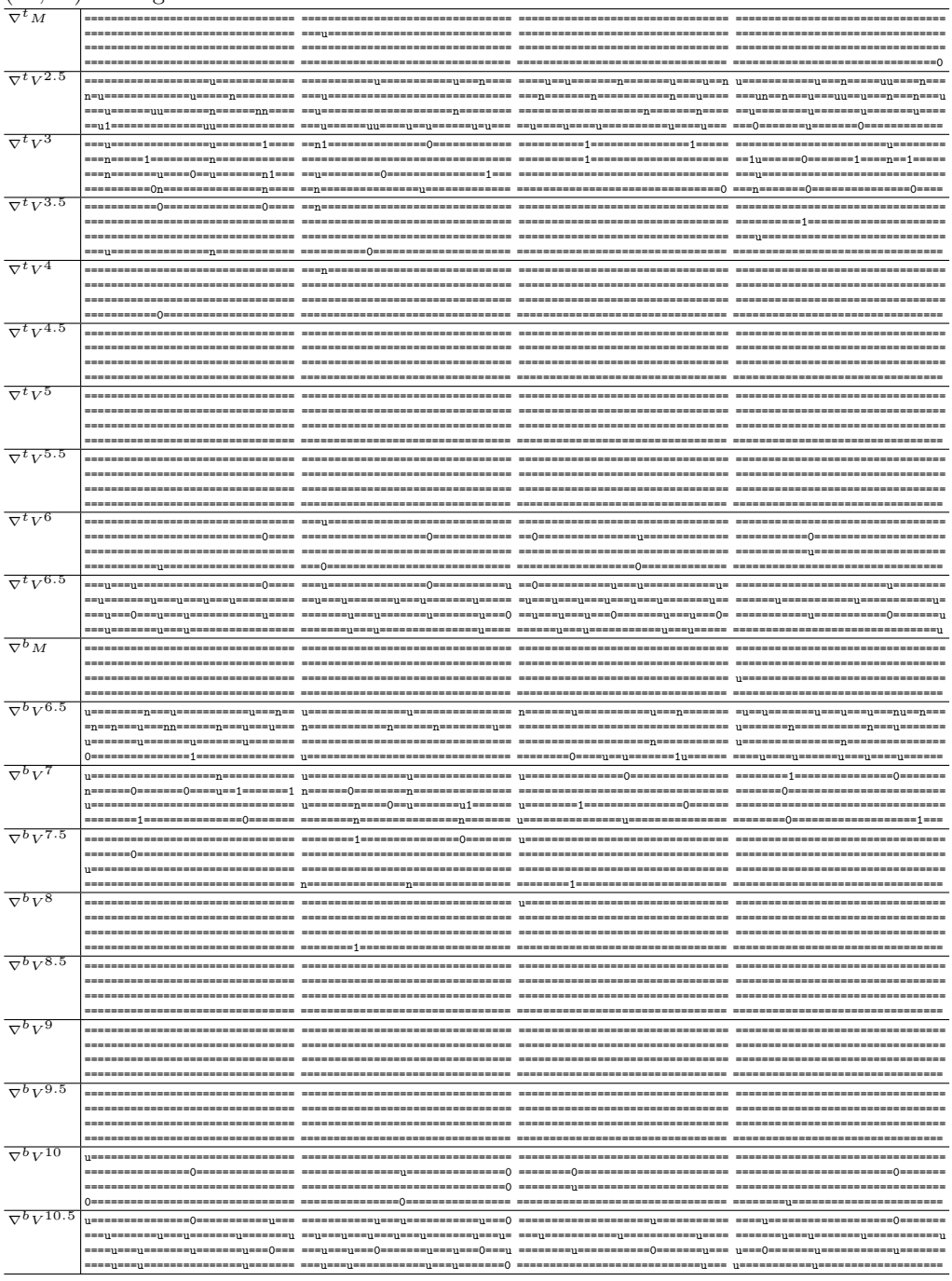

Table 17: The signed differential path for 8-round BLAKE2s with the $(i, j) = (31, 30)$ setting.

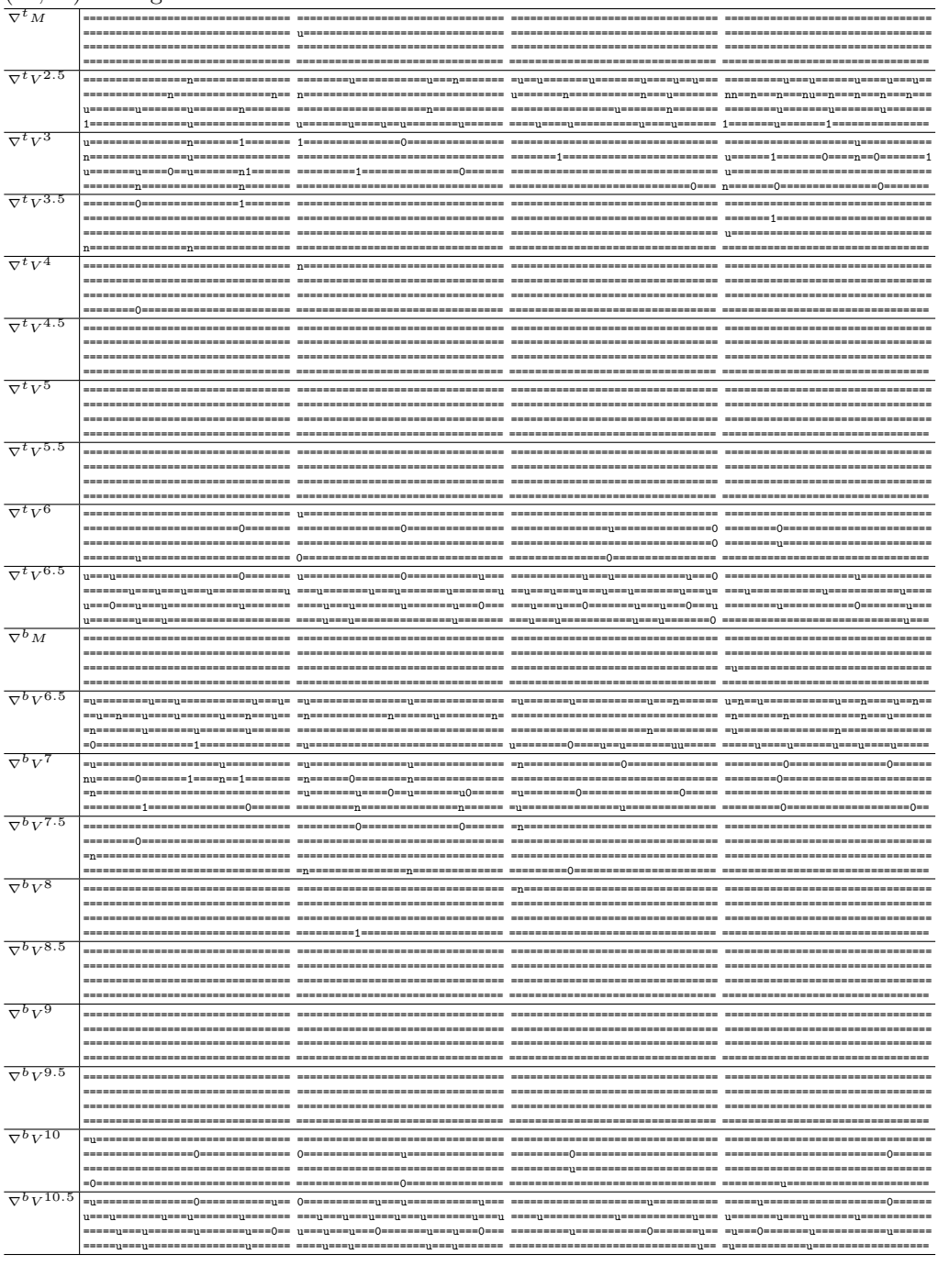

Table 18: The signed differential path for 8-round BLAKE-256 with the $(i, j) = (31, 30)$ setting.