Analysis of RIPEMD-160: New Collision Attacks and Finding Characteristics with MILP

Fukang Liu[1,2], Gaoli Wang[3,4], Santanu Sarkar[6], Ravi Anand[2], Willi Meier[7], Yingxin Li[3], Takanori Isobe[2,5]

[1] Tokyo Institute of Technology, Tokyo, Japan,
[2] University of Hyogo, Hyogo, Japan
[3] Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
[4] State Key Laboratory of Cryptology, Beijing, China
[5] NICT, Tokyo, Japan
[6] Indian Institute of Technology Madras, Chennai, India
[7] FHNW, Windisch, Switzerland
liufukangs@gmail.com, glwang@sei.ecnu.edu.cn, santanu@iitm.ac.in, ravianandsps@gmail.com, liyx1140@163.com, willi.meier@fhnw.ch, takanori.isobe@ai.u-hyogo.ac.jp

Abstract. The hash function RIPEMD-160 is an ISO/IEC standard and is being used to generate the bitcoin address together with SHA-256. Despite the fact that many hash functions in the MD-SHA hash family have been broken, RIPEMD-160 remains secure and the best collision attack could only reach up to 34 out of 80 rounds, which was published at CRYPTO 2019. In this paper, we propose a new collision attack on RIPEMD-160 that can reach up to 36 rounds with time complexity $2^{64.5}$. This new attack is facilitated by a new strategy to choose the message differences and new techniques to simultaneously handle the differential conditions on both branches. Moreover, different from all the previous work on RIPEMD-160, we utilize a MILP-based method to search for differential characteristics, where we construct a model to accurately describe the signed difference transitions through its round function. As far as we know, this is the first model targeting the signed difference transitions for the MD-SHA hash family. Indeed, we are more motivated to design this model by the fact that many automatic tools to search for such differential characteristics are not publicly available and implementing them from scratch is too time-consuming and difficult. Hence, we expect that this can be an alternative easy tool for future research, which only requires to write down some simple linear inequalities.

Keywords: RIPEMD-160, collision attack, signed difference, modular difference, MILP

1 Introduction

Background. The most powerful technique to mount collision attacks on the MD-SHA hash family is to carefully trace the evolutions of the signed difference through the round functions [29,30,31,32]. The feature of the signed difference is that it can capture how a bit is changed, i.e. from 1 to 0 or from 0 to 1. This makes it interact well with the modular difference because each specified signed difference can uniquely determine the corresponding modular difference and XOR difference. It is thus clear that the signed difference carries the information of both the XOR difference and modular difference.

Based on the above crucial observations, in Wang et al.'s seminal work [29, 30, 31, 32], they deduced all the collision-generating differential characteristics by hand for a series of famous hash functions, including MD4, MD5, SHA-0 and SHA-1. However, such hand-crafted work is too technical and time-consuming. Therefore, several automatic tools [2, 6, 14, 15, 16, 17, 18, 23, 24, 25] to search for these differential characteristics have been developed and they have even been applied to much more complex hash functions like SHA-2 [5, 6, 15, 17] and RIPEMD-160 [14, 18]. However, most of these tools [2, 6, 14, 15, 16, 17, 18] are not made publicly available. As far as we know, only the tools [23, 24, 25] developed by Stevens are open-source. A similar tool developed by Leurent for the ARX cipher Skein is also open-source [9]. However, the tools developed by Stevens are only for MD5 and SHA-1. Tweaking Stevens's tools for different hash functions is not easy because it requires deep understanding of their implementations and there are a few structured documents for the codes. Especially for RIPEMD-160 and SHA-2, their round functions are more complex than those of MD5 and SHA-1, which further increases the difficulty.

On RIPEMD-160. The hash function RIPEMD-160 [4] was proposed at FSE 1996, whose overall structure can be viewed as two parallel MD5-like instances. Such a double-branch structure makes it well resist against Wang et al.'s powerful techniques for the MD-SHA hash family. The main difficulty is to construct suitable collision-generating differential characteristics and to perform the message modification to fulfill the differential conditions on both branches simultaneously.

Due to the increasing difficulty of analyzing the double-branch structure, the progress in analyzing the security of RIPEMD-160 is slow, as can be seen in Table 1. For example, the first practical collision attacks on 30 and 31 rounds of RIPEMD-160 were demonstrated in 2019 and the best collision attack with the same technique could only reach up to 34 rounds [10]. For the semi-free-start (SFS) collision attack, the best attack could only reach up to 40 rounds [11], which was published also in 2019.

As RIPEMD-160 is an ISO/IEC standard and is being used in bitcoin, we believe further understanding its (second-)preimage and collision resistance is of practical interest. In this work, we target the collision resistance, which is generally more meaningful than the SFS collision resistance.

Our contributions. The contributions of this work are fourfold. Specifically, we propose:

1. A new strategy to choose the message differences which allows to mount a collision attack on 36-round RIPEMD-160.

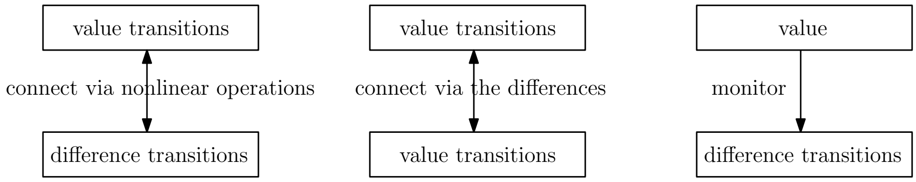

Fig. 1: The comparison between different models (left: [12],middle: [19, 24], right: this paper)

2. A state-of-the-art method to efficiently perform the message modification on both branches simultaneously by carefully exploiting the feature of the differential characteristic.
3. A new methodology to search for differential characteristics for RIPEMD-160 that relies on off-the-shelf solvers. This is achieved by constructing a model to describe the signed difference transitions through the round function of RIPEMD-160. As far as we know, it is the first time to use the MILP-based method to search for a *pure signed* differential characteristic
4. A new method to automatically detect the contradictions in the search for signed differential characteristics. Specifically, we propose to use *monitoring variables* representing the values of the internal states to monitor the inconsistency appearing in the signed difference transitions over different rounds. This should be distinguished from Liu et al.'s technique [12] where both the value transitions and difference transitions are involved in a model to avoid the inconsistency, i.e. we do not care about the value transitions because they are costly. This should also be distinguished from the techniques [19, 24] where only a model to simply describe two parallel value transitions is used, which is inefficient as no feature of the signed difference propagations is exploited in such a model. The comparison between different methods is shown in Fig. 1.

The source code to search for signed differential characteristics is available at `https://github.com/LFKOKAMI/Find_RIPEMD_Trail.git`.

Outline of the paper. In Section 2, we introduce the notations and some preliminary works. In Section 3, the MILP model to describe the signed difference transitions through RIPEMD-160's round function is detailed. Then, we show the 36-round collision attack in Section 4. Finally, we end this paper with some discussions on our techniques in Section 5.

2 Preliminaries

2.1 Notation

The following notations are used throughout this paper. $\boxplus$ and $\boxminus$ represent the modular addition and substraction modulo 2^{32}, respectively. $x[i]$ denotes the i-th bit of x and $x[0]$ is the least significant bit. Δx denotes the XOR difference of

Table 1: Summary of preimage and (SFS) collision attack on RIPEMD-160

Attack Type	Rounds	Time	Memory	Reference	Year
Preimage	31[a]	2^{155}	unknown	[21]	2010
	34	$2^{158.91}$	unknown	[27]	2014
	35[a]	$2^{159.38}$	unknown	[22]	2018
SFS collision	36[a]	practical		[14]	2012
	42[a]	$2^{75.5}$	2^{64}	[18]	2013
	48[a]	$2^{76.4}$	2^{64}	[28]	2017
	36	$2^{70.4}$	2^{64}	[18]	2013
	36	$2^{55.1}$	2^{32}	[13]	2017
	36/37	practical		[11]	2019
	40	$2^{74.6}$	negligible	[11]	2019
collision	30/31	practical		[10]	2019
	34	$2^{74.3}$	2^{32}	[10]	2019
	36	$2^{64.5}$	2^{24}	this work	2022

[a] An attack starting at an intermediate round.

x' and x, i.e. $\Delta x = x' \oplus x$. δx denotes the modular difference, i.e. $\delta x = x' \boxminus x$. ∇x denotes the signed difference between x' and x, i.e. $\nabla x[i] = [\texttt{=}]$ if $x'[i] = x[i]$, $\nabla x[i] = [\texttt{0}]$ if $x'[i] = x[i] = 0$, $\nabla x[i] = [\texttt{1}]$ if $x'[i] = x[i] = 1$, $\nabla x[i] = [\texttt{n}]$ if $(x'[i] = 1, x[i] = 0)$, $\nabla x[i] = [\texttt{u}]$ if $(x'[i] = 0, x[i] = 1)$. $[a, b]$ denotes the set $\{i|a \leq i \leq b\}$. $\overline{x}$ denotes the bitwise NOT operation on x. Moreover, x^T denotes a column vector and we simply use $x^T[i]$ to represent the i-th element of x^T. Especially, $x^T \geq y^T$ iff $x^T[i] \geq y^T[i]$ for all i, e.g. $(1, 2, 3)^T \geq (0, 2, 1)^T$ as $(1 \geq 0, 2 \geq 2, 3 \geq 1)$.

Definition 1. *The signed difference ∇x is said to be an expansion of the modular difference δx only when ∇x corresponds to the modular difference δx.*

Definition 2. *The hamming weight of the signed difference ∇x is denoted by $\mathbb{H}(\nabla x)$ and $\mathbb{H}(\nabla x)$ is the number of indices i such that $\nabla x[i] \in \{\texttt{n}, \texttt{u}\}$.*

For example, let

$$\nabla x_0 = [\texttt{=n== ==== ==== ==== ==== ==== ==== ====}],$$
$$\nabla x_1 = [\texttt{nu== ==== ==== ==== ==== ==== ==== ====}].$$

Then, both ∇x_0 and ∇x_1 are the expansions of $\delta x = 2^{30}$. Moreover, we have $\mathbb{H}(\nabla x_0) = 1$ and $\mathbb{H}(\nabla x_1) = 2$.

As each signed difference corresponds to a unique modular difference, for convenience, when computing $\delta x \boxplus \delta y$ for a given $(\nabla x, \nabla y)$, we also simply denote $\delta x \boxplus \delta y$ by $\nabla x \boxplus \nabla y$. For the above example, we have $\nabla x_0 \boxplus \nabla x_1 = 2^{31}$.

2.2 Description of RIPEMD-160

RIPEMD-160 [4] was proposed at FSE 1996 by Dobbertin et al. and it is built on the Merkle-Damgård structure. To compress an arbitrary-length message with

RIPEMD-160, the message will be first padded and then divided into several message blocks and each block is of size 512 bits. Supposing there are $\gamma+1$ message blocks and they are denoted by $M^0, M^1, \ldots, M^\gamma$, the 80-bit hash value $h=(h_0,h_1,h_2,h_3,h_4)$ is computed as follows:

$$IV^{j+1} = H(IV^j, M^j) \text{ for } j \in [0,\gamma],$$
$$h = IV^{\gamma+1},$$

where $H(IV^j, M^j)$ is the compression function of RIPEMD-160, IV^j is a 160-bit chaining variable and IV^0 is a predetermined constant value.

In our collision attack, we aim to find (M^0, M^1) and $(M^0, M^{1\prime})$ such that

$$H(H(IV_0, M^0), M^1) = H(H(IV_0, M^0), M^{1\prime})$$

where the number of rounds of H is reduced. In this way, a colliding message pair for the round-reduced RIPEMD-160 can be easily derived.

Let $M=(m_0,m_1,\ldots,m_{15})$ be the 16 message words of size 32 bits each and $IV^0=(IV_0^0, IV_1^0, \ldots, IV_4^0)$. The specification of the compression function $H(IV^0, M)$ is described below:

$$X_{-5}=Y_{-5}=IV_0^0 \ggg 10, X_{-4}=Y_{-4}=IV_4^0 \ggg 10, X_{-3}=Y_{-3}=IV_3^0 \ggg 10,$$
$$X_{-2}=Y_{-2}=IV_2^0, X_{-1}=Y_{-1}=IV_1^0,$$
$$Q_i^l = X_{i-5} \lll 10 \boxplus \phi_j^l(X_{i-1}, X_{i-2}, X_{i-3} \lll 10) \boxplus m_{\pi_l(i)} \boxplus K_j^l,$$
$$X_i = X_{i-4} \lll 10 \boxplus Q_i^l \lll s_i^l,$$
$$Q_i^r = Y_{i-5} \lll 10 \boxplus \phi_j^r(Y_{i-1}, Y_{i-2}, Y_{i-3} \lll 10) \boxplus m_{\pi_r(i)} \boxplus K_j^r,$$
$$Y_i = Y_{i-4} \lll 10 \boxplus Q_i^r \lll s_i^r,$$

where $i \in [0,79]$ and $j = \lfloor \frac{i}{16} \rfloor$. Due to the page limit, the specification of $\phi_j^l, \phi_j^r, K_j^l, K_j^r$ can be found in Table 2. $\pi_l(i), \pi_r(i), s_i^l, s_i^r$ can be referred to [4].

Table 2: Boolean functions and round constants in RIPEMD-160

j	ϕ_j^l	ϕ_j^r	K_j^l	K_j^r	Function	Expression
0	XOR	ONX	0x00000000	0x50a28be6	$XOR(x,y,z)$	$x \oplus y \oplus z$
1	IFX	IFZ	0x5a827999	0x5c4dd124	$IFX(x,y,z)$	$(x \wedge y) \oplus (\overline{x} \wedge z)$
2	ONZ	ONZ	0x6ed9eba1	0x6d703ef3	$IFZ(x,y,z)$	$(x \wedge z) \oplus (y \wedge \overline{z})$
3	IFZ	IFX	0x8f1bbcdc	0x7a6d76e9	$ONX(x,y,z)$	$x \oplus (y \vee \overline{z})$
4	ONX	XOR	0xa953fd4e	0x00000000	$ONZ(x,y,z)$	$(x \vee \overline{y}) \oplus z$

After 80 rounds of update, the output of $H(IV^0, M)$ denoted by $IV^1 = (IV_0^1, IV_1^1, \ldots, IV_4^1) \in \mathbb{F}_{2^{32}}^5$ is computed as follows:

$$IV_0^1 = IV_1^0 \boxplus X_{78} \boxplus Y_{77} \lll 10,\ IV_1^1 = IV_2^0 \boxplus X_{77} \lll 10 \boxplus Y_{76} \lll 10,$$
$$IV_2^1 = IV_3^0 \boxplus X_{76} \lll 10 \boxplus Y_{75} \lll 10,\ IV_3^1 = IV_4^0 \boxplus X_{75} \lll 10 \boxplus Y_{79},$$
$$IV_4^1 = IV_0^0 \boxplus X_{79} \boxplus Y_{78}.$$

2.3 The Differential Conditions for RIPEMD-160

Given a specified signed differential characteristic of RIPEMD-160, it has been shown in [13] that there should also be additional conditions on the modular difference. Specifically, apart from the bit conditions imposed by the differential characteristic, there will also be implicit conditions on each Q_i^l and Q_k^r, which are intermediate values during the round update of RIPEMD-160 as stated above. These implicit conditions are of the following forms:

$$(Q_i^l \boxplus \alpha_i^l) \lll s_i^l = Q_i^l \lll s_i^l \boxplus \beta_i^l,$$
$$(Q_k^r \boxplus \alpha_k^r) \lll s_k^l = Q_k^l \lll s_k^l \boxplus \beta_k^r,$$

where $(\alpha_i^l, \alpha_k^r, \beta_i^l, \beta_k^r)$ are constants and they can be easily derived from the specified differential characteristic. For convenience, we call these implicit conditions and the bit conditions **the differential conditions for a differential characteristic**.

It is possible that the conditions on these (Q_i^l, Q_k^r) contradict with the bit conditions, especially for the dense parts where many bits of the internal states (X_i, X_{i-4}) or (Y_k, Y_{k-4}) are fixed by the differential characteristic due to $Q_i^l = (X_i \boxminus X_{i-4} \lll 10) \ggg s_i^l$ and $Q_k^r = (Y_k \boxminus Y_{k-4} \lll 10) \ggg s_k^r$. Therefore, we should take this into account when searching for a valid differential characteristic. We note that many valid differential characteristics used for the (SFS) collision attacks on round-reduced RIPEMD-160 have been found with Mendel et al.'s tool [10,11,13,14,18]. However, it is unclear how this problem is handled in their tool as the implementation is not publicly available and only a few details of the tool are given in the corresponding papers.

2.4 Previous Methods to Search for Differential Characteristics

In the automatic tools [2,6,14,15,16,17,18,23,25] and Wang et al.'s hand-crafted work, it is common to first linearly propagate the message differences through the internal states backward and forward for several rounds, which can be easily finished either by hand or in a simple automatic way. Then, the signed differences for many internal states are fixed, while there are still some internal states whose signed differences are unknown.

For example, $(\nabla X_{i_0}, \nabla X_{i_0+1}, \ldots, \nabla X_{i_0+i_1})$ and $(\nabla X_{k_0}, \nabla X_{k_0+1}, \ldots, \nabla X_{k_0+k_1})$ are determined at the linear propagation phase where $k_0 > i_0 + i_1$. Then, the aim is to find a valid solution of $(\nabla X_{i_0+i_1+1}, \nabla X_{i_0+i_1+2}, \ldots, \nabla X_{k_0-1})$ to connect $(\nabla X_{i_0}, \nabla X_{i_0+1}, \ldots, \nabla X_{i_0+i_1})$ and $(\nabla X_{k_0}, \nabla X_{k_0+1}, \ldots, \nabla X_{k_0+k_1})$. Achieving the connection is the most technical component in these automatic tools. Its efficiency directly affects the overall performance. The main difficulty to achieve the connection is that many differential conditions are suddenly forced, which makes invalid solutions easily occur.

The most commonly used method for this connection problem is the guess-and-determine technique combined with some heuristic early-stop strategies [2,6,9,14, 15,16,17,18,23,25]. However, the implementation for RIPEMD-160 is not publicly available. There are also some tools [19, 24] relying on off-the-shelf solvers for this problem. However, in these tools, the idea is to construct a model to describe two parallel instances of the value transitions. Specifically, does there exist a solution of $(X_{i_0+i_1+1}, X_{i_0+i_1+2}, \ldots, X_{k_0-1})$ and $(X'_{i_0+i_1+1}, X'_{i_0+i_1+2}, \ldots, X'_{k_0-1})$ such that the predetermined signed differences $(\nabla X_{i_0}, \nabla X_{i_0+1}, \ldots, \nabla X_{i_0+i_1})$ and $(\nabla X_{k_0}, \nabla X_{k_0+1}, \ldots, \nabla X_{k_0+k_1})$ can be connected? This can be easily converted into a SAT problem by modelling the value transitions. We tried this method but we could not find desired differential characteristics in practical time. We believe this is mainly because the information of the signed difference propagations cannot be efficiently encoded in such a model.

2.5 On MILP/SAT-based Automatic Methods

It has become popular to utilize some off-the-shelf solvers to reduce the workload of cryptanalysis in the symmetric-key community. Depending on the used solvers, different languages are required to describe a target problem. Among these automatic methods, the SAT-based and MILP-based methods are mostly used [7, 20, 26]. For SAT-based methods, it is required to describe the target problem in the Conjunctive Normal Form (CNF) such that the solvers can handle them. For MILP-based methods, it is then required to describe the problem with linear inequalities.

With the software LogicFriday[8], by importing a truth table for some variables, one can easily obtain the minimized CNF in terms of these variables and then convert it into linear inequalities [1]. For example, suppose (x_0, x_1, x_2, x_3) can only take 3 values $\{(0,0,1,1), (1,0,1,0), (1,1,1,1)\}$. With LogicFriday, we can obtain the following equivalent minimized CNF to describe this constraint:

$$x_2 \wedge (x_0 \vee \overline{x_1}) \wedge (\overline{x_1} \vee x_3) \wedge (x_0 \vee x_3) \wedge (\overline{x_0} \vee x_1 \vee \overline{x_3}),$$

i.e. only the above 3 possible values of (x_0, x_1, x_2, x_3) can make the above boolean expression output 1 (true), while the remaining 13 values will make it output 0 (false). The above CNF can be converted into the following linear inequality system:

$$\begin{array}{c} x_2 \geq 1,\ x_0 + (1 - x_1) \geq 1,\ (1 - x_1) + x_3 \geq 1, \\ x_0 + x_3 \geq 1,\ (1 - x_0) + x_1 + (1 - x_3) \geq 1. \end{array}$$

For convenience, we also describe this system with the help of a matrix, as shown below:

$$\mathcal{H} \cdot (x_0, x_1, x_2, x_3)^T \geq (1, 0, 0, 1, -1)^T,$$

where

$$\mathcal{H} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & -1 \end{bmatrix}.$$

[8] You can easily download it from `https://download.cnet.com/`.

3 Finding Signed Differential Characteristics with MILP

In this work, we consider the MILP-based methods to search for signed differential characteristics for RIPEMD-160. To achieve this, the first step is to formulate the problem and the second step is to model the problem with linear inequalities. We emphasize that we tried several different modelling methods before we eventually identified the method described in the paper. Due to the page limit, we only describe the most successful and efficient modelling method.

Formulating the problem is easy. Take the left branch of RIPEMD-160 as an example and it also can be applied to the right branch due to the similarity. Specifically, given $(\nabla X_i, \nabla X_{i+1}, \ldots, \nabla X_{i+4}, \nabla m_{\pi_l(i)})$, how to describe the possible values of ∇X_{i+5} with linear inequalities? In other words, how do the signed differences propagate through the round function and how to describe it with linear inequalities? Once this problem is solved, searching for collision-generating differential characteristics with some chosen message differences is easy as the signed difference transitions through the round function are known and one only needs to add some extra simple constraints to obtain a desired differential characteristic.

3.1 Modelling Signed Difference Transitions

The round function of RIPEMD-160 is of the following form:

$$d_{i+5} = (d_{i+1} \lll 10) \boxplus (F(d_{i+4}, d_{i+3}, d_{i+2} \lll 10) \boxplus (d_i \lll 10) \boxplus m \boxplus c) \lll s.$$

When considering the signed differences, the operation $\lll 10$ only affects the order of variables. From this perspective, to study the signed difference propagation $(\nabla d_i, \nabla d_{i+1}, \nabla d_{i+2}, \nabla d_{i+3}, \nabla d_{i+4}, \nabla m) \to \nabla d_{i+5}$, we indeed only need to study the signed difference propagation $(\nabla a_0, \nabla a_1, \nabla a_2, \nabla a_3, \nabla a_4, \nabla m) \to \nabla a_5$, where

$$a_5 = a_1 \boxplus (F(a_4, a_3, a_2) \boxplus a_0 \boxplus m \boxplus c) \lll s. \tag{1}$$

With some intermediate variables $(b_0, b_1, b_2, b_3, b_4, b_5)$, Equation 1 can be decomposed as

$$b_0 = m \boxplus c, b_1 = F(a_4, a_3, a_2), b_2 = b_0 \boxplus b_1,$$
$$b_3 = b_2 \boxplus a_0, b_4 = b_3 \lll s, b_5 = a_1 \boxplus b_4, a_5 = b_5.$$

As (b_0, b_1, b_2, b_3) are all intermediate state values and m is a free variable that can be controlled by attackers, we only care about their modular differences. In other words, we can arbitrarily choose only one expansion of δb_i $(0 \leq i \leq 3)$ when constructing the model because one expansion is sufficient to describe the corresponding modular difference. For example, to describe $\delta b_i =$ `0x1`, we can constrain that ∇b_i only takes [`==== ==== ==== ==== ==== ==== ==== ===n`] even though ∇b_i indeed can take many possible values. This is because one possible ∇b_i is sufficient to describe the modular difference `0x1`. This is critical to improve the whole efficiency as invalid modular differences can be filtered in a much faster way. Our basic idea to construct the model is as follows:

1. Deterministically compute the signed difference transitions for $b_0 = m \boxplus c$, $b_2 = b_0 \boxplus b_1$ and $b_3 = b_2 \boxplus a_0$. Specifically, for each given $(\nabla x, \nabla y)$, uniquely compute one ∇z such that $\delta z = \delta x \boxplus \delta y$, even though there are many such possible ∇z.
2. Compute the signed difference transitions for $b_1 = F(a_4, a_3, a_2)$, where F is a boolean function.
3. Handle the signed difference transitions for $b_4 = b_3 \lll s$, $b_5 = a_1 \boxplus b_4$ and $a_5 = b_5$ according to different situations.

3.2 Describing Signed Differences

To construct the model, we first need to properly describe the signed difference. Different from the XOR difference which can be trivially described with a binary variable, there are 3 important statuses for the signed difference, namely $\{\texttt{=}, \texttt{n}, \texttt{u}\}$ and we cannot simply describe them with a binary variable. One may think that it can be described with a variable taking the value from $\{-1, 0, 1\}$, which is also supported by Gurobi. However, such a method is unfriendly to model the signed difference transitions through the boolean functions and the whole performance is bad even if we try some other strategies to make it work.

Finally, we choose to use two binary variables (v, d) to describe a 1-bit signed difference. Moreover, we restrict that (v, d) can only take 3 possible values[9], i.e. $(v, d) \in \{(0, 1), (1, 1), (0, 0)\}$. Specifically, $(v, d) = (0, 1)$ corresponds to `n`, $(v, d) = (1, 1)$ corresponds to `u`, and $(v, d) = (0, 0)$ corresponds to `=`. Note that we do not allow $(v, d) = (1, 0)$ because this is redundant and will affect the overall performance. This trick is important to improve the performance.

For convenience, when describing the signed difference of a binary variable κ, we simply use $(\kappa_v, \kappa_d) \in \{(0, 1), (1, 1), (0, 0)\}$ to represent the signed difference $\nabla\kappa$. In many of the following algorithms, we also say such a variable $\nabla\kappa$ is a signed difference variable and it should be viewed as a structure $\nabla\kappa = (\kappa_v, \kappa_d)$.

3.3 Modelling the Modular Addition

We consider the signed difference transition through $z = x \boxplus y$ bit by bit. Moreover, as stated above, we are interested in only one ∇z for a given $(\nabla x, \nabla y)$. To achieve this purpose, we introduce an additional variable ∇c of size 33 to represent the signed differences of the carry bits when computing $\nabla x \boxplus \nabla y$. Then, we use deterministic propagation rules for $(\nabla x[i], \nabla y[i], \nabla c[i]) \to (\nabla z[i], \nabla c[i+1])$, i.e. each $(\nabla x[i], \nabla y[i], \nabla c[i])$ corresponds to a unique $(\nabla z[i], \nabla c[i+1])$, as shown in Table 3. In this way, ∇z is uniquely determined for each given $(\nabla x, \nabla y)$ and it corresponds to the modular difference $\delta z = \delta x \boxplus \delta y$, which can be easily observed from the propagation rules.

[9] Here, it can be found that $d = 1$ means there is a difference and v is the initial value to be changed. Hence, $(v, d) = (0, 1)$ means 0 is changed to 1 and $(v, d) = (1, 1)$ means 1 is changed to 0. We exclude $(v, d) = (1, 0)$ because $(v, d) = (0, 0)$ can carry the same information as $(v, d) = (1, 0)$, i.e. both mean there is no difference.

Let us take [nnu → n=] and [u=u → =u] as examples. For [nnu → n=], it means $2^i \boxplus 2^i \boxminus 2^i = 2^i$. For [u=u → =u], it means $\boxminus 2^i \boxminus 2^i = \boxminus 2^{i+1}$. It is then clear that the modular difference $\delta x \boxplus \delta y$ is correctly recorded by the computed ∇z.

Table 3: The propagation rules for $(\nabla x[i], \nabla y[i], \nabla c[i]) \rightarrow (\nabla z[i], \nabla c[i+1])$

[=== → ==],	[==n → n=],	[==u → u=],	[=n= → n=],
[=u= → u=],	[=nn → =n],	[=un → ==],	[=nu → ==],
[=uu → =u],	[n== → n=],	[u== → u=],	[n=n → =n],
[u=n → ==],	[n=u → ==],	[u=u → =u],	[nn= → =n],
[nu= → ==],	[un= → ==],	[uu= → =u],	[nnn → nn],
[nun → n=],	[unn → n=],	[nnu → n=],	[uun → u=],
[unu → u=],	[nuu → u=],	[uuu → uu]	

According to our way to describe {n, u, =}, we can convert the above 27 propagation rules in Table 3 into 27 possible values of

$$V_{\texttt{ADD}} = (x_v[i], x_d[i], y_v[i], y_d[i], c_v[i], c_d[i], z_v[i], z_d[i], c_v[i+1], c_d[i+1]).$$

For example, [n=u → ==] corresponds to the possible value $(0, 1, 0, 0, 1, 1, 0, 0, 0, 0)$. With LogicFriday, we can obtain the corresponding linear inequality system:

$$\mathcal{H}_{\texttt{ADD}} \cdot V_{\texttt{ADD}}^T \geq \mathcal{C}_{\texttt{ADD}}.$$

Algorithm 1 describes how to model the deterministic modular addition. Note that we do not make $\nabla c[0] =$ [=] in Algorithm 1 to increase its flexibility and hence before calling it, the value of $\nabla c[0]$ should be clearly specified.

Algorithm 1 Model $\delta z = \delta x \boxplus \delta y$

procedure MODADD_MODEL($\nabla x, \nabla y, \nabla c, \nabla z$)
  for $i = 0$ to 32 **do**
    $V_{\texttt{ADD}} = (x_v[i], x_d[i], y_v[i], y_d[i], c_v[i], c_d[i], z_v[i], z_d[i], c_v[i+1], c_d[i+1])$
    add constraints $\mathcal{H}_{\texttt{ADD}} \cdot V_{\texttt{ADD}}^T \geq \mathcal{C}_{\texttt{ADD}}$.

3.4 Modelling the Expansions of the Modular Difference

Given an arbitrary δz, there are many expansions of δz. For example, there are 3 possible expansions of $\delta z = 2^{30}$ as shown below:

=n== ==== ==== ==== ==== ==== ==== ====,

nu== ==== ==== ==== ==== ==== ==== ====,

uu== ==== ==== ==== ==== ==== ==== ====.

Due to our deterministic way to compute the signed difference transitions through the modular addition $z = x \boxplus y$, we lose many possible ∇z. When it is necessary to compute all possible forms of ∇z, we need to tackle the problem of how to model all the possible $\nabla\xi$ from a given ∇z such that $\delta\xi = \delta z$.

To achieve this, we again introduce an additional variable ∇c with $\nabla c[0] = [\texttt{=}]$. Based on the basic fact that $2^i = 2^{i+1} \boxminus 2^i$, $\boxminus 2^i = \boxminus 2^{i+1} + 2^i$, $0 = 0$, $2^{i+1} = 2^{i+1}$ and $\boxminus 2^{i+1} = \boxminus 2^{i+1}$, we can use the following propagation rules in Table 4 to compute all possible $\nabla\xi$ from ∇z.

Table 4: The propagation rules for $(\nabla z[i], \nabla c[i]) \to (\nabla\xi[i], \nabla c[i+1])$

[nn $\to$ =n], [uu $\to$ =u], [nu $\to$ ==], [un $\to$ ==],
[n= $\to$ (n=, un)], [u= $\to$ (u=, nu)],
[=n $\to$ (n=, un)], [=u $\to$ (u=, nu)],
[== $\to$ ==]

Similarly, the propagation rules in Table 4 can be converted into 13 possible values of

$$V_{\texttt{EXP}} = (z_v[i], z_d[i], c_v[i], c_d[i], \xi_v[i], \xi_d[i], c_v[i+1], c_d[i+1]),$$

Note that [n= $\to$ (n=, un)] corresponds to two possible transitions [n= $\to$ n=] and [n= $\to$ un]. Similar representations will be used throughout this paper. Then, we can obtain the linear inequality system $\mathcal{H}_{\texttt{EXP}} \cdot V_{\texttt{EXP}}^T \geq \mathcal{C}_{\texttt{EXP}}$ to describe Table 4 with LogicFriday.

A slightly different problem. In the procedure to search for signed differential characteristics for the MD-SHA family, it is common to first fix the signed differences of some internal states in advance. In other words, we now consider how to efficiently determine whether a computed ∇z satisfies $\delta\xi \boxminus \delta z = 0$ when $\nabla\xi$ is known and fixed. This is indeed the same with the problem to model the expansions of the modular difference, but we prefer a different method because it does not rely only on a tree structure, i.e. there is no branch.

The following propagation rules for $(\nabla\xi[i], \nabla z[i], \nabla c[i]) \to (\nabla c[i+1])$ are sufficient to constrain $\delta\xi \boxminus \delta z = 0$, where ∇c is the signed difference of the carry bits when computing $\nabla\xi \boxminus \nabla z$ and $\nabla c[0] = [\texttt{=}]$.

[=== $\to$ =],
[=un $\to$ n], [=nn $\to$ =], [=uu $\to$ =], [=nu $\to$ u],
[u=n $\to$ =], [n=n $\to$ n], [u=u $\to$ u], [n=u $\to$ =],
[nu= $\to$ n], [nn= $\to$ =], [uu= $\to$ =], [un= $\to$ u].

These 13 propagations rules can be converted into 13 possible values of

$$V_{\texttt{ZERO}} = (\xi_v[i], \xi_d[i], z_v[i], z_d[i], c_v[i], c_d[i], c_v[i+1], c_d[i+1]).$$

With LogicFriday, we can obtain the corresponding $\mathcal{H}_{\texttt{ZERO}} \cdot V_{\texttt{ZERO}}^T \geq \mathcal{C}_{\texttt{ZERO}}$.

Algorithm 2 describes how to model the expansion of the modular difference. The input `isK` is a binary variable and is used to provide an option to choose different models.

Algorithm 2 Expansion: derive $\nabla\xi$ from ∇z

procedure EXPAND_MODEL($\nabla z, \nabla\xi,$ isK)
  Claim a signed difference vector ∇c of size 33
  $\nabla c[0] = [=]$
  for $i = 0$ to 32 **do**
    if isK $= 1$ **then**
      $V_{\texttt{ZERO}} = (\xi_v[i], \xi_d[i], z_v[i], z_d[i], c_v[i], c_d[i], c_v[i+1], c_d[i+1])$
      $\mathcal{H}_{\texttt{ZERO}} \cdot V_{\texttt{ZERO}}^T \geq \mathcal{C}_{\texttt{ZERO}}$
    else
      $V_{\texttt{EXP}} = (z_v[i], z_d[i], c_v[i], c_d[i], \xi_v[i], \xi_d[i], c_v[i+1], c_d[i+1])$
      add constraints $\mathcal{H}_{\texttt{EXP}} \cdot V_{\texttt{EXP}}^T \geq \mathcal{C}_{\texttt{EXP}}$

3.5 Modelling Boolean Functions

Using some simple boolean functions in the round function is a basic operation in the MD-SHA hash family. For RIPEMD-160, the used boolean functions are shown in Table 2: XOR, ONX, IFZ, IFX and ONZ. Especially, we have

$$w = IFX(x, y, z) = IFZ(y, z, x), \quad w = ONZ(x, y, z) = ONX(z, x, y).$$

The strategies to handle these boolean functions are the same. Due to the space limit, we only explain the difference transitions through $w = ONX(x, y, z)$.

Table 5: The valid values of $(\nabla x[i], \nabla y[i], \nabla z[i], \nabla w[i])$

[====],
[==u=], [==uu], [==un], [==n=], [==nn], [==nu],
[=n==], [=n=n], [=n=u], [=u==], [=u=u], [=u=n],
[n==u], [n==n], [u==n], [u==u],
[=nn=], [=uu=], [=nun], [=nuu], [=unn], [=unu],
[nn=u], [nn==], [nu=u], [nu==], [uu=n], [uu==], [un=n], [un==],
[n=nu], [n=n=], [n=uu], [n=u=], [u=nn], [u=n=], [u=un], [u=u=],
[nnnu], [nnu=], [nun=], [unnn], [uun=], [unu=], [nuuu], [uuun].

The fast filtering model. First, we list all possible $(\nabla x[i], \nabla y[i], \nabla z[i], \nabla w[i])$, as shown in Table 5. Similarly, we can obtain the corresponding inequality system

$$\begin{aligned} \mathcal{H}_{\texttt{ONX}} \cdot V_{\texttt{DF}}^T &\geq \mathcal{C}_{\texttt{ONX}}, \\ V_{\texttt{DF}} &= (x_v[i], x_d[i], y_v[i], y_d[i], z_v[i], z_d[i], w_v[i], w_d[i]). \end{aligned} \tag{2}$$

The full model. In the fast filtering model, we only consider signed difference transitions and ignore the implicit conditions. For example, for $w = ONX(x, y, z)$, when $(\nabla x[i], \nabla y[i], \nabla z[i], \nabla w[i]) =$ [=n==], there is an implicit condition $z[i] = 1$. Ignoring such implicit conditions will cause invalid differential characteristics because each internal state is used three times in such boolean functions to update different internal states at 3 consecutive rounds. To capture such implicit conditions, a full list of possible $(\nabla x[i], \nabla y[i], \nabla z[i], \nabla w[i], x[i], y[i], z[i])$ is provided in Table 6. For convenience, we call $(x[i], y[i], z[i])$ **monitoring variables** as they are used to store the implicit conditions and hence to monitor the contradictions.

Table 6: The valid values of $(\nabla x[i], \nabla y[i], \nabla z[i], \nabla w[i], x[i], y[i], z[i])$, where * represents that the bit value can take either 0 or 1.

[====,*,*,*],
[==u=,*,1,*], [==uu,1,0,*], [==un,0,0,*], [==n=,*,1,*], [==nn,1,0,*], [==nu,0,0,*],
[=n==,*,*,0], [=n=n,0,*,1], [=n=u,1,*,1], [=u==,*,*,0], [=u=u,0,*,1], [=u=n,1,*,1],
[n==u,*,1,*], [n==u,*,0,0], [n==n,*,0,1], [u==n,*,1,*], [u==n,*,0,0], [u==u,*,0,1],
[=nn=,*,*,*], [=uu=,*,*,*], [=nun,0,*,*], [=nuu,1,*,*], [=unn,1,*,*], [=unu,0,*,*],
[nn=u,*,*,0], [nn==,*,*,1], [nu=u,*,*,0], [nu==,*,*,1], [uu=n,*,*,0], [uu==,*,*,1],
[un=n,*,*,0], [un==,*,*,1],
[n=nu,*,1,*], [n=n=,*,0,*], [n=uu,*,1,*], [n=u=,*,0,*], [u=nn,*,1,*], [u=n=,*,0,*],
[u=un,*,1,*], [u=u=,*,0,*],
[nnnu,*,*,*], [nnu=,*,*,*], [nun=,*,*,*], [unnn,*,*,*], [uun=,*,*,*], [unu=,*,*,*],
[nuuu,*,*,*], [uuun,*,*,*].

Similarly, based on Table 6, we can obtain the corresponding

$$\begin{aligned} &\mathcal{H}_{\texttt{ONXFull}} \cdot V_{\texttt{DFC}}^T \geq \mathcal{C}_{\texttt{ONXFull}}, \\ &V_{\texttt{DFC}} = (x_v[i], x_d[i], y_v[i], y_d[i], z_v[i], z_d[i], w_v[i], w_d[i], x[i], y[i], z[i]). \end{aligned} \tag{3}$$

Note that in Table 6, * means it can take either 0 or 1, e.g. [==u=,*,1,*] corresponds to 4 possible values: $(0,0,0,0,1,1,0,0,0,1,0)$, $(0,0,0,0,1,1,0,0,0,1,1)$, $(0,0,0,0,1,1,0,0,1,1,0)$ and $(0,0,0,0,1,1,0,0,1,1,1)$.

It is found that some inequalities appear in both Equation 2 and Equation 3. This is indeed as expected since the information of Table 5 is fully encoded in Table 6. Therefore, to filter invalid signed difference transitions in a faster way, we will actually use the following linear inequality system

$$\begin{cases} \mathcal{H}_{\texttt{ONX}} \cdot V_{\texttt{DF}}^T \geq & \mathcal{C}_{\texttt{ONX}} \\ \mathcal{H}_{\texttt{ONXCut}} \cdot V_{\texttt{DFC}}^T \geq & \mathcal{C}_{\texttt{ONXCut}} \end{cases} \tag{4}$$

to describe Table 6. Specifically, $(\mathcal{H}_{\texttt{ONXCut}}, \mathcal{C}_{\texttt{ONXCut}})$ is obtained by removing the inequalities appearing in Equation 2 from Equation 3. Specifically, we check the inequalities specified by $(\mathcal{H}_{\texttt{ONXFull}}, \mathcal{C}_{\texttt{ONXFull}})$ one by one. If it does not appear in $(\mathcal{H}_{\texttt{ONX}}, \mathcal{C}_{\texttt{ONX}})$, add it to $(\mathcal{H}_{\texttt{ONXCut}}, \mathcal{C}_{\texttt{ONXCut}})$.

In this way, we can equivalently say that $(\mathcal{H}_{\texttt{ONXCut}}, \mathcal{C}_{\texttt{ONXCut}})$ is purely utilized to describe the implicit conditions as $(\mathcal{H}_{\texttt{ONX}}, \mathcal{C}_{\texttt{ONX}})$ can fully describe valid signed difference transitions. This is very important to increase the flexibility of the model as we can add $\mathcal{H}_{\texttt{ONXCut}} \cdot V_{\texttt{DFC}}^T \geq \mathcal{C}_{\texttt{ONXCut}}$ to the model depending on different situations while $\mathcal{H}_{\texttt{ONX}} \cdot V_{\texttt{DF}}^T \geq \mathcal{C}_{\texttt{ONX}}$ is always added. Moreover, the lazy constraint[10] can be applied to $\mathcal{H}_{\texttt{ONXCut}} \cdot V_{\texttt{ONXCut}}^T \geq \mathcal{C}_{\texttt{ONXCut}}$ to improve the performance for some problems. For simplicity, Equation 2 is called the fast filtering model, while Equation 4 is called the full model.

Modelling other boolean functions. The above procedure is rather general and we can apply it to other boolean functions.

For $w = XOR(x, y, z)$, the full model can be described with $\mathcal{H}_{\texttt{XOR}} \cdot V_{\texttt{DF}}^T \geq \mathcal{C}_{\texttt{XOR}}$ and $\mathcal{H}_{\texttt{XORCut}} \cdot V_{\texttt{DFC}}^T \geq \mathcal{C}_{\texttt{XORCut}}$, where the fast filtering model is $\mathcal{H}_{\texttt{XOR}} \cdot V_{\texttt{DF}}^T \geq \mathcal{C}_{\texttt{XOR}}$.

For $w = IFZ(x, y, z)$, the full model can be described with: $\mathcal{H}_{\texttt{IFZ}} \cdot V_{\texttt{DF}}^T \geq \mathcal{C}_{\texttt{IFZ}}$ and $\mathcal{H}_{\texttt{IFZCut}} \cdot V_{\texttt{DFC}}^T \geq \mathcal{C}_{\texttt{IFZCut}}$, where the fast filtering model is $\mathcal{H}_{\texttt{IFZ}} \cdot V_{\texttt{DF}}^T \geq \mathcal{C}_{\texttt{IFZ}}$.

Algorithm 3 describes how to model the signed difference signed difference transitions through Boolean functions.

Algorithm 3 Model the signed difference transitions through Boolean functions

procedure BOOLFAST_MODEL(fNa, $\nabla x, \nabla y, \nabla z, \nabla w$)
  for $i = 0$ to 32 **do**
    $V_{\texttt{DF}} = (x_v[i], x_d[i], y_v[i], y_d[i], z_v[i], z_d[i], w_v[i], w_d[i])$
    if fNa = "ONX" **then**
      add constraints $\mathcal{H}_{\texttt{ONX}} \cdot V_{\texttt{DF}}^T \geq \mathcal{C}_{\texttt{ONX}}$
    else if fNa = "XOR" **then**
      add constraints $\mathcal{H}_{\texttt{XOR}} \cdot V_{\texttt{DF}}^T \geq \mathcal{C}_{\texttt{XOR}}$
    else if fNa = "IFZ" **then**
      add constraints $\mathcal{H}_{\texttt{IFZ}} \cdot V_{\texttt{DF}}^T \geq \mathcal{C}_{\texttt{IFZ}}$
procedure BOOLFULL_MODEL(funName, $\nabla x, \nabla y, \nabla z, \nabla w, x, y, z$)
  for $i = 0$ to 32 **do**
    $V_{\texttt{DFC}} = (x_v[i], x_d[i], y_v[i], y_d[i], z_v[i], z_d[i], w_v[i], w_d[i], x[i], y[i], z[i])$
    if funName="ONX" **then**
      add constraints $\mathcal{H}_{\texttt{ONXCut}} \cdot V_{\texttt{DFC}}^T \geq \mathcal{C}_{\texttt{ONXCut}}$
    else if funName="XOR" **then**
      add constraints $\mathcal{H}_{\texttt{XORCut}} \cdot V_{\texttt{DFC}}^T \geq \mathcal{C}_{\texttt{XORCut}}$
    else if funName="IFZ" **then**
      add constraints $\mathcal{H}_{\texttt{IFZCut}} \cdot V_{\texttt{DFC}}^T \geq \mathcal{C}_{\texttt{IFZCut}}$

3.6 Modelling $a_5 = a_1 \boxplus b_3 \lll s$

This is the special operation in RIPEMD-160 and is another place where contradictions easily occur especially when there are many bit conditions on (a_1, a_5).

[10] In Gurobi, the lazy constraint means the constraints that are checked only after a solution is found.

We also note that we will sometimes decompose this computation as

$$b_4 = b_3 \lll s, b_5 = a_1 \boxplus b_4, a_5 = b_5.$$

Due to our deterministic way to compute ∇b_3, many possible ∇b_3 are lost. One idea is to first compute all possible expansions of δb_3 from ∇b_3. Then, the bitwise rotation only affects the order of variables and we immediately obtain all possible ∇b_4. However, what we need is all possible ∇a_5 where $a_5 = a_1 \boxplus b_4$. If we compute $\nabla b_5 = \nabla a_1 \boxplus \nabla b_4$ for each ∇b_4 with the deterministic model for the modular addition and then compute all possible ∇a_5 from ∇b_5 with the model for the expansion, the expansion is used twice and it is too costly because there are too many combinations. However, in some extreme cases, we will use this idea to avoid the contradictions, i.e. the second strategy stated below.

Indeed, it has been studied in [3,13] that $\delta b_4 = ((b_3 \boxplus \delta b_3) \lll s) \boxminus (b_3 \lll s)$ has at most four possible values for a given δb_3. Therefore, ∇b_4 can be divided into four classes and each class corresponds to different δb_4.

The first strategy. We always choose some δb_4 that hold with a high probability. Then, for each of them, randomly pick one of its expansions ∇b_4. Next, according to $(\nabla a_1, \nabla b_4)$, uniquely determine ∇b_5 with the model for the modular addition. Finally, compute all possible ∇a_5 from ∇b_5 with the model for the expansion. Describing the strategy in words is easy, but how to encode it with linear inequalities?

The most important step is to use linear inequalities to describe how to pick some ∇b_4 holding with a high probability. According to [13], the branch is mainly caused by the carries from the 31st bit and the $(31 - s)$-th bit when computing $b_3 \boxplus \delta b_3$. Therefore, we introduce two variables $(\nabla c_h, \nabla c_m)$ to denote the signed difference of these two carry bits, respectively. Although the two carry bits depend on many bits, we restrict ourselves to only $(\nabla b_3[31], \nabla b_3[30])$ and $(\nabla b_3[31 - s], \nabla b_3[30 - s])$. Then, we fix the propagation rules for

$$(\nabla b_3[31], \nabla b_3[30]) \rightarrow (\nabla b_4[31 + s], \nabla b_4[30 + s], \nabla c_h),$$
$$(\nabla b_3[31 - s], \nabla b_3[30 - s]) \rightarrow (\nabla b_4[31], \nabla b_4[30], \nabla c_m),$$

where the indices are within modulo 32. As the propagation rules are the same for both cases and they are of the same form $(\nabla u, \nabla t) \rightarrow (\nabla \mu, \nabla \tau, \nabla \iota)$, for simplicity, these rules are specified in Table 7. With LogicFriday, Table 7 can be equivalently

Table 7: The propagation rules for $(\nabla u, \nabla t) \rightarrow (\nabla \mu, \nabla \tau, \nabla \iota)$

[== → ===],
[n= → (n==, u=n)], [u= → (u==, n=u)],
[un → =u=], [nu → =n=],[=u → =u=], [=n → =n=],
[nn → =un], [uu → =nu].

described with:

$$\mathcal{H}_{\texttt{ROT}} \cdot V_{\texttt{ROT}}^T \geq \mathcal{C}_{\texttt{ROT}},$$
$$V_{\texttt{ROT}} = (u_v, u_d, t_v, t_d, \mu_v, \mu_d, \tau_v, \tau_d, \iota_v, \iota_d).$$

For the remaining $\nabla b_4[i]$, they are uniquely determined with

$$\nabla b_4[i] = \nabla b_3[i - s] \text{ for } i \notin \{31, 30, 31 + s, 30 + s\}.$$

An algorithmic description of the first strategy can be referred to Algorithm 4. Later, we need to use $\nabla q[0 : s]$ where $\delta q = \delta a_5 \boxminus \delta a_1 = \delta b_4$ to help detect contradictions (ref. Section 3.7). Therefore, we also take ∇q as an input to `ROTATE_DIFF_FIRST` and whether we compute it depends on the variable `isV`.

The second strategy. For the second strategy, we will allow some low-probability propagations $\delta b_3 \to \delta b_4$. This is because when there are many bit conditions on (a_1, a_5), it is possible that such a propagation $\delta b_3 \to \delta b_4$ indeed holds with probability close to 1 under these conditions.

Still we consider $\nabla z = \nabla x \boxplus \nabla y$ and use the variable ∇c to denote the signed differences of the carry bits where $\nabla c[0] = [\texttt{=}]$. The new propagation rules for $(\nabla x[i], \nabla y[i], \nabla c[i]) \to (\nabla z[i], \nabla c[i+1])$ are listed in Table 8. In these new rules, the previous rules for the modular addition and the rules for the expansion are combined in a way, i.e. we will consider branches for the modular addition this time because a_5 is no more an intermediate variable but the final output of the round function. As a result, the new model for the modular addition will become much heavier.

Table 8: The new propagation rules for $(\nabla x[i], \nabla y[i], \nabla c[i]) \to (\nabla z[i], \nabla c[i+1])$

[=== → ==], [(==n, =n=, n==) → (n=, un)], [(==u, =u=, u==) → (u=, nu)],
[(=un, un=, u=n, =nu, nu=, n=u) → ==], [(=uu, uu=, u=u) → =u],
[(=nn, nn=, n=n) → =n], [nnn → nn], [uuu → uu],
[(nnu, unn, nun) → un], [(uun, nuu, unu) → nu].

With LogicFriday, Table 8 can be equivalently described with

$$\mathcal{H}_{\texttt{EXPAdd}} \cdot V_{\texttt{EXPAdd}}^T \geq \mathcal{C}_{\texttt{EXPAdd}}$$
$$V_{\texttt{EXPAdd}} = (x_v[i], x_d[i], y_v[i], y_d[i], c_v[i], c_d[i], z_v[i], z_d[i], c_v[i+1], c_d[i+1]).$$

The model for the signed difference transitions through $a_5 = a_1 \boxplus (b_3 \lll s)$ with the second strategy is also described in Algorithm 4.

3.7 Detecting More Contradictions

It has been stated in Section 2.3 that there are additional implicit conditions. Specifically, for

$$b_4 = b_3 \lll s, b_5 = a_1 \boxplus b_4, a_5 = b_5,$$

Algorithm 4 Model $a_5 = a_1 \boxplus (b_3 \lll s)$

procedure ROTATE_DIFF_FIRST($s, \nabla b_3, \nabla a_1, \nabla a_5, \nabla q$, isV, isK)
    Claim two signed difference vectors $\nabla b_4, \nabla b_5$ of size 32
    for $i = 0$ to $30 - s$ **do**
        $\nabla b_4[i + s \mod 32] = \nabla b_3[i]$
    for $i = 32 - s$ to 30 **do**
        $\nabla b_4[i + s \mod 32] = \nabla b_3[i]$
    Claim a signed difference vector ∇c_0 of size 33
    Claim a signed difference vector ∇c_h
    ROTATE_MODEL($\nabla b_3[31 - s], \nabla b_3[30 - s], \nabla b_4[31], \nabla b_4[30], \nabla c_0[0]$)
    ROTATE_MODEL($\nabla b_3[31], \nabla b_3[30], \nabla b_4[31 + s], \nabla b_4[30 + s], \nabla c_h$)
    MODADD_MODEL($\nabla a_1, \nabla b_4, \nabla c_0, \nabla b_5$)// $\nabla c_0[0]$ is no longer always [=]
    EXPAND_MODEL($\nabla b_5, \nabla a_5$, isK)
    if isV $= 1$ **then**
        Claim a signed difference vector ∇c_1 of size $s + 2$
        $\nabla c_1[0] =$ [=]
        SIGNED_Q_MODEL($\nabla b_4, \nabla c_0[0], \nabla c_1, \nabla q, s$)//$\nabla q[0 : s] = (\nabla b_4 \boxplus \nabla c_0[0])[0 : s]$

procedure ROTATE_MODEL($\nabla u, \nabla t, \nabla \mu, \nabla \tau, \nabla \iota$)
    $V_{\texttt{ROT}} = (u_v, u_d, t_v, t_d, \mu_v, \mu_d, \tau_v, \tau_d, \iota_v, \iota_d)$
    add constraints $\mathcal{H}_{\texttt{ROT}} \cdot V_{\texttt{ROT}}^T \geq \mathcal{C}_{\texttt{ROT}}$

procedure SIGNED_Q_MODEL($\nabla x, \nabla y, \nabla c, \nabla z, s$)
    $V_{\texttt{ADD}} = (x_v[0], x_d[0], y_v, y_d, c_v[0], c_d[0], z_v[0], z_d[0], c_v[1], c_d[1])$
    add constraint $\mathcal{H}_{\texttt{ADD}} \cdot V_{\texttt{ADD}}^T \geq \mathcal{C}_{\texttt{ADD}}$
    for $i = 1$ to $s + 1$ **do**
        $V_{\texttt{ADD}} = (x_v[i], x_d[i], 0, 0, c_v[i], c_d[i], z_v[i], z_d[i], c_v[i + 1], c_d[i + 1])$
        add constraint $\mathcal{H}_{\texttt{ADD}} \cdot V_{\texttt{ADD}}^T \geq \mathcal{C}_{\texttt{ADD}}$

procedure ROTATE_DIFF_SECOND($s, \nabla b_3, \nabla a_1, \nabla a_5, \nabla q$, isV)
    Claim a signed difference vector ∇b_4 of size 32
    EXPAND_MODEL($\nabla b_4, \nabla b_3, 0$)
    ADDEXP_MODEL($\nabla a_1, \nabla b_4 \lll s, \nabla a_5$) //$\nabla b_4 \lll s$ only changes the order of ∇b_4
    if isV $= 1$ **then**
        for $i = 0$ to $s + 1$ **do**
            $\nabla q[i] = \nabla b_4[i - s]$

procedure ADDEXP_MODEL($\nabla x, \nabla y, \nabla z$)
    Claim a signed difference vector ∇c of size 33
    $\nabla c[0] =$ [=]
    for $i = 0$ to 32 **do**
        $V_{\texttt{EXPAdd}} = (x_v[i], x_d[i], y_v[i], y_d[i], c_v[i], c_d[i], z_v[i], z_d[i], c_v[i + 1], c_d[i + 1])$
        add constraints $\mathcal{H}_{\texttt{EXPAdd}} \cdot V_{\texttt{EXPAdd}}^T \geq \mathcal{C}_{\texttt{EXPAdd}}$.

due to the probabilistic propagation $\delta b_3 \to \delta b_4$, there will be conditions on $q = a_5 \boxminus a_1 = b_3 \lll s$, i.e. there should exist a solution of q to the following equations

$$q = a_5 \boxminus a_1, \delta q = \delta a_5 \boxminus \delta a_1, \delta b_3 \boxplus q \ggg s = (\delta q \boxplus q) \ggg s,$$

where $(\delta b_3, \delta a_5, \delta a_1)$ are fixed according to their specified signed differences $(\nabla b_3, \nabla a_5, \delta a_1)$.

Algorithm 5 Detect more contradictions in $a_5 = a_1 \boxplus (b_3 \lll s)$

procedure ROTATE_DIFF_FILTER$(s, \nabla a_5, \nabla a_1, \nabla b_3, \nabla q, a_5, a_1)$
    Claim a binary vector q of size 32
    COMPUTE_Q$(\nabla a_5, \nabla a_1, a_5, a_1, q)$ //compute q
    Claim a binary vector v_0 of size $s+1$
    VAL_DIFF_ADD_MODEL$(\nabla q, q, v_0, s+1)$//compute $v_0 = (\delta q \boxplus q)[0:s]$
    Claim a binary vector v_1 of size $33-s$
    VAL_DIFF_ADD_MODEL$(\nabla b_3, q \ggg s, v_1, 33-s)$//compute v_1
    add constraint $v_0[0] = v_1[32-s]$
    add constraint $v_0[s] = v_1[0]$
procedure COMPUTE_Q$(\nabla z, \nabla x, z, x, q)$
    for $i = 0$ to 32 **do**
        DERIVE_COND$(x[i], \nabla x[i])$//derive conditions on x from ∇x
        DERIVE_COND$(z[i], \nabla z[i])$//derive conditions on z from ∇z
    VAL_ADD_MODEL$(x, q, z, 32)$//$x \boxplus q = z$
procedure DERIVE_COND$(x, \nabla x)$
    //$x = 0$ if $(\nabla x = \texttt{n})$; $x = 1$ if $(\nabla x = \texttt{u})$; x is free if $(\nabla x = \texttt{=})$
    add constraint $-x_v + x \geq 0$
    add constraint $x_v - x_d - x \geq -1$
procedure VAL_DIFF_ADD_MODEL$(\nabla a, b, v, l)$//compute $v = (\delta a \boxplus b)[0:l-1]$
    Claim a signed difference vector ∇c of size l
    $\nabla c[0] = [\texttt{=}]$
    for $i = 0$ to l **do**
        add constraint $2(c_d[i+1] - 2c_v[i+1]) + v[i] = (a_d[i] - 2a_v[i]) + b[i] + (c_d[i] - 2c_v[i])$
        add constraint $c_d[i+1] \geq c_v[i+1]$
procedure VAL_ADD_MODEL(a, b, v, l)//compute $v = (a \boxplus b)[0:l-1]$
    Claim a binary vector c of size l
    $c[0] = 0$
    for $i = 0$ to l **do**
        add constraint $2c[i+1] + v[i] = a[i] + b[i] + c[i]$

In our model, the constraints have ensured that δb_4 is one of the 4 possible values computed from δb_3. Since $\delta b_4 = \delta a_5 \boxminus \delta a_1$, there are always solutions to

$$\delta b_3 \boxplus q \ggg s = (\delta q \boxplus q) \ggg s. \quad (5)$$

The problem exists in the additional constraint $q = a_5 \boxminus a_1$. When there are many bit conditions on (a_5, a_1), the number of possible values of q is significantly reduced and it is possible that none of them can make Equation 5 hold.

As $\delta b_3 \rightarrow \delta b_4$ is a possible propagation, taking the careful analysis in [13] into account, we only need to add the following constraints to make the model automatically detect such contradictions:

$$\begin{aligned} q &= a_5 \boxminus a_1, \\ (\delta q \boxplus q)[0] &= (\delta b_3 \boxplus q \ggg s)[32 - s], \\ (\delta q \boxplus q)[s] &= (\delta b_3 \boxplus q \ggg s)[0]. \end{aligned}$$

In ROTATE_DIFF_FIRST and ROTATE_DIFF_SECOND, we have provided an option to compute $\nabla q[0 : s]$ according to the binary variable isV and therefore it can be viewed as known. Modelling $\delta q \boxplus q$ and $q = a_5 \boxminus a_1$ is trivial, the details of which can be found from the algorithmic description to detect more contradictions, as shown in Algorithm 5.

Algorithm 6 Model the signed difference transitions for $a_5 = a_1 \boxplus (F(a_4, a_3, a_2) \boxplus a_0 \boxplus m \boxplus c) \lll s$.

procedure R(fNa,isC,isF,isV,isK, $s, \nabla m, \nabla a_0, \nabla a_1, \nabla a_2, \nabla a_3, \nabla a_4, \nabla a_5, a_4, a_3, a_2, a_5, a_1$)
    Claim signed difference vectors $\nabla b_0, \nabla b_1, \nabla b_2, \nabla b_3$ of size 32
    Claim signed difference vectors $\nabla c_2, \nabla c_3$ of size 33.
    Claim a signed difference vector ∇q of size $s + 1$.
    $\nabla b_0 = \nabla m$
    BOOLFAST_MODEL(fNa,$\nabla a_4, \nabla a_3, \nabla a_2, \nabla b_1$)
    if isC $= 1$ **then** //involve conditions into the model
        BOOLCOND_MODEL(fNa,$\nabla a_4, \nabla a_3, \nabla a_2, \nabla b_1, a_4, a_3, a_2$)
    $\nabla c_2[0] = [=]$, $\nabla c_3[0] = [=]$//no carry for the least significant bit
    MODADD_MODEL($\nabla b_0, \nabla b_1, \nabla c_2, \nabla b_2$)//$\delta b_2 = \delta b_0 \boxplus \delta b_1$
    MODADD_MODEL($\nabla b_2, \nabla a_0, \nabla c_3, \nabla b_3$)//$\delta b_3 = \delta a_0 \boxplus \delta b_2$
    if isF $= 1$ **then**//use the first strategy
        ROTATE_DIFF_FIRST($s, \nabla b_3, \nabla a_1, \nabla a_5, \nabla q$, isV,isK)
    else//the second strategy
        ROTATE_DIFF_SECOND($s, \nabla b_3, \nabla a_1, \nabla a_5, \nabla q$, isV)
    if isV $= 1$ **then**//further detect contradictions
        ROTATE_DIFF_FILTER($s, \nabla a_5, \nabla a_1, \nabla b_3, \nabla q, a_5, a_1$)

3.8 The Full Model for RIPEMD-160

With the model for all operations known, it is straightforward to combine them to describe the propagation

$$(\nabla a_0, \nabla a_1, \nabla a_2, \nabla a_3, \nabla a_4, \nabla m) \rightarrow \nabla a_5,$$

as shown in Algorithm 6. In the input parameters, `fNa` is the name of the boolean function, `isC` is the option to involve the implicit conditions for the boolean functions, `isF` is the option to use the first or the second strategy to compute ∇b_4, `isV` is the option to perform the further detection of contradictions, and `isK` is the option to use different models for the expansions of the modular difference. In other words, depending on the target parts of the differential characteristics, one can flexibly choose different values for these options.

4 Collision Attacks on 36-Round RIPEMD-160

In our new collision attacks on round-reduced RIPEMD-160, we choose to inject differences in (m_0, m_6, m_9) because this choice can allow a 36-round collision attack. The pattern of the differential characteristic under such message differences is shown in Fig. 2.

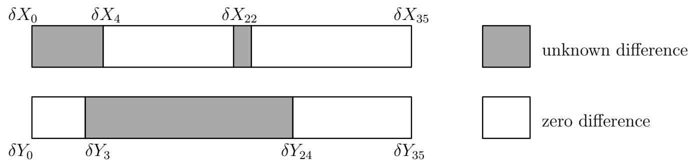

Fig. 2: The pattern of the 36-round differential characteristic

Although we found (m_0, m_6, m_9) according to our experience to analyze the MD-SHA hash family and it is not related to our MILP model, this model is particularly useful when determining their actual modular differences. Specifically, we first considered the message differences of the following form:

$$\delta m_0 = 2^i, \delta m_6 = 0 \boxminus 2^{i+25}, \delta m_9 = 2^{i+12},$$

where the addition in the exponents is modulo 32. However, the obtained differential characteristics are quite unfriendly to the message modification and the probability of the uncontrolled parts is too low. In many cases, the model even outputs that there is no solution for the left branch.

Then, we choose to inject differences in 2 bits of m_0, m_6 and m_9, respectively. For each possible choice, we use the model to minimize $\sum_{i=16}^{24} \mathbb{H}(\nabla Y_i)$. It is found that among all possible choices, the minimal value of $\sum_{i=16}^{24} \mathbb{H}(\nabla Y_i)$ is 12 and we eventually identified the following message differences

$$\delta m_0 = 2^3 \boxplus 2^{22}, \delta m_6 = 0 \boxminus 2^{15} \boxminus 2^{28}, \delta m_9 = 2^2 \boxplus 2^{15}.$$

In addition, with the above message differences, we can also find a suitable solution for the left branch.

In general, with the above message differences, we search for the corresponding collision-generating differential characteristic as follows:

Step 1: Find a valid solution of ∇X_i $(0 \leq i \leq 4)$ and check the differential conditions. If the number of conditions is not that large, just use this solution of ∇X_i $(0 \leq i \leq 4)$ for left branch.

Step 2: Find a valid solution of ∇Y_i $(16 \leq i \leq 24)$ with the MILP model such that $\Delta Y_i = 0$ for $25 \leq i \leq 35$ and we minimize $\sum_{i=16}^{24} \mathbb{H}(\nabla Y_i)$.

Step 3: Find a valid solution of ∇Y_i $(11 \leq i \leq 15)$ with the MILP model such that it can propagate to ∇Y_i $(16 \leq i \leq 24)$ and we minimize $\sum_{i=11}^{15} \mathbb{H}(\nabla Y_i)$.

Step 4: Choose a sparse differential characteristic manually for ∇Y_i $(3 \leq i \leq 5)$ and fix it.

Step 5: Find a solution of ∇Y_i $(6 \leq i \leq 10)$ with the MILP model such that $(\nabla Y_1, \nabla Y_2, \nabla Y_3, \nabla Y_4, \nabla Y_5)$ and $(\nabla Y_{11}, \nabla Y_{12}, \nabla Y_{13}, Y_{14}, \nabla Y_{15})$ can be connected, i.e. the differential characteristic for the right branch is valid.

The found 36-round differential characteristic is displayed in Table 9.

4.1 Fulfilling Differential Conditions

Fulfilling the differential conditions for the 36-round differential characteristic in Table 9 requires nontrivial efforts. Different from the collision attacks on round-reduced RIPEMD-160 [10,11] where the attackers only need to perform the message modification for one branch, we now need to handle the conditions in both branches simultaneously [8] and the differential characteristic is very dense at the first few rounds for both branches.

The general procedure to fulfill the differential conditions is summarized as follows. As in most collision attacks on MD-SHA hash functions, some minor details for the message modification are omitted here because they are trivial.

Step 1: Exhaust all possible solutions of $(Y_4, Y_5, Y_6, Y_7, Y_8, Y_9)$ and compute the corresponding m_6. Store these m_6s in a table denoted by `TAB_M6` and store the tuples $(Y_4, Y_5, Y_6, Y_7, Y_8, Y_9, m_6)$ in a sorted table denoted by `TAB_Y_M6`, which is sorted according to m_6.

Step 2: Exhaust all possible solutions of $(X_1, X_2, X_3, X_4, X_5, X_6)$ and compute the corresponding m_6. If the obtained m_6 is in `TAB_M6_F`, store X_1 in a table denote by `TAB_X1`.

Step 3: Exhaust all possible solutions of $(ONX(Y_{11}, Y_{10}, Y_9 \lll 10), Y_7, Y_8, Y_{12})$ and compute the corresponding m_1. Store these m_1s in a table denoted by `TAB_M1`.

Step 4: Find a valid M^0 such that the conditions on the newly-obtained chaining variable $(X_{-5}, X_{-4}, X_{-3}, X_{-2}, X_{-1}) = H(CV_0, M^0)$ can hold.

Step 5: For the obtained $(X_{-5}, X_{-4}, \ldots, X_{-1})$, exhaust all possible solutions of (X_0, X_1) and compute the corresponding (m_0, m_1). If m_1 is in `TAB_M1` and X_1 is in `TAB_X1`, move to Step 6. Otherwise, try another (X_0, X_1). If all possible values of (X_0, X_1) are traversed, return to Step 4.

Step 6: Exhaust all possible solutions of $(X_2, X_3, X_4, X_5, X_6)$ and compute the corresponding $(m_2, m_3, m_4, m_5, m_6)$. For each obtained m_6, if it is in

Table 9: The 36-round differential characteristic, where $\delta m_0 = 2^3 \boxplus 2^{22}$, $\delta m_6 = 0 \boxminus 2^{15} \boxminus 2^{28}$ and $\delta m_9 = 2^2 \boxplus 2^{15}$.

i	∇X_i	$\pi_l(i)$	i	∇Y_i	$\pi_r(i)$
-5	================================		-5	================================	
-4	================================		-4	================================	
-3	================================		-3	================================	
-2	================================		-2	================================	
-1	================================		-1	================================	
0	nuuuuuuuuuuuuuuuuu=nuuuuuuuuuuu=	0	0	================================	5
1	n===u=u==n=un====uuu=u=nn===u=uu	1	1	================================	14
2	=nun=u=n==n==nn==u==uun==nnu=un=	2	2	========0=====1============1====	7
3	====nu===========nu=============	3	3	====0=========1==n=========0==n1	0
4	nnnnnnnn===unnnnnnnnnnnnnnnunnnn	4	4	=10=n==0=======1=n1=101===1=0010	9
5	================================	5	5	=10=10=0010001=101000n0001110010	2
6	================================	6	6	10001nuunnnnnnnnnnnnnn=un1101110	11
7	================================	7	7	0u0n1uun00n10nu01nnun=nuuuuuuuuu	4
8	================================	8	8	n1un0nuuuu1=0u0un0unnnn1nn0nunuu	13
9	================================	9	9	=1=010u1000n00u01uu010n101=n100n	6
10	================================	10	10	u1=0u0110uu=u011=0=1=0=u1=1=0111	15
11	================================	11	11	111n==0=1=1=0n===11==10100n00==0	8
12	================================	12	12	==00==0=0===10==1=01=n0=1100===1	1
13	================================	13	13	==00=0==u==11===0n=1===1u===u01=	10
14	================================	14	14	==u==0===n===n==1========n===01=	3
15	================================	15	15	======u========1=0=uu====1=n==10	12
16	================================	7	16	===============n=1=============1	6
17	================================	4	17	==0====u=========1==1===========	11
18	================================	13	18	==1==========00=====1===========	3
19	================================	1	19	==========n==11=========n=======	7
20	================================	10	20	===nu=========================0=	0
21	=======u==================u=====	6	21	==========0===========01=0====1=	13
22	=======0==================0=====	15	22	====1=====1======0==u=11=1======	5
23	================1============1==	3	23	n===1=======nu===1==============	10
24	================================	12	24	=======u==============0===u=====	14
25	================================	0	25	======================1======0==	15
26	================================	9	26	=============================1==	8
27	================================	5	27	================================	12
28	================================	2	28	================================	4
29	================================	14	29	================================	9
30	================================	11	30	================================	1
31	================================	8	31	================================	2
32	================================	3	32	================================	15
33	================================	10	33	================================	5
34	================================	14	34	================================	1
35	================================	4	35	================================	3

Table 10: A partial solution for the 36-round differential characteristic

i	∇X_i	$\pi_l(i)$	i	∇Y_i	$\pi_r(i)$
-5	10100101010010101101011111001000		-5	10100101010010101101011111001000	
-4	11101110001000000011110110000011		-4	11101110001000000011110110000011	
-3	11111010101100010100111101100010		-3	11111010101100010100111101100010	
-2	00011100010010000100111100010010		-2	00011100010010000100111100010010	
-1	00111011110101101010010000011111		-1	00111011110101101010010000011111	
0	nuuuuuuuuuuuuuuuu1nuuuuuuuuuuuu1	0	0	10111000110000010010000010111011	5
1	n010u0u11n1un0100uuu1u1nn000u1uu	1	1	11111010100111011001011011100001	14
2	1nun1u0n10n00nn10u10uun01nnu1un1	2	2	01101001000000101010110011010110	7
3	1011nu11111110010nu1110011100011	3	3	0010011111011110 1n001101000010n1	0
4	nnnnnnnn000unnnnnnnnnnnnnnnunnnn	4	4	0101n00001001001 1n10101010110010	9
5	11111110100000100000101011100010	5	5	0100101001000111 01000n0001110010	2
6	00110001111011101111011010111010	6	6	10001nuunnnnnnnnnnnnnnn0un1101110	11
7	01101000000111101011001111000001	7	7	0u0n1uun00n10nu01nnun0nuuuuuuuuu	4
8	10011000010111000010010111111011	8	8	n1un0nuuuu110u0un0unnnn1nn0nunuu	13
9	01111100011111000110101010010100	9	9	110010u1000n00u01uu010n1010n100n	6
10	10000100100000000111100110011011	10	10	u100u0110uu0u0111001101u10100111	15
11	00000110100010001011100111011111	11	11	111n110011110n000110110100n00010	8
12	10100111100100110101011100111110	12	12	000001010010101110010n0111001111	1
13	10011000000000010001000010001011	13	13	10001011u11111010n010001u011u010	10
14	00011010101011010110100101110110	14	14	01u000011n010n01111001101n010010	3
15	00100001000111110011110010111100	15	15	101110u110100001101uu110010n0010	12
16	================================	7	16	110100101100111n0101101000001011	6
17	================================	4	17	0001101u110111011110111101011110	11
18	================================	13	18	10100001110000011100110111111110	3
19	================================	1	19	0111011100n1111001010011n0111111	7
20	================================	10	20	010nu010011000010100110010100101	0
21	=======u==================u=====	6	21	00001110010001011100110110011010	13
22	=======0==================0=====	15	22	0001101111101001101 0u11101001000	5
23	================1============1==	3	23	n===1=======nu===1==============	10
24	================================	12	24	=======u==============0===u=====	14
25	================================	0	25	======================1======0==	15
26	================================	9	26	=============================1==	8
27	================================	5	27	================================	12
28	================================	2	28	================================	4
29	================================	14	29	================================	9
30	================================	11	30	================================	1
31	================================	8	31	================================	2
32	================================	3	32	================================	15
33	================================	10	33	================================	5
34	================================	14	34	================================	1
35	================================	4	35	================================	3
m_0	1111101nuu1111010101110111 00n000		m_8	10001000100110111010111000011100	
m_1	11100010100010101000011010001010		m_9	0110100100111101nu11000111 0001n01	
m_2	01110100001111011001110110000001		m_{10}	10011100001110000100101111001101	
m_3	01100101011111001001111010101101		m_{11}	00100100001110000011000100111110	
m_4	01011010111100011001011001010001		m_{12}	10000110011010100101011001001110	
m_5	10000010000110010100000110001110		m_{13}	00100011101101110011111000101001	
m_6	00un10111111110u011100000100101		m_{14}	10100010111011001101010111011101	
m_7	11011100001000100110001010001000		m_{15}	11010001001001110100011001001011	

TAB_M6, move to Step 7. Otherwise, try another $(X_2, X_3, X_4, X_5, X_6)$. If all possible $(X_2, X_3, X_4, X_5, X_6)$ are traversed, return to Step 5.

Step 7: Compute Y_0 using $(Y_{-5}, Y_{-4}, Y_{-3}, Y_{-2}, Y_{-1}) = (X_{-5}, X_{-4}, X_{-3}, X_{-2}, X_{-1})$ and m_5.

Step 8: Retrieve from TAB_Y_M6 the corresponding $(Y_4, Y_5, Y_6, Y_7, Y_8, Y_9)$ according to m_6. For each possible value, move to Step 9. If all possible values are traversed, return to Step 6.

Step 9: Determine (Y_1, Y_2, Y_3) to connect Y_i $(-5 \leq i \leq 0)$ and Y_j $(4 \leq j \leq 8)$ by using the degrees of freedom provided by $(m_{14}, m_7, m_9, m_{11}, m_{13})$, the details of which will be explained later. If there exists no solution of (Y_1, Y_2, Y_3), return to Step 8.

Step 10: Traverse all possible values of (Y_{10}, Y_{11}) and compute Y_{12} using

$$(Y_7, Y_8, Y_9, Y_{10}, Y_{11}, m_1).$$

Check the conditions[11] on (Y_{12}, Q_8^l) and if they hold, move to Step 11.

Step 11: Traverse all possible values of Y_{13} and compute Y_{14} using

$$(Y_9, Y_{10}, Y_{11}, Y_{12}, Y_{13}, m_3).$$

Check the conditions on Y_{14} and if they hold, move to Step 12.

Step 12: Traverse all possible values of Y_{15} and compute the corresponding m_{12}. Then, Y_i $(-5 \leq i \leq 15)$ are all fixed and therefore all m_j $(0 \leq i \leq 15)$ are fixed. Hence, the remaining internal states X_i $(i \geq 7)$ and Y_j $(j \geq 16)$ can be computed and we check whether the differential conditions on them hold. If they hold, a collision for 36-round RIPEMD-160 is found.

More details about the connection (Step 9). Given Y_i $(-5 \leq i \leq 0)$, Y_j $(4 \leq j \leq 8)$ and (m_0, m_2, m_4), we aim to find a solution of (Y_1, Y_2, Y_3) such that the computed value of (m_0, m_2, m_4) based on Y_i $(-5 \leq i \leq 8)$ is consistent with its given value. This is achieved by using the degrees of freedom provided by $(m_{14}, m_7, m_9, m_{11}, m_{13})$. The procedure is described as follows.

Step 9.1. Exhaust all possible valid Y_3. For each valid Y_3, compute Y_2 using $(Y_3, Y_4, Y_5, Y_6, Y_7, m_4)$. If the conditions on (Q_7^r, Y_2, Q_6^r) hold, move to Step 9.2. Otherwise, try another Y_3 until all possible Y_3 are traversed.

Step 9.2. Note that

$$\begin{aligned} Q_3^r &= ONX(Y_2, Y_1, Y_0 \lll 10) \boxplus Y_{-2} \lll 10 \boxplus K_0^r \boxplus m_0, \\ Y_3 &= Y_{-1} \lll 10 \boxplus Q_3^r \lll s_3^r. \end{aligned} \tag{6}$$

In the above equation, only Y_1 is not yet determined. As

$$ONX(x, y, z) = x \oplus (y \wedge \overline{z}),$$

[11] After computing Y_{11}, m_8 can be computed using Y_i $(6 \leq i \leq 11)$. Then, X_8 and Q_8^l can be computed using X_i $(3 \leq i \leq 7)$ and m_8.

we can uniquely determine $Y_1 \wedge \overline{Y_0 \lll 10}$ according to

$$Q_3^r = (Y_3 \boxminus Y_{-1} \lll 10) \ggg s_3^r,$$
$$ONX(Y_2, Y_1, Y_0 \lll 10) = Q_3^r \boxminus (Y_{-2} \lll 10 \boxplus K_0^r \boxplus m_0),$$
$$Y_1 \wedge \overline{Y_0 \lll 10} = ONX(Y_2, Y_1, Y_0 \lll 10) \oplus Y_2.$$

However, as Y_0 has already been determined, the computed $Y_1 \wedge \overline{Y_0 \lll 10}$ may contradict with Y_0. Specifically, if $Y_0[i] = 0$ and $(Y_1 \wedge \overline{Y_0 \lll 10})[(i + 10) \mod 32] = 1$, the current Y_3 is invalid and we need to try another Y_3. Otherwise, the current Y_3 is correct and we can simply move to Step 9.3 to enumerate valid Y_1 to ensure Equation 6 holds. Specifically, if there are n_0 different indices $\{i_1, i_2, \ldots, i_{n_0}\}$ such that

$$Y_0[i_j] = 0, (Y_1 \wedge \overline{Y_0 \lll 10})[(i_j + 10) \mod 32] = 0 \text{ for } 1 \leq j \leq n_0,$$

there will be 2^{n_0} possible Y_1 and they can be simply enumerated.

Step 9.3. Enumerate all valid Y_1 as explained above. For each Y_1, check the condition on it, i.e. the condition on $Q_5^r = (Y_5 \boxminus Y_1 \lll 10) \ggg s_5^r$. If it holds, compute a new value of m_2 using $(Y_0, Y_1, Y_2, Y_3, Y_4, Y_5)$ and check whether this computed m_2 is consistent with the predetermined m_2. If it is, compute$(m_{14}, m_7, m_9, m_{11}, m_{13})$ using

$$(Y_{-4}, Y_{-3}, Y_{-2}, Y_{-1}, Y_0, Y_1), (Y_{-3}, Y_{-2}, Y_{-1}, Y_0, Y_1, Y_2),$$
$$(Y_{-1}, Y_0, Y_1, Y_2, Y_3, Y_4), (Y_1, Y_2, Y_3, Y_4, Y_5, Y_6), (Y_3, Y_4, Y_5, Y_6, Y_7, Y_8),$$

respectively. Then, compute X_7 and check the conditions on Q_7^l. If the conditions on Q_7^l holds, the connection succeeds and move to Step 10. Otherwise, try another Y_1 until all Y_1 are traversed.

4.2 Complexity Evaluations and Simulations

Let us highlight what we can benefit from our message modification technique. First, we aim to find a valid solution for

$$(X_{-5}, X_{-4}, \ldots, X_6), (Y_{-5}, Y_{-4}, \ldots, Y_9),$$

which corresponds to Step 1 to Step 9. For convenience, we call its solution a **starting point** in our collision attacks. Then, the remaining work is to make the exhaustive search over (Y_{10}, Y_{11}), Y_{13} and Y_{15} in a sequential manner, which is to utilize the degrees of freedom provided by these internal states to fulfill the remaining differential conditions.

On the exhaustive search over $(Y_{10}, Y_{11}, Y_{13}, Y_{15})$***.*** Based on the number of bit conditions, there are in total 2^{20}, 2^{18} and 2^{20} possible values of (Y_{10}, Y_{11}), Y_{13} and Y_{15}, respectively. Suppose there are on average 2^{n_1} possible (Y_{10}, Y_{11}) that can pass Step 10. Moreover, for each valid solution (Y_{10}, Y_{11}, Y_{12}), suppose there are on average 2^{n_2} possible Y_{13} that can pass Step 11. Then, the time complexity to exhaust all possible $(Y_{10}, Y_{11}, \ldots, Y_{15})$ is $2^{20} + 2^{18+n_1} + 2^{20+n_1+n_2} \approx 2^{20+n_1+n_2}$.

Experimental results suggest that $n_1 + n_2 \approx 16.5$ where we performed the exhaustive search over (Y_{10}, Y_{11}, Y_{13}) as stated above for 110 valid starting points. We should note that for some starting points, there is no valid solution of (Y_{10}, Y_{11}, Y_{13}) and the probability that there are valid solutions of them is about 0.36.

From the above analysis, we can equivalently say that for each starting point where $(m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_9, m_{11}, m_{13}, m_{14})$ are fixed, there are about $2^{20+16.5} = 2^{36.5}$ valid values for $(m_{15}, m_8, m_{10}, m_{12})$. More importantly, these $2^{36.5}$ values can be efficiently enumerated by performing the exhaustive search over $(Y_{10}, Y_{11}, Y_{13}, Y_{15})$ as stated above with time complexity $2^{36.5}$, i.e. our exhaustive search strategy is optimal.

On the required number of starting points. If the differential conditions on the uncontrolled internal states Y_i $(i \geq 16)$ and X_j $(j \geq 9)$ hold with probability 2^{-p}, we will need to generate $2^{p-36.5}$ starting points to find a collision. According to the calculation, $p = 8 + 55 + 1.5 = 64.5$ where there are 8 bit conditions on $(X_{19}, X_{20}, \ldots, X_{23})$, 55 bit conditions on $(Y_{16}, Y_{17}, \ldots, Y_{26})$ and about 1.5 bit conditions on (Q^l_{21}, Q^l_{25}) and $(Q^r_{16}, Q^l_{17}, \ldots, Q^l_{28})$. Hence, **it is required to generate about $2^{64.5-36.5} = 2^{28}$ starting points**.

On the complexity of the connection. At Step 9, we will need to exhaust 2^{26} possible values of Y_3 as there are 6 bit conditions on it. Then, we need to check the conditions on $(Q^r_7, Y_2, Q^r_6, Q^r_5)$ which hold with probability of about 2^{-4}. Finally, we need to check the consistency in m_2 which holds with probability 2^{-32} and check the condition on Q^l_7 holding with probability close to 1.

Moreover, even if the conditions on (Q^r_7, Y_2, Q^r_6) hold, Y_3 is still likely to be invalid due to the contradiction between Y_0 and $Y_0 \wedge \overline{Y_1 \lll 10}$. However, this happens only when there exists i such that $Y_0[i] = 0$ and $(Y_0 \wedge \overline{Y_1 \lll 10})[i] = 1$. On the other hand, if $Y_0[i] = 0$ and $(Y_0 \wedge \overline{Y_1 \lll 10})[i] = 0$, we then obtain one free bit in Y_1 and the free bit will be exhausted. Therefore, it is equivalent to stating that there are on average 2^{26} possible (Y_1, Y_3) and they can be exhausted in time 2^{26}.

For each trial of (Y_3, Y_1), the success probability is $2^{-4-32} = 2^{-36}$. Therefore, to generate 2^{28} starting points, we need to try $2^{28+36} = 2^{64}$ times. Hence, **the total time complexity of Step 9 is $2^{28+36} = 2^{64}$.**

On the complexity of Step 8. As there are on average 2^{26} possible valid values for (Y_1, Y_3), the time complexity of Step 8 is $2^{64-26} = 2^{38}$.

On the complexity to exhaust $(X_2, X_3, \ldots, X_6)$. We now evaluate the cost of **Step 6−7** where we need to exhaust all possible values of $(X_2, X_3, \ldots, X_6)$ for a valid $(X_{-5}, X_{-4}, \ldots, X_1)$ obtained at Step 5. By counting the bit conditions, we find that there are in total 28 free bits in $(X_2, X_3, \ldots, X_6)$ for a fixed $(X_{-5}, X_{-4}, \ldots, X_1)$. Hence, the time complexity of this phase is 2^{28}.

For each possible value of $(X_2, X_3, \ldots, X_6)$, m_6 will be computed and checked against `TAB_M6`. Since the size of `TAB_M6` is `0x23a000` $\approx 2^{21.15}$, without considering the conditions on Q_i^l $(2 \leq i \leq 6)$, the matching probability is about $2^{-32+21.15} \approx 2^{-10.9}$. Therefore, we can expect to obtain $2^{28-10.9} = 2^{17.1}$ valid solutions of $(X_{-5}, X_{-4}, \ldots, X_6)$ after the exhaustive search. Experiments suggest that there are about 2^{16} such valid solutions. For each such valid value, we need to move to Step 8.

At Step 8, since each m_6 in `TAB_M6` corresponds to on average 7 different values of $(Y_4, Y_5, \ldots, Y_9)$ in `TAB_Y_M6`, Step 8 can also provide about 2.8 free bits. Hence, **the total time complexity of Step 6−7 is** $2^{38-2.8-16+28} = 2^{47.2}$.

On the complexity to find $(X_{-5}, X_{-4}, \ldots, X_1)$**.** We now evaluate the cost of **Step 4−5**. First, according to the bit conditions on $(X_{-2}, X_{-1}, X_0, X_1)$, for each $(X_{-5}, X_{-4}, \ldots, X_{-1})$ computed from M^0, all state bits of (X_1, X_0) will be directly fixed to fulfill their bit conditions. Hence, we are left to verify whether the computed (X_0, X_1) is valid. First, we need to check whether X_1 is in `TAB_X1` and check whether the corresponding m_1 is in `TAB_M1`. As the size of `TAB_X1` is `7800` $\approx 2^{14.9}$ and the size of `TAB_M1` is `0x1676000` $\approx 2^{24.5}$, the probability it can pass this test is $2^{-17+14.9} \times 2^{-32+24.5} = 2^{-9.6}$. Second, we need to check the conditions on (Q_0^l, Q_1^l) which hold with probability of about $2^{-1.5}$. Therefore, for each computed (X_0, X_1), it is valid with probability $2^{-11.1}$.

Finally, we need to verify the conditions on (X_{-2}, X_{-1}) computed from each M^0 which hold with probability 2^{-30}. Hence, finding a valid $(X_{-5}, X_{-4}, \ldots, X_1)$ requires to try about $2^{30+11.1} = 2^{41.1}$ random M^0. As we need $2^{38-2.8-16} = 2^{19.2}$ such valid solutions, it is required to try $2^{19.2+41.1} = 2^{60.3}$ different M^0. Consequently, **the total time complexity of Step 4−5 is** $2^{60.3}$.

On the complexity of Step 1−3. We only need to perform Step 1−3 once and we have finished Step 1-3 in practical time. Hence, **the total cost of Step 1−3 is negligible**.

The total complexity. According to the above analysis, the time complexity and memory complexity of about collision attacks on 36 rounds of RIPEMD-160 are $2^{64.5}$ and $2^{21.15+2.8} \approx 2^{24}$, respectively.

Simulations. To verify our theoretical analysis and the correctness of our message modification technique, we perform the experiments in the following way. First, we randomly generate $(X_{-5}, X_{-4}, \ldots, X_{-1})$ by always making the conditions on (X_{-2}, X_{-1}) hold because finding their valid values from random M^0 is costly. Then, we compute (X_0, X_1) and check the conditions until we obtain a valid (X_0, X_1). Experimental results match our theoretical analysis for Step 4−5. Next, for each valid $(X_{-5}, X_{-4}, \ldots, X_1)$, we move to Step 6 and try to find valid solutions for $(X_2, X_3, \ldots, X_6)$ and experiments also confirmed our analysis of the time complexity. Then, we move to Step 7−9 to achieve the connection. We find that the success probability of connection is about 2^{-36} and it matches well with our analysis. In this way, we succeed in generating many valid starting points. At last, for each of the obtained starting points, we perform the exhaustive search over $(Y_{10}, Y_{11}, Y_{13}, Y_{15})$ in our way and aim to find a solution for $(Y_{10}, Y_{11}, \ldots, Y_{22})$. The expected time complexity to find a valid $(Y_{10}, Y_{11}, \ldots, Y_{22})$ is about 2^{40} as the conditions on $(Y_{16}, Y_{17}, \ldots, Y_{22})$ hold with probability of about 2^{-40}. Experiments have confirmed this value and we provide a solution of $(m_0, m_1, \ldots, m_{15})$ and $(X_{-5}, X_{-4}, \ldots, X_{-1}) = (Y_{-5}, Y_{-4}, \ldots, Y_{-1})$ which can make the conditions on X_i $(0 \leq i \leq 8)$ and Y_j $(0 \leq j \leq 22)$ hold, as shown in Table 10.

5 Further Works and Discussions

As the round functions of the MD-SHA hash family are very similar, we expect that some of our techniques to model the signed difference transitions can be applied to other hash functions that have not yet been broken. The most important target should be SHA-2. However, there are several obstacles to directly apply our techniques to SHA-2. Specifically, in our model, we implicitly rely on the fact that each 32-bit message word is used to update one 32-bit internal state. When it comes to SHA-2, each message word of 32 (resp. 64) bits will be used to update two different internal states of 32 (resp. 64) bits at the same round. In this case, contradictions will much more easily occur and our techniques to detect the inconsistency are insufficient. How to adapt our techniques to SHA-2 is an interesting and meaningful work.

We also notice that in the paper [6] to improve the automatic tool for SHA-2, it is mentioned that relying on off-the-shelf solvers to search for such differential characteristics is inefficient because the information of the signed difference propagations cannot be well exploited. We believe they referred to the models where two parallel instances of value transitions are considered. Obviously, in our model, we have efficiently encoded the information of the signed difference propagations and we believe this is the first important step towards this problem, i.e. how to efficiently rely on off-the-shelf solvers to find such signed differential characteristics.

For RIPEMD-160, we further made some progress by improving the best collision attack by 2 rounds and we believe this work advances the understanding of RIPEMD-160 further.

Acknowledgement. We thank the reviewers of EUROCRYPT 2023 for improving the quality of this paper. Gaoli Wang is supported by the National Key Research and Development Program of China (No. 2022YFB2701900), National Natural Science Foundation of China (No. 62072181), NSFC-ISF Joint Scientific Research Program (No. 61961146004), Shanghai Trusted Industry Internet Software Collaborative Innovation Center. Fukang Liu is supported by Grant-in-Aid for Research Activity Start-up (Grant No. 22K21282). Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031. This research was in part conducted under a contract of "Research and development on new generation cryptography for secure wireless communication services" among "Research and

Development for Expansion of Radio Wave Resources (JPJ000254)", which was supported by the Ministry of Internal Affairs and Communications, Japan.

References

1. Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP modeling for (large) s-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. **2017**(4), 99–129 (2017). https://doi.org/10.13154/tosc.v2017.i4.99-129
2. Cannière, C.D., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 4284, pp. 1–20. Springer (2006). https://doi.org/10.1007/11935230_1
3. Daum, M.: Cryptanalysis of Hash functions of the MD4-family. Ph.D. thesis, Ruhr University Bochum (2005)
4. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A Strengthened Version of RIPEMD. In: FSE. Lecture Notes in Computer Science, vol. 1039, pp. 71–82. Springer (1996). https://doi.org/10.1007/3-540-60865-6_44
5. Dobraunig, C., Eichlseder, M., Mendel, F.: Analysis of SHA-512/224 and SHA-512/256. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 9453, pp. 612–630. Springer (2015). https://doi.org/10.1007/978-3-662-48800-3_25
6. Eichlseder, M., Mendel, F., Schläffer, M.: Branching Heuristics in Differential Collision Search with Applications to SHA-512. In: FSE. Lecture Notes in Computer Science, vol. 8540, pp. 473–488. Springer (2014). https://doi.org/10.1007/978-3-662-46706-0_24
7. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON Block Cipher Family. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 9215, pp. 161–185. Springer (2015). https://doi.org/10.1007/978-3-662-47989-6_8
8. Landelle, F., Peyrin, T.: Cryptanalysis of Full RIPEMD-128. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7881, pp. 228–244. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_14
9. Leurent, G.: Construction of Differential Characteristics in ARX Designs Application to Skein. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 8042, pp. 241–258. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_14
10. Liu, F., Dobraunig, C., Mendel, F., Isobe, T., Wang, G., Cao, Z.: Efficient Collision Attack Frameworks for RIPEMD-160. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 11693, pp. 117–149. Springer (2019). https://doi.org/10.1007/978-3-030-26951-7_5
11. Liu, F., Dobraunig, C., Mendel, F., Isobe, T., Wang, G., Cao, Z.: New Semi-Free-Start Collision Attack Framework for Reduced RIPEMD-160. IACR Trans. Symmetric Cryptol. **2019**(3), 169–192 (2019). https://doi.org/10.13154/tosc.v2019.i3.169-192
12. Liu, F., Isobe, T., Meier, W.: Automatic Verification of Differential Characteristics: Application to Reduced Gimli. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 12172, pp. 219–248. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_8
13. Liu, F., Mendel, F., Wang, G.: Collisions and Semi-Free-Start Collisions for Round-Reduced RIPEMD-160. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 10624, pp. 158–186. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_6

14. Mendel, F., Nad, T., Scherz, S., Schläffer, M.: Differential Attacks on Reduced RIPEMD-160. In: ISC. Lecture Notes in Computer Science, vol. 7483, pp. 23–38. Springer (2012). https://doi.org/10.1007/978-3-642-33383-5_2
15. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 7073, pp. 288–307. Springer (2011). https://doi.org/10.1007/978-3-642-25385-0_16
16. Mendel, F., Nad, T., Schläffer, M.: Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128. In: FSE. Lecture Notes in Computer Science, vol. 7549, pp. 226–243. Springer (2012). https://doi.org/10.1007/978-3-642-34047-5_14
17. Mendel, F., Nad, T., Schläffer, M.: Improving Local Collisions: New Attacks on Reduced SHA-256. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7881, pp. 262–278. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_16
18. Mendel, F., Peyrin, T., Schläffer, M., Wang, L., Wu, S.: Improved Cryptanalysis of Reduced RIPEMD-160. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 8270, pp. 484–503. Springer (2013). https://doi.org/10.1007/978-3-642-42045-0_25
19. Mironov, I., Zhang, L.: Applications of SAT Solvers to Cryptanalysis of Hash Functions. In: SAT. Lecture Notes in Computer Science, vol. 4121, pp. 102–115. Springer (2006). https://doi.org/10.1007/11814948_13
20. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In: Inscrypt. Lecture Notes in Computer Science, vol. 7537, pp. 57–76. Springer (2011), https://doi.org/10.1007/978-3-642-34704-7_5
21. Ohtahara, C., Sasaki, Y., Shimoyama, T.: Preimage Attacks on Step-Reduced RIPEMD-128 and RIPEMD-160. In: Inscrypt. Lecture Notes in Computer Science, vol. 6584, pp. 169–186. Springer (2010). https://doi.org/10.1007/978-3-642-21518-6_13
22. Shen, Y., Wang, G.: Improved preimage attacks on RIPEMD-160 and HAS-160. KSII Trans. Internet Inf. Syst. **12**(2), 727–746 (2018). https://doi.org/10.3837/tiis.2018.02.011
23. Stevens, M.: New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7881, pp. 245–261. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_15
24. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The First Collision for Full SHA-1. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 10401, pp. 570–596. Springer (2017). https://doi.org/10.1007/978-3-319-63688-7_19
25. Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 4515, pp. 1–22. Springer (2007). https://doi.org/10.1007/978-3-642-03356-8_4
26. 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: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 8873, pp. 158–178. Springer (2014). https://doi.org/10.1007/978-3-662-45611-8_9
27. Wang, G., Shen, Y.: (Pseudo-) Preimage Attacks on Step-Reduced HAS-160 and RIPEMD-160. In: ISC. Lecture Notes in Computer Science, vol. 8783, pp. 90–103. Springer (2014). https://doi.org/10.1007/978-3-319-13257-0_6

28. Wang, G., Shen, Y., Liu, F.: Cryptanalysis of 48-step RIPEMD-160. IACR Trans. Symmetric Cryptol. **2017**(2), 177–202 (2017). https://doi.org/10.13154/tosc.v2017.i2.177-202
29. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 3494, pp. 1–18. Springer (2005). https://doi.org/10.1007/11426639_1
30. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: CRYPTO. Lecture Notes in Computer Science, vol. 3621, pp. 17–36. Springer (2005). https://doi.org/10.1007/11535218_2
31. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 3494, pp. 19–35. Springer (2005). https://doi.org/10.1007/11426639_2
32. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In: CRYPTO. Lecture Notes in Computer Science, vol. 3621, pp. 1–16. Springer (2005). https://doi.org/10.1007/11535218_1