Efficient Collision Attack Frameworks for RIPEMD-160

Fukang Liu[1,6], Christoph Dobraunig[2,3], Florian Mendel[4], Takanori Isobe[5,6], Gaoli Wang[1(✉)], and Zhenfu Cao[1(✉)]

[1] Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
liufukangs@163.com, {glwang,zfcao}@sei.ecnu.edu.cn

[2] Graz University of Technology, Graz, Austria

[3] Radboud University, Nijmegen, The Netherlands
cdobraunig@cs.ru.nl

[4] Infineon Technologies AG, Ludwigsburg, Germany
florian.mendel@gmail.com

[5] National Institute of Information and Communications Technology, Tokyo, Japan

[6] University of Hyogo, Kobe, Japan
takanori.isobe@ai.u-hyogo.ac.jp

Abstract. RIPEMD-160 is an ISO/IEC standard and has been applied to generate the Bitcoin address with SHA-256. Due to the complex dual-stream structure, the first collision attack on reduced RIPEMD-160 presented by Liu, Mendel and Wang at Asiacrypt 2017 only reaches 30 steps, having a time complexity of 2^{70}. Apart from that, several semi-free-start collision attacks have been published for reduced RIPEMD-160 with the start-from-the-middle method. Inspired from such start-from-the middle structures, we propose two novel efficient collision attack frameworks for reduced RIPEMD-160 by making full use of the weakness of its message expansion. Those two frameworks are called dense-left-and-sparse-right (DLSR) framework and sparse-left-and-dense-right (SLDR) framework. As it turns out, the DLSR framework is more efficient than SLDR framework since one more step can be fully controlled, though with extra 2^{32} memory complexity. To construct the best differential characteristics for the DLSR framework, we carefully build the linearized part of the characteristics and then solve the corresponding nonlinear part using a guess-and-determine approach. Based on the newly discovered differential characteristics, we provide colliding messages pairs for the first practical collision attacks on 30 and 31 (out of 80) steps of RIPEMD-160 with time complexity $2^{35.9}$ and $2^{41.5}$ respectively. In addition, benefiting from the partial calculation, we can attack 33 and 34 (out of 80) steps of RIPEMD-160 with time complexity $2^{67.1}$ and $2^{74.3}$ respectively. When applying the SLDR framework to the differential characteristic used in the Asiacrypt 2017 paper, we significantly improve the time complexity by a factor of 2^{13}. However, it still cannot compete with the results obtained from the DLSR framework. To the best of our knowledge, these are the best collision attacks on reduced RIPEMD-160 with respect to the number of steps, including the first colliding message pairs for 30 and 31 steps of RIPEMD-160.

© International Association for Cryptologic Research 2019
A. Boldyreva and D. Micciancio (Eds.): CRYPTO 2019, LNCS 11693, pp. 117–149, 2019.
https://doi.org/10.1007/978-3-030-26951-7_5

Keywords: Hash function · RIPEMD-160 · Start-from-the-middle · Collision attack · Collision

1 Introduction

A cryptographic hash function is a function which takes arbitrary long messages as input and output a fixed-length hash value. Traditionally, such a cryptographic hash function has to fulfill the three basic requirements of collision resistance, preimage resistance and second preimage resistance in order to be considered secure. Most standardized hash functions, like SHA-1, SHA-2, HAS-160, or RIPEMD-160 are based on the Merkle-Damgård paradigm [3,22] which iterates a compression function with fixed-size input to compress arbitrarily long messages. Furthermore, the aforementioned hash functions have in common that their compression function is built by utilization of additions, rotations, xor and boolean functions in an unbalanced Feistel network. This way of designing hash functions has been greatly threatened, starting with a series of results as well as advanced message modification techniques by Wang et al. [28–31].

Before Wang et al. proposed a series of collision attacks on MD-SHA hash family, there existed substantial efforts to analyze the security of MD-SHA hash functions. Historically, the start-from-the-middle structure was first exploited by den Boer et al. at Eurocrypt 1993 to break the compression function of MD5 [6]. Later at FSE 1996, Dobbertin applied the start-from-the-middle approach to break full MD4 [7]. Since the target is the hash function rather than the compression function, the initial value must be consistent with its definition of the primitive, which is costly under the start-from-the-middle structure. To overcome this obstacle, Dobbertin introduced a connecting phase to connect the correct initial value with the starting point in the middle by exploiting the property of the round boolean function and the freedom of message words [7]. As will be shown, our SLDR framework is almost the same with Dobbertin's structure to break MD4. Moreover, the neutral bits introduced by Biham and Chen [1] at Crypto 2004 serve as an important tool to analyze MD-SHA hash family as well till now. A message bit is neutral up to step n if flipping this bit does not influence the differential characteristic conditions up to step n with a high probability. Due to the low diffusion of SHA-0/SHA-1's step functions, there exist many neutral bits up to a few steps.

Soon after Wang et al. presented their exciting work on MD4/MD5/SHA-0/SHA-1, where all the differential characteristics were hand-crafted, De Cannière and Rechberger invented the first automatic search tool to solve the nonlinear part of the differential characteristic of SHA-1 with the guess-and-determine technique [5]. With such a guess-and-determine technique, Mendel et al. designed a tool to find the differential characteristic of SHA-2 at Asiacrypt 2011 [18]. Later, tools to solve the nonlinear characteristics of SHA-2, RIPEMD-128 and RIPEMD-160 progressed well and a series of results were published [10,11,16,17,19–21]. After Wang et al. presented the differential characteristic as well as the corresponding sufficient conditions used to break MD5 in [30], cryptographers soon observed that the differential characteristic conditions were not sufficient in [30]. Specifically, Stevens revealed that the differential rotations must hold if the differential characteristic hold [24]. Consequently, Stevens further investigated the influence of the carry and added some extra bit conditions to have the differential rotations hold with probability close to 1. A highly-related work is the recently proposed method to theoretically calculate the probability of the step function of RIPEMD-160 at Asiacrypt 2017 [16], where the authors introduced the influence of the modular difference propagation and also presented how to add extra conditions for RIPEMD-160 to ensure the modular difference propagates correctly.

The very first theoretical collision attack on full SHA-1 was achieved by Wang et al. at Crypto 2005 [29], which required about 2^{69} calls to SHA-1's compression function. However, practical collisions were still out-of-reach. After a decade's effort, Stevens et al. presented the first practical collision of full SHA-1 at Crypto 2017 [25]. In that work, Stevens et al. utilized the SHA-1 collision search GPU framework [13] and the speed-up techniques such as neutral bits and boomerangs and finally found the practical collision of SHA-1. Boomerangs were introduced by Joux and Peyrin at Crypto 2007 [12] to speed up the collision search for SHA-1. It consists in carefully selecting a few bits that are all flipped together in a way that this effectively flips only one state bit in the first 16 steps, and therefore the diffusion of uncontrollable changes is greatly slowed down.

The RIPEMD family can be considered as a subfamily of the MD-SHA-family, since, for instance, RIPEMD [2] consists of two MD4-like functions computed in parallel with totally 48 steps. The security of RIPEMD was first put into question by Dobbertin [8] and a practical collision attack on it was proposed by Wang et al. [28]. In order to reinforce the security of RIPEMD, Dobbertin, Bosselaers and Preneel [9] proposed two strengthened versions of RIPEMD in 1996, which are RIPEMD-128 and RIPEMD-160 with 128/160 bits output and 64/80 steps, respectively. In order to make both computation branches more distinct from each other, not only different constants, but also different rotation values, message expansions and boolean functions are used for RIPEMD-128 and RIPEMD-160 in both of their branches.

Due to the complicated structure of the dual-stream RIPEMD-128 and RIPEMD-160, collision attacks on the two primitives progressed slowly. For RIPEMD-128, a practical collision attack on 38 steps was achieved at FSE 2012 with a new structure [19]. Later, a practical collision attack on 40 steps was achieved at CT-RSA 2014 [26]. A break-through was made at Eurocrypt 2013, when Landelle and Peyrin employed the start-from-the-middle approach to break the compression function of full RIPEMD-128 [14]. As for RIPEMD-160, no collision attack was presented before Asiacrypt 2017 [16]. However, several results of semi-free-start collision attacks on the compression function of RIPEMD-160 were obtained with the start-from-the-middle approach [17,21], only one of them started from the first step and the remaining started from the middle, further showing the difficulty to cryptanalyze the collision resistance of RIPEMD-160. In the work of [21], a partial calculation to ensure that more uncontrolled bit conditions hold was also introduced with a few statements. Later, a thorough discussion was presented at ToSC 2017 [27].

At Asiacrypt 2017, the authors proposed a strategy to mount collision attacks on the dual-stream RIPEMD-160 [16]. Specifically, they inserted the difference at the message word m_{15}, which is used to update the last internal state of the left branch in the first round. Then, they utilized the search tool [21] to find a differential characteristic whose left branch was linear and sparse and the right branch was as sparse as possible. At last, they applied single-step and multi-step message modification only to the dense right branch to make as many bit conditions as possible hold in a very traditional way, i.e. starting modification from the first step. Typically, multi-step message modification requires a lot of complicated hand-crafted work for different discovered differential characteristics and therefore is very time-consuming. This motivates us to come up with two efficient collision attack frameworks.

Since SHA-3 does not provide the 160-bit digest and the first collision of full SHA-1 has been presented [25], as an ISO/IEC standard, RIPEMD-160 is often used as a drop-in replacement of SHA-1 and therefore worth analyzing. For instance, RIPEMD-160 and SHA-256 have been used to generate the Bitcoin address. For completeness, we list some related work of RIPEMD-160 in Table 1.

This paper is organized as follows. The preliminaries of this paper are introduced in Sect. 2, including some notations, description of RIPEMD-160, start-from-the-middle structure to find collisions, single-step message modification, and how to estimate the probability of the uncontrolled part. In Sect. 3, the details of the two efficient collision attack frameworks are explained. Then, we will show how to construct suitable differential characteristics for the DLSR framework and report the newly discovered 30/31/33/34-step differential characteristics in Sect. 4. The application of the frameworks to the differential characteristics is shown in Sect. 5. Finally, our paper is summarized in Sect. 6.

Table 1. Summary of preimage and collision attack on RIPEMD-160.

Target	Attack Type	Steps	Time	Memory	Ref
comp. function	preimage	31	2^{148}	2^{17}	[23]
hash function	preimage	31	2^{155}	2^{17}	[23]
comp. function	semi-free-start collision	36[a]	low	negligible	[17]
	semi-free-start collision	36	$2^{70.4}$	2^{64}	[21]
	semi-free-start collision	36	$2^{55.1}$	2^{32}	[16]
	semi-free-start collision	42[a]	$2^{75.5}$	2^{64}	[21]
	semi-free-start collision	48[a]	$2^{76.4}$	2^{64}	[27]
hash function	collision	30	2^{70}	negligible	[16]
	collision	30[b]	2^{57}	negligible	Appendix A
	collision	30	$2^{35.9}$	2^{32}	Sect. 5.1
	collision	31	$2^{41.5}$	2^{32}	Sect. 5.2
	collision	33	$2^{67.1}$	2^{32}	Sect. 5.3
	collision	34	$2^{74.3}$	2^{32}	Sect. 5.4

[a] An attack starting at an intermediate step.
[b] Based on the differential characteristic in [16].

Our Contributions. With the start-from-the-middle structure, we propose two efficient collision attack frameworks for reduced RIPEMD-160. For the sake of clearness, we differentiate the two frameworks by dense-left-and-sparse-right (DLSR) framework and sparse-left-and-dense-right (SLDR) framework. The two frameworks significantly simplify the procedure of finding collisions after a differential characteristic is discovered and provide an efficient way to choose the best differential characteristic from many candidates discovered by a search tool. To the best of our knowledge, we obtained the best collision attacks on reduced RIPEMD-160 with respect to the number of steps, including the first practical attack. Specifically, the contribution of this paper can be summarized as follows.

- Two novel efficient collision attack frameworks for reduced RIPEMD-160 are proposed. The DLSR framework is much more efficient than SLDR framework since one more step can be fully controlled, though with extra 2^{32} memory complexity.
- With a guess-and-determine technique, new 30/31/33/34-step differential characteristics of RIPEMD-160 are discovered, whose left branch is dense and right branch is linear and sparse.
- By applying the DLSR framework to the newly discovered 30-step and 31-step differential characteristics, practical collision attacks on 30 and 31 steps of RIPEMD-160 are achieved. The instances of collision are provided as well.
- With the partial calculation technique that fully exploits the property of the round boolean function of RIPEMD-160 and the differential characteristic conditions, we introduce a clever way to dynamically choose the value of free message words under the DLSR framework. Thus, based on the newly discovered 33-step and 34-step differential characteristics, we can mount collision attack on 33 and 34 steps of RIPEMD-160 with time complexity $2^{67.1}$ and $2^{74.3}$ respectively.
- Applying the SLDR framework to the discovered 30-step differential characteristic of Liu, Mendel and Wang [16], we improve the collision attack on 30 steps of RIPEMD-160 by a factor of 2^{13}.

2 Preliminaries

In this section, several preliminaries of this paper will be introduced.

2.1 Notation

For a better understanding of this paper, we introduce the following notations.

1. $\gg$, $\lll$, $\ggg$, $\oplus$, $\vee$, $\wedge$ and $\neg$ represent respectively the logic operation: *shift right, rotate left, rotate right, exclusive or, or, and, negate.*
2. $\boxplus$ and $\boxminus$ represent respectively the modular addition and modular substraction on 32 bits.
3. $M = (m_0, m_1, ..., m_{15})$ and $M' = (m'_0, m'_1, ..., m'_{15})$ represent two 512-bit message blocks.

4. K^l_j and K^r_j represent the constant used at the left and right branch for round j.
5. Φ^l_j and Φ^r_j represent respectively the 32-bit boolean function at the left and right branch for round j.
6. s^l_i and s^r_i represent respectively the rotation constant used at the left and right branch during step i.
7. $\pi_1(i)$ and $\pi_2(i)$ represent the index of the message word used at the left and right branch during step i.
8. X_i, Y_i represent respectively the 32-bit internal state of the left and right branch updated during step i for compressing M.
9. V^j represent the $(j+1)$-th bit of V (V can be $X_i, Y_i, Q_i, F...$), where the least significant bit is the 1st bit and the most significant bit is the 32nd bit. For example, X^0_i represents the least significant bit of X_i.
10. $V^{p\sim q}(0 \leq q < p \leq 31)$ represents the $(q+1)$-th bit to the $(p+1)$-th bit of the 32-bit word V (V can be $X_i, Y_i, Q_i, F...$). For example, $X^{1\sim 0}_i$ represents the two bits X^1_i and X^0_i of X_i.

Moreover, we adopt the concept of generalized conditions in [5]. Some related notations for differential characteristics are presented in Table 2.

Table 2. Notations for differential characteristics

(x, x^*)	(0,0)	(1,0)	(0,1)	(1,1)	(x, x^*)	(0,0)	(1,0)	(0,1)	(1,1)
?	✓	✓	✓	✓	3	✓	✓	–	–
–	✓	–	–	✓	5	✓	–	✓	–
x	–	✓	✓	–	7	✓	✓	✓	–
0	✓	–	–	–	A	–	✓	–	✓
u	–	✓	–	–	B	✓	✓	–	✓
n	–	–	✓	–	C	–	–	✓	✓
1	–	–	–	✓	D	✓	–	✓	✓
♯	–	–	–	–	E	–	✓	✓	✓

• x represents one bit of the first message and x^* represents the same bit of the second message.

2.2 Description of RIPEMD-160

RIPEMD-160 is a 160-bit hash function that uses the Merkle-Damgård construction as domain extension algorithm: the hash function is built by iterating a 160-bit compression function H which takes as input a 512-bit message block M_i and a 160-bit chaining variable CV_i:

$$CV_{i+1} = H(CV_i, M_i)$$

where a message to hash is padded beforehand to a multiple of 512 bits and the first chaining variable is set to the predetermined initial value IV, that is $CV_0 = IV$. We refer to [9] for a detailed description of RIPEMD-160.

The RIPEMD-160 compression function is a wider version of RIPEMD-128 and is based on MD5, but with the particularity that it consists of two different and almost independent parallel instances of it. We differentiate the two computation branches by left and right branch. The compression function consists of 80 steps divided into 5 rounds of 16 steps each in both branches.

Initialization. The 160-bit input chaining variable CV_i is divided into five 32-bit words h_i $(i = 0, 1, 2, 3, 4)$, initializing the left and right branch 160-bit internal state in the following way:

$$X_{-4} = h_0^{\ggg 10}, \quad X_{-3} = h_4^{\ggg 10}, \quad X_{-2} = h_3^{\ggg 10}, \quad X_{-1} = h_2, \quad X_0 = h_1.$$
$$Y_{-4} = h_0^{\ggg 10}, \quad Y_{-3} = h_4^{\ggg 10}, \quad Y_{-2} = h_3^{\ggg 10}, \quad Y_{-1} = h_2, \quad Y_0 = h_1.$$

Particularly, CV_0 corresponds to the following five 32-bit words:

$X_{-4} = Y_{-4} =$ `0xc059d148`, $X_{-3} = Y_{-3} =$ `0x7c30f4b8`, $X_{-2} = Y_{-2} =$ `0x1d840c95`, $X_{-1} = Y_{-1} =$ `0x98badcfe`, $X_0 = Y_0 =$ `0xefcdab89`.

Message Expansion. The 512-bit input message block is divided into 16 message words m_i of size 32 bits. Each message word m_i will be used once in every round in a permuted order π for both branches.

Step Function. At round j, the internal state is updated in the following way.

$$\begin{aligned}
LQ_i &= X_{i-5}^{\lll 10} \boxplus \Phi_j^l(X_{i-1}, X_{i-2}, X_{i-3}^{\lll 10}) \boxplus m_{\pi_1(i)} \boxplus K_j^l,\\
X_i &= X_{i-4}^{\lll 10} \boxplus (LQ_i)^{\lll s_i^l},\\
RQ_i &= Y_{i-5}^{\lll 10} \boxplus \Phi_j^r(Y_{i-1}, Y_{i-2}, Y_{i-3}^{\lll 10}) \boxplus m_{\pi_2(i)} \boxplus K_j^r,\\
Y_i &= Y_{i-4}^{\lll 10} \boxplus (RQ_i)^{\lll s_i^r},
\end{aligned}$$

where i = (1, 2, 3, ..., 80) and j = (0, 1, 2, 3, 4). The details of the boolean functions and round constants for RIPEMD-160 are displayed in Table 3. The other parameters can be found in the specification [9].

Table 3. Boolean Functions and Round Constants in RIPEMD-160

Round 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)	$(x \wedge y) \oplus (\neg x \wedge z)$
2	ONZ	ONZ	0x6ed9eba1	0x6d703ef3	IFZ(x,y,z)	$(x \wedge z) \oplus (y \wedge \neg z)$
3	IFZ	IFX	0x8f1bbcdc	0x7a6d76e9	ONX(x,y,z)	$x \oplus (y \vee \neg z)$
4	ONX	XOR	0xa953fd4e	0x00000000	ONZ(x,y,z)	$(x \vee \neg y) \oplus z$

Finalization. A finalization and a feed-forward is applied when all 80 steps have been computed in both branches. The five 32-bit words h'_i composing the output chaining variable are computed in the following way.

$$\begin{aligned}
h'_0 &= h_1 \boxplus X_{79} \boxplus Y_{78}^{\lll 10},\\
h'_1 &= h_2 \boxplus X_{78}^{\lll 10} \boxplus Y_{77}^{\lll 10},\\
h'_2 &= h_3 \boxplus X_{77}^{\lll 10} \boxplus Y_{76}^{\lll 10},\\
h'_3 &= h_4 \boxplus X_{76}^{\lll 10} \boxplus Y_{80},\\
h'_4 &= h_0 \boxplus X_{80} \boxplus Y_{79}.
\end{aligned}$$

2.3 Start-from-the-Middle Structure

The start-from-the-middle structure was first used to break the compression function of MD5 [6]. However, when applying such a structure to find collisions, an extra phase is essential to match the correct initial value. Historically, Dobbertin was the first to use it to find real collisions [7]. In order to match the correct initial value of MD4, Dobbertin introduced a connecting phase in the framework. Exploting the property of the boolean function and the freedom degree of message words, Dobbertin could achieve a connection with a very low cost. Due to the high cost once there is no efficient approach to achieve a connection, the start-from-the-middle structure is generally applied to find semi-free-start or free-start collisions, which do not require the match with the predefined initial value. Although such a structure has been used to find collisions in [10,15], the situation is much simpler than Dobbertin's work [7]. Specifically, since the length of the middle part is short, only a few message words are fixed [10,15] and the connection can be achieved trivially.

Formally, suppose there are r consecutive internal states $s_1, s_2, ..., s_r$ to be connected, which are updated with the messages words $m_{w_1}, m_{w_2}, ..., m_{w_r}$ respectively. In [7], one of $m_{w_1}, m_{w_2}, ..., m_{w_r}$ is fixed so as to extend the length of the middle part. Therefore, an efficient approach to solve it is non-trivial. For the start-from-the-middle structure used in [10,15] to find real collisions, none of $m_{w_1}, m_{w_2}, ..., m_{w_r}$ are fixed in order to obtain a solution of the middle part. In this situation, they could achieve connection trivially when computing from the first step, i.e. obtain the value of m_{w_i} according to the already computed $s_i, s_{i-1}, ..., s_{i-r}$. However, the length of the middle part is greatly limited, thus leaving more uncontrolled conditions in such a situation. Or else, the authors made a tradeoff and finally determined not to consider the complex situation.

As will be shown in our two frameworks, we also use the start-from-the-middle approach to find real collisions in a complex situation similar to Dobbertin's work [7]. Our motivation is to ensure that as many conditions as possible hold in the second round, which sometimes is almost impossible with multi-step message modification or requires sophisticated and time-consuming manual work. Therefore, in the SLDR framework, one of the message words used to update the internal states to be connected will be fixed. In the DLSR framework, we even fix two of the message words used to update the internal states to be connected, thus greatly extending the controllable part of the differential characteristic and leaving fewer uncontrolled conditions. Fortunately, because of the property of the round boolean function and the weakness of the message expansion of RIPEMD-160, we can manage to achieve a connection with a low cost for the two frameworks.

2.4 Single-Step Message Modification

Since only single-step message modification [28] will be used in this paper, we give a brief description of it. Generally, single-step message modification can ensure all the conditions in the first round for most MD-SHA-like hash functions. The implied reason is that the message words are used for the first time in the first round. Therefore, the attackers can randomly choose values for the internal states while satisfying the conditions in the first round, i.e. randomly choose values for the free bits of internal states. Then, the corresponding message words can be computed according to the already fixed internal states in the first round. For the sake of clearness, we take the step function of RIPEMD-160 as instance.

Suppose the following pattern represents the conditions on X_i.

$$X_i = \texttt{-11- ---- ---- -1-- 1--- n-un -u-- --11}.$$

Then, we can first choose a random value for X_i and then correct it in the following way to ensure the conditions on it hold.

$$\begin{aligned} X_i &\leftarrow X_i \wedge \texttt{0xffff9ff}, \\ X_i &\leftarrow X_i \vee \texttt{0x60048243}. \end{aligned}$$

If there are two-bit conditions on X_i, we then check them and correct them. Suppose $X_i^4 = X_{i-1}^4$ is one two-bit condition, we first check whether $X_i^4 = X_{i-1}^4$ holds. If it does not hold, we simply flip X_i^4. In this way, all conditions on X_i can hold. Finally, we compute the corresponding message word to update X_i with $X_i, X_{i-1}, ..., X_{i-5}$. The above description of single-step message modification is different from the original one [28], but the implied idea is the same.

2.5 Propagation of Modular Difference

At Asiacrypt 2017, theoretical calculation of the probability of the step function of RIPEMD-160 was described by introducing the influence of the propagation of modular difference [16]. The complete description of the calculation procedure is complex. Generally, the authors divided the problem into two parts. The first part is to calculate the characteristics of Q_i (LQ_i/RQ_i for the left/right branch) which satisfies an equation like $(Q_i \boxplus c_0)^{\lll s} = Q_i^{\lll s} \boxplus c_1$ (c_0 and c_1 are constants) to ensure the correct propagation of modular difference. Then, they calculate the probability that the bit conditions on the internal state (X_i/Y_i for left/right branch) hold under the condition that Q_i satisfies the equation $(Q_i \boxplus c_0)^{\lll s} = Q_i^{\lll s} \boxplus c_1$. In other words, they considered the dependency between the bit conditions and the propagation of modular difference and this obviously is a more accurate assumption.

In this paper, since the dense part of the differential characteristic will be first fixed and the remaining part is very sparse and short, we can simply assume the independency between the bit conditions and the propagation of modular difference. Thus, the product of the probability of correct propagation of modular difference and the probability of bit conditions will represent the final probability of the uncontrolled part. Specifically, supposing Q_i (LQ_i/RQ_i for left/right branch) satisfies the equation $(Q_i \boxplus c_0)^{\lll s} = Q_i^{\lll s} \boxplus c_1$ with probability p and there are q bit conditions on the corresponding internal state (X_i/Y_i for left/right branch), then the final probability is $p \times 2^{-q}$. According to our experiments, such an assumption is reasonable and almost consistent with the experiments.

Calculating the probability $(Q_i \boxplus c_0)^{\lll s} = Q_i^{\lll s} \boxplus c_1$ can be found in Daum's Ph.D thesis [4], which was well illustrated in [16] with the help of a table. Due to the space limitation, we refer the readers to Table 3 in [16].

3 Efficient Collision Attack Frameworks

In this section, we will present the details of the two efficient collision attack frameworks. Both frameworks aim at ensuring as many conditions as possible in an efficient way for specific strategies to construct differential characteristics. For the SLDR framework, the differential characteristic is constructed by inserting a difference at the message word m_{15}, which is used to update the last internal state in the first round on the left branch. Moreover, the differential characteristic on the left branch should be linear and sparse. For the DLSR framework, the differential characteristic is constructed by inserting difference at the message word m_{12}, which is used to update the last internal state in the first round on the right branch. In addition, the differential characteristic on the right branch should be linear and sparse. For both frameworks, the linear and sparse branch remains fully probabilistic. The differential characteristic patterns for SLDR and DLSR framework are depicted in Fig. 1.

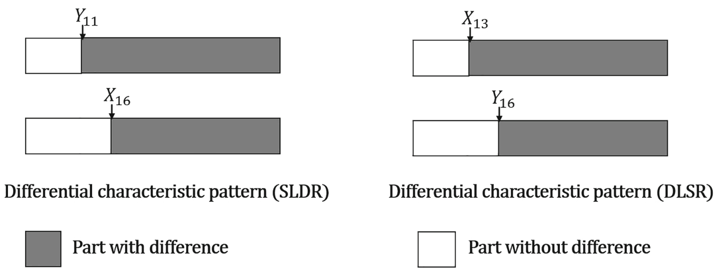

Y_{11}
X_{16}
X_{13}
Y_{16}
Differential characteristic pattern (SLDR)
Differential characteristic pattern (DLSR)
Part with difference
Part without difference

Fig. 1. Illustration of the differential characteristic patterns for both frameworks

3.1 SLDR Collision Attack Framework

Since m_{15} is firstly used to update Y_{11}, for the strategy to build differential characteristic by inserting difference only at m_{15} and making the left branch sparse at Asiacrypt 2017 [16], the following two observations can be obtained.

Observation 1. *There is no condition on* Y_i $(1 \leq i \leq 8)$.

Observation 2. *The first internal state with difference on the right branch is* Y_{11}. *When considering the difference propagating to* Y_{12}, *we are actually considering the difference propagation of* $Y_{11} \oplus (Y_{10} \vee \neg Y_9^{\lll 10})$ *where only* Y_{11} *has differences. If all the bits*$(p_i, p_{i+1}, ..., p_j)$ *with difference in* Y_{11} *are flipped by adding conditions* $Y_{10}^{p_i} = 1$, $Y_{10}^{p_{i+1}} = 1$, ..., $Y_{10}^{p_j} = 1$ *when constructing the differential characteristic, there will not be conditions on* Y_9 *either.*

The above two observations motivate us to consider the start-from-the-middle structure to find collisions. Therefore, we carefully investigated the message expansion on the right branch and finally found an efficient collision attack framework for such a strategy to construct differential characteristics.

The overview of SLDR attack framework is illustrated in Fig. 2. Such a framework contains 4 steps, as specified below and illustrated in Fig. 3.

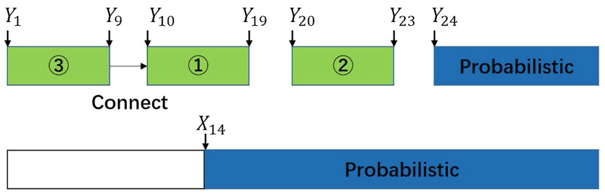


Fig. 2. Overview of SLDR collision attack framework for RIPEMD-160

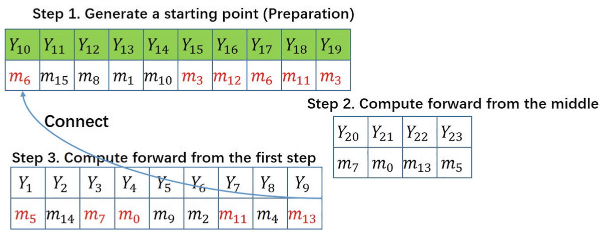

Step 1. Generate a starting point (Preparation)
Y_{10} Y_{11} Y_{12} Y_{13} Y_{14} Y_{15} Y_{16} Y_{17} Y_{18} Y_{19}
m_6 m_{15} m_8 m_1 m_{10} m_3 m_{12} m_6 m_{11} m_3
Step 2. Compute forward from the middle
Y_{20} Y_{21} Y_{22} Y_{23}
m_7 m_0 m_{13} m_5
Connect
Step 3. Compute forward from the first step
Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8 Y_9
m_5 m_{14} m_7 m_0 m_9 m_2 m_{11} m_4 m_{13}

Fig. 3. Specification of SLDR collision attack framework for RIPEMD-160. Message words in red at Step 1 and Step 3 represent their values will be fixed.

Step 1: Fix the internal states located in the middle part from Y_{10} to Y_{19}, which can be easily finished via single-step message modification since only m_3 is used twice to update the internal states. Specifically, randomly choose values for Y_i $(10 \leq i \leq 18)$ while keeping their conditions hold via single-step message modification since $(m_3, m_{12}, m_6, m_{11})$ are used for the first time. Then, we reuse m_3 to compute Y_{19} and check its condition. If the condition does not hold, choose another solution of Y_i $(10 \leq i \leq 18)$ and repeat until we find a solution of Y_i $(10 \leq i \leq 19)$. We call a solution of Y_i $(10 \leq i \leq 19)$ a starting point.

Step 2: Apply single-step message modification to ensure the conditions on Y_i $(20 \leq i \leq 23)$ since their corresponding message words (m_7, m_0, m_{13}, m_5) are used for the first time.

Step 3: Randomly choose values for the free message words m_{14} and m_9. Compute from the first step until Y_5. Then achieve connection in Y_{10}, whose corresponding message word m_6 has been fixed in the starting point. The costly condition $Y_7 = 0$ is used to ensure Y_{10} is irrelevant to Y_8, which can be satisfied by consuming the freedom degree of m_2.

$$\begin{aligned}
Y_7 &= 0.\\
Y_6 &= ((Y_7 \boxminus Y_3^{\lll 10})^{\ggg 15} \boxminus (m_{11} \boxplus K_0^r)) \oplus (Y_5 \vee \overline{Y_4^{\lll 10}}).\\
m_2 &= (Y_6 \boxminus Y_2^{\lll 10})^{\ggg 15} \boxminus (ONX(Y_5, Y_4, Y_3^{\lll 10}) \boxplus Y_1^{\lll 10} \boxplus K_0^r).\\
Y_9 &= ((Y_{10} \boxminus Y_6^{\lll 10})^{\ggg 7} \boxminus (Y_5^{\lll 10} \boxplus m_6 \boxplus K_0^r)) \oplus \texttt{0xffffffff}.\\
Y_8 &= ((Y_9 \boxminus Y_5^{\lll 10})^{\ggg 7} \boxminus (Y_4^{\lll 10} \boxplus m_{13} \boxplus K_0^r)) \oplus (Y_7 \vee Y_6^{\lll 10}),\\
m_4 &= (Y_8 \boxminus Y_4^{\lll 10})^{\ggg 5} \boxminus (ONX(Y_7, Y_6, Y_5^{\lll 10}) \boxplus Y_3^{\lll 10} \boxplus K_0^r).
\end{aligned}$$

Compute m_{15}, m_8, m_1, m_{10} to achieve connection in Y_i $(11 \leq i \leq 14)$. More specifically, m_{15} is computed by Y_i $(6 \leq i \leq 11)$, m_8 is computed by Y_i $(7 \leq i \leq 12)$, m_1 is computed by Y_i $(8 \leq i \leq 13)$ and m_{10} is computed by Y_i $(9 \leq i \leq 14)$.

Step 4: All message words have been fixed after connection. Then we verify the probabilistic parts in both branches. If they do not hold, return Step 2 until we find colliding messages. The degree of freedom is provided by m_0, m_5, m_7, m_9, m_{13} and m_{14}.

Such a general framework can ensure all the bit conditions on Y_i $(10 \leq i \leq 23)$ trivially, which is almost impossible via multi-step message modification once the conditions are dense. However, more attention should be paid when applying it to a specific differential characteristic. In this framework, Y_7 is fixed to zero to achieve an efficient connection in Y_{10}, thus resulting in $RQ_{11} = Y_{11}^{\ggg s_{11}^r}$. If the differential characteristic conditions on Y_{11} always make RQ_{11} fail to satisfy its corresponding equation, this framework cannot be applied directly. Although we can fix some bits of Y_7 to one to solve it, this will influence the success probability of connection. Therefore, when constructing the differential characteristic, such a bad case should be considered and avoided.

3.2 DLSR Collision Attack Framework

Now, we consider an opposite strategy to construct differential characteristics by inserting difference only at m_{12} and making the right branch sparse. In this way, X_{13} is the first internal state with difference. To propagate the difference in X_{13} to X_{14}, we are actually propagating the difference of $X_{13} \oplus X_{12} \oplus X_{11}^{\lll 10}$. Since there is no difference in X_{11} or X_{12} and it is an XOR operation, there will be always conditions on X_{11} and X_{12}. However, there will not be conditions on X_i $(1 \leq i \leq 10)$. This also motivates us to consider the start-from-the-middle approach.

The overview of DLSR framework is shown in Fig. 4. The attack procedure can be divided into four steps as well, as illustrated in Fig. 5.

Step 1: Fix the internal states located in the middle part from X_{11} to X_{23}, which can be easily finished via single-step message modification since only m_{15} is used twice to update the internal states. If there are too many bit conditions on X_{23}, we can firstly fix the internal states from X_{12} to X_{23} via single-step message modification since all the corresponding message words (m_7, m_4, m_{13}, m_1, m_{10}, m_6 and m_{15}) are used for the first time. Then, we compute X_{11} by using X_i $(12 \leq i \leq 16)$ and m_{15}. At last, we check the conditions on X_{11} and the modular difference of X_{15}. If they do not hold, choose another solution of X_i $(12 \leq i \leq 23)$ via single-step message modification and repeat until we can find a solution for the starting point X_i $(11 \leq i \leq 23)$. After a starting point is fixed, we have to achieve connection in five consecutive internal states X_i $(11 \leq i \leq 15)$. However, m_{10} and m_{13} have been already fixed. Thus, an efficient approach to achieve connection in X_{11} and X_{14} is quite important and non-trivial.
To achieve connection in X_{14}, we pre-compute a solution set $\mathbf{S}$ for (X_9, X_{10}) according to the following equation by exhausting all possible values of X_9. For each X_9, compute the corresponding X_{10} and store X_9 in a two-dimensional array with $X_9 \oplus X_{10}$ denoting the row number. Both the time complexity and memory complexity of the pre-computation are 2^{32}.

$$X_{14} = X_{10}{}^{\lll 10} \boxplus (XOR(X_{13}, X_{12}, X_{11}{}^{\lll 10}) \boxplus X_9{}^{\lll 10} \boxplus m_{13} \boxplus K_0^l)^{\lll 7}.$$

Step 2: Apply single-step message modification to ensure the conditions on X_{24} since its corresponding message word m_3 is not fixed in the starting point and is used for the first time. We have to stress that we have considered the influence of the propagation of modular difference and have added extra bit conditions to control its correct propagation with probability 1.

Step 3: Randomly choose values for the free message words m_0, m_2 and m_5. Compute from the first step until X_8 and then achieve connection in X_{11} and X_{14} as follows. First, we calculate the value of $\mathbf{var}$.

$$\mathbf{var} = ((X_{11} \boxminus X_7^{\lll 10})^{\ggg 14} \boxminus (X_6^{\lll 10} \boxplus m_{10} \boxplus K_0^l)) \oplus X_8^{\lll 10}.$$

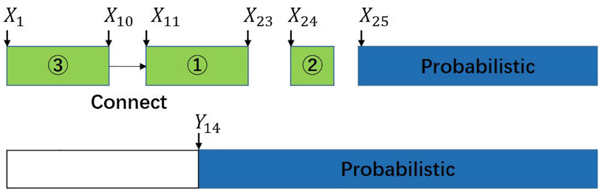
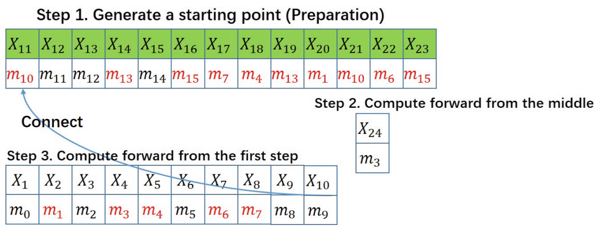


Fig. 4. Overview of DLSR collision attack framework for RIPEMD-160



Step 1. Generate a starting point (Preparation)
X_{11} X_{12} X_{13} X_{14} X_{15} X_{16} X_{17} X_{18} X_{19} X_{20} X_{21} X_{22} X_{23}
m_{10} m_{11} m_{12} m_{13} m_{14} m_{15} m_7 m_4 m_{13} m_1 m_{10} m_6 m_{15}
Step 2. Compute forward from the middle
X_{24}
m_3
Connect
Step 3. Compute forward from the first step
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10}
m_0 m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9

Fig. 5. Specification of DLSR collision attack framework for RIPEMD-160. Message words in red at Step 1 and Step 3 represent their values will be fixed.

Second, find solutions of (X_9, X_{10}) from $\mathbf{S}$ which satisfy $X_9 \oplus X_{10} = \mathbf{var}$. The corresponding solutions are stored in the row numbered $\mathbf{var}$. In this way, each solution of (X_9, X_{10}) will ensure the connection in X_{11} and X_{14}. At last, compute m_8 and m_9 as follows to ensure X_9 and X_{10} can be the computed value for connection. Since there are 2^{32} valid pairs of (X_9, X_{10}) in $\mathbf{S}$ and $\mathbf{var}$ is a random 32-bit variable, we expect one solution of (X_9, X_{10}) for a random $\mathbf{var}$ on average.

$$\begin{aligned} m_8 &= (X_9 \boxminus X_5^{\lll 10})^{\ggg 11} \boxminus (XOR(X_8, X_7, X_6^{\lll 10}) \boxplus X_4^{\lll 10} \boxplus K_0^l). \\ m_9 &= (X_{10} \boxminus X_6^{\lll 10})^{\ggg 13} \boxminus (XOR(X_9, X_8, X_7^{\lll 10}) \boxplus X_5^{\lll 10} \boxplus K_0^l). \end{aligned}$$

Compute m_{11}, m_{12} and m_{14} to achieve connection in X_{12}, X_{13} and X_{15}. Specifically, m_{11} is computed by X_i $(7 \le i \le 12)$, and m_{12} is computed by X_i $(8 \le i \le 13)$, and m_{14} is computed by X_i $(10 \le i \le 15)$.

Step 4: All message words have been fixed after connection. Then we verify the probabilistic part in both branches. If they do not hold, return Step 2 until we find colliding messages. The degree of freedom is provided by m_0, m_2, m_3 and m_5.

However, observe that there will be difference in X_{13} and X_{14} when inserting difference at m_{12}. Therefore, $LQ_{13} = (X_{13} \boxminus X_9^{\lll 10})^{\ggg 6}$ and $LQ_{14} = (X_{14} \boxminus X_{10}^{\lll 10})^{\ggg 7}$ have to satisfy their corresponding equations to ensure the correct propagation of modular difference. Since X_9 and X_{10} cannot be controlled, we have to verify whether LQ_{13} and LQ_{14} satisfy their corresponding equations when obtaining a solution of (X_9, X_{10}). A way to reduce the verifying phase is to filter the wrong pair of (X_9, X_{10}) in the pre-computing phase. However, we cannot expect one solution of (X_9, X_{10}) for a random **var** anymore. In other words, whatever the case is, the influence of the correct propagation of modular difference of X_{13} and X_{14} must be taken into account when estimating the success probability.

Therefore, under our DLSR framework, except the modular difference of X_{13} and X_{14}, all the conditions on X_i ($11 \leq i \leq 24$) can hold trivially with an efficient method, which sometimes is almost impossible with multi-step message modification or at least very time-consuming and requires sophisticated manual work, especially when the conditions are dense in the second round. For the dense left branch, since there is no condition on X_i ($1 \leq i \leq 10$), we only need focus on the the uncontrolled conditions on internal states X_i ($i \geq 25$) and the modular difference of X_{13} and X_{14}. Thus, to construct the best differential characteristic for this framework, there should be minimum active bits in X_i ($i \geq 23$) and the modular difference of X_{13} and X_{14} should hold with a high probability. Moreover, to select the best differential characteristic from many discovered candidates, we only need to analyze the probability of the conditions on X_i ($i \geq 25$), consisting of the number of bit conditions and the influence of the modular difference propagation, as well as the probability of the correct propagation of the modular difference of X_{13} and X_{14}. Obviously, we significantly simplify the procedure to construct and select differential characteristics as well as find collisions with the DLSR framework.

3.3 Comparison

Under the SLDR framework, we can only control until Y_{23} by adding an extra costly condition $Y_7 = 0$ to achieve efficient connection. For the DLSR framework, we can control until X_{24} by consuming extra 2^{32} memory to achieve efficient connection. Hence, the SLDR framework has the obvious advantage of having no memory requirement. However, when there is sufficient memory available, there is a great advantage to leverage the DLSR framework, since we can control the internal state until the 24^{th} step. In other words, one more step can be fully controlled with the DLSR framework, thus having the potential to leave fewer uncontrolled conditions. It should be noted that the number of steps that can be controlled highly depends on the message expansion. Thus, we rely on the specifics of RIPEMD-160's message expansion and extend to more steps as well as find an efficient approach to achieve connection in the complex situation.

A direct application of the SLDR framework to the 30-step differential characteristic in [16] will improve the collision attack by a factor of 2^{11}. With a partial calculation technique, two more uncontrolled bit conditions can be controlled. Thus, the collision attack on 30 steps of RIPEMD-160 is improved to 2^{57}. Actually, the 30-step differential characteristic in [16] is not perfect under our

SLDR framework since there are three bit conditions on Y_9. Although the three bit conditions can be eliminated by generating a new differential characteristic with **Observation 2** taken into account, the time complexity is still too high. As will be shown, we can attack 30 steps of RIPEMD-160 with time complexity $2^{35.9}$ under the DLSR framework. Therefore, considering its improving factor, we decided not to generate a new differential characteristic for the SLDR framework and we refer the readers to Appendix A for the details of the improvement for the collision attack at Asiacrypt 2017 [16]. The source code to verify the correctness of the SLDR framework is available at https://github.com/Crypt-CNS/Improved_Collision_Attack_on_Reduced_RIPEMD-160.git.

Actually, not only the framework but also the characteristic of the fully probabilistic branch has influences on the final effect of the collision attack. Taking the two factors into consideration, we finally determined to utilize the DLSR framework.

4 Differential Characteristics

As stated in the previous section, to construct the best differential characteristic for the DLSR framework, the uncontrolled part should hold with a high probability. To achieve this, according to the boolean function IFX used in the second round on the left branch, we have to control that there are a few active bits in X_i ($i \geq 23$) so that the number of bit conditions on X_i ($i \geq 25$) is minimal. Suppose we will analyze the collision resistance of t steps of RIPEMD-160. According to the finalization phase of the compression function of RIPEMD-160, to achieve a minimal number of active bits in X_i ($i \geq 23$), it is better to let only one of $Y_{t-1}, Y_{t-2}, Y_{t-3}, Y_{t-4}$ have differences and $\Delta Y_t = 0$. In this way, only one of $X_t, X_{t-1}, X_{t-2}, X_{t-3}$ has differences and $\Delta X_{t-4} = 0$.

Based on such a strategy to construct differential characteristics, we firstly determine the characteristics on the fully probabilistic right branch for 30/31/33/34 steps of RIPEMD-160, which can be found in Tables 11, 12, 13 and 14 respectively.

Then, we construct the sparse characteristics on the left branch starting from X_{23} for 30/31/33/34 steps of RIPEMD-160, which are displayed in Table 4.

At last, we utilize a search tool [11,18–21] to solve the nonlinear characteristic located at X_i ($11 \leq i \leq 22$) based on a guess-and-determine technique [5]. To choose the best nonlinear characteristic from many candidates, we only need focus on the conditions on X_i ($i \geq 25$), consisting of the number of bit conditions and the probability of the correct propagation of the modular difference, as well as the probability that LQ_{13} and LQ_{14} satisfy their corresponding equations. The best 30-step, 31-step, 33-step and 34-step differential characteristics for RIPEMD-160 that we eventually determined are displayed in Tables 11, 12, 13 and 14 respectively. To save space, we only list the uncontrolled two-bit conditions located at the fully probabilistic right branch and X_i ($i \geq 25$), which

Table 4. Sparse characteristics on the left branch

i	30 steps of RIPEMD-160	31 steps of RIPEMD-160
23	u------- -------- -------- -----u--	-------n --n----- -------- --------
24	-------- -------- ----n--n --------	-u------ -------- -------- ------u-
25	-------- -------- -------- --------	-------- -------- -------- --------
26	-------- -------- -------- --------	-------- -------- -------- --------
27	-------- -------- -------- --------	-------- -------- -------- --------
28	-------- -------- -------- --------	-------- -------- -------- --------
29	--n----- -------- -------- -------n	-------- -u--u--- -------- --------
30	-------- -------- -------- --------	-------- -------- -------- --------
31		-------- -------- -------- --------

i	33 steps of RIPEMD-160	34 steps of RIPEMD-160
23	-------- u--u---- -------n --------	-n------ -----u-- -------- ------u-
24	--u----- -----u-- -------- -------u	---n---- -------- -----u-- n-------
25	-------- -------- ----n--n --------	-------- --n--u-- -------- --------
26	-------- -----u-- u------- --------	---u--n- -------- -------- --------
27	n--n---- -------- -------- --------	-------- -------- ------n- -u------
28	-------- -------- ----u--u --------	-------- --u--n-- -------- --------
29	-------- -------- -------- --------	-------- -------- -------- --------
30	-------- -------- -------- --------	-------- -------- -------- --------
31	-------- -------- -------- --------	-------- -------- -------- --------
32	-------- -------- -------- --------	-------- -------- -------- --------
33	--u----- -------- -------- -------u	-------- -------- -----u-- n-------
34		-------- -------- -------- --------

cannot be denoted by generalized conditions. The two-bit conditions located at X_i $(11 \leq i \leq 24)$ as well as the equations to ensure the correct propagation of modular difference of X_i $(15 \leq i \leq 24)$ are not listed in the four tables since all these conditions can hold trivially under the DLSR framework. In addition, from the differential characteristics and the corresponding starting points in next section, it is not difficult to extract all these information.

If we construct characteristic for 32 steps of RIPEMD-160 in a similar way, there will be many bit conditions in X_i $(i \geq 23)$, which is even greater than that of 33 steps. This is because $\Delta X_{28} \neq 0$ and $\Delta X_{29} \neq 0$. Therefore, for the attack with high time complexity, we only provide the results for more steps.

Thanks to the efficiency of our DLSR framework, once a differential characteristic for collision attack is determined, the uncontrolled probability can be calculated immediately. Therefore, for each characteristic in Tables 11, 12, 13 and 14, we also present the corresponding total uncontrolled probability in these tables, consisting of the number of bit conditions on the right branch and X_i $(i \geq 25)$, as well as the equations to ensure the correct propagation of modular difference on the right branch and of X_{13}, X_{14} and X_i $(i \geq 25)$. The probability estimated in these four tables represents the success probability to find the collision when the DLSR framework is directly applied to the differential characteristics.

For the best 34-step differential characteristic given in Table 14, a direct application of the DLSR framework is infeasible since it is beyond the birthday attack. However, by benefiting from the partial calculation, which fully exploits the property of the round boolean function and the existing differential characteristic conditions, we significantly improve this probability. Such a technique will be also used to improve the collision attack on 31 and 33 steps of RIPEMD-160. The details will be explained in next section. It should be noted that the effect of partial calculation highly depends on the existing differential characteristic conditions. Therefore, when selecting differential characteristics from many candidates, we actually have taken the effect of partial calculation into account as well.

5 Application

5.1 Practical Collision Attack on 30 Steps of RIPEMD-160

By applying the DLSR framework to the discovered 30-step differential characteristic in Table 11, we can mount collision attack on 30 steps of RIPEMD-160 with time complexity $2^{35.9}$ and memory complexity 2^{32}. It should be noted that there are sufficient free bits in m_0, m_2, m_3 and m_5 to generate a collision. The collision is displayed in Table 5. For completeness, the starting point can be found in Table 7.

5.2 Collision Attack on 31 Steps of RIPEMD-160

According to Table 12, the time complexity to mount collision attack on 31 steps is $2^{42.5}$ if the DLSR framework is directly applied. However, we can make it slightly better with partial calculation technique by using the property of the boolean function IFX. This is based on the following observation.

Table 5. Collision for 30 steps of RIPEMD-160

M	1fbb5316	8ad15821	bf04a498	b85ed58f	4d2d28f6	977b64cd	8c7769dc	961cce16
	9d7a5bc6	f6519d38	37316e69	206d429	2f451be9	e748e57f	5c73a141	e753c86
M'	1fbb5316	8ad15821	bf04a498	b85ed58f	4d2d28f6	977b64cd	8c7769dc	961cce16
	9d7a5bc6	f6519d38	37316e69	206d429	2f449be9	e748e57f	5c73a141	e753c86
hash value	cdcf5aec	cf44ca54	70a8cdbb	e1fd7e6d	bea2687d			

Table 6. Collision for 31 steps of RIPEMD-160

M	3d604874	ff13f724	d60f43b4	c02645eb	a9df768c	172f15dc	d8cfa4bb	edb8f36f
	c898dd5e	71c62ade	d13c6647	bfa932ef	fc2b5325	fc5c01e5	5f7658c8	e5e50cc1
M'	3d604874	ff13f724	d60f43b4	c02645eb	a9df768c	172f15dc	d8cfa4bb	edb8f36f
	c898dd5e	71c62ade	d13c6647	bfa932ef	fc2bd325	fc5c01e5	5f7658c8	e5e50cc1
hash value	5244127c	c976d649	362154bb	59070fc	8e5212e1			

Table 7. Starting points for differential characteristics

30 steps		31 steps	
m_1 = 0x8ad15821, m_4 = 0x4d2d28f6, m_6 = 0x8c7769dc, m_7 = 0x961cce16, m_{10} = 0x37316e69, m_{13} = 0xe748e57f, m_{15} = 0xe753c86.		m_1 = 0xff13f724, m_4 = 0xa9df768c, m_6 = 0xd8cfa4bb, m_7 = 0xedb8f36f, m_{10} = 0xd13c6647, m_{13} = 0xfc5c01e5, m_{15} = 0xe5e50cc1.	
X_{11}	11111100011000001100000101110001	X_{11}	01011100111011101010100010101010
X_{12}	01101010111011111100011001000101	X_{12}	01010101010111001010110111110101
X_{13}	1000111011u100000000010101101011 0	X_{13}	1101nuuuuuuu00000111100011111011 0
X_{14}	010n1110101011100110010000001000	X_{14}	101u0101010001000101101011100101
X_{15}	1n00010011110000100111010 1u11101	X_{15}	0u100010101001100101100111110101
X_{16}	10101111111000000011n110110u0n0110	X_{16}	00100000001000110 01n1100101nn1001
X_{17}	n000010010 1u0110110001010101 1101	X_{17}	u00101010101100nuuuu0100110101nu
X_{18}	01100n001100110000unnu01nun00101	X_{18}	0011010110u100101u001nun1n111001
X_{19}	1100001u00n1nuun0un010000000 10nun	X_{19}	0nu1100un1011110101110u000001111
X_{20}	1u1nn0u011u00110110110101nu1100n	X_{20}	010110111uu11001nn1un00101n10011
X_{21}	10011101100101000001011un100u0uu	X_{21}	01n11n11100110101001110101001000
X_{22}	1001011111uuu00n0011u10010001000	X_{22}	000001100111unnnnnnnnnn01u111100
X_{23}	u110100011110001101000011010 0u11	X_{23}	1001001n11n001000111101011011100
X_{24}	011--1----1------110n01n-----0--	X_{24}	1u10001111111--0-1001101011011u1

33 steps		34 steps	
m_1 = 0xf2470729, m_4 = 0xd19ebad5, m_6 = 0x1f2c0d0e, m_7 = 0xc4f488a9, m_{10} = 0x236883a, m_{13} = 0x8425047b, m_{15} = 0x6458c5e3.		m_1 = 0x58a0be2, m_4 = 0x8d38c100, m_6 = 0x7214c160, m_7 = 0xea755943, m_{10} = 0xa6a0ee3e, m_{13} = 0xb9e9de76, m_{15} = 0xb949ab42.	
X_{11}	10010000100111111001111000001111	X_{11}	10111101100001000010011010001000
X_{12}	00101100110001010110101110010100	X_{12}	01010010100101101101010100000010
X_{13}	010110011 1n0001001101111010 10111	X_{13}	1100001 1nuu11010010001010001010 1
X_{14}	110u0110110001011011110110000111	X_{14}	110n0000000101101000100100100000
X_{15}	1u000010011001111110111001u11101	X_{15}	0n001011110001101101110000u000n0
X_{16}	01111110000000000100n10011un1u0001	X_{16}	00000011001001000 1n0111u00uu0110
X_{17}	n0010111111n1110000u010110u00011	X_{17}	u0010001011010 0n0uuu101n1000101n
X_{18}	000unn1unnn1011101u1u11unn001111	X_{18}	0101101000uu10110111 0110nuu1011u
X_{19}	u0nn101111un110u01011001000u0101	X_{19}	u1nuun01010n01011nnuu0100010111n
X_{20}	0u101110u1110010nn0n1n0u1110u011	X_{20}	1001u00n0nu1u11u0001n11n100u000u
X_{21}	n1n110000uu1u10100n1000n010001un	X_{21}	u1nu0010u1000u001000nu10u100nu10
X_{22}	0100unn010101111 0001111nunnnu0un	X_{22}	01001001 0010n0uuu00uu0uu011nu101
X_{23}	00110111u11u10101101010n00100001	X_{23}	0n01010111001u00111111111 010111u1
X_{24}	11u0101101011u01000101-111100--u	X_{24}	100n011--11111110101-u00n1010111

Observation 3. Let $F = X_{25}X_{24} \oplus \overline{X_{25}}X_{23}^{\lll 10}$, then

$$F^i = \begin{cases} X_{24}^i & (X_{24}^i = X_{23}^{i-10}) \\ X_{24}^i & (X_{25}^i = 1) \\ X_{23}^{i-10} & (X_{25}^i = 0). \end{cases}$$

Note that X_{26} is updated by the free message word m_0 and X_i $(21 \leq i \leq 24)$ can be fully controlled. Although X_{25} cannot be controlled and unknown, we can use partial calculation to ensure several bit conditions on X_{26} hold.

Specifically, consider the 31-step differential characteristic in Table 12. We write X_{25}, X_{24}, $X_{23}^{\lll 10}$ in binary as follows for a better understanding. Consider the following calculation of F, we can know several bits of F if the conditions on X_{25} hold, where a denotes that the bit value is possible to be determined by carefully choosing values of X_{24} and X_{23}, and b denotes that the bit value cannot be determined with existing differential characteristic conditions.

$$\begin{aligned}
X_{25} &= \texttt{10-- ---- ---- ---- ---- ---- ---- -10-}.\\
X_{24} &= \texttt{1u10 0011 1111 1--0 -1-0 1-01 0--- --u1}.\\
X_{23}^{\lll 10} &= \texttt{n001 0001 1110 1-11 01-- ---0 ---- -n-1}.\\
F &= \texttt{10bb 00b1 111b 1aab a1aa aaab aaaa aaa1}.
\end{aligned}$$

Consider the calculation of $sum_0 = X_{21}^{\lll 10} \boxplus K_1^l$ after adding four bit conditions on X_{21}. In this way, the higher 12 bits of sum_0 are constant.

$$\begin{aligned}
X_{21}^{\lll 10} &= \texttt{0110 1010 0110 010- 0--0 0-01 n1-n ---0}.\\
K_1^l &= \texttt{0101 1010 1000 0010 0111 1001 1001 1001}.\\
sum_0 &= \texttt{1100 0100 1110 bbbb bbbb bbbb bbbb bbb1}.
\end{aligned}$$

Then, we consider the calculation of $sum_1 = sum_0 \boxplus m_0$ by pre-fixing the pattern of m_0 as follows.

$$\begin{aligned}
sum_0 &= \texttt{1100 0100 1110 bbbb bbbb bbbb bbbb bbb1}.\\
m_0 &= \texttt{0-11 110- ---- ---- ---- ---- ---- ----}.\\
sum_1 &= \texttt{0b00 00bb bbbb bbbb bbbb bbbb bbbb bbbb}.
\end{aligned}$$

Next, we consider the calculation of $sum_2 = sum_1 \boxplus F$ as follows.

$$\begin{aligned}
sum_1 &= \texttt{0b00 00bb bbbb bbbb bbbb bbbb bbbb bbbb}.\\
F &= \texttt{10bb 00b1 111b 1aab a1aa aaab aaaa aaa1}.\\
sum_2 &= \texttt{1bbb 0bbb bbbb bbbb bbbb bbbb bbbb bbbb}.
\end{aligned}$$

At last, consider the calculation of X_{26} after adding three extra bit conditions on X_{22}.

$$\begin{aligned}
X_{26} &= X_{22}^{\lll 10} \boxplus (F \boxplus X_{21}^{\lll 10} \boxplus K_1^l \boxplus m_0)^{\lll 12} = X_{22}^{\lll 10} \boxplus sum_2^{\lll 12}.\\
sum_2^{\lll 12} &= \texttt{bbbb bbbb bbbb bbbb bbbb 1bbb 0bbb bbbb}.\\
X_{22}^{\lll 10} &= \texttt{11un nnnn nnnn n0-u ---- 0000 01-1 10-1}.\\
X_{26} &= \texttt{bbbb bbbb bbbb bbbb bbbb 1bbb bbbb bbbb}.
\end{aligned}$$

Therefore, $X_{26}^{11} = 1$ can hold with probability 1. In the same procedure to perform the partial calculation, if we choose the following pattern of m_0, $X_{26}^{11} = 1$ can always hold as well.

$$m_0 = \texttt{0100 000- ---- ---- ---- ---- ---- ----}.$$

It should be noted that m_0 is randomly chosen at the third step when applying the DLSR framework. Therefore, with our partial calculation, we can choose the value for m_0 in a clever way to have the condition $X_{26}^{11} = 1$ always hold. Therefore, the time complexity of a collision attack on 31 steps of RIPEMD-160 is improved to $2^{41.5}$.

According to the above analysis, it is not difficult to observe that such an approach to make only one bit condition hold is costly since at least 6 bits of

m_0 have to be fixed. In the case when there are sufficient free bits in the free message words, such a method is feasible. However, when the success probability is low, we have to carefully consume the degree of freedom. As will be shown in the collision attack on 33/34 steps of RIPEMD-160, we dynamically choose a value for m_0 to save the degree of freedom. Moreover, partial calculation will show its significant effect to decrease the time complexity when attacking 33 and 34 steps of RIPEMD-160.

Verification. Both the correctness of the framework and the partial calculation are fully verified. The collision for 31 steps of RIPEMD-160 is displayed in Table 6 and the corresponding starting point is provided in Table 7.

5.3 Collision Attack on 33 Steps of RIPEMD-160

If we directly apply the DLSR framework to the discovered 33-step differential characteristic in Table 13, the time complexity is $2^{71.6}$ and the memory complexity is 2^{32}. With the partial calculation, we can choose m_0 in a clever way to ensure more uncontrolled bit conditions hold.

Write X_{25}, X_{24}, $X_{23}^{\lll 10}$ in binary according to Table 13 as follows for a better understanding. Thus, several bits of $F = X_{25}X_{24} \oplus \overline{X_{25}}X_{23}^{\lll 10}$ can be known if the conditions on X_{25} hold based on **Observation 3**.

$$\begin{aligned}
X_{25} &= \texttt{-11- ---- ---- -1-- 1--- n--n ---- --11.}\\
X_{24} &= \texttt{11u0 10-- 0--1 1u01 0001 01-1 1110 0--u.}\\
X_{23}^{\lll 10} &= \texttt{1u10 --1- 0-01 0n00 100- 0100 1-01 1-u1.}\\
F &= \texttt{1110 aaaa 0aa1 b10b 000a 01a0 1abb baa1.}
\end{aligned}$$

Consider the calculation of X_{26},

$$X_{26} = X_{22}^{\lll 10} \boxplus (F \boxplus X_{21}^{\lll 10} \boxplus K_1^l \boxplus m_0)^{\lll 12}.$$

Observe that the higher 12 bits of F can be fully fixed by properly setting values for X_{24} and X_{23}. Moreover, $X_{21}^{\lll 10} \boxplus K_1^l$ and $X_{22}^{\lll 10}$ are all constants after a starting point is found. Therefore, it is feasible to have a clever choice of the higher 12 bits of m_0 rather than in a random way to ensure the conditions on the lower 12 bits of X_{26}. To explain more precisely, we firstly present the starting point of the 33-step differential characteristic in Table 7.

From this starting point, the following information can be extracted.

$$\begin{aligned}
F \wedge \texttt{0xfffc0000} &= \texttt{1110 1011 0101 b100 0000 0000 0000 0000.}\\
X_{21}^{\lll 10} \boxplus K_1^l &= \texttt{0100 1110 1100 0011 1001 0010 1111 1010.}
\end{aligned}$$

Then, we add some extra conditions on m_0 to ensure that there is always a carry from the 20th bit to 21st bit when calculating $F \boxplus X_{21}^{\lll 10} \boxplus K_1^l \boxplus m_0$. The reason why there is a carry is as follows. Suppose $sum_3 = X_{21}^{\lll 10} \boxplus K_1^l \boxplus m_0$. When m_0

satisfies such a pattern, $sum_3^{19\sim18} = 11_2$. Since $F^{18} = 1$, there will be always carry from the 20th bit when calculating $F \boxplus sum_3$.

$$\begin{aligned} F \wedge \texttt{0xfffc0000} &= \texttt{1110 1011 0101 b100 0000 0000 0000 0000}. \\ X_{21}^{\lll 10} \boxplus K_1^l &= \texttt{0100 1110 1100 0011 1001 0010 1111 1010}. \\ m_0 &= \texttt{---- ---- ---- 101- ---- ---- ---- ----}. \end{aligned}$$

Therefore,

$$(F \wedge \texttt{0xfff00000}) \boxplus ((X_{21}^{\lll 10} \boxplus K_1^l) \wedge \texttt{0xfff00000}) \boxplus \texttt{0x100000} = \texttt{0x3a200000}.$$

Moreover, to ensure that the modular difference of X_{26} can hold with a probability close to 1, we add an extra bit condition $X_{26}^9 = 1$. The reason can be found in the following calculation of $LQ_{26}^{\lll 12} = X_{26} \boxminus X_{22}^{\lll 10}$. In this way, $LQ_{26}^{31\sim30} = 00_2$ can hold with probability 1, thus resulting $(LQ_{26} \boxplus \texttt{0x407fff7e})^{\lll 12} = LQ_{26}^{\lll 12} \boxplus \texttt{0xfff7e408}$ holds with a probability close to 1.

$$\begin{aligned} X_{26} &= \texttt{---1 ---- ---- -u-- u--- 1010 1--- ----}. \\ X_{22}^{\lll 10} &= \texttt{1011 ---- 0--- -nun nnu0 un0- 0-un n010}. \\ LQ_{26}^{\lll 12} &= \texttt{---- ---- ---- ---- ---- 00-- ---- ----}. \end{aligned}$$

After the above preparation, we give a complete description of how to choose m_0 in a clever way to ensure the bit conditions on the lower 12 bits of X_{26}. After choosing values for X_{24} via single-step message modification and computing the corresponding m_3, we will determine the value of m_0 according to the following procedure.

step 1: Randomly choose values for the lower 12 bits of X_{26} while keeping the conditions on this part hold.
step 2: Compute the lower 12 bits of $X_{26} \boxminus X_{22}^{\lll 10}$. Then, the higher 12 bits of LQ_{26} are known.
step 3: Based on $LQ_{26} = m_0 \boxplus F \boxplus X_{21}^{\lll 10} \boxplus K_1^l$, we can compute the higher 12 bits of m_0 since the higher 12 bits of LQ_{26} and $F \boxplus X_{21}^{\lll 10} \boxplus K_1^l$ as well as the carry from the 20-th bit are all known. The remaining free bits of m_0 are set to a random value.

In this way, we can ensure that 4 extra bit conditions on X_{26} and the modular difference of it hold. Therefore, the time complexity of collision attack on 33 steps of RIPEMD-160 becomes $2^{71.6-4.5} = 2^{67.1}$. It should be noted that there are sufficient free bits in m_0, m_2, m_3 and m_5 to generate a collision even though m_0 is not fully random anymore. Specifically, it is equivalent to fixing 8 bits of m_0.

Verification. Our program has verified the correctness of the above optimizing strategy of partial calculation. Moreover, due to the low time complexity of the left branch after applying such a strategy, we can find a group solution of message words to ensure the dense left branch as shown in Table 8.

Table 8. Solution of dense left branch

Solution for 33-step left branch							
m_0	0xdc0b0468	m_1	0xf2470729	m_2	0xee83478c	m_3	0x3c25962
m_4	0xd19ebad5	m_5	0x1aed1d2b	m_6	0x1f2c0d0e	m_7	0xc4f488a9
m_8	0x586e5bed	m_9	0x1a444ebb	m_{10}	0x236883a	m_{11}	0xd38ea539
m_{12}	0x61e4d55f	m_{13}	0x8425047b	m_{14}	0xe8649646	m_{15}	0x6458c5e3
Solution for 34-step left branch							
m_0	0xc2056cdf	m_1	0x58a0be2	m_2	0xe114b874	m_3	0xb7f045ff
m_4	0x8d38c100	m_5	0x4e926b96	m_6	0x7214c160	m_7	0xea755943
m_8	0x496a5788	m_9	0x857f0518	m_{10}	0xa6a0ee3e	m_{11}	0xcd1f88a9
m_{12}	0x14a4951c	m_{13}	0xb9e9de76	m_{14}	0x65df3f3a	m_{15}	0xb949ab42

5.4 Collision Attack on 34 Steps of RIPEMD-160

The best 34-step differential characteristic is displayed in Table 14. A direct application of the DLSR framework to this differential characteristic is infeasible since the uncontrolled part holds with probability $2^{-81.4}$. Fortunately, we can exploit the partial calculation of X_{26} as above to ensure a lot of bit conditions on X_{26} hold. Different from the 33-step differential characteristic where the lower 12 bits of X_{26} can be controlled with probability 1, only the higher 20 bits of X_{26} can be controlled with probability 2^{-2} for the discovered 34-step differential characteristic. However, there are a lot of conditions on the higher 20 bits of X_{26}. Therefore, there is a great advantage if exploiting such a strategy even though it succeeds with probability 2^{-2}. The details will be explained in the following, which share many similarities with the procedure for the 33-step differential characteristic.

Let $F = X_{25}X_{24} \oplus \overline{X_{25}}X_{23}^{\lll 10}$. We write X_{25}, X_{24}, $X_{23}^{\lll 10}$ in binary according to Table 14 as follows. Thus, many bits of F can be controlled by properly choosing values for the free bits of X_{24} and X_{23}.

$$
\begin{aligned}
X_{25} &= \texttt{---1 ---- --n- -u0- ---- 00-1 1--- ----}.\\
X_{24} &= \texttt{100n 011- -111 111- -10- -u00 n10- -111}.\\
X_{23}^{\lll 10} &= \texttt{001u -01- 1-11 -101 011- u1-n -1-- ----}.\\
F &= \texttt{b0b0 ab1a aa11 a10a a1ba 11a0 01aa aaaa}.
\end{aligned}
$$

Consider the calculation of X_{26},

$$X_{26} = X_{22}^{\lll 10} \boxplus (F \boxplus X_{21}^{\lll 10} \boxplus K_1^l \boxplus m_0)^{\lll 12}.$$

Observe that there are only two possible values for the lower 20 bits of F depending on X_{25}^{13} after setting values for X_{24} and X_{23} properly. Moreover, $X_{21}^{\lll 10} \boxplus K_1^l$ and $X_{22}^{\lll 10}$ are all constants after a starting point is found. Therefore, it is feasible to have a clever choice of the lower 20 bits of m_0 rather than in a random way to ensure the conditions on the higher 20 bits of X_{26}. To explain more precisely, we firstly present the starting point of the 34-step differential characteristic in Table 7.

From this starting point, the following information can be extracted.

$$F \wedge \texttt{0x000fffff} = \texttt{0000 0000 0000 1101 01b1 1100 0101 0111}.$$
$$X_{21}^{\lll 10} \boxplus K_1^l = \texttt{0110 1100 1001 1101 1001 0100 1110 0100}.$$

Therefore, $(F \boxplus X_{21}^{\lll 10} \boxplus K_1^l) \wedge \texttt{0x000fffff}$ can only take two possible values, which are `0xaf13b` and `0xb113b`.

Moreover, it should be observed that the modular difference of X_{26} holds with a very low probability of $2^{-3.1}$. Therefore, adding extra bit conditions to control the modular difference is vital as well. We add four extra bit conditions $X_{26}^{31} = 1$, $X_{26}^{30} = X_{22}^{20}$, $X_{26}^{29} = 0$ and $X_{26}^{27} = 0$, all of which are located at the higher 20 bits of X_{26}. The reason can be found in the following calculation of $LQ_{26}^{\lll 12} = X_{26} \boxminus X_{22}^{\lll 10}$. In this way, $LQ_{26}^{19\sim 16} = 0000_2$ can hold with probability 1, thus resulting $(LQ_{26} \boxplus \texttt{0xe06be})^{\lll 12} = LQ_{26}^{\lll 12} \boxplus \texttt{0xe06be000}$ holds with probability 1.

$$X_{26} = \texttt{1-0u 0-n- --00 -11- ---- ---- -1-- ----}.$$
$$X_{22}^{\lll 10} = \texttt{1-n0 uuu0 -uu0 uu01 1nu1 01-1 00-0 --0-}.$$
$$LQ_{26}^{\lll 12} = \texttt{0000 ---- ---- ---- ---- ---- ---- ----}.$$

Since we are trying to control the higher 20 bits of X_{26}, the influence of the carry from the 12th bit must be taken into account when calculating $X_{22}^{\lll 10} \boxplus LQ_{26}^{\lll 12}$. The carry behaves randomly since m_0 is random and the higher 12 bits of $F \boxplus X_{21}^{\lll 10} \boxplus K_1^l$ are random. However, since $X_{22}^{1\sim 0} = 01_2$, there is a bias that there is no carry from the 12th bit. Therefore, in the implementation, we always assume there is no carry, which holds with probability slightly higher than 2^{-1}.

After the above preparation, we give a complete description of how to choose m_0 in a clever way to ensure the 10 bit conditions on the higher 20 bits of X_{26}. After choosing values for X_{24} via single-step message modification and computing the corresponding m_3, we will determine the value of m_0 in the following procedure.

step 1: Randomly choose values of the higher 20 bits of X_{26} while keeping the 10 bit conditions on this part hold.

step 2: Compute the higher 20 bits of $X_{26} \boxminus X_{22}^{\lll 10}$ by assuming there is no carry from the 12th bit. Then, the lower 20 bits of LQ_{26} are known.

step 3: Based on $LQ_{26} = m_0 \boxplus F \boxplus X_{21}^{\lll 10} \boxplus K_1^l$, we can compute the lower 20 bits of m_0 since the lower 20 bits of LQ_{26} and $F \boxplus X_{21}^{\lll 10} \boxplus K_1^l$ are known. Randomly choose one value of the 20 bits of $F \boxplus X_{21}^{\lll 10} \boxplus K_1^l$ from the two possible values and compute the corresponding lower 20 bits of m_0. The remaining free bits of m_0 are set to a random value.

In this way, we can ensure that 6 bit conditions on X_{26} and the modular difference of it hold. Therefore, the time complexity of collision attack on 33 steps of RIPEMD-160 is improved to $2^{81.4-9.1+2} = 2^{74.3}$. It should be noted that there are sufficient free bits in m_0, m_2, m_3 and m_5 to generate a collision even though m_0 is not fully random anymore. Specifically, it is equivalent to fixing 10 bits of m_0.

Verification. The above partial calculation to ensure 10 bit conditions on the higher 20 bits of X_{26} has been verified with the program, which is consistent with our estimated success probability $2^{-1-1} = 2^{-2}$. In addition, we also found a solution for the dense left branch as shown in Table 8.

Experiment Details. The verification is briefly described above. To make this paper more complete, we give a relatively detailed description of our experiments. For the efficiency of the search, we store the solutions for (X_9, X_{10}) in RAM. However, due to the memory limit of our PC (Linux system) or Linux server, we could only store 2^{28} solutions for (X_9, X_{10}) in a two-dimensional dynamic array in RAM for one program, thus resulting that the success probability of connection becomes 2^{-4}.

Therefore, for our DLSR framework, we count the total times T_1 to start from Step 2 (where we start choosing another random values for free message words) and the total times T_2 to start verifying the probabilistic part X_i $(i \geq 25)$ and Y_j $(j \geq 14)$ after the connection succeeds. It is found that $T_1/T_2 = 17$, which is consistent with the success probability of connection. Obviously, it is expected that the total number of attempts to find the collision is T_2 when all the 2^{32} solutions can be stored in RAM for one program.

To find the collision for 30 steps of RIPEMD-160 in this paper, $T_2 =$ `0x4c11e4a5` and $T_1/T_2 = 17$. To find the collision for 31 steps of RIPEMD-160 in this paper, $T_2 =$ `0xfa3bab4a47` and $T_1/T_2 = 17$.

Note that the estimated probability to find the collision for 30/31 steps of RIPEMD-160 is $2^{-35.9}$ and $2^{-41.5}$ when all the 2^{32} solutions can be stored in RAM. Therefore, according the value of T_2, we believe that the estimated probability is reasonable. Similar experiments have been conducted for the collision attack on 33 and 34 steps of RIPEMD-160. The source code can be found at https://github.com/Crypt-CNS/DLSR_Framework_RIPEMD160.

6 Conclusion

Inspired from the start-from-the-middle approach, we discovered two efficient collision frameworks for reduced RIPEMD-160 by making full use of the weakness of message expansion. With the DLSR framework, we achieved the first practical collision attack on 30 and 31 steps of RIPEMD-160. Benefiting from the partial calculation techniques, the random message word can be chosen in a clever way so as to ensure more uncontrolled bit conditions hold. In this way, with the newly discovered 33-step and 34-step differential characteristics, collision attack on 33 and 34 steps of RIPEMD-160 can be achieved with time complexity $2^{67.1}$ and $2^{74.3}$ respectively. When applying the SLDR framework to the differential characteristic at Asiacrypt 2017, the time complexity is significantly improved, though it still cannot compete with the result obtained from the DLSR framework.

Acknowledgements. We thank the anonymous reviewers of CRYPTO 2019 for their insightful comments and suggestions. Fukang Liu and Zhenfu Cao are supported by National Natural Science Foundation of China (Grant No. 61632012, 61672239). In addition, Fukang Liu is also supported by Invitation Programs for Foreigner-based Researchers of the National Institute of Information and Communications Technology (NICT). Takanori Isobe is supported by Grant-in-Aid for Scientific Research (B) (KAKENHI 19H02141) for Japan Society for the Promotion of Science. Gaoli Wang is supported by the National Natural Science Foundation of China (No. 61572125) and National Cryptography Development Fund (No. MMJJ20180201).

A Application of the SLDR Framework

A direct application of this framework to the 30-step differential characteristic in [16] will improve the collision attack by a factor of 2^{11}. The constraints on RQ_i and the starting point are displayed in Tables 9 and 10 respectively.

Observe that m_{14} is randomly chosen in the SLDR framework and used to update Y_{25}. When the starting point is extended to Y_{20}, $sum_0 = Y_{20}^{\lll 10} \boxplus K_1^r = \texttt{0xf45c8129}$ is constant. Let $F = IFZ(Y_{24}, Y_{23}, Y_{22}^{\lll 10}) = (Y_{24} \bigwedge Y_{22}^{\lll 10}) \oplus (Y_{23} \bigwedge \overline{Y_{22}^{\lll 10}})$. Adding six extra bit conditions on Y_{23} ($Y_{23}^{26\sim 24} = 000_2$) and Y_{22} ($Y_{22}^{16\sim 14} = 000_2$) will make $F^{26\sim 24} = 000_2$. Then, adding four bit conditions on m_{14} ($m_{14}^{26\sim 23} = 1000_2$) will make $RQ_{25}^{26\sim 25} = 00_2$ since $RQ_{25} = F \boxplus sum_0 \boxplus m_{14}$. In this way, the condition $Y_{25}^{1\sim 0} = 01_2$ can always hold. Since all the newly added conditions can be fully controlled under this framework, two more probabilistic bit conditions are controlled, thus improving the collision attack by a factor of 2^{13} in total. A solution for the dense right branch is as follows: $m_0 = \texttt{0x284ca581}$, $m_1 = \texttt{0x55fd6120}$, $m_2 = \texttt{0x694b052c}$, $m_3 = \texttt{0xd5f43d9f}$, $m_4 = \texttt{0xa064a7c8}$, $m_5 = \texttt{0xb9f7b3cd}$, $m_6 = \texttt{0x1221b7bb}$, $m_7 = \texttt{0x42156657}$, $m_8 = \texttt{0x121ecfee}$, $m_9 = \texttt{0xce7a7105}$, $m_{10} = \texttt{0xf2d47e6f}$, $m_{11} = \texttt{0xf567ac2e}$, $m_{12} = \texttt{0x20d0d1cb}$, $m_{13} = \texttt{0x9d928b7d}$, $m_{14} = \texttt{0x5c6ff19b}$, $m_{15} = \texttt{0xc306e50f}$.

Table 9. Starting point for the differential characteristic presented at Asiacrypt'17

Y_{10}	01110000001111110100000010001010	Y_{16}	1111n1uu000n1n110001n1111nuuuuuu
Y_{11}	101101110000110110010000000nuuuu	Y_{17}	1u10111un110111100u10unnn0nnn011
Y_{12}	nuuuuuuuuuuuuuuuu0n0n00100001100	Y_{18}	010010000n1011111n00001001000001
Y_{13}	0unn1uu0111110100nuunn11011011un	Y_{19}	1u000101100100100101001000011101
Y_{14}	010000111111111110nu101011nu1111	Y_{20}	0000000101100110000000nu110101100
Y_{15}	000010111100u1u11010000u11010101		

Table 10. Information of RQ_i

Equation: $(\mathrm{RQ_i} \boxplus \mathrm{in})^{\lll \mathrm{shift}} = \mathrm{RQ_i}^{\lll \mathrm{shift}} \boxplus \mathrm{out}$									
i	shift	in	out	Pr.	i	shift	in	out	Pr.
11	8	0x1000000	0x1	1[a]	26	7	0x1000800	0x80040000	$\approx 2^{-1}$
12	11	0x15	0xa800	0.999	27	12	0x7ffc0000	0xbffff800	$\approx 2^{-1.4}$
13	14	0x6ffba800	0xea001bff	$\approx 2^{-1}$	28	7	0x0	0x0	1
24	11	0xffffff00	0xfff80000	0.999	29	6	0xc0000000	0xfffffff0	$\approx 2^{-0.4}$
25	7	0x80000	0x4000000	$\approx 2^{-0.02}$	30	15	0x10	0x80000	0.999

[a] The condition $Y_7 = 0$ makes it hold with probability 1.

B Differential Characteristics

We present the differential characteristics used for collision attack in this section.

Table 11. 30-Step differential characteristic

	$\Delta m_{12} = -2^{15}$			
i	X	$\pi_1(i)$	Y	$\pi_2(i)$
1	--------------------------------	0	--------------------------------	5
2	--------------------------------	1	--------------------------------	14
3	--------------------------------	2	--------------------------------	7
4	--------------------------------	3	--------------------------------	0
5	--------------------------------	4	--------------------------------	9
6	--------------------------------	5	--------------------------------	2
7	--------------------------------	6	--------------------------------	11
8	--------------------------------	7	--------------------------------	4
9	--------------------------------	8	--------------------------------	13
10	--------------------------------	9	--------------------------------	6
11	---0----------------------------	10	--------------------------------	15
12	--1-----------------------------	11	--------------------------------	8
13	----------u--0------------------	12	--------------------------------	1
14	---n-1----------------------1---	13	--------------------0-----------	10
15	-n000----1-100001001-1--01u1----	14	--------------------1-----------	3
16	-010111-11-0---011n1----0u0n0110	15	----------u---------------------	12
17	n----1-010-u----1100010101011---	7	--------------------------------	6
18	0--00n-01100110000unnu--nun--101	4	----------1---------------------	11
19	-100001u00n-nuun0un0100-000--nun	13	----------1---------------------	3
20	-u-nn-u011u00110110110-01nu1100n	1	u-------------------------------	7
21	----11011-0----0---101-un100u0uu	10	--------------------------------	0
22	10--------uuu--n0--1u10-100--000	6	1------------------------------0	13
23	u-----001-110001--1000-1-----u--	15	1------------------------------1	5
24	011--1----1------110n0-n-----0--	3	---------------------un---------	10
25	-------------------00-00--------	12	--------------------------------	14
26	----------0--1------------------	0	-------------------0-01---------	15
27	--------------------------------	9	-------------------1-11---------	8
28	--------------------------------	5	---------u-un-------------------	12
29	--n----------------------------n	2	--------------------------------	4
30	--------------------------------	14	--------------------------------	9

Other uncontrolled bit conditions on the left branch	$X_{28}^{29} = X_{27}^{19}, X_{28}^{0} = X_{27}^{22}$.	
Other uncontrolled bit conditions on the right branch	$Y_{18}^{31} = Y_{17}^{31}, Y_{22}^{9} = Y_{21}^{9}, Y_{26}^{20} = Y_{25}^{20}, Y_{26}^{19} = Y_{25}^{19}$.	
Modular difference	$(RQ_{16} \boxplus \texttt{0xffff8000})^{\lll 6} = RQ_{16}^{\lll 6} \boxplus \texttt{0xffe00000}$.	Pr. : Negligible.
	$(RQ_{28} \boxplus \texttt{0xffff8000})^{\lll 7} = RQ_{28}^{\lll 7} \boxplus \texttt{0xffc00000}$.	Pr. : Negligible.
	$(LQ_{13} \boxplus \texttt{0xffff8000})^{\lll 6} = LQ_{13}^{\lll 6} \boxplus \texttt{0xffe00000}$.	Pr. : Negligible.
	$(LQ_{14} \boxplus \texttt{0x200000})^{\lll 7} = LQ_{14}^{\lll 7} \boxplus \texttt{0x10000000}$.	Pr. = $2^{-0.1}$.
	$(LQ_{25} \boxplus \texttt{0x458})^{\lll 7} = LQ_{25}^{\lll 7} \boxplus \texttt{0x22c00}$.	Pr. : Negligible.
	$(LQ_{26} \boxplus \texttt{0xfffdc200})^{\lll 12} = LQ_{26}^{\lll 12} \boxplus \texttt{0xdc200000}$.	Pr. = $2^{-0.3}$.
	$(LQ_{27} \boxplus \texttt{0x24000000})^{\lll 15} = LQ_{27}^{\lll 15} \boxplus \texttt{0x1200}$.	Pr. = $2^{-0.3}$.
	$(LQ_{28} \boxplus \texttt{0xffffee00})^{\lll 9} = LQ_{28}^{\lll 9} \boxplus \texttt{0xffdc0000}$.	Pr. : Negligible.
	$(LQ_{29} \boxplus \texttt{0x240000})^{\lll 11} = LQ_{29}^{\lll 11} \boxplus \texttt{0x20000001}$.	Pr. = $2^{-0.2}$.
Total uncontrolled probability	$2^{-(10+0.9+25)} = 2^{-35.9}$	

Table 12. 31-Step differential characteristic

$\Delta m_{12}=2^{15}$				
i	X	$\pi_1(i)$	Y	$\pi_2(i)$
1	--------------------------------	0	--------------------------------	5
2	--------------------------------	1	--------------------------------	14
3	--------------------------------	2	--------------------------------	7
4	--------------------------------	3	--------------------------------	0
5	--------------------------------	4	--------------------------------	9
6	--------------------------------	5	--------------------------------	2
7	--------------------------------	6	--------------------------------	11
8	--------------------------------	7	--------------------------------	4
9	--------------------------------	8	--------------------------------	13
10	--------------------------------	9	--------------------------------	6
11	--------------------------------	10	--------------------------------	15
12	---1----------0-----------------	11	--------------------------------	8
13	----nuuuuuu---------------------	12	--------------------------------	1
14	---u0---------------------1-0---	13	--------------------0-----------	10
15	-u------10---110010------11-0---	14	--------------------1-----------	3
16	-0-0-----------00-n-1----1nn--01	15	----------n---------------------	12
17	u---------01---nuuuu01-0-101--nu	7	--------------------------------	6
18	00-----11-u----01u001nun-n-1-001	4	----------1---------------------	11
19	-nu1100un10-1----0-110u0-0------	13	----------1---------------------	3
20	010-----1uu110-1nn-un-0--1n1-01-	1	n-------------------------------	7
21	01n--n---0011--010-11-010-0--00-	10	--------------------------------	0
22	0-0--110-111unnnnnnnnnn0-u-----0	6	1------------------------------0	13
23	-0-----n-1n001000111101-1101----	15	1------------------------------1	5
24	1u10001111111--0-1-01-010-----u1	3	---------------------nu---------	10
25	10---------------------------10-	12	--------------------------------	14
26	--------------------1--1--------	0	-------------------0-01---------	15
27	--------------------------------	9	-------------------1-11---------	8
28	--------------------------------	5	---------n-nu-------------------	12
29	---------u--u-------------------	2	--------------------------------	4
30	---------0--0-------------------	14	--------------------------------	9
31	--------------------------------	11	--------------------------------	1

Other uncontrolled bit conditions on the left branch	$X_{28}^{19}=X_{27}^{9}, X_{28}^{22}=X_{27}^{12}$.	
Other uncontrolled bit conditions on the right branch	$Y_{18}^{31}=Y_{17}^{31}, Y_{22}^{9}=Y_{21}^{9}, Y_{26}^{20}=Y_{25}^{20}, Y_{26}^{19}=Y_{25}^{19}, Y_{30}^{0}=Y_{29}^{0}, Y_{30}^{29}=Y_{29}^{29}, Y_{30}^{30}=Y_{29}^{30}$.	
Modular difference	$(RQ_{16} \boxplus \texttt{0x8000})^{\lll 6} = RQ_{16}^{\lll 6} \boxplus \texttt{0x200000}$.	Pr. : Negligible.
	$(RQ_{28} \boxplus \texttt{0x8000})^{\lll 7} = RQ_{28}^{\lll 7} \boxplus \texttt{0x400000}$.	Pr. : Negligible.
	$(LQ_{13} \boxplus \texttt{0x8000})^{\lll 6} = LQ_{13}^{\lll 6} \boxplus \texttt{0x200000}$.	Pr. : Negligible.
	$(LQ_{14} \boxplus \texttt{0xffe00000})^{\lll 7} = LQ_{14}^{\lll 7} \boxplus \texttt{0xf0000000}$.	Pr. $= 2^{-0.1}$.
	$(LQ_{25} \boxplus \texttt{0xdfffffff})^{\lll 7} = LQ_{25}^{\lll 7} \boxplus \texttt{0xffffff70}$.	Pr. $= 2^{-0.2}$.
	$(LQ_{26} \boxplus \texttt{0x90})^{\lll 12} = LQ_{26}^{\lll 12} \boxplus \texttt{0x90000}$.	Pr. : Negligible.
	$(LQ_{27} \boxplus \texttt{0xfff70000})^{\lll 15} = LQ_{27}^{\lll 15} \boxplus \texttt{0x7ffffffc}$.	Pr. $= 2^{-1.1}$.
	$(LQ_{28} \boxplus \texttt{0x80000004})^{\lll 9} = LQ_{28}^{\lll 9} \boxplus \texttt{0x900}$.	Pr. $= 2^{-1.1}$.
	$(LQ_{29} \boxplus \texttt{0xfffff700})^{\lll 11} = LQ_{29}^{\lll 11} \boxplus \texttt{0xffb80000}$.	Pr. : Negligible.
Total uncontrolled probability	$2^{-(12+2.5+28)} = 2^{-42.5}$	

Table 13. 33-Step differential characteristic

$\Delta m_{12} = 2^{15}$				
i	X	$\pi_1(i)$	Y	$\pi_2(i)$
1	--------------------------------	0	--------------------------------	5
2	--------------------------------	1	--------------------------------	14
3	--------------------------------	2	--------------------------------	7
4	--------------------------------	3	--------------------------------	0
5	--------------------------------	4	--------------------------------	9
6	--------------------------------	5	--------------------------------	2
7	--------------------------------	6	--------------------------------	11
8	--------------------------------	7	--------------------------------	4
9	--------------------------------	8	--------------------------------	13
10	--------------------------------	9	--------------------------------	6
11	--------------------------------	10	--------------------------------	15
12	---------------------------1----	11	--------------------------------	8
13	----------n--0--------11--0-0---	12	--------------------------------	1
14	---u-----------------------001--	13	--------------------0-----------	10
15	-u000------0------1--1---1u--1--	14	--------------------1-----------	3
16	-111-------0-0-100n10--1un1u0-01	15	----------n---------------------	12
17	n0-1-1-1111n11--0-0u01-110u0----	7	--------------------------------	6
18	000unn1unnn1--1101u1u1-unn00-1-1	4	----------1---------------------	11
19	u0nn10-111un1--u01011001000u0-01	13	----------1---------------------	3
20	0u101110u1110010nn0n-n0u-110u011	1	n-------------------------------	7
21	n1n-100-0uu1u--100n1-00n01000-un	10	--------------------------------	0
22	0-0-unn0101011----0----nunnnu0un	6	1------------------------------0	13
23	001-011-u11u10--1-0-010n00100-01	15	1------------------------------1	5
24	11u010--0--11u01000101-111100--u	3	---------------------nu---------	10
25	-11----------1--1---n--n------11	12	--------------------------------	14
26	---1---------u--u---10-01-------	0	-------------------0-01---------	15
27	n--n------1--0--0---------------	9	-------------------1-11---------	8
28	0--1--1-------------u--u--------	5	---------n-nu-------------------	12
29	--------------------0-10-1------	2	--------------------------------	4
30	----------1--1------------------	14	---------1--1-------------------	9
31	--------------------------------	11	--1----------------------------1	1
32	--------------------------------	8	--n----------------------------n	2
33	--u----------------------------u	3	--------------------------------	15

Other uncontrolled bit conditions on the left branch	$X_{26}^{31} = X_{25}^{21}, X_{27}^{11} = X_{26}^{1}, X_{27}^{8} = X_{26}^{30}.$	
Other uncontrolled bit conditions on the right branch	$Y_{18}^{31} = Y_{17}^{31}, Y_{22}^{9} = Y_{21}^{9}, Y_{26}^{20} = Y_{25}^{20}, Y_{26}^{19} = Y_{25}^{19}, Y_{30}^{0} = Y_{29}^{0}, Y_{30}^{29} = Y_{29}^{29}, Y_{30}^{30} = Y_{29}^{30}.$	
Modular difference	$(RQ_{16} \boxplus \texttt{0x8000})^{\lll 6} = RQ_{16}^{\lll 6} \boxplus \texttt{0x200000}.$	Pr. : Negligible.
	$(RQ_{28} \boxplus \texttt{0x8000})^{\lll 7} = RQ_{28}^{\lll 7} \boxplus \texttt{0x400000}.$	Pr. : Negligible.
	$(LQ_{13} \boxplus \texttt{0x8000})^{\lll 6} = LQ_{13}^{\lll 6} \boxplus \texttt{0x200000}.$	Pr. : Negligible.
	$(LQ_{14} \boxplus \texttt{0xffe00000})^{\lll 7} = LQ_{14}^{\lll 7} \boxplus \texttt{0xf0000000}.$	Pr. = $2^{-0.1}$.
	$(LQ_{25} \boxplus \texttt{0x33ef815})^{\lll 7} = LQ_{25}^{\lll 7} \boxplus \texttt{0x9f7c0a81}.$	Pr. = $2^{-1.5}$.
	$(LQ_{26} \boxplus \texttt{0x407fff7e})^{\lll 12} = LQ_{26}^{\lll 12} \boxplus \texttt{0xfff7e408}.$	Pr. = $2^{-0.5}$.
	$(LQ_{27} \boxplus \texttt{0x39ff8})^{\lll 15} = LQ_{27}^{\lll 15} \boxplus \texttt{0xcffc0002}.$	Pr. = $2^{-0.3}$.
	$(LQ_{28} \boxplus \texttt{0xc007fffe})^{\lll 9} = LQ_{28}^{\lll 9} \boxplus \texttt{0xffffb80}.$	Pr. = $2^{-0.6}$.
	$(LQ_{29} \boxplus \texttt{0xfffffb80})^{\lll 11} = LQ_{29}^{\lll 11} \boxplus \texttt{0xffdc0000}.$	Pr. : Negligible.
	$(LQ_{30} \boxplus \texttt{0x240000})^{\lll 7} = LQ_{30}^{\lll 7} \boxplus \texttt{0x12000000}.$	Pr. = $2^{-0.2}$.
	$(LQ_{31} \boxplus \texttt{0xee000000})^{\lll 13} = LQ_{31}^{\lll 13} \boxplus \texttt{0xffffdc0}.$	Pr. = $2^{-0.2}$.
	$(LQ_{32} \boxplus \texttt{0x240})^{\lll 12} = LQ_{32}^{\lll 12} \boxplus \texttt{0x240000}.$	Pr. : Negligible.
	$(LQ_{33} \boxplus \texttt{0xffdc0000})^{\lll 11} = LQ_{33}^{\lll 11} \boxplus \texttt{0xdfffffff}.$	Pr. = $2^{-0.2}$.
Total uncontrolled probability	$2^{-(31+3+3.6+34)} = 2^{-71.6}$	

Table 14. 34-Step differential characteristic

$\varDelta m_{12} = 2^{15}$				
i	X	$\pi_1(i)$	Y	$\pi_2(i)$
1	--------------------------------	0	--------------------------------	5
2	--------------------------------	1	--------------------------------	14
3	--------------------------------	2	--------------------------------	7
4	--------------------------------	3	--------------------------------	0
5	--------------------------------	4	--------------------------------	9
6	--------------------------------	5	--------------------------------	2
7	--------------------------------	6	--------------------------------	11
8	--------------------------------	7	--------------------------------	4
9	--------------------------------	8	--------------------------------	13
10	--------------------------------	9	--------------------------------	6
11	--------------------------------	10	--------------------------------	15
12	-------------------10------0----	11	--------------------------------	8
13	--------nuu1-0------------0-----	12	--------------------------------	1
14	-1-n00---00101101-0010----------	13	--------------------0-----------	10
15	-n0010--1100-----------0--u000n-	14	--------------------1-----------	3
16	-00-0----010010--1n0111u-0uu0110	15	----------n---------------------	12
17	u00100---110100n0uuu101n1-00101n	7	--------------------------------	6
18	0101101-00uu-01101110-10nuu1-11u	4	----------1---------------------	11
19	u1nuun01010n01-11nnuu0100010--1n	13	----------1---------------------	3
20	1001u00n0nu1u11u0001n11n100u000u	1	u-------------------------------	7
21	u1nu001-u1-00u001--0nu10u100nu-0	10	--------------------------------	0
22	-100-0--0-1-n0uuu0-uu0uu011nu101	6	1------------------------------0	13
23	-n-1------001u-01-1-11-101011-u1	15	1------------------------------1	5
24	100n011--111111--10--u00n10--111	3	---------------------un---------	10
25	---1------n--u0-----00-11-------	12	--------------------------------	14
26	---u--n---00-11----------1------	0	-------------------0-01---------	15
27	1-----0---------------n--u------	9	-------------------1-11---------	8
28	----------u--n--------0-----1---	5	---------n-un-------------------	12
29	----------0-10-1----------------	2	--------------------------------	4
30	1--1----------------------------	14	---------1--1-------------------	9
31	--------------------------------	11	--1----------------------------1	1
32	---------------------0--0-------	8	--u----------------------------n	2
33	---------------------u--n-------	3	--1----------------------------1	15
34	--------------------------------	10	--------------------------------	5

Other uncontrolled bit conditions on the left branch	$X_{25}^{25} = X_{24}^{15}, X_{26}^{9} = X_{25}^{31}, X_{27}^{18} = X_{26}^{8}, X_{27}^{21} = X_{26}^{11}$.	
Other uncontrolled bit conditions on the right branch	$Y_{18}^{31} = Y_{17}^{31}, Y_{22}^{9} = Y_{21}^{9}, Y_{26}^{20} = Y_{25}^{20}, Y_{26}^{19} = Y_{25}^{19}, Y_{30}^{0} = Y_{29}^{0}, Y_{30}^{29} = Y_{29}^{29}, Y_{30}^{30} = Y_{29}^{30}$.	
Modular difference	$(RQ_{16} \boxplus \mathtt{0x8000})^{\lll 6} = RQ_{16}^{\lll 6} \boxplus \mathtt{0x200000}$.	Pr. : Negligible.
	$(RQ_{28} \boxplus \mathtt{0x8000})^{\lll 7} = RQ_{28}^{\lll 7} \boxplus \mathtt{0x400000}$.	Pr. : Negligible.
	$(LQ_{13} \boxplus \mathtt{0x8000})^{\lll 6} = LQ_{13}^{\lll 6} \boxplus \mathtt{0x200000}$.	Pr. : Negligible.
	$(LQ_{14} \boxplus \mathtt{0x200000})^{\lll 7} = LQ_{14}^{\lll 7} \boxplus \mathtt{0x10000000}$.	Pr. = $2^{-0.1}$.
	$(LQ_{25} \boxplus \mathtt{0x84201be3})^{\lll 7} = LQ_{25}^{\lll 7} \boxplus \mathtt{0x100df1c2}$.	Pr. = $2^{-1.2}$.
	$(LQ_{26} \boxplus \mathtt{0xe06be})^{\lll 12} = LQ_{26}^{\lll 12} \boxplus \mathtt{0xe06be000}$.	Pr. = $2^{-3.1}$.
	$(LQ_{27} \boxplus \mathtt{0x11802000})^{\lll 15} = LQ_{27}^{\lll 15} \boxplus \mathtt{0x100008c0}$.	Pr. = $2^{-0.2}$.
	$(LQ_{28} \boxplus \mathtt{0xdffff900})^{\lll 9} = LQ_{28}^{\lll 9} \boxplus \mathtt{0xfff1ffc0}$.	Pr. = $2^{-0.2}$.
	$(LQ_{29} \boxplus \mathtt{0xfff20000})^{\lll 11} = LQ_{29}^{\lll 11} \boxplus \mathtt{0x90000000}$.	Pr. = $2^{-0.9}$.
	$(LQ_{30} \boxplus \mathtt{0x70000000})^{\lll 7} = LQ_{30}^{\lll 7} \boxplus \mathtt{0x38}$.	Pr. = $2^{-0.9}$.
	$(LQ_{31} \boxplus \mathtt{0xffffffc8})^{\lll 13} = LQ_{31}^{\lll 13} \boxplus \mathtt{0xfff90000}$.	Pr. : Negligible.
	$(LQ_{32} \boxplus \mathtt{0x70000})^{\lll 12} = LQ_{32}^{\lll 12} \boxplus \mathtt{0x70000000}$.	Pr. = $2^{-0.9}$.
	$(LQ_{33} \boxplus \mathtt{0x90000000})^{\lll 11} = LQ_{33}^{\lll 11} \boxplus \mathtt{0xffffff c80}$.	Pr. = $2^{-0.9}$.
Total uncontrolled probability	$2^{-8.4-37-36} = 2^{-81.4}$	

References

1. Biham, E., Chen, R.: Near-collisions of SHA-0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 290–305. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_18
2. Bosselaers, A., Preneel, B. (eds.): Integrity Primitives for Secure Information Systems. LNCS, vol. 1007. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60640-8
3. Damgård, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_39
4. Daum, M.: Cryptanalysis of Hash functions of the MD4-family. Ph.D. thesis, Ruhr University Bochum (2005)
5. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: general results and applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006). https://doi.org/10.1007/11935230_1
6. den Boer, B., Bosselaers, A.: Collisions for the compression function of MD5. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 293–304. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_26
7. Dobbertin, H.: Cryptanalysis of MD4. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 53–69. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60865-6_43
8. Dobbertin, H.: RIPEMD with two-round compress function is not collision-free. J. Cryptol. **10**(1), 51–70 (1997)
9. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: a strengthened version of RIPEMD. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 71–82. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-60865-6_44
10. Dobraunig, C., Eichlseder, M., Mendel, F.: Analysis of SHA-512/224 and SHA-512/256. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 612–630. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_25
11. 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. LNCS, vol. 8540, pp. 473–488. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46706-0_24
12. Joux, A., Peyrin, T.: Hash functions and the (amplified) boomerang attack. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 244–263. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_14
13. Karpman, P., Peyrin, T., Stevens, M.: Practical free-start collision attacks on 76-step `SHA-1`. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 623–642. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_30
14. Landelle, F., Peyrin, T.: Cryptanalysis of full RIPEMD-128. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 228–244. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_14

15. Leurent, G.: Message freedom in MD4 and MD5 collisions: application to APOP. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 309–328. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74619-5_20
16. 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. LNCS, vol. 10624, pp. 158–186. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_6
17. Mendel, F., Nad, T., Scherz, S., Schläffer, M.: Differential attacks on reduced RIPEMD-160. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 23–38. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33383-5_2
18. 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. LNCS, vol. 7073, pp. 288–307. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_16
19. Mendel, F., Nad, T., Schläffer, M.: Collision attacks on the reduced dual-stream hash function RIPEMD-128. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 226–243. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_14
20. Mendel, F., Nad, T., Schläffer, M.: Improving local collisions: new attacks on reduced SHA-256. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 262–278. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_16
21. Mendel, F., Peyrin, T., Schläffer, M., Wang, L., Wu, S.: Improved cryptanalysis of reduced RIPEMD-160. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 484–503. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_25
22. Merkle, R.C.: One way hash functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_40
23. Ohtahara, C., Sasaki, Y., Shimoyama, T.: Preimage attacks on the step-reduced RIPEMD-128 and RIPEMD-160. IEICE Trans. **95-A**(10), 1729–1739 (2012)
24. Stevens, M.: Fast collision attack on MD5. Cryptology ePrint Archive, Report 2006/104 (2006). https://eprint.iacr.org/2006/104
25. 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. LNCS, vol. 10401, pp. 570–596. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_19
26. Wang, G.: Practical collision attack on 40-Step RIPEMD-128. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 444–460. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04852-9_23
27. Wang, G., Shen, Y., Liu, F.: Cryptanalysis of 48-step RIPEMD-160. IACR Trans. Symmetric Cryptol. **2017**(2), 177–202 (2017)
28. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_1

29. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_2
30. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_2
31. Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 1–16. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_1