Improved Collision Attack on Reduced RIPEMD-160

Fukang Liu, Gaoli Wang, Zhenfu Cao*

Shanghai Key Laboratory of Trustworthy Computing, School of Computer Science and Software Engineering, East China Normal University, Shanghai, China
liufukangs@163.com, glwang@sei.ecnu.edu.cn,
zfcao@sei.ecnu.edu.cn

Abstract. In this paper, we propose a new cryptanalysis method to mount collision attack on RIPEMD-160. Firstly, we review two existent cryptanalysis methods to mount (semi-free-start) collision attack on MD-SHA hash family and briefly explain their advantages and disadvantages. To make the best use of the advantages of the two methods, we come up with a new model to mount collision attack. Applying the new technique, we improve the only existent collision attack on the first 30-step RIPEMD-160 presented at Asiacrypt 2017 by a factor of 2^{11}. Moreover, our new method is much simpler than that presented at Asiacrypt 2017 and there is no need to pre-determine many bit conditions for multi-step modification, thus leaving sufficient freedom degree of the message words. Besides, we further evaluate the pros and cons of the new method and describe how to carefully apply it in future research. We also implement this attack in C++ and can find the message words to ensure the dense right branch with time complexity 2^{30}.

Keywords: RIPEMD-160, collision, hash function.

1 Introduction

A cryptographic hash function is a function which takes arbitrary long messages as input and output a fixed-length hash value of size n bits. There are three basic requirements for a hash function, which are preimage resistance, second-preimage resistance and collision resistance. Most standardized hash functions are based on the Merkle-Damgård paradigm[Dam89,Mer89] and iterate a compression function H with fixed-size input to compress arbitrarily long messages. Therefore, the compression function itself should satisfy equivalent security requirements so that the hash function can inherit from it. There are two attack models on the compression function. One is called free-start collision attack, the other is semi-free-start collision attack. The free-start collision attack is to find two different pairs of message and chaining value (CV, M), (CV', M') which satisfy $\mathrm{H}(CV, M) = \mathrm{H}(CV', M')$. The semi-free-start collision attack works in the same way apart from an additional condition that $CV = CV'$.

The last decade has witnessed the fall of a series of hash functions such as MD4, MD5, SHA-0 and SHA-1 since many break-through results on hash functions cryptanalysis [SBK+17,WLF+05,WY05,WYY05b,WYY05a] were obtained. All of these hash

* Corresponding author.

functions belong to the MD-SHA family, whose design strategy is based on the utilization of additions, rotations, xor and boolean functions in an unbalanced Feistel network.

RIPEMD family can be considered as a subfamily of the MD-SHA-family. Dobbertin was the first one to doubt the security of RIPEMD-0 [Dob97] and the first practical collision attack on it was presented by Wang et al. [WLF+05]. In order to strengthen the security of RIPEMD-0, Dobbertin, Bosselaers and Preneel [DBP96] proposed two strengthened versions of RIPEMD-0 in 1996, which are RIPEMD-128 and RIPEMD-160 with 128/160 bits output and 64/80 steps, respectively. Different from the simple design strategy of RIPEMD-0, more complex design strategies are adopted for the two new hash functions. More specifically, different constants, rotation values, message insertion schedules and boolean functions are used for RIPEMD-128 and RIPEMD-160 in their both branches.

At Eurocrypt 2013, semi-free-start collision attack on full RIPEMD-128 was presented [LP13], thus threatening its security claim. Before it, there also exists some results on RIPEMD-128 [MNS12,Wan14,WY15]. As for RIPEMD-160, the first security analysis for collision resistance of the reduced version was presented in [MNSS12]. In that work, Mendel et al. implemented a tool and used it to find a differential path. Then in the next year after Landelle and Peyrin presented a new method to mount the semi-free-start collision attack on full RIPEMD-128 [LP13], Mendel et al. [MPS+13] improved the tool in [MNSS12] and it was utilized to find the differential path of RIPEMD-160 at Asiacrypt 2013. With their new tool, they found a 48-step differential path and a 36-step differential path. Based on the two differential paths, the semi-free-start collision attack on 42-step RIPEMD-160 and the first 36-step RIPEMD-160 was mounted. In addition, they also proposed an open problem to theoretically calculate the step differential probability. Four years later, Liu et al. solved this problem by modeling the propagation of the modular difference using an equation at first and then calculating the probability of the bit conditions [LMW17]. The semi-free-start collision attack on the first 36-step RIPEMD-160 was improved as well in [LMW17] by choosing different free message words for merge. What's more, the first collision attack on step-reduced RIPEMD-160 was presented [LMW17]. However, the authors neglect three bit conditions and therefore the probability to find a collision becomes 2^{-70} (They have corrected this mistake). The strategy to mount the collision attack is to apply the message modification techniques on the dense right branch while keeping the sparse left branch probabilistic. For the semi-free-start collision attack on 42-step RIPEMD-160 from the middle, it was improved by a factor of 2^{10} and therefore can be extended to 48-step RIPEMD-160 [WSL17]. Besides, there are also some other analytical results on RIPEMD-160, such as a preimage attack [OSS12] on 31-step RIPEMD-160, a distinguisher on up to 51 steps of the compression function [SW12]. We summarize existent results in Table 1. Although there are several results on RIPEMD-160, it is yet unbroken and is widely used in the implementations of security protocols as a ISO/IEC standard.

This paper is organized as follows. The algorithm of RIPEMD-160 is briefly described in Section 2. Then, we review the work to model the propagation of modular difference using an equation in Section 3 for a better understanding of this paper. In Section 4, we propose a new cryptanalysis method to mount collision attack on the first

30-step RIPEMD-160. Then, a further discussion of our new method is presented in Section 5. Finally, we conclude the paper in Section 6.

Our Contributions. In this paper, we review two existent cryptanalysis methods to mount (semi-free-start) collision attack on MD-SHA hash family [LP13,WLF+05,WY05]. And then, we point out the advantages and disadvantages of the two methods. To leverage the advantages of the two methods, we propose a new cryptanalysis method to mount collision attack. Applying this new technique, we improve the only existent collision attack on the first 30-step RIPEMD-160 [LMW17] by a factor of 2^{11}. Moreover, our new method is much simpler than that presented in [LMW17] and there is no need to pre-determine many bit conditions for multi-step modification, thus leaving large freedom degree of the message words. The attack is implemented in C++ and can find the message words to ensure the dense right branch with time complexity 2^{30}.

At last, we give a further evaluation of our new method and consider the ideal application that maybe exist in the future. It reveals some lights on how to choose secure message insertion schedule for dual-stream hash functions like RIPEMD-128 and RIPEMD-160 to a certain degree. What's more, we also illustrate the importance to make a trade-off to obtain the optimal attack under our attack model.

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

Target	Attack Type	Steps	Complexity	Ref.
comp. function	preimage	31	2^{148}	[OSS12]
hash function	preimage	31	2^{155}	[OSS12]
comp. function	semi-free-start collision	36[a]	low	[MNSS12]
comp. function	semi-free-start collision	36	$2^{70.4}$	[MPS+13]
comp. function	semi-free-start collision	36	$2^{55.1}$	[LMW17]
comp. function	semi-free-start collision	42[a]	$2^{75.5}$	[MPS+13]
comp. function	semi-free-start collision	48[a]	$2^{76.4}$	[WSL17]
hash function	collision	30	2^{70}	[LMW17]
hash function	collision	30	2^{59}	new

[a] An attack starts at an intermediate step.

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 variables CV_i :

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

where a message M 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 [DBP96] for a detailed description of RIPEMD-160.

2.1 Notations

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

1. $\lll$, $\ggg$, $\oplus$, $\vee$, $\wedge$ and $\neg$ represent respectively the logic operation: *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. $\Delta m_i = m'_i - m_i$ represents the modular difference between two message words m_i and m'_i.
5. K^l_j and K^r_j represent the constant used at the left and right branch for round j.
6. Φ^l_j and Φ^r_j represent respectively the 32-bit boolean function at the left and right branch for round j.
7. X_i, Y_i represent respectively the 32-bit internal state of the left and right branch updated during step i for compressing M.
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. $X_{i,j}$, $Y_{i,j}$ represent respectively the j-th bit of X_i and Y_i, where the least significant bit is the 0th bit and the most significant bit is the 31st bit.
10. Q_i represents the 32-bit temporary state of the right branch updated during step i for compressing M.
11. s^l_i and s^r_i represent respectively the rotation constant used at the left and right branch during step i.
12. $\pi_1(i)$ and $\pi_2(i)$ represent the index of the message word used at the left and right branch during step i.
13. $[Z]_i$ represents the i-th bit of the 32-bit Z.
14. $[Z]_{j\sim i}$ $(0 \leq i < j \leq 31)$ represents the i-th bit to the j-th bit of the 32-bit word Z.
15. P(A) is the probability of the event A.

2.2 RIPEMD-160 Compression Function

The RIPEMD-160 compression function is a wider version of RIPEMD-128, which is based on MD4, 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`.

The 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.

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

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

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 2. As for other parameters, you can refer to [DBP96].

Table 2: 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)	$(x \vee \neg y) \oplus z$

The 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.

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

3 Propagation of the Modular Difference

Mendel et al. point out that it is not as easy to calculate the differential probability for each step of a given differential path of RIPEMD-160 as that of RIPEMD-128 [MPS+13]. The main reason is that the step function in RIPEMD-160 is no longer a T-function. Therefore, the accurate calculation of the differential probability becomes very hard. Then, Liu et al. solve this problem by modeling the propagation of the modular difference using an equation at first and then calculate the probability of the bit conditions [LMW17]. For a better understanding of this paper, we review how the modular difference of the internal states propagates and how to model the propagation by an equation as presented in [LMW17]. We use the step function of the right branch for explanation.

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

Let

$$
\begin{aligned}
\varDelta(Y_i) &= Y_i' \boxminus Y_i,\\
\varDelta(Y_{i-5}^{\lll 10}) &= Y_{i-5}'^{\lll 10} \boxminus Y_{i-5}^{\lll 10},\\
\varDelta(Y_{i-4}^{\lll 10}) &= Y_{i-4}'^{\lll 10} \boxminus Y_{i-4}^{\lll 10},\\
\varDelta F &= \Phi_j^l(Y_{i-1}', Y_{i-2}', Y_{i-3}'^{\lll 10}) \boxminus \Phi_j^l(Y_{i-1}, Y_{i-2}, Y_{i-3}^{\lll 10}),\\
\varDelta m &= m_{\pi_2(i)}' - m_{\pi_2(i)}.
\end{aligned}
$$

Given the differential path and the bit conditions to control the differential propagation, $\varDelta(Y_i)$, $\varDelta(Y_{i-5}^{\lll 10})$, $\varDelta(Y_{i-4}^{\lll 10})$, $\varDelta F$ and $\varDelta m$ are all fixed. Let

$$
\begin{aligned}
\texttt{in} &= \varDelta(Y_{i-5}^{\lll 10}) \boxplus \varDelta F \boxplus \varDelta m,\\
\texttt{out} &= \varDelta(Y_i) \boxminus \varDelta(Y_{i-4}^{\lll 10}).
\end{aligned}
$$

Hence, `in` and `out` are constants. To ensure $\varDelta(Y_i)$ can be the correct value, an equation is constructed as follows.

$$
(Q_i \boxplus \texttt{in})^{\lll s_i^r} = Q_i^{\lll s_i^r} \boxplus \texttt{out}.
$$

Since Q_i is a variable, the probability that $\varDelta(Y_i)$ is correct is equal to the probability that Q_i satisfies the equation $(Q_i \boxplus \texttt{in})^{\lll s_i^r} = Q_i^{\lll s_i^r} \boxplus \texttt{out}$. Calculating this probability of such an equation has been solved by Daum [Dau05]. Then, Liu et al. [LMW17] consider this problem from a different perspective and can obtain some useful information of Q_i. We can refer to [LMW17] for more details. Next, we present the example in [LMW17] so as to introduce the concept of possible and impossible characteristics of Q_i, which is vital to message modification.

Example. Let `in` = 0x80bfd9ff, `out` = 0x0xfd9ff80c and $s_i^r = 12$. Then, the equation becomes

$$(Q_i \boxplus \texttt{0x80bfd9ff})^{\lll 12} = Q_i^{\lll 12} \boxplus \texttt{0x0xfd9ff80c}.$$

To have a better understanding of the method to calculate the probability, we explain it by Table 3.

Table 3: Calculation of the Probability

Q_i	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q_i																																
`in`	1	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	1	1	1	1	1	1
	R_0												R_1																			
$Q_i^{\lll 12}$	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20
$Q_i^{\lll 12}$																																
`out`	1	1	1	1	1	1	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0
	R'_1																				R'_0											

First of all, we give a description of the notations in Table 3. To illustrate it more clearly, R_0, R_1, R'_0 and R'_1 are introduced. Let

$$\begin{aligned} R_0 \| R_1 &= Q_i \boxplus \texttt{0x80bfd9ff}, \\ R'_1 \| R'_0 &= Q_i^{\lll 12} \boxplus \texttt{0x0xfd9ff80c}. \end{aligned}$$

where R_0, R'_0 are 12-bit variables, and R_1, R'_1 are 20-bit variables. Then, the goal is to calculate the probability $\mathrm{P}(R_0 = R'_0 \texttt{ and } R_1 = R'_1)$. Observing the values of `in` and `out`, it is easy to find the following relationship:

$$[\texttt{in}]_{19\sim 0} = [\texttt{out}]_{31\sim 12},\ [\texttt{in}]_{31\sim 20} + 1 \equiv [\texttt{out}]_{11\sim 0} \ mod\ (2^{12}).$$

Therefore, to ensure $R_0 = R'_0$ and $R_1 = R'_1$, there must be carry from the 19-th bit to the 20-th bit when calculating $Q_i \boxplus \texttt{0x80bfd9ff}$, while there must be no carry from the 11-th bit to the 12-th bit when calculating $Q_i^{\lll 12} \boxplus \texttt{0x0xfd9ff80c}$. For example, $[Q_i]_{19} = 1$ can ensure $R_0 = R'_0$, and we call $[Q_i]_{19} = 1$ one possible characteristic of Q_i. However, $[Q_i]_{31} = 1$ will cause $R_1 \neq R'_1$ and we call $[Q_i]_{31} = 1$ one impossible characteristic of Q_i. By considering all cases, we can obtain the characteristics of Q_i as listed in Table 4. Then,

$$\begin{aligned} P(R_1 = R'_1) &= 1 - (2^{-1} + 2^{-9} + 2^{-10}). \\ P(R_0 = R'_0) &= \Sigma_{i=1}^{6} 2^{-i} + 2^{-8} + 2^{-9} + \Sigma_{i=12}^{20} 2^{-i}. \end{aligned}$$

Therefore, $\mathrm{P}(R_0 = R'_0 \texttt{ and } R_1 = R'_1) = P(R_1 = R'_1) \times P(R_0 = R'_0) \approx 2^{-1}$.

Table 4: The Characteristics of Q_i

Num	Characteristic	Type	Num	Characteristic	Type
1	$[Q_i]_{31} = 1$	Impossible	11	$[Q_i]_{19\sim11} = 000000101$	Possible
2	$[Q_i]_{31\sim23} = 011111111$	Impossible	12	$[Q_i]_{19\sim8} = 000000100111$	Possible
3	$[Q_i]_{31\sim22} = 0111111101$	Impossible	13	$[Q_i]_{19\sim7} = 0000001001101$	Possible
4	$[Q_i]_{19} = 1$	Possible	14	$[Q_i]_{19\sim6} = 00000010011001$	Possible
5	$[Q_i]_{19\sim18} = 01$	Possible	15	$[Q_i]_{19\sim5} = 000000100110001$	Possible
6	$[Q_i]_{19\sim17} = 001$	Possible	16	$[Q_i]_{19\sim4} = 0000001001100001$	Possible
7	$[Q_i]_{19\sim16} = 0001$	Possible	17	$[Q_i]_{19\sim3} = 00000010011000001$	Possible
8	$[Q_i]_{19\sim15} = 00001$	Possible	18	$[Q_i]_{19\sim2} = 000000100110000001$	Possible
9	$[Q_i]_{19\sim14} = 000001$	Possible	19	$[Q_i]_{19\sim1} = 0000001001100000001$	Possible
10	$[Q_i]_{19\sim12} = 00000011$	Possible	20	$[Q_i]_{19\sim0} = 00000010011000000001$	Possible

4 Improved Collision Attack on the First 30-Step RIPEMD-160

At Asiacrypt 2017, Liu et al. proposed the first collision attack on the first 30-step RIPEMD-160 [LMW17]. The differential path used in [LMW17] for collision attack is shown in Table 5. The strategy to construct this differential path can be divided into two phases. At first, one bit difference of m_{15} is chosen and then the difference is propagated in the left branch as sparsely as possible. This phase can be done manually. At the second phase, the tool invented by Mendel et al. [LMW17] is utilized to find a compatible differential path for the right branch. When the differential path is constructed, the message modification techniques [WLF+05] is then applied on the right branch until Y_{23}. Due to the difficulty to correct the conditions on both branches, the differential path in the left branch remains probabilistic.

However, although they apply the message modification techniques on the right branch until Y_{23}, they can't ensure all the bit conditions on Y_i nor have all the equations in terms of Q_i hold for $1 \leq i \leq 23$. More specifically, 13 bit conditions on Y_{23} and 1 bit condition on Y_{19} can't be satisfied by message modification. In addition, Q_{20} satisfies its corresponding equation with probability 2^{-1}. It seems rather hard to correct the bit conditions on Y_{23} since there are many bit conditions on Y_i ($12 \leq i \leq 18$). Hence, overcoming this obstacle is quite important so as to improve the collision attack. Since the multi-step modification techniques have its limitation, a new method is essential to solve this problem.

4.1 Overview of Our Method to Find Collisions

Our new method is inspired by Landelle's and Peyrin's idea [LP13]. Therefore, first of all, we give a comparison between Wang's method, Landelle's and Peyrin's method and our new method to find collisions as illustrated in Figure 1.

For Wang's method [WY05,LMW17], the computation starts from the first step and then the message modification techniques are applied. For Landelle's and Peyrin's method [LP13], the dense nonlinear part is determined by simple message modification at first. Then, the computation have two directions. One is backward from the nonlinear

Table 5: 30-step Differential Path, where $m'_{15} = m_{15} \boxplus 2^{24}$, and $\Delta m_i = 0$ $(0 \leqslant i \leqslant 14)$. Note that the symbol n represents that a bit changes to 1 from 0, u represents that a bit changes to 0 from 1, and - represents that the bit value is free.

X_i		$\pi_1(i)$	Y_i		$\pi_2(i)$
-4	--------------------------------		-4	--------------------------------	
-3	--------------------------------		-3	--------------------------------	
-2	--------------------------------		-2	--------------------------------	
-1	--------------------------------		-1	--------------------------------	
00	--------------------------------	00	00	--------------------------------	05
01	--------------------------------	01	01	--------------------------------	14
02	--------------------------------	02	02	--------------------------------	07
03	--------------------------------	03	03	--------------------------------	00
04	--------------------------------	04	04	--------------------------------	09
05	--------------------------------	05	05	--------------------------------	02
06	--------------------------------	06	06	--------------------------------	11
07	--------------------------------	07	07	--------------------------------	04
08	--------------------------------	08	08	--------------------------------	13
09	--------------------------------	09	09	-----1-1-1----------------------	06
10	--------------------------------	10	10	----000000-1--1---0000--1-001010	15
11	--------------------------------	11	11	-0--0---00001101100100000000nuuuu	08
12	--------------------------------	12	12	nuuuuuuuuuuuuuuuu0n0n00----01100	01
13	--------------------------------	13	13	0unn1uu-111-1-1--nuunn11011011un	10
14	--------------------------------	14	14	-100001111----1-10nu10101-nu1-11	03
15	--------------------------------	15	15	00---01111-0u-u-101000-u----0-01	12
16	-------------------------------n	07	16	111-n1uu000n1n--0001n----nuuuuuu	06
17	-------------------------------0	04	17	1u1-1--un--0111-00u10unnn-nnn01-	11
18	---------------------1---------1	13	18	01------0n-011--1n0000----0-00-1	03
19	---------------------0----------	01	19	1u------1--100--010----------1-1	07
20	---------------------n----------	10	20	-0--------1-----------0nu11---11-0	00
21	---------------------0----------	06	21	-1-----1011-----11111-101-------	13
22	-----------1---------1----------	15	22	u-----001-u-----------1u------00	05
23	n----------0--------------------	03	23	1--------------0-----01-------n-	10
24	0----------n--------------------	12	24	1--------------1----0-1-------00	14
25	1----------0----------1---------	00	25	1----n-----0--------1---------01	15
26	-1---------1----------0---------	09	26	-----------0--------unn---------	08
27	-0--------------------n---------	05	27	-u------------------------------	12
28	-n--------------------0---------	02	28	--------------------------------	04
29	-0----------1-------------------	14	29	--------------------------------	09
30	--------------------------------	11	30	--------------------------------	01

Other Conditions
$Y_{11,31} \bigvee \neg Y_{10,21} = 1$, $Y_{11,29} \bigvee \neg Y_{10,19} = 1$, $Y_{11,28} \bigvee \neg Y_{10,18} = 1$, $Y_{11,26} \bigvee \neg Y_{10,16} = 1$, $Y_{11,25} \bigvee \neg Y_{10,15} = 1$, $Y_{11,24} \bigvee \neg Y_{10,14} = 1$.
$Y_{14,21} = 1$, $Y_{14,20} = 1$, $Y_{14,19} = 1$ (We use the three conditions); Or $Y_{15,21} = 1$, $Y_{14,21} = 0$, $Y_{14,20} = 0$, $Y_{14,19} = 0$.
$Y_{15,6} = 1$, $Y_{14,6} = 0$, $Y_{15,5} = 1$; Or $Y_{14,6} = 1$, $Y_{15,5} = 0$ (We use the two conditions).
$Y_{15,29} = 0$, $Y_{15,28} = 0$, $Y_{15,27} = 1$.
$Y_{18,28} = Y_{17,28}$, $Y_{18,21} = Y_{17,21}$, $Y_{18,16} = Y_{17,16}$.
$Y_{19,17} = Y_{18,17}$, $Y_{19,8} = Y_{18,8}$, $Y_{19,1} = Y_{18,1}$.
$Y_{20,24} = Y_{19,24}$.
$Y_{22,19} = Y_{21,19}$, $Y_{22,20} = Y_{21,20}$.
$Y_{24,18} = Y_{23,18}$.
$Y_{27,4} = Y_{26,4}$.
$Y_{28,19} = Y_{27,19}$, $Y_{28,20} = Y_{27,20}$, $Y_{28,21} = Y_{27,21}$.
$Y_{29,8} = Y_{28,8}$.
$X_{15,0} = X_{14,22}$.
$X_{22,31} = X_{21,21}$.

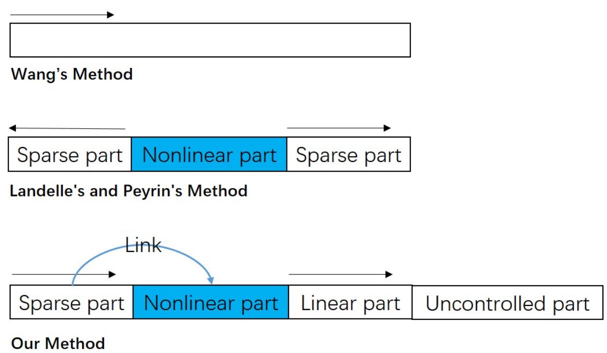

Fig. 1: Comparison between three methods

part and the other is forward from the nonlinear part. The advantage of Landelle's and Peyrin's method is obvious. That's, it can skip the the sophisticated multi-step modification and determining the dense nonlinear part is the pre-computation with a relatively low time complexity. Then at Asiacrypt 2013, such a method is applied by Mendel et al. to find semi-free-start collisions for RIPEMD-160 [MPS+13]. However, its disadvantage is also evident. It seems impossible to use such a method to find collisions since the computation doesn't start from the first step. Therefore, we propose a new model, which keeps the advantages of Landelle's and Peyrin's method and make the collision attack become possible.

Our new method can be devided into four steps.

Step 1: Preparation: Use the single-step modification to ensure the dense nonlinear part. We call the dense nonlinear part a starting point.

Step 2: Compute forward from the nonlinear part and use the message modification to ensure the corresponding conditions.

Step 3: Compute forward from the first step and use the message modification to ensure the corresponding conditions. Link the this part with the nonlinear part by leveraging remaining free message words. In other words, we leverage the remaining free message words to achieve the consistency between the nonlinear part and this part.

Step 4: The conditions located in the uncontrolled part are verified probabilistically.

4.2 Deducing Extra Bit Conditions to Control the Characteristics of Q_i

To mount the collision attack on the first 30-step RIPEMD-160, the first work is to identify the bit conditions. As observed in [LMW17], not only the bit conditions but also the modular difference of the internal states should be satisfied. However, message modification techniques are only useful to ensure the bit conditions. It is rather difficult to directly use this technique to ensure the modular difference of the internal states. Fortunately, Liu et al. note that by adding extra bit conditions on the internal states, the modular difference can be correctly propagated with probability 1 or close to

1 [LMW17]. The reason is that the newly-added bit conditions can be satisfied by message modification techniques. For completeness, we explain the two examples showed in [LMW17] once again.

Based on the 30-step differential path in Table 5, we can obtain that Q_{13} has to satisfy the equation $(Q_{13} \boxplus \texttt{0x6ffba800})^{\lll 14} = Q_{13}^{\lll 14} \boxplus \texttt{0xea001bff}$ so that the modular difference $\varDelta Y_{13}$ holds. As described previously, we can deduce the characteristics of Q_{13}. We only choose two possible characteristics of Q_{13}, which are $[Q_{13}]_{31} = 0$ and $[Q_{13}]_{17} = 1$. Considering the relationship between Y_{13} and Y_9 :

$$Q_{13}^{\lll 14} = Y_{13} \boxminus Y_9^{\lll 10},$$

our goal is to ensure the two bit conditions on Q_{13} are satisfied under the condition that some bits of Y_{13} and Y_9 are already fixed. We show the calculation of $Q_{13}^{\lll 14} = Y_{13} \boxminus Y_9^{\lll 10}$ in Table 6, which will help understand how to accurately deduce the extra bit conditions.

Table 6: The Calculation of $Q_{13}^{\lll 14} = Y_{13} \boxminus Y_9^{\lll 10}$

Y_{13}	0 1 0 0	1 u u -	1 1 1 -	1 - 1 -	- n u u	n n 1 1	0 1 1 0	1 1 u n
$Y_9^{\lll 10}$	1 0 - -	- - - -	- - - -	- - - -	- - 1 0	- - - -	- - - 1	- 1 - 1
$Q_{13}^{\lll 14}$	1 - - -	- - - -	- - - -	- - - -	- - 0 -	- - - -	- - - -	- - - -

If we impose four bit conditions on Y_9, which are $Y_{9,2} = 0$, $Y_{9,3} = 1$, $Y_{9,20} = 0$, $Y_{9,21} = 1$, the two bit conditions on Q_{13} will hold with probability 1.

In order to ensure that the modular difference $\varDelta Y_{23}$ holds, Q_{23} has to satisfy the equation $(Q_{23} \boxplus \texttt{0x81000001})^{\lll 9} = Q_{23}^{\lll 9} \boxplus \texttt{0x102}$, from which we can deduce the characteristics of Q_{23}. Then, we choose one possible characteristic, which is $[Q_{23}]_{31} = 1$. In this way, Q_{23} satisfies its corresponding equation with probability $1 - 2^{-23} \approx 1$. By considering the calculation of $Q_{23}^{\lll 9} = Y_{23} \boxminus Y_{19}^{\lll 10}$ as shown in Table 7, we describe how to dynamically determine the bit conditions on Y_{23}.

Table 7: The Calculation of $Q_{23}^{\lll 9} = Y_{23} \boxminus Y_{19}^{\lll 10}$

Y_{23}	1 - - -	- - - -	- - - -	- - - 0	- - - -	- 0 1 -	- - - -	- - n -
$Y_{19}^{\lll 10}$	1 u - -	- - - -	1 - - 1	0 0 - -	0 1 0 -	- - - -	- - - -	- 1 - 1
$Q_{23}^{\lll 9}$	- - - -	- - - -	- - - -	- - - -	- - - -	- - - 1	- - - -	- - - -

After $Y_{23,1}$ is corrected, compare $[Y_{23}]_{7\sim 0}$ with $[Y_{19}^{\lll 10}]_{7\sim 0}$. For different relationships between them, different bit conditions are used.

1. If $[Y_{23}]_{7\sim 0} \geq [Y_{19}^{\lll 10}]_{7\sim 0}$, we add a condition $Y_{23,8} \bigoplus Y_{19,30} = 1$.
2. If $[Y_{23}]_{7\sim 0} < [Y_{19}^{\lll 10}]_{7\sim 0}$, we add a condition $Y_{23,8} \bigoplus Y_{19,30} = 0$.

By determining the conditions on Y_{23} in this way dynamically, we can ensure Q_{23} satisfies its corresponding equation with probability close to 1 by applying the message modification to correct $Y_{23,8}$.

As described above, we can deduce many extra bit conditions on the internal states, which are displayed in Table 8. Different from [LMW17], we don't add extra bit conditions to control the characteristics of Q_i ($11 \le i \le 13$). The reason will be explained later. Then we can take these newly added bit conditions into consideration when applying the message modification techniques. In this way, both the bit conditions and the modular difference of the internal states can be satisfied at the same time.

Table 8: Equations of Q_i for the 30-Step Differential Path and Extra Conditions to Control the Equations

Equation: $(Q_i \boxplus in)^{\lll shift} = Q_i^{\lll shift} \boxplus out$				
i	shift	in	out	Extra conditions
11	8	`0x1000000`	`0x1`	
12	11	`0x15`	`0xa800`	
13	14	`0x6ffba800`	`0xea001bff`	
14	14	`0x40400001`	`0x1010`	$Y_{10,31} = 0$
15	12	`0xafffff5f`	`0xfff5fb00`	$Y_{15,9} = 0$, $Y_{11,31} = 1$
16	6	`0x9d020`	`0x2740800`	
17	9	`0x85f87f2`	`0xbf0fe410`	$Y_{13,20} = 1$, $Y_{13,18} = 0$, $Y_{17,28} = 0$, $Y_{17,26} = 1$, $Y_{13,16} = 0$.
18	7	`0x0`	`0x0`	
19	15	`0xffffd008`	`0xe8040000`	$Y_{15,21} = 0$
20	7	`0xd75fbffc`	`0xafdffdec`	
21	12	`0x10200813`	`0x812102`	$Y_{21,6} = 1$, $Y_{17,28} = 0$, $Y_{21,10} = Y_{17,0}$
22	8	`0xff7edffe`	`0x7edffeff`	$Y_{22,30} = 1$, $Y_{18,21} = 1$, $Y_{22,2} = Y_{18,24}$, $Y_{22,3} = Y_{18,25}$, $Y_{22,4} = Y_{18,26}$, $Y_{22,5} = Y_{18,27}$, $Y_{22,6} = Y_{18,28}$, $Y_{22,7} = Y_{18,29}$
23	9	`0x81000001`	`0x102`	If $[Y_{23}]_{7\sim0} \ge [Y_{19}^{\lll 10}]_{7\sim0}$, then $Y_{23,8} \bigoplus Y_{19,30} = 1$. If $[Y_{23}]_{7\sim0} < [Y_{19}^{\lll 10}]_{7\sim0}$, then $Y_{23,8} \bigoplus Y_{19,30} = 0$.
24	11	`0xffffff00`	`0xfff80000`	
25	7	`0x80000`	`0x4000000`	
26	7	`0x1000800`	`0x80040000`	
27	12	`0x7ffc0000`	`0xbffff800`	
28	7	`0x0`	`0x0`	
29	6	`0xc0000000`	`0xfffffff0`	
30	15	`0x10`	`0x80000`	

4.3 Finding Collisions for 30-step RIPEMD-160

Due to the difficulty to modify two branches simultaneously, we only consider the dense right branch and the sparse left branch is left probabilistic. Then, we specify our method to find collisions as illustrated in Figure 2. The procedure is divided into four steps.

Step 1. At this step, the goal is to generate a starting point. The technique used here is the single-step modification and the details are as follows.

- S1: Randomly choose Y_{10}, Y_{11}, Y_{12}, Y_{13} and Y_{14}. Then, it is very easy to correct Y_i ($10 \le i \le 14$) to ensure the bit conditions on them. For instance, suppose there are two bit conditions on Y_{10}, which are $Y_{10,2} = 0$, $Y_{10,5} = 1$. Then, we can correct Y_{10} in this way: $Y_{10} = Y_{10} \oplus (Y_{10,2} << 2) \oplus (\overline{Y_{10,5}} << 5)$.
- S2: Randomly choose Y_{15}, Y_{16}, Y_{17} and Y_{18}. In the same way as above, correct Y_i ($15 \le i \le 18$) to ensure the bit conditions on them. Then, compute m_3 by using Y_{10}, Y_{11},

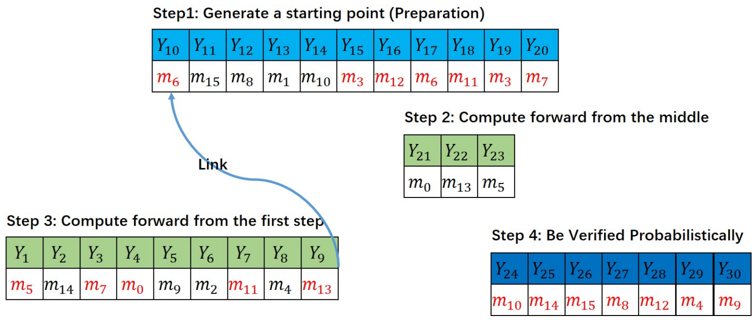

Fig. 2: Outline of our method

Y_{12}, Y_{13}, Y_{14} and Y_{15}.
$m_3 = (Y_{15} \boxminus Y_{11}^{\lll 10})^{\ggg 12} \boxminus (ONX(Y_{14}, Y_{13}, Y_{12}^{\lll 10}) \boxplus Y_{10}^{\lll 10} \boxplus K_0^r)$. Similarly, compute m_{12}, m_6, and m_{11}.

S3: Compute Y_{19} by using Y_{14}, Y_{15}, Y_{16}, Y_{17}, Y_{18} and m_3. If all the bit conditions on Y_{19} can hold, goto S4. Otherwise, goto S1.

S4: Randomly choose Y_{20} and correct Y_{20} to ensure the bit conditions on it. Compute m_7 and Q_{20} by using Y_{15}, Y_{16}, Y_{17}, Y_{18}, Y_{19} and Y_{20}.
Compute $Q_{20} = (Y_{20} \boxminus Y_{16}^{\lll 10})^{\ggg 7}$. If Q_{20} doesn't satisfy its corresponding equation, goto S1. Otherwise, a starting point is found.

Since there are 14 bit conditions on Y_{19} and the probability that Q_{20} satisfies its corresponding equation is about 2^{-1}, the time complexity of finding a starting point is 2^{15}.

Step 2. At this step, the goal is to ensure the bit conditions on Y_{21}, Y_{22} and Y_{23}. It is easy to achieve this goal since m_0, m_{13} and m_5 are free. More specifically, randomly choose Y_{21}, Y_{22} and Y_{23}, and then apply the single-step modification to correct Y_{21}, Y_{22} and Y_{23} by modifying m_0, m_{13} and m_5.

Step 3. At this step, there are two goals. One is to ensure the bit conditions on the internal states. The other is to link this part with the nonlinear part. Observe that, after fixing the value of Y_i $(1 \leq i \leq 9)$ by computing forward from the first step, we have to ensure that the determined Y_i $(10 \leq i \leq 14)$ have to be consistent with the values obtained by computing forward from the first step. In this way, the two parts can be linked. The ideal case is that m_6, m_{15}, m_8, m_1, m_{10} are all free. In this case, by modifying these five consecutive free message words, we can achieve the consistency in Y_i $(10 \leq i \leq 14)$.

However, m_6 is already fixed at **Step 1**. Therefore, the case is not ideal. Fortunately, we still can link the two parts by using the property of the boolean function in the first round on the right branch. Note that Y_{10} and m_6 is determined. Consider the calculation of Y_{10} when computing forward from the first step.

$$Y_{10} = Y_6^{\lll 10} \boxplus ((Y_9 \oplus (Y_8 \bigvee \overline{Y_7^{\lll 10}})) \boxplus Y_5^{\lll 10} \boxplus m_6 \boxplus K_0^r)^{\lll 7}.$$

The first idea to have this equation hold is to only change the value of Y_9 while keeping Y_i ($5 \le i \le 8$) the same. Then, compute a new m_{13} by using Y_i ($4 \le i \le 9$). However, m_{13} is already determined. Therefore, this idea doesn't work.

However, if we make $Y_7 = 0$, then the calculation of Y_{10} is changed as follows.

$$Y_{10} = Y_6^{\lll 10} \boxplus ((Y_9 \oplus \mathtt{0xffffffff}) \boxplus Y_5^{\lll 10} \boxplus m_6 \boxplus K_0^r)^{\lll 7}.$$

Therefore, we can calculate a new Y_9 as follows to have the above equation hold.

$$Y_9 = ((Y_{10} \boxminus Y_6^{\lll 10})^{\ggg 7} \boxminus (Y_5^{\lll 10} \boxplus m_6 \boxplus K_0^r)) \oplus \mathtt{0xffffffff}.$$

Observe that m_{13} is not free but m_4 is free. Therefore, we can modify m_4 to achieve that Y_9 can be the value as computed in the above equation.

$$\begin{aligned} Y_8 &= ((Y_9 \boxminus Y_5^{\lll 10})^{\ggg 7} \boxminus (Y_4^{\lll 10} \boxplus m_{13} \boxplus K_0^r)) \oplus (Y_7 \bigvee Y_6^{\lll 10}), \\ m_4 &= (Y_8 \boxminus Y_4^{\lll 10})^{\ggg 5} \boxminus (ONX(Y_7, Y_6, Y_5^{\lll 10}) \boxplus Y_3^{\lll 10} \boxplus K_0^r). \end{aligned}$$

Based on our analysis, by making $Y_7 = 0$, we have a way to ensure Y_{10} is consistent with the value obtained by computing forward from the first step. In other words, Y_{10} can be linked even though m_6 is already fixed. For Y_i ($11 \le i \le 14$), they can be linked by using the free message words m_{15}, m_8, m_1, m_{10}.

Now, we give a complete description of how to compute the internal states at **Step 3**.

S1: Since m_5 is already fixed, we can compute Y_1 by using m_5 and Y_i ($-4 \le i \le 0$).

S2: Randomly choose m_{14} and compute Y_2.

S3: Since m_7 and m_0 are already fixed, we can compute Y_3 and Y_4.

S4: Randomly choose m_9 and compute Y_5.

S5: Let $Y_7 = 0$, and then Y_6 can be computed as follows.
$Y_6 = ((Y_7 \boxminus Y_3^{\lll 10})^{\ggg 15} \boxminus (m_{11} \boxplus K_0^r)) \oplus (Y_5 \bigvee \overline{Y_4^{\lll 10}})$.

S6: Compute m_2 based on the following equation.
$m_2 = (Y_6 \boxminus Y_2^{\lll 10})^{\ggg 15} \boxminus (ONX(Y_5, Y_4, Y_3^{\lll 10}) \boxplus Y_1^{\lll 10} \boxplus K_0^r)$.

S7: In order to ensure Y_{10} is consistent with the value obtained by computing forward from the first step, we firstly compute the value of Y_9 since the condition $Y_7 = 0$ makes Y_{10} won't be influenced by the Y_8.
$Y_9 = ((Y_{10} \boxminus Y_6^{\lll 10})^{\ggg 7} \boxminus (Y_5^{\lll 10} \boxplus m_6 \boxplus K_0^r)) \oplus \mathtt{0xffffffff}$.
Since there are three bit conditions on Y_9, we have to goto S2 if these three bit conditions don't hold.

S8: Compute Y_8 and the corresponding m_4 as follows.
$Y_8 = ((Y_9 \boxminus Y_5^{\lll 10})^{\ggg 7} \boxminus (Y_4^{\lll 10} \boxplus m_{13} \boxplus K_0^r)) \oplus (Y_7 \bigvee \overline{Y_6^{\lll 10}})$.
$m_4 = (Y_8 \boxminus Y_4^{\lll 10})^{\ggg 5} \boxminus (ONX(Y_7, Y_6, Y_5^{\lll 10}) \boxplus Y_3^{\lll 10} \boxplus K_0^r)$.

S9: Check whether Q_i ($11 \le i \le 13$) satisfy their corresponding equations since the characteristics of Q_i ($11 \le i \le 13$) are not controlled. If they don't hold, goto S2.

S10: Compute the free message words m_{15}, m_8, m_1 and m_{10} to ensure that Y_i ($11 \le i \le 14$) are consistent with the values obtained by computing forward from the first

step.
$m_{15} = (Y_{11} \boxminus Y_7{}^{\lll 10})^{\ggg 8} \boxminus (ONX(Y_{10}, Y_9, Y_8^{\lll 10}) \boxplus Y_6^{\lll 10} \boxplus K_0^r)$.
$m_8 = (Y_{12} \boxminus Y_8^{\lll 10})^{\ggg 11} \boxminus (ONX(Y_{11}, Y_{10}, Y_9{}^{\lll 10}) \boxplus Y_7{}^{\lll 10} \boxplus K_0^r)$.
$m_1 = (Y_{13} \boxminus Y_9{}^{\lll 10})^{\ggg 14} \boxminus (ONX(Y_{12}, Y_{11}, Y_{10}{}^{\lll 10}) \boxplus Y_8^{\lll 10} \boxplus K_0^r)$.
$m_{10} = (Y_{14} \boxminus Y_{10}{}^{\lll 10})^{\ggg 14} \boxminus (ONX(Y_{13}, Y_{12}, Y_{11}{}^{\lll 10}) \boxplus Y_9{}^{\lll 10} \boxplus K_0^r)$.

After presenting the procedure of computation at **Step 3**, we explain the reason why we don't control the characteristics of Q_i ($11 \leq i \leq 13$). If we add some extra bit conditions on Y_i ($7 \leq i \leq 9$) to control their characteristics as in [LMW17], since Y_7 is set to 0 and it is difficult to ensure the bit conditions on Y_8 and Y_9, it will lower the success probability at **Step 3**. On the other hand, Q_{11} and Q_{12} satisfy their corresponding equations with probability close to 1 and therefore can be neglected. For Q_{13}, it satisfies its corresponding equation with probability of about 2^{-1}. In addition, we can't ensure the three bit conditions on Y_9. Hence, the success probability at **Step 3** is 2^{-4}.

Attack Procedure. Now, we give the procedure to mount collision attack on the first 30-step RIPEMD-160.

S1: **Preparation.** Generate a starting point as described in **Step 1**.
S2: Compute as described in **Step 2**.
S3: Compute as described in **Step 3**.
S4: Check the conditions on Y_i ($24 \leq i \leq 30$). If they don't hold, goto S2.
S5: Check whether the left branch can hold. If not, goto S2.

Observing the attack procedure, it is easy to find that we don't need goto S1 if the uncontrolled part don't hold. This is important to improve the probability of the attack since the success probability at S1 is 2^{-15}. Why do we only need go back to S2? The reason is that the freedom degree of the message words (m_0, m_{13}, m_5, m_{14} and m_9) is sufficient to ensure that uncontrolled part can be satisfied.

Implementation. We implement the attack in C++ and obtain one instance on the right branch as showed in Table 9.

4.4 Complexity Evaluation

As described in [LMW17], the left branch holds with probability 2^{-29}. For the right branch, the time complexity to generate a starting point is 2^{15} and the success probability at **Step 3** is 2^{-4}. For Y_i ($24 \leq i \leq 30$), there are 23 bit conditions on them. In addition, Q_i ($24 \leq i \leq 30$) satisfy their corresponding equations with probability about 2^{-3}. Therefore, the right branch holds with probability about $2^{-23-3-4} = 2^{-30}$ after applying our method. Therefore, a collision can be found with probability $2^{-29-30} = 2^{-59}$. Moreover, it is easy to observe that one starting point is enough due to the large freedom degree of the message words. Hence, the time complexity to find a collision is $2^{15} + 2^{59} \approx 2^{59}$. We have to stress that the authors [LMW17] neglected three bit conditions (marked in red in Table 5). That's, the success probability of collision attack in [LMW17] should be 2^{-70}. Obviously, our new method performs better than [LMW17]. Moveover, our method is simpler compared with the sophisticated multi-step modification and there is no need to pre-determine many bit conditions for multi-step modification as in [LMW17].

Table 9: One Instance on the Right Branch, where $m'_{15} = m_{15} \boxplus 2^{14}$, and $\Delta m_i = 0$ ($i \neq 7, 0 \leqslant i \leqslant 15$). The starting point is marked in red.

Y_i		$\pi_2(i)$
−4	11000000 01011001 11010001 01001000	
−3	01111100 00110000 11110100 10111000	
−2	00011101 10000100 00001100 10010101	
−1	10011000 10111010 11011100 11111110	
00	11101111 11001101 10101011 10001001	05
1	11100101 10010011 00000110 10000000	14
2	00001111 10000111 10001110 10100011	7
3	01101000 10100010 00100111 01101100	0
4	00100100 11010111 00101001 10011000	9
5	01001011 10111011 01110110 01011011	2
6	11010011 10000011 11010101 00001101	11
7	00000000 00000000 00000000 00000000	4
8	00111011 10101011 00000111 01100010	13
9	11010111 11011011 11011010 10111100	6
10	01110000 00111111 01000000 10001010	15
11	10110111 00001101 10010000 000nuuuu	8
12	nuuuuuuu uuuuuuuu u0n0n001 00001100	1
13	0unn1uu0 11111010 0nuunn11 011011un	10
14	01000011 11111111 10nu1010 11nu1111	3
15	00001011 1100u1u1 101000u 11010101	12
16	1111n1uu 000n1n11 0001n111 1nuuuuuu	5
17	1u10111u n1101111 00u10unn n0nnn011	11
18	01001000 0n101111 1n000010 01000001	3
19	1u000101 10010010 01010010 00011101	7
20	00000001 01100110 00000nu1 10101100	0
21	11000101 01101000 11111110 11011111	13
22	u1010100 10u01001 0011001u 00100000	5
23	11010111 01101100 01000010 100001n1	10
24	11011001 10110101 00100110 10001000	14
25	10111n10 00100111 00101011 01101101	15
26	00100010 11001100 0001unn0 11010010	8
27	0u111000 00010001 01110000 11010001	12
28	00001001 10010011 10110010 10011010	4
29	01011111 01111011 00000100 00111010	9
30	01000100 11111111 00001100 10001010	1

Message Words	m_0	m_1	m_2	m_3
Value	0x27e16551	0x3b7d1236	0xb896c190	0xd5f43d9f

Message Words	m_4	m_5	m_6	m_7
Value	0xd2100fd1	0xc807bcc7	0x1221b7bb	0x42156657

Message Words	m_8	m_9	m_{10}	m_{11}
Value	0x6b8513b4	0x41aeeaf1	0x5369aa6f	0xf567ac2e

Message Words	m_{12}	m_{13}	m_{14}	m_{15}
Value	0x20d0d1cb	0xa58a488f	0xe82b93aa	0x7fb8b25

5 Further Discussion of Our Method

In this section, we evaluate the pros and cons of our new method and describe how it can be applied in future research. First of all, we explain the advantages of our new method, which is highly related to the application.

As the application on collision attack on 30-step RIPEMD-160 in Section 4 shows, our new method to find a collision can be divided into two phases. The first phase is to generate a starting point, which is used to ensure the dense nonlinear part. The second phase is to leverage the free message words to find a collision. Suppose the time complexity of the two phases are TC_0 and TC_1 respectively, the total time complexity becomes $TC_0 + TC_1$. In many cases, we can use $\max\{TC_0, TC_1\}$ to represent the total time complexity since it is very common that one of TC_0 and TC_1 is far greater than the other. Therefore, we can make a trade-off between the two phases. In some bad cases, the freedom degree at the second phase is not sufficient enough to generate a collision. Therefore, one starting point is not enough and we have to generate several starting points. In these cases, we have to carefully analyze the time complexity. This is much like the collision attack on SHA-3 by constructing a 3-round connector at Crypto 2017 [SLG17], which requires several times to construct this connector due to the size of the solution space.

The advantage of our method is that the dense nonlinear part can be efficiently satisfied and there is no need to add extra bit conditions on the first round so as to ensure the conditions on the second round. However, as we know, although the single-step modification is powerful, the multi-step modification requires a lot of sophisticated manual work and many other bit conditions must be pre-determined on the first round if the bit conditions are dense on the second round. In addition, it is quite difficult to ensure the bit conditions on both branches simultaneously if directly applying Wang's method [WLF+05] to find a collision by computing from the first step. However, our method may have the potential to overcome this obstacle. We present an ideal case in Figure 3 to explain it.

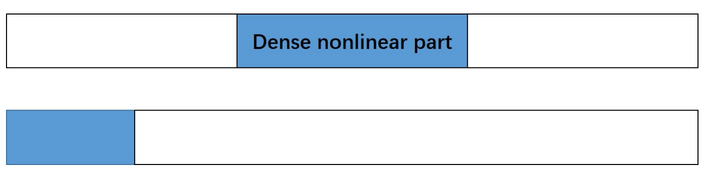

Fig. 3: Ideal case

To ensure the dense nonlinear part on one branch, some message words will be fixed. If some of these fixed message words are also used in the very first part on the other branch, then we can check whether these messages words can ensure the conditions on this part. Finally, our starting point can not only ensure the nonlinear dense part on one branch, but also ensure some parts on the other branch. Then, we leverage free message words to link the starting point with the previous part and ensure some other bit conditions. Finally, uncontrolled conditions will be verified probabilistically. Since the total time complexity is the sum of that at two phases, if the probability of finding a starting point is relatively high, our method will overcome the obstacle to a certain degree. Besides, this also reveals some principles on how to choose secure message insertion schedules for dual-stream hash functions like RIPEMD-128 and RIPEMD-160.

We have to stress that the similar cases actually exist in literatures [MPS+13,LMW17]. Specifically, the starting points in [MPS+13] and [LMW17] are illustrated in Figure 4 and Figure 5 respectively. Obviously, the shorter the starting point is, the more freedom degree of message words is, and the lower probability of the uncontrolled part. Therefore, it is a trade-off when choosing the position of the starting point. Based on the trade-off , Liu et al. improved the semi-free-start collision attack on 36-step RIPEMD-160 [MPS+13]. Although they are not the collision attacks, the some trade-off should be considered in our model as well.

On the other hand, the disadvantage of our model is also quite obvious. Linking the starting point in the middle with previous part decreases the freedom degree dramatically. Besides, the consecutive internal states to be linked should be carefully chosen in order that we can use free message words to ensure their consistency. Moreover, to reduce the times to generate a starting point, we have to make a trade-off so as to provide sufficient freedom degree of message words to generate a collision at the second phase.

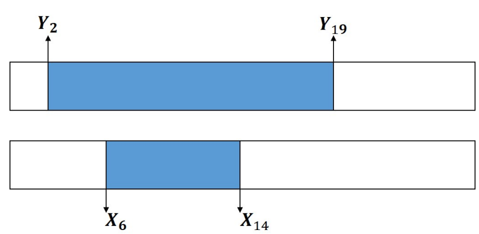

Fig. 4: The starting Point at Asiacrypt 2013

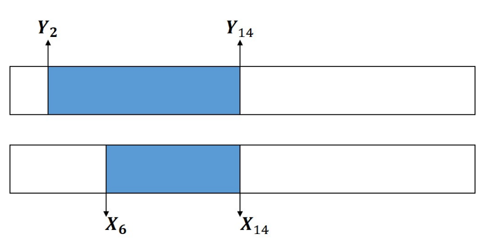

Fig. 5: The starting Point at Asiacrypt 2017

6 Conclusion

In this paper, we propose a new cryptanalysis method to find collision for reduced RIPEMD-160. Wang's method to find a collision is by computing forward from the first step, and the message modification techniques are applied on the first round and second round. Landelle's and Peyrin's method to find a semi-free-start collision is by computing in two directions after a starting point is fixed. However, for Wang's method, although the single-step modification is powerful to ensure all the bit conditions on the first round, the multi-step modification technique is sophisticated and requires a lot of manual work as well as many pre-determined bit conditions. For Landelle's and Peyrin's method, although there is no sophisticated multi-step modification, it seems hard to directly use it to find a collision since the computation doesn't start from the first step. To make the best of the advantages and bypass the disadvantages of the two methods, we propose a new model to find a collision. In our method, we firstly generate a starting point with relatively low time complexity. Then, we compute forward from the middle to ensure some latter parts. Next, we compute forward from the first step to ensure the previous part and use the free message words to link this part with the starting point so as to achieve the consistency. Applying our new technique, the only existent collision attack on the first 30-step RIPEMD-160 is improved by a factor of 2^{11}. The time complexity to mount the collision attack is improved from 2^{70} to 2^{59}. We hope that our new method can be used in future research.

References

Dam89. Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 416–427, 1989.

Dau05. Magnus Daum. *Cryptanalysis of Hash functions of the MD4-family*. PhD thesis, Ruhr University Bochum, 2005.

DBP96. Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, pages 71–82, 1996.

Dob97. Hans Dobbertin. RIPEMD with two-round compress function is not collision-free. *J. Cryptology*, 10(1):51–70, 1997.

LMW17. Fukang Liu, Florian Mendel, and Gaoli Wang. Collisions and semi-free-start collisions for round-reduced RIPEMD-160. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 158–186, 2017.

LP13. Franck Landelle and Thomas Peyrin. Cryptanalysis of full RIPEMD-128. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 228–244, 2013.

Mer89. Ralph C. Merkle. One way hash functions and DES. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 428–446, 1989.

MNS12. Florian Mendel, Tomislav Nad, and Martin Schläffer. Collision attacks on the reduced dual-stream hash function RIPEMD-128. In *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, pages 226–243, 2012.

MNSS12. Florian Mendel, Tomislav Nad, Stefan Scherz, and Martin Schläffer. Differential attacks on reduced RIPEMD-160. In *Information Security - 15th International Conference, ISC 2012, Passau, Germany, September 19-21, 2012. Proceedings*, pages 23–38, 2012.

MPS+13. Florian Mendel, Thomas Peyrin, Martin Schläffer, Lei Wang, and Shuang Wu. Improved cryptanalysis of reduced RIPEMD-160. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pages 484–503, 2013.

OSS12. Chiaki Ohtahara, Yu Sasaki, and Takeshi Shimoyama. Preimage attacks on the step-reduced RIPEMD-128 and RIPEMD-160. *IEICE Transactions*, 95-A(10):1729–1739, 2012.

SBK+17. Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 570–596, 2017.

SLG17. Ling Song, Guohong Liao, and Jian Guo. Non-full sbox linearization: Applications to collision attacks on round-reduced keccak. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 428–451, 2017.

SW12. Yu Sasaki and Lei Wang. Distinguishers beyond three rounds of the RIPEMD-128/-160 compression functions. In *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings*, pages 275–292, 2012.

Wan14. Gaoli Wang. Practical collision attack on 40-step RIPEMD-128. In *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, pages 444–460, 2014.

WLF+05. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 1–18, 2005.

WSL17. Gaoli Wang, Yanzhao Shen, and Fukang Liu. Cryptanalysis of 48-step RIPEMD-160. *IACR Trans. Symmetric Cryptol.*, 2017(2):177–202, 2017.

WY05. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 19–35, 2005.

WY15. Gaoli Wang and Hongbo Yu. Improved cryptanalysis on RIPEMD-128. *IET Information Security*, 9(6):354–364, 2015.

WYY05a. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 17–36, 2005.

WYY05b. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International*

Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings, pages 1–16, 2005.