Efficient Collision Attack Frameworks for RIPEMD-160

Fukang Liu

Shanghai Key Laboratory of Trustworthy Computing, School of Computer Science and Software Engineering, East China Normal University, Shanghai, China
liufukangs@163.com

Abstract. In this paper, we re-consider the connecting techniques to find collisions, which is achieved by connecting the middle part with the initial part. To obtain the best position of middle part, we propose two principles to deal with the case that is not ideal.
Then, we reviewed the searching strategy to find a differential path presented at Asiacrypt 2017, we observe some useful characteristics of the path which is not used in their work. To fully capture the characteristics of the differential path discovered by the searching strategy, we find an efficient attack framework under the guidance of the two principles, which in turn helps improve the searching strategy. Under our efficient attack framework, we easily improve the collision attack on 30-step RIPEMD-160 by a factor of 2^{13}. And we believe that the collision attack can be further improved under this efficient framework if the differential path is discovered by taking the new strategies into consideration.
For some interest, we also consider an opposite searching strategy and propose another efficient attack framework special for the differential path discovered by the new searching strategy. Under this new framework, we find we can control one more step than that special for the original searching strategy. Therefore, we expect that we can obtain better collision attack by adopting the new searching strategy and attack framework.
Moreover, combining with the searching tool, it is potential to give a tight upper bound of steps to mount collision attack on reduced RIPEMD-160 when adopting the two searching strategies.

Keywords: RIPEMD-160, collision, hash function, attack framework, searching strategy

1 Introduction

A cryptographic hash function is a function which takes arbitrary long messages as input and output a fixed-length hash value. Collision resistance and (second-) preimage resistance are three basic requirements for a secure hash function. For most standardized hash functions, they are based on the Merkle-Damgård paradigm[2,15] which iterates a compression function with fixed-size input to compress arbitrarily long messages.

The history of the great progress on MD-SHA hash family was impressive that Wang et al. published a series of results as well as some message modification techniques [20,22,23,21]. These results greatly threaten the security of design strategy for hash functions by utilization of additions, rotations, xor and boolean functions in an unbalanced Feistel network. On the other hand, it also provides a generic framework to evaluate the security of these hash functions, which is to find a good differential path as well as an efficient method to control the probability of the path. Therefore, there are also two directions for cryptanalysis of hash functions. One is to invent automatic tools for searching a good differential path. The other is to design strategies to make the discovered differential path hold with as a high probability as possible.

The searching tool for differential path progresses very well in recent years. At Asiacrypt 2011, Mendel et al. invented a searching tool to find good characteristics for SHA-2 [10] based on [1]. Since then, similar (improved) tools were utilized to find good differential characteristics for several hash functions and a series of results on RIPEMD-128, RIPEMD-160, SHA-2, SM3 were published [11,9,14,8,13,5,4,12]. However, all of them are not publicly available.

On the other hand, the strategies to make the discovered differential path hold with a high probability also progress well recently. In [9], Mendel et al. proposed a method to find semi-free-start collisions for reduced RIPEMD-160. More specifically, they invent a method by firstly fixing the dense part in the middle and then compute backward to achieve merging. Such a method was later used to mount full round semi-free-start collision attack on RIPEMD-128 at Eurocrypt 2013 [6]. Later, such a method was also applied to improve the semi-free-start collision attack on reduce RIPEMD-160 [14,8]. For SHA2, [4] also discovered a method to match IV and achieve the practical collision attack on 27-step SHA-512/224, SHA-512/256 and SHA-512/256. The technique to match IV is to adopt the idea by firstly fixing the heavy internal states in the middle of the first round and then connecting it with the initial part using free message words. In fact, similar method by connecting the initial part with the middle part has been used several years ago [7]. However, according to our understanding of the method presented in [4,7], we find they only consider the ideal case. To be more accurate, suppose the are t internal states $(S_0, S_1, ..., S_{t-1})$ to be connected and they are updated by message words $m_{k_0}, m_{k_1}, ..., m_{k_{t-1}}$, then $m_{k_0}, m_{k_1}, ..., m_{k_{t-1}}$ must be set free. In this way, it is quite straightforward to achieve connection. In this paper, we don't consider the ideal case since it can't help improve the efficiency to find collisions. Therefore, we have to determine the position of middle part carefully.

Since our paper focuses on the collision attack for RIPEMD-160, we also introduce some related results on RIPEMD-160. We have to stress that it is still meaningful to analyze the security of RIPEMD-160 since it is still an ISO/IEC standard. Moreover, since SHA-1 has been proven to be not secure [18,21] and SHA-3 doesn't provide the 160-bit digest, RIPEMD-160 may be used in the future to provide 160-bit digest. As far as we know, RIPEMD-160 and SHA-256 have been used to generate the bitcoin address.

Boomerang attack on RIPEMD-128/160 [17]. The framework of boomerang attack on RIPEMD-160/128 is illustrated in Fig. 1. The attacker tries to find out four pairs (IV, M), $(IV \boxplus \Delta, M \boxplus \Delta_0)$, $(IV \boxplus \nabla, M \boxplus \nabla_0)$ and $(IV \boxplus \nabla \boxplus \Delta, M \boxplus \nabla_0 \boxplus \Delta_0)$, supposing the compression function is denoted by H(IV, M), then a distinguishing property is obtained:

$$H(IV, M) \boxplus H(IV \boxplus \nabla \boxplus \Delta, M \boxplus \nabla_0 \boxplus \Delta_0) \boxminus$$
$$H(IV \boxplus \Delta, M \boxplus \Delta_0) \boxminus H(IV \boxplus \nabla, M \boxplus \nabla_0) = 0.$$

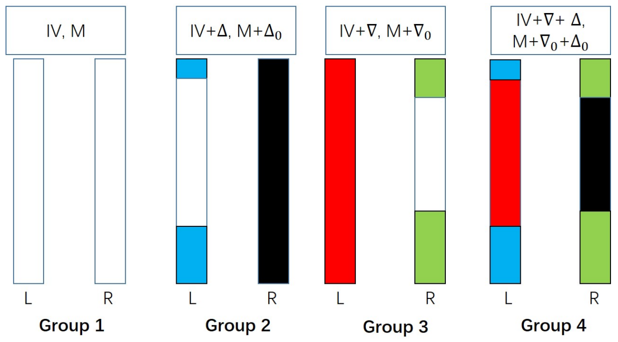


Fig. 1: Boomerang attack on RIPEMD-128/160

Semi-free-start collision attack on RIPEMD-160 [14,8]. The framework of semi-free-start collision attack on RIPEMD-160 is illustrated in Fig. 2. The attacker firstly fixes some heavy middle parts in both branches and then compute backward to merge both branches by leveraging the remaining free message words. At last, the uncontrolled part is verified probabilistically.

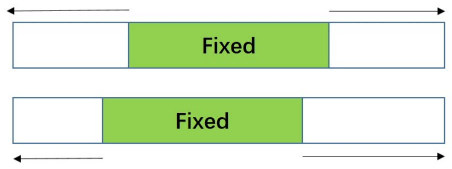


Fig. 2: Semi-free-start collision attack on RIPEMD-160

Collision attack on RIPEMD-160 [8]. The framework of collision attack on RIPEMD-160 is illustrated in Fig. 3. The attacker applies single-step modification and multi-step modification only on the dense branch in the first two rounds to make as many bit conditions as possible hold. Then, the conditions that can't be satisfied by message modification will hold probabilistically. The computation starts from the first step.

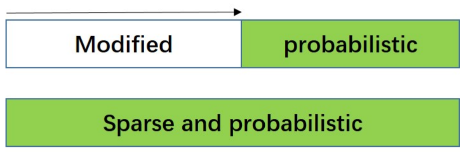


Fig. 3: Collision attack on RIPEMD-160

We recall the strategy presented at Asiacrypt 2017 to find differential path for RIPEMD-160 [8]. Since it is difficult to ensure the conditions in both branches, the authors let one branch remain fully probabilistic. Therefore, they choose the message word used to update the last internal state X_{16} in the first round to generate a difference. Then, they utilize the searching tool to find a good differential characteristic. Since only m_{15} is chosen to generate the difference, there won't be bit conditions on Y_i ($1 \leq i \leq 8$). In some cases, it is also possible there won't be conditions on Y_9. The reason is that to update Y_{12}, we have to control the difference generated by $Y_{11} \oplus (Y_{10} \bigvee \overline{Y_9^{\lll 10}})$. Since there is no difference in Y_{10} and Y_9, we control the bit i with difference in Y_{11} always flip by adding $Y_{10,i} = 1$. In this way, Y_9 can also be fully free. We will use such an observation in our attack framework.

1.1 Our Contributions

In this paper, we recall the strategy to find a differential path for RIPEMD-160 presented at Asiacrypt 2017 [8].

Firstly, we observe that the very initial part in the right branch is fully free if adopting such a searching strategy. Then, we try to capture such a characteristic of the differential path and find the most efficient way to make the most conditions hold. Then, the problem comes how to fully use such an observed characteristic. Inspired by the idea to mount collision attack through connecting the initial part with the middle part, we finally come up with an efficient attack framework for such a searching strategy. However, the previous constraint to choose the position of middle part to be connected is too strict and they only consider the ideal case [4,7], which is explained in previous part. According to our trial, we find it is impossible to reach the ideal case if we want to achieve the highest efficiency for the attack framework. Therefore, we propose two principles to guide us to choose the optimal position of middle part.

The two principles are quite straightforward. Since we don't consider the idea case, after we find a candidate of the position of middle part, we have to check whether we can achieve connection with a low cost. And this is the first principle. To make the uncontrolled probability hold with the highest probability, we have to ensure that after the middle part is fixed, there is an efficient method to make as many bit conditions as possible hold. And this is the second principle.

Following the two principles, we finally find an efficient attack framework special for the searching strategy to find differential path in [8]. Our attack framework can also provide some other useful searching strategies to find an optimal differential path utilizing the searching tool.

Then, we apply such an attack framework to the 30-step differential path in [8] and achieve a very simple and efficient way to find collisions. The time complexity is improved by a factor of 2^{13}. We list some results of RIPEMD-160 in Table 1.

For some interest, we also try to find whether there exists an efficient attack framework in the left branch if we have the right branch holding probabilistic. That's, we consider an opposite case to [8]. It is interesting that we find the new searching strategy [8] may be better than the original strategy.

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

Target	Attack Type	Steps	Complexity	Ref.
comp. function	preimage	31	2^{148}	[16]
hash function	preimage	31	2^{155}	[16]
comp. function	semi-free-start collision	36[a]	low	[9]
comp. function	semi-free-start collision	36	$2^{70.4}$	[14]
comp. function	semi-free-start collision	36	$2^{55.1}$	[8]
comp. function	semi-free-start collision	42[a]	$2^{75.5}$	[14]
comp. function	semi-free-start collision	48[a]	$2^{76.4}$	[19]
hash function	collision	30	2^{70}	[8]
hash function	collision	30	2^{57}	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 [3] for a detailed description of RIPEMD-160.

2.1 Notations

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

1. $\ll$, $\lll$, $\ggg$, $\oplus$, $\vee$, $\wedge$ and $\neg$ represent respectively the logic operation: $shift\ left$, $rotate\ left$, $rotate\ right$, $exclusive\ or$, 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. $X_{i,j}$, $Y_{i,j}$ 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.

9. $[Z]_i$ represents the i-th bit of the 32-bit Z.
10. $[Z]_{j\sim i}$ $(0 \leq i < j \leq 31)$ represents the i-th bit to the j-th bit of the 32-bit word Z (include bit i and j).

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 [3].

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 chain-

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⊕y⊕z
1	IFX	IFZ	0x5a827999	0x5c4dd124	IFX(x,y,z)	(x∧y)⊕(¬x∧z)
2	ONZ	ONZ	0x6ed9eba1	0x6d703ef3	IFZ(x,y,z)	(x∧z)⊕(y∧¬z)
3	IFZ	IFX	0x8f1bbcdc	0x7a6d76e9	ONX(x,y,z)	x⊕(y∨¬z)
4	ONX	XOR	0xa953fd4e	0x00000000	ONZ(x,y,z)	(x∨¬y)⊕ z

ing 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 Connecting Techniques

In this section, we give a brief description of the connecting techniques used to find collisions. For most hash functions in MD-SHA hash family, the internal states S_i is updated by a function f with a message word w_i and t consecutive internal states $S_{i-1}, ..., S_{i-t}$ as input. Besides, we can always compute w_i through another function g with $S_i, S_{i-1}, ..., S_{i-t}$ as input. Formally, we can express it as the following equation.

$$S_i = f(w_i, S_{i-1}, ..., S_{i-t}). \tag{1}$$

$$w_i = g(S_i, S_{i-1}, ..., S_{i-t}). \tag{2}$$

After Wang et al. presented some impressive attacks on MD4/MD5/SHA-0/SHA-1, the procedure to find collisions also developed. Specifically, Wang et al. start computation from the first step and then apply single-step and multi-step modification. Then, some cryptologists firstly fix some middle part and then connect it with the initial part as shown in [4,7]. Formally, their methods share some similarities. Suppose they choose $S_{i-1}, ..., S_{i-t}$ to be connected, then $w_{i-1}, ..., w_{i-t}$ are all set free. Since at the phase of connection, all the internal states and the middle part are known, it is trivial to calculate $w_{i-1}, ..., w_{i-t}$ and achieve connection.

However, what will happen if one or two of $w_{i-1}, ..., w_{i-t}$ are not free? The probability of successful connection is then dramatically decreased if without any strategy to solve it. Specifically, if one of $w_{i-1}, ..., w_{i-t}$ is fixed and the message word is an n-bit value, then the success probability of connection becomes 2^{-n}. This fact may prevent cryptologists from considering the case which is not ideal. However, we claim that this can be changed to one principle to finally determine the position of middle part.

Now, we give the first principle to guide us to choose the position of middle part.

Principle 1. When we consider a candidate of the position of middle part, we firstly consider whether it is efficient to achieve connection in the first t consecutive internal states located in the middle part when the case is not ideal. If not, we shorten the length of the middle part and repeat until we can find a solution.

Since the middle part will be fixed, some message words will be fixed as well. Suppose w_i is fixed in the middle part, and then w_i is also used to update the internal state S_t which is not in the middle part and there are some conditions on it. Then, it is hard to ensure these conditions. We have to stress that this may seem to be the case which can be solved by multi-step modification, it actually is hard to solve since w_i is already fixed and can't be changed. If we want to change it, we have to restart finding a solution for the middle part, which success with some probability. Therefore, we can hardly ensure these bit conditions. This will provide the second principle to guide us to determine the position of middle part.

Principle 2. When we consider a candidate of the position of middle part, we have to record the message words being fixed. Then, we observe the internal states not in the middle part and check whether these fixed message words will greatly decrease the probability. If so, we have to extend the middle part until the internal state which is also updated using the recorded message words.

We have to stress that the two principles should be considered simultaneously when considering a candidate of the position of middle part. Besides, observe that **Principle 1** is used to shorten the middle part and **Principle 2** is used to extend the middle part. The final determined position of middle part should be a tradeoff between two principles.

4 Optimizing the Attack Framework

In the above section, we propose two principles to find optimal position of middle part. In this section, combing the principles with the searching strategy proposed in [8], we present how to obtain an efficient attack framework and also provide some additional strategies to find an optimal differential path when using the searching tool.

4.1 Searching Strategy in Previous Research

At Asiacrypt 2017, [8] proposed a searching strategy to find a differential path for reduced RIPEMD-160. More specifically, they choose m_{15} as the message word to generate a difference. In this way, the first internal state with difference is X_{16} and Y_{11} in the left/right branch respectively. Due to the difficulty to modify both branches simultaneously, they only apply message modification on one branch. Therefore, the left branch is set very sparse and fully probabilistic. For the right branch, it is very dense and advanced message modification techniques are applied to ensure as many bit conditions as possible. However, the authors in [8] didn't fully capture the potentially useful characteristics of the differential path obtained by using such a searching strategy and directly applied a tradition attack framework (start modification from the first step) to find collisions.

Now, we give some important observations when adopting such a searching strategy.

Observation 1. There are not conditions on Y_i $(1 \leq i \leq 8)$.

Observation 2. The first internal state with difference in the right branch is Y_{11}. When considering the difference propagating to Y_{12}, we are actually considering the differential propagation of $Y_{11} \oplus (Y_{10} \bigvee \overline{Y_9^{\lll 10}})$ where only Y_{11} has difference. If we have all the bits $(p_i, p_{i+1}, ..., p_j)$ with difference in Y_{11} flipped by adding conditions $Y_{10,p_i} = 1$, $Y_{10,p_{i+1}} = 1$, ..., $Y_{10,p_j} = 1$ when searching the differential path, there won't be conditions on Y_9 either.

4.2 Determining the Position of Middle Part

To well illustrate the procedure of determining the position of middle part, we give partial information of the order of message words used in RIPEMD-160 in Table 3.

Table 3: Order of the Message Words in the First Two round

	i															
X_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
m_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X_i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
m_i	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
Y_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
m_i	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
Y_i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
m_i	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2

According to the searching strategy in [8], the left branch is set fully probabilistic and they only apply message modification techniques in the dense right branch. We also only consider how to ensure as many bit conditions as possible hold in the dense right branch. Based on our observations in above section, the first internal state with conditions is Y_{10} if we also use **Observation 2** to find a differential path. Therefore, we firstly choose Y_{10} as the starting position of middle part. In this case, the five consecutive internal states to be connected is Y_i $(10 \leq i \leq 14)$ and the corresponding message word are $m_6, m_{15}, m_8, m_1, m_{10}$.

After choosing the starting position, we choose a candidate for the ending position. We only consider the case that one or two of $m_6, m_{15}, m_8, m_1, m_{10}$ are fixed at the middle part since the success probability of connection will be dramatically decreased if more than three of $m_6, m_{15}, m_8, m_1, m_{10}$ are fixed. Therefore, we choose Y_{25} as a candidate.

Now, we explain the procedure to find an optimal middle position to achieve the most efficient attack framework.

Case 1: Choose Y_{25} as the ending position of middle part. Then, m_{10} will be fixed in the middle part. In this case, Y_9 is known according to m_{10} and Y_i $(10 \leq i \leq 14)$.

Thus, the starting position becomes Y_9 and the internal states to be connected become Y_i ($9 \leq i \leq 13$). Since two of the message words (m_6, m_{13}) used to update these 5 internal states are fixed, we have to consider whether there is an efficient way to achieve connection in such a bad case. It easy to achieve connection in Y_9 since m_4 is free. However, it will be difficult to achieve connection in Y_{10}. Therefore, the success probability of connection is 2^{-32}. Since the cost to connect is too high, we have to shorten the length of middle part based on **Principle 1**.

Case 2: Choosing Y_{24} as the ending position will be the same with Case 1 apart from m_{14} becomes free in the initial part, which can't be used to achieve an efficient connection. In this case, the success probability of connection is also 2^{-32}.

Case 3: Choose Y_{23} as the ending position of middle part. In this case, the starting position won't be changed since m_{10} won't be fixed in the middle part. Then, only one of the message words (m_6) used to update the 5 internal states to be connected is fixed. Next, we consider whether there exists an efficient method to achieve connection in Y_{10}. And we actually find an efficient method to make it. After choosing Y_{23} as the ending position, m_{14}, m_9, m_2, m_4 are free in the initial part, i.e., from Y_1 to Y_9. In other words, we can consider whether there is a method to achieve connection in Y_{10} by leveraging these free message words. Considering the calculation of Y_{10} as follows.

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

If we make $Y_7 = 0$, then the above equation becomes

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

Then, after we have computed until Y_7, we can efficiently find the solution of Y_9 to achieve connection in Y_{10} by using the freedom of m_4. The details are shown below:

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

Then, the problem becomes how to ensure the condition $Y_7 = 0$. Suppose we have computed until Y_5, and then this condition can be solved as follows by using the freedom of m_2.

$$\begin{aligned} Y_6 &= ((Y_7 \boxminus Y_3^{\lll 10})^{\ggg 15} \boxminus (m_{11} \boxplus K_0^r)) \oplus (Y_5 \bigvee \overline{Y_4^{\lll 10}}). \\ m_2 &= (Y_6 \boxminus Y_2^{\lll 10})^{\ggg 15} \boxminus (ONX(Y_5, Y_4, Y_3^{\lll 10}) \boxplus Y_1^{\lll 10} \boxplus K_0^r). \end{aligned}$$

In a word, by using the freedom of m_2 and m_4, we can achieve connection with probability 1. Therefore, we find a possible good position of middle part in this case. However, is it the optimal position? We have to further consider this question.

Observe that m_7, m_0, m_{13} and m_5 are only used once to update the internal states in the middle part. The conditions on Y_i $(20 \leq i \leq 23)$ can be easily satisfied by single-step modification. In other words, we don't necessarily fix their values in the middle part and this will provide more freedom to find collisions. Therefore, we consider Case 4.

Case 4: Choose Y_{19} as the ending position of middle part. After fixing the values in the middle part, we use single-step modification to ensure the conditions on Y_i $(20 \leq i \leq 23)$. Then we deal with the connection in the same way as Case 3.

A natural question is whether it is possible to further shorten the length of the middle part to provide more freedom. Then, we follow **Principle 2** to answer such an question.

If we further shorten the length of the middle part, then m_3 is fixed in the middle part while m_3 is also used to update Y_{19}, which is not included in the middle part. Perhaps, one may try to use idea like multi-step modification techniques to ensure the conditions on Y_{19} and this may be feasible. However, it differs for different discovered differential path and requires a lot of sophisticated manual work. Moreover, in some bad cases, the modification techniques can't ensure all the conditions on Y_{19}. Why not sacrifice the freedom of m_3 to achieve a simple attack framework? Therefore, we finally choose Y_i $(10 \leq i \leq 19)$ as the position of middle part.

4.3 Formalizing the Optimal Attack Framework

In this section, we formalize the optimal attack framework corresponding to the optimal position of middle part found based on the two principles, which fully uses the two observations of the searching strategy.

The attack framework is illustrated in Fig. 4. It contains four 4 steps.

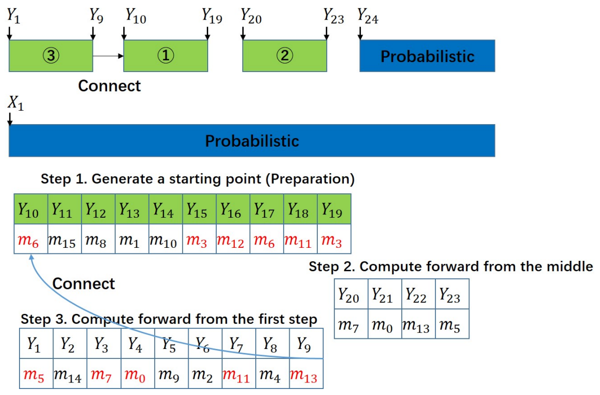

Y_1
Y_9
Y_{10}
Y_{19}
Y_{20}
Y_{23}
Y_{24}
③
①
②
Probabilistic
Connect
X_1
Probabilistic
Step 1. Generate a starting point (Preparation)
Y_{10} Y_{11} Y_{12} Y_{13} Y_{14} Y_{15} Y_{16} Y_{17} Y_{18} Y_{19}
m_6 m_{15} m_8 m_1 m_{10} m_3 m_{12} m_6 m_{11} m_3
Step 2. Compute forward from the middle
Y_{20} 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. 4: Attack Framework for RIPEMD-160

Step 1: Fix the internal states located in the middle part from Y_{10} to Y_{19}, which can be easily done using single-step modification since only m_3 is used twice to update the internal states.

Step 2: Apply single-step modification to ensure the conditions on Y_{20} to Y_{23} since their corresponding message words have not been fixed in the middle part.

Step 3: Randomly choose values for message words which have not been fixed and compute from the first step until Y_5. Then achieve connection in Y_{10} as follows:

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

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 don't hold, go to Step 2 until we find collisions. The freedom is provided by m_0, m_5, m_7, m_9, m_{13} and m_{14}.

Based on this attack framework, it is quiet simple and efficient to find collisions. The reason is Step 3 can succeed with a probability close to 1 if the differential path is discovered based on **Observation 2**. Besides, only single-step modification technique is enough to fix the middle part and only one solution for the middle part is enough if the degree of freedom is sufficient. For Step 2, based on [8], we can add additional bit conditions to ensure the modular difference and therefore we only need move forward by single-step modification technique. In other words, there is no need to do multi-step modification which requires a lot of sophisticated manual work for each discovered differential path. The only thing we need to concern about is whether the freedom of message words is sufficient to find collisions after the middle part is fixed.

4.4 Improving the Searching Strategies

Based on our efficient framework to mount collision attack on RIPEMD-160, we also add some constraints to find an optimal differential path in the searching phase.

Strategy 1. The number of conditions on the internal states Y_i $(i \geq 24)$ should be as small as possible.

Strategy 2. According to **Observation 2**, we should make all the bits with difference in Y_{11} flip. In this way, there will be no conditions on Y_9.

Combining the two strategies, it is potential to obtain a better collision attack and give a tight upper bound of steps to mount collision attack on reduced RIPEMD-160 when adopting such a strategy to find a differential path.

5 Application on 30-Step RIPEMD-160

We apply this efficient framework on the 30-step differential path found in [8] as shown in Table 4. Firstly, we find a solution for the middle part, which is marked in red in Table 6. To make some conditions on Y_{25} hold, we extend the middle part to Y_{20}, which only decreases the freedom of m_7. The details are as follows.

5.1 Slight Improvement to Make More Conditions Hold

Observe that m_{14} is fully free in the initial part and it is used to update Y_{25}.

$$\begin{aligned} Q_{25} &= IFZ(Y_{24}, Y_{23}, Y_{22}^{\lll 10}) \boxplus Y_{20}^{\lll 10} \boxplus m_{14} \boxplus K_1^r. \\ Y_{25} &= Y_{21}^{\lll 10} \boxplus Q_{25}^{\lll 7}. \end{aligned}$$

According to Table 4, we can find that there are two bit conditions on $Y_{25,0}$ and $Y_{25,1}$, which are $Y_{25,1} = 0$ and $Y_{25,0} = 1$. After the middle part is fixed, $Y_{20}^{\lll 10}$ is known. Besides, we also know the pattern of $Y_{21}^{\lll 10}$ as follows:

$$Y_{21}^{\lll 10} = \texttt{1-----11 111-101- -------1 -----101}.$$

Therefore, if $[Q_{25}^{\lll 7}]_0 = 0$ and $[Q_{25}^{\lll 7}]_1 = 0$ can hold, $Y_{25,1} = 0$ and $Y_{25,0} = 1$ will hold with probability 1. Then, our goal is to ensure $[Q_{25}^{\lll 7}]_0 = 0$ and $[Q_{25}^{\lll 7}]_1 = 0$. After fixing the middle part as shown in Table 6, $Y_{20}^{\lll 10}$ is known and we can compute that

$$\begin{aligned} \texttt{Temp} &= Y_{20}^{\lll 10} \boxplus K_1^r = \texttt{0xf45c8129}. \\ \texttt{0xf45c8129} &= \texttt{11110100 01011100 10000001 00101001}. \end{aligned}$$

Consider the calculation of IFZ(Y_{24}, Y_{23}, $Y_{22}^{\lll 10}$).

$$IFZ(Y_{24}, Y_{23}, Y_{22}^{\lll 10}) = (Y_{24} \bigwedge Y_{22}^{\lll 10}) \oplus (Y_{23} \bigwedge \overline{Y_{22}^{\lll 10}})$$

We write Y_{24}, Y_{23}, $Y_{22}^{\lll 10}$ in binary according to Table 4 as follows for a better understanding.

$$\begin{aligned} Y_{24} &= \texttt{1------- -------1 ----0-1- ------00}. \\ Y_{23} &= \texttt{1------- -------0 -----01- ------n-}. \\ Y_{22}^{\lll 10} &= \texttt{u------- ----1u-- ----00u- ----001-}. \end{aligned}$$

Add the following bit conditions on Y_{23} and Y_{22} marked in red.

$$\begin{aligned} Y_{24} &= \texttt{1------- -------1 ----0-1- ------00}. \\ Y_{23} &= \texttt{1----000 -------0 -----01- ------n-}. \\ Y_{22}^{\lll 10} &= \texttt{u----000 ----1u-- ----00u- ----001-}. \end{aligned}$$

Table 4: 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.

```
Xi                                       π1(i) Yi                                       π2(i)
-4 -------- -------- -------- --------         -4 -------- -------- -------- --------
-3 -------- -------- -------- --------         -3 -------- -------- -------- --------
-2 -------- -------- -------- --------         -2 -------- -------- -------- --------
-1 -------- -------- -------- --------         -1 -------- -------- -------- --------
00 -------- -------- -------- --------         00 -------- -------- -------- --------
01 -------- -------- -------- --------   00    01 -------- -------- -------- --------   05
02 -------- -------- -------- --------   01    02 -------- -------- -------- --------   14
03 -------- -------- -------- --------   02    03 -------- -------- -------- --------   07
04 -------- -------- -------- --------   03    04 -------- -------- -------- --------   00
05 -------- -------- -------- --------   04    05 -------- -------- -------- --------   09
06 -------- -------- -------- --------   05    06 -------- -------- -------- --------   02
07 -------- -------- -------- --------   06    07 -------- -------- -------- --------   11
08 -------- -------- -------- --------   07    08 -------- -------- -------- --------   04
09 -------- -------- -------- --------   08    09 -----1-1 -1------ -------- --------   13
10 -------- -------- -------- --------   09    10 ----0000 00-1--1- --0000-- 1-001010   06
11 -------- -------- -------- --------   10    11 -0--0--- 00001101 10010000 000nuuuu   15
12 -------- -------- -------- --------   11    12 nuuuuuuu uuuuuuuu u0n0n00- ---01100   08
13 -------- -------- -------- --------   12    13 0unn1uu- 111-1-1- -nuunn11 011011un   01
14 -------- -------- -------- --------   13    14 -1000011 11----1- 10nu1010 1-nu1-11   10
15 -------- -------- -------- --------   14    15 00---011 11-0u-u- 101000-u ----0-01   03
16 -------- -------- -------- -------n   15    16 111-n1uu 000n1n-- 0001n--- -nuuuuuu   12
17 -------- -------- -------- -------0   07    17 1u1-1--u n--0111- 00u10unn n-nnn01-   06
18 -------- -------- -----1-- -------1   04    18 01------ 0n-011-- 1n0000-- --0-00-1   11
19 -------- -------- -----0-- --------   13    19 1u------ 1--100-- 010----- -----1-1   03
20 -------- -------- -----n-- --------   01    20 -0------ --1----- ----0nu1 1---11-0   07
21 -------- -------- -----0-- --------   10    21 -1-----1 011----- 11111-10 1-------   00
22 -------- ---1---- -----1-- --------   06    22 u-----00 1-u----- ------1u ------00   13
23 n------- ---0---- -------- --------   15    23 1------- -------0 -----01- ------n-   05
24 0------- ---n---- -------- --------   03    24 1------- -------1 ----0-1- ------00   10
25 1------- ---0---- ------1- --------   12    25 1----n-- ---0---- ----1--- ------01   14
26 -1------ ---1---- ------0- --------   00    26 -------- ---0---- ----unn- --------   15
27 -0------ -------- ------n- --------   09    27 -u------ -------- -------- --------   08
28 -n------ -------- ------0- --------   05    28 -------- -------- -------- --------   12
29 -0------ ----1--- -------- --------   02    29 -------- -------- -------- --------   04
30 -------- -------- -------- --------   14    30 -------- -------- -------- --------   09
```

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).
$Y_{15,29} = 0$, $Y_{15,28} = 0$, $Y_{15,27} = 1$.
$Y_{18,28} = Y_{17,28}$, $Y_{18,21} = Y_{17,21}$, $Y_{18,16} = Y_{17,16}$.
$Y_{19,17} = Y_{18,17}$, $Y_{19,8} = Y_{18,8}$, $Y_{19,1} = Y_{18,1}$.
$Y_{20,24} = Y_{19,24}$.
$Y_{22,19} = Y_{21,19}$, $Y_{22,20} = Y_{21,20}$.
$Y_{24,18} = Y_{23,18}$.
$Y_{27,4} = Y_{26,4}$.
$Y_{28,19} = Y_{27,19}$, $Y_{28,20} = Y_{27,20}$, $Y_{28,21} = Y_{27,21}$.
$Y_{29,8} = Y_{28,8}$.
$X_{15,0} = X_{14,22}$.
$X_{22,31} = X_{21,21}$.

Let $\mathtt{F} = \mathrm{IFZ}(Y_{24}, Y_{23}, Y_{22}^{\lll 10})$ and then we can know $[F]_{26\sim24} = 000$. Next, we add some conditions on m_{14} when randomly choosing its value. Consider the following pattern of m_{14}.

$$m_{14} = \texttt{-----100 0------- -------- --------}.$$

Then, we consider the calculation of Q_{25}.

$$Q_{25} = \mathtt{F} \boxplus \mathtt{Temp} \boxplus m_{14}.$$

We have known the pattern of F, Temp and m_{14}. More specifically, they are as follows:

$$\begin{aligned}
\mathtt{F} &= \texttt{-----000 -------- -------- --------}.\\
\mathtt{Temp} &= \texttt{11110100 01011100 10000001 00101001}.\\
m_{14} &= \texttt{-----100 0------- -------- --------}.\\
Q_{25} &= \texttt{-----00- -------- -------- --------}.
\end{aligned}$$

Therefore, as shown in the above equation, $[Q_{25}^{\lll 7}]_0 = 0$ and $[Q_{25}^{\lll 7}]_1 = 0$ can always hold. In other words, by adding three bit conditions on Y_{23} and three bit conditions on Y_{22} ($Y_{23,24} = 0$, $Y_{23,25} = 0$, $Y_{23,26} = 0$, $Y_{22,14} = 0$, $Y_{22,15} = 0$ and $Y_{22,16} = 0$) and by adding four bit conditions on m_{14} when randomly choosing its value, for the solution of the middle part in Table 6, $Y_{25,1} = 0$ and $Y_{25,0} = 1$ will hold with probability 1. All these newly-added conditions can be satisfied with probability 1 using single-step modification.

Using the same strategy, we can ensure as many as 5 bit bit conditions on Y_{25}. However, this will require more than 30 extra bit conditions. Strangely, according to our experiment, after adding these extra bit conditions, some bit conditions on Y_{24} hold with a very low probability, thus increasing the time to find the message words to ensure the dense right branch. In our opinions, we think this is caused by that the randomness of the the uncontrolled internal states is greatly decreased when too many bits are fixed in the controlled part. In conclusion, to ensure five bit conditions on Y_{25}, more than 30 bit conditions are needed, which reduces the freedom of message words dramatically and the randomness of the uncontrolled part is greatly influenced. Therefore, we only control two bit conditions on Y_{25} as above. That's, we make a trade-off between the theoretical result and the experimental result.

5.2 Complexity Evaluation

As described in [8], the left branch holds with probability 2^{-29}.

For Y_i ($24 \le i \le 30$), since we can ensure two bit conditions on Y_{25}, there are 21 bit conditions on them remaining uncontrolled. In addition, Q_i ($24 \le i \le 30$) satisfy their corresponding equations with probability about 2^{-3}.

For the initial part, $Y_7 = 0$ will always make the condition on Q_{11} hold. Different from [8], we don't control characteristics of Q_{12} and Q_{13}. Q_{12} satisfies its corresponding equations with probability close to 1 and therefore can be neglected. Q_{13} 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 right branch holds with probability of $2^{-21-3-1-4} = 2^{-28}$. The details of the information of Q_i are displayed in Table 5, which is slightly different from [8], i.e., we add some other bit conditions to ensure Q_{23} and Q_{20} hold with probability 1.

Totally, the success probability to find colliding messages using the 30-step differential path found in [8] is 2^{-57}. The degree of freedom in m_0, m_{13}, m_5, m_{14}, m_9 is sufficient to generate a collision. Therefore, under our efficient attack framework, we improve the original time complexity by a factor of 2^{13}.

Table 5: Information of Q_i

Equation: $(Q_i \boxplus in)^{\lll shift} = Q_i^{\lll shift} \boxplus out$					
i	shift	in	out	Extra conditions to control characteristics of Q_i	Probability
11	8	`0x1000000`	`0x1`	$Y_7 = 0$. (Added in the connecting phase.)	1
12	11	`0x15`	`0xa800`		0.99998
13	14	`0x6ffba800`	`0xea001bff`		$\approx 2^{-1}$
14	14	`0x40400001`	`0x1010`	$Y_{10,31} = 0$.	1
15	12	`0xafffff5f`	`0xfff5fb00`	$Y_{15,9} = 0$, $Y_{11,31} = 1$.	1
16	6	`0x9d020`	`0x2740800`	The conditions on the internal states are sufficient.	1
17	9	`0x85f87f2`	`0xbf0fe410`	$Y_{13,20} = 1$, $Y_{13,18} = 0$, $Y_{17,28} = 0$, $Y_{17,26} = 1$, $Y_{13,16} = 0$.	1
18	7	`0x0`	`0x0`		1
19	15	`0xffffd008`	`0xe8040000`	$Y_{15,21} = 0$.	1
20	7	`0xd75fbffc`	`0xafdffdec`	If $[Y_{20}]_{5\sim0} \geq [Y_{16}^{\lll 10}]_{5\sim0}$, then $Y_{20,6} \bigoplus Y_{16,28} = 1$. If $[Y_{20}]_{5\sim0} < [Y_{16}^{\lll 10}]_{5\sim0}$, then $Y_{20,6} \bigoplus Y_{16,28} = 0$. If $[Y_{20}]_{30\sim0} \geq [Y_{16}^{\lll 10}]_{30\sim0}$, then $Y_{20,31} \bigoplus Y_{16,21} = 1$. If $[Y_{20}]_{30\sim0} < [Y_{16}^{\lll 10}]_{30\sim0}$, then $Y_{20,31} \bigoplus Y_{16,21} = 0$.	1
21	12	`0x10200813`	`0x812102`	$Y_{21,6} = 1$, $Y_{17,28} = 0$, $Y_{21,10} = Y_{17,0}$.	1
22	8	`0xff7edffe`	`0x7edffeff`	$Y_{22,30} = 1$, $Y_{18,21} = 1$, $Y_{22,2} = Y_{18,24}$, $Y_{22,3} = Y_{18,25}$, $Y_{22,4} = Y_{18,26}$, $Y_{22,5} = Y_{18,27}$, $Y_{22,6} = Y_{18,28}$, $Y_{22,7} = Y_{18,29}$.	1
23	9	`0x81000001`	`0x102`	If $[Y_{23}]_{7\sim0} \geq [Y_{19}^{\lll 10}]_{7\sim0}$, then $Y_{23,8} \bigoplus Y_{19,30} = 1$. If $[Y_{23}]_{7\sim0} < [Y_{19}^{\lll 10}]_{7\sim0}$, then $Y_{23,8} \bigoplus Y_{19,30} = 0$. $Y_{19,21} = 0$.	1
24	11	`0xffffff00`	`0xfff80000`		0.99987
25	7	`0x80000`	`0x4000000`		$\approx 2^{-0.02}$
26	7	`0x1000800`	`0x80040000`		$\approx 2^{-1}$
27	12	`0x7ffc0000`	`0xbffff800`		$\approx 2^{-1.4}$
28	7	`0x0`	`0x0`		1
29	6	`0xc0000000`	`0xfffffff0`		$\approx 2^{-0.4}$
30	15	`0x10`	`0x80000`		0.99987

5.3 Verification

After adding some additional conditions, we can finally know all the conditions on the internal states and message words. Then, under our attack framework, we find a solution for the right branch as shown in Table 6. The estimated probability is consistent with the experiments.

Table 6: One Instance on the Right Branch, where $m'_{15} = m_{15} \boxplus 2^{24}$, and $\varDelta m_i = 0$ $(0 \leqslant i \leqslant 14)$.

Y_i					$\pi_2(i)$
−4	1 1 0 0 0 0 0 0	0 1 0 1 1 0 0 1	1 1 0 1 0 0 0 1	0 1 0 0 1 0 0 0	
−3	0 1 1 1 1 1 0 0	0 0 1 1 0 0 0 0	1 1 1 1 0 1 0 0	1 0 1 1 1 0 0 0	
−2	0 0 0 1 1 1 0 1	1 0 0 0 0 1 0 0	0 0 0 0 1 1 0 0	1 0 0 1 0 1 0 1	
−1	1 0 0 1 1 0 0 0	1 0 1 1 1 0 1 0	1 1 0 1 1 1 0 0	1 1 1 1 1 1 1 0	
00	1 1 1 0 1 1 1 1	1 1 0 0 1 1 0 1	1 0 1 0 1 0 1 1	1 0 0 0 1 0 0 1	
1	1 1 0 1 0 1 0 1	1 0 0 0 1 0 1 0	0 0 0 0 1 1 0 0	0 1 1 1 0 0 1 0	5
2	0 1 1 0 1 0 1 0	0 0 1 1 1 0 0 1	0 0 0 1 0 1 0 1	1 0 1 0 1 1 0 0	14
3	1 0 0 0 0 1 0 1	1 0 0 1 0 1 0 1	0 0 0 1 1 1 0 1	0 0 0 1 0 1 1 1	7
4	1 0 1 0 0 0 0 1	0 1 0 1 0 0 1 0	0 1 0 0 0 1 1 0	1 0 1 1 0 1 0 1	0
5	0 1 1 1 0 0 0 0	0 0 0 0 0 0 1 1	0 1 0 1 1 1 0 1	1 1 1 0 1 1 0 1	9
6	1 1 1 0 1 1 1 1	1 0 0 1 0 0 1 1	0 0 0 1 0 0 0 0	1 0 1 0 0 1 0 0	2
7	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	11
8	1 1 1 0 1 0 1 1	1 1 1 1 1 0 0 0	0 1 1 0 0 1 1 0	1 1 1 0 1 1 0 1	4
9	1 1 0 1 0 1 1 1	1 1 1 1 0 1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 1 1	13
10	0 1 1 1 0 0 0 0	0 0 1 1 1 1 1 1	0 1 0 0 0 0 0 0	1 0 0 0 1 0 1 0	6
11	1 0 1 1 0 1 1 1	0 0 0 0 1 1 0 1	1 0 0 1 0 0 0 0	0 0 0 n u u u u	15
12	n u u u u u u u	u u u u u u u u	u 0 n 0 n 0 0 1	0 0 0 0 1 1 0 0	8
13	0 u n n 1 u u 0	1 1 1 1 1 0 1 0	0 n u u n n 1 1	0 1 1 0 1 1 u n	1
14	0 1 0 0 0 0 1 1	1 1 1 1 1 1 1 1	1 0 n u 1 0 1 0	1 1 n u 1 1 1 1	10
15	0 0 0 0 1 0 1 1	1 1 0 0 u 1 u 1	1 0 1 0 0 0 0 u	1 1 0 1 0 1 0 1	3
16	1 1 1 1 n 1 u u	0 0 0 n 1 n 1 1	0 0 0 1 n 1 1 1	1 n u u u u u u	12
17	1 u 1 0 1 1 1 u	n 1 1 0 1 1 1 1	0 0 u 1 0 u n n	n 0 n n n 0 1 1	6
18	0 1 0 0 1 0 0 0	0 n 1 0 1 1 1 1	1 n 0 0 0 0 1 0	0 1 0 0 0 0 0 1	11
19	1 u 0 0 0 1 0 1	1 0 0 1 0 0 1 0	0 1 0 1 0 0 1 0	0 0 0 1 1 1 0 1	3
20	0 0 0 0 0 0 0 1	0 1 1 0 0 1 1 0	0 0 0 0 0 n u 1	1 0 1 0 1 1 0 0	7
21	0 1 1 1 1 0 0 1	0 1 1 0 1 0 1 1	1 1 1 1 1 1 1 0	1 1 1 0 0 1 0 1	0
22	u 1 0 1 1 1 0 0	1 1 u 0 1 1 0 0	0 0 0 0 1 1 1 u	0 0 1 0 0 0 0 0	13
23	1 1 0 0 0 0 0 0	1 1 1 1 1 0 1 0	0 1 0 1 1 0 1 0	1 0 1 1 0 0 n 1	5
24	1 1 0 1 1 1 1 1	0 0 0 1 1 0 0 1	0 0 0 0 0 0 1 1	0 1 0 1 0 0 0 0	10
25	1 0 0 0 0 n 1 1	0 1 1 0 0 0 1 0	1 1 0 1 1 1 1 1	1 1 1 1 0 1 0 1	14
26	0 0 0 1 0 0 1 0	0 1 0 0 0 1 1 0	0 0 1 1 u n n 0	0 0 1 0 0 1 1 0	15
27	1 u 1 0 0 1 0 1	0 1 1 1 1 1 0 1	0 1 1 1 0 0 0 1	0 0 1 0 0 0 1 1	8
28	0 1 1 0 0 1 1 1	0 1 1 1 1 1 1 0	0 1 0 0 1 1 1 0	0 1 1 1 1 0 0 0	12
29	0 0 0 1 1 0 1 0	1 1 0 0 1 0 0 1	0 1 0 0 1 0 1 0	1 0 0 1 1 1 1 0	4
30	0 1 0 0 1 0 1 0	0 1 1 1 1 0 0 1	1 1 1 1 1 1 0 0	1 1 1 0 1 0 1 0	9

Message Words	m_0	m_1	m_2	m_3
Value	0x284ca581	0x55fd6120	0x694b052c	0xd5f43d9f

Message Words	m_4	m_5	m_6	m_7
Value	0xa064a7c8	0xb9f7b3cd	0x1221b7bb	0x42156657

Message Words	m_8	m_9	m_{10}	m_{11}
Value	0x121ecfee	0xce7a7105	0xf2d47e6f	0xf567ac2e

Message Words	m_{12}	m_{13}	m_{14}	m_{15}
Value	0x20d0d1cb	0x9d928b7d	0x5c6ff19b	0xc306e50f

6 Opposite Searching Strategy

For some interest, we also consider an opposite searching strategy and the corresponding attack framework which captures the characteristics of the differential path found by such a searching strategy. We hope the readers can refer to Table 3 for a better understanding when reading this part.

The opposite searching strategy is that the right branch is set sparse and fully probabilistic and message modification is only applied on the dense left branch. Therefore, we choose m_{12} as the message word to generate difference. 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 $X_{13} \oplus X_{12} \oplus \overline{X_{11}^{\lll 10}}$. Since there is no difference in X_{11} and X_{12} and it is an XOR operation, there will be always conditions on X_{11} and X_{12}. However, there won't be conditions on X_i $(1 \leq i \leq 10)$. This seems quiet good compared with the original searching strategy where Y_i $(1 \leq i \leq 8)$ are fully free.

Then, we capture this characteristics of the differential path and find the corresponding attack framework under the guidance of the two principles. The optimal position of middle part we finally determined is X_i $(11 \leq i \leq 23)$. In this case, the 5 consecutive internal states to be connected become X_i $(11 \leq i \leq 15)$. However, two of message words (m_{10}, m_{13}) used to update these five internal states are fixed in the middle part. Now, we describe how to use an efficient method to achieve connection in X_{11} and X_{14}.

Observe that m_8 and m_9 is fully free. Consider the calculation of X_{14}, which is fixed in the middle part.

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

Since m_{13} and X_i $(11 \leq i \leq 14)$ are all fixed in the middle part, we can exhaust all 2^{32} possible values of X_9 and obtain 2^{32} possible pairs (X_9, X_{10}) satisfying the above equation.

Consider the calculation of X_{11}, which is also fixed in the middle part.

$$X_{11} = X_7^{\lll 10} \boxplus (XOR(X_{10}, X_9, X_8^{\lll 10}) \boxplus X_6^{\lll 10} \boxplus m_{10} \boxplus K_0^l)^{\lll 14}.$$

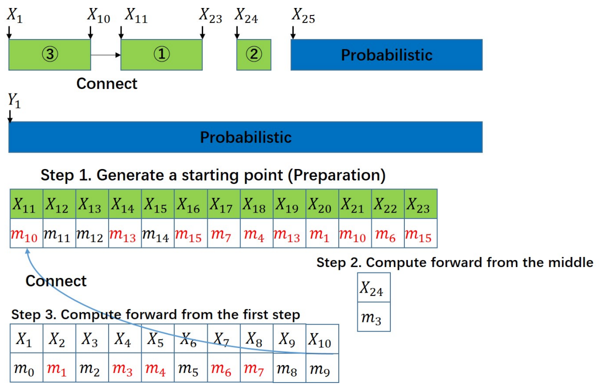
Let

$$\begin{aligned} \mathbf{var} &= ((X_{11} \boxminus X_7^{\lll 10})^{\ggg 14} \boxminus (X_6^{\lll 10} \boxplus m_{10} \boxplus K_0^l)) \oplus X_8^{\lll 10}. \\ X_{10} \oplus X_9 &= \mathbf{var}. \end{aligned}$$

Suppose we have computed until X_8 in the initial part, we then compute the value of **var** and then find a solution of (X_9, X_{10}) from the pre-computed solution set which will make the connection in X_{11} and X_{14} succeed. It is expected that we can find one solution for a random **var** since there are 2^{32} solutions in the pre-computed solution set. Besides, we can obtain the solution quickly by storing it in a table in memory.

Now, we give the attack framework illustrated in Fig. 5 to mount collision attack on RIPEMD-160 whose differential path follows the new searching strategy.

Step 1: Fix the internal states located in the middle part from X_{11} to X_{23}, which can be easily done using single-step modification since only m_{15} is used twice to



X_1
X_{10}
X_{11}
X_{23}
X_{24}
X_{25}
③
①
②
Probabilistic
Connect
Y_1
Probabilistic
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: Attack Framework for RIPEMD-160

update the internal states. Then, pre-compute the solution set **S** for (X_9, X_{10}) based on the following equation.

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

Step 2: Apply single-step modification to ensure the conditions on X_{24} since its corresponding message word m_3 hasn't been fixed in the middle part.

Step 3: Compute from the first step until X_8 and then achieve connection in X_{11} and X_{14} as follows:

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

Find solutions of (X_9, X_{10}) from **S** and then compute m_8 and m_9.

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

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

Comparing this attack framework with that presented in previous part special for the original searching strategy in [8], we find that we can extend one more step that can be controlled. Therefore, we expect to obtain a better collision attack under our attack framework if the differential path is found by taking Strategy 3 into consideration when using the tool.

Strategy 3. The number of conditions on the internal states X_i $(i \geq 25)$ should be as small as possible.

7 Conclusion

In this paper, we re-consider the connecting techniques used to find collisions, which is achieved by connecting the middle part with the initial part. To obtain the most efficient attack framework, we don't consider the ideal case. Motivated by this, we propose two principles which will guide us to find an optimal position of middle part even though the case is not ideal. Following the two principles, we find an optimal position of middle part and the corresponding attack framework which works quite efficiently for the differential path discovered by using the strategy in [8]. Under this attack framework, the 30-step collision attack is also improved by a factor of 2^{13}.

Following our efficient attack framework, we also propose another two strategies when adopting the original strategy in [8] to find a differential path. And we believe that it is potential to obtain a better collision attack on 30-step RIPEMD-160 and extend it to more steps.

For some interest, we also consider an opposite searching strategy to [8]. To capture the characteristics of the differential path discovered using the new strategy, we also propose another efficient attack framework. This framework also provides one more strategy when searching a differential path. Besides, we observe that we can control one more step than the framework special for the original searching strategy and therefore it is possible to obtain a better collision attack on reduced RIPEMD-160.

In conclusion, we propose two efficient attack frameworks special for two different searching strategies to find a differential path. Then, we also give some additional strategies to help find an optimal differential path, which may help improve existing attack. Combining with the searching tool, it is potential to give a tight upper bound of steps to mount collision attack on reduced RIPEMD-160 when adopting the two strategies.

References

1. Christophe De Cannière and Christian Rechberger. Finding SHA-1 characteristics: General results and applications. In *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, pages 1–20, 2006.
2. 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.
3. 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.
4. Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Analysis of SHA-512/224 and SHA-512/256. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 612–630, 2015.
5. Maria Eichlseder, Florian Mendel, and Martin Schläffer. Branching heuristics in differential collision search with applications to SHA-512. In *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, pages 473–488, 2014.
6. 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.

7. Gaëtan Leurent. Message freedom in MD4 and MD5 collisions: Application to APOP. In *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, pages 309–328, 2007.
8. 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.
9. 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.
10. Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding SHA-2 characteristics: Searching through a minefield of contradictions. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 288–307, 2011.
11. 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.
12. Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding collisions for round-reduced SM3. In *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, pages 174–188, 2013.
13. Florian Mendel, Tomislav Nad, and Martin Schläffer. Improving local collisions: New attacks on reduced SHA-256. 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 262–278, 2013.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. Gaoli Wang, Yanzhao Shen, and Fukang Liu. Cryptanalysis of 48-step RIPEMD-160. *IACR Trans. Symmetric Cryptol.*, 2017(2):177–202, 2017.
20. 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.
21. 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.

22. 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.
23. 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.