Improved Cryptanalysis of Reduced RIPEMD-160

Florian Mendel[1], Thomas Peyrin[2], Martin Schläffer[1],
Lei Wang[2], and Shuang Wu[2]

[1] IAIK, Graz University of Technology, Austria
florian.mendel@iaik.tugraz.at martin.schlaeffer@iaik.tugraz.at

[2] Nanyang Technological University, Singapore
thomas.peyrin@gmail.com
wang.lei@ntu.edu.sg wushuang@ntu.edu.sg

Abstract. In this article, we propose an improved cryptanalysis of the double-branch hash function standard RIPEMD-160. Using a carefully designed non-linear path search tool, we study the potential differential paths that can be constructed from a difference in a single message word and show that some of these message words can lead to very good differential path candidates. Leveraging the recent freedom degree utilization technique from Landelle and Peyrin to merge two branch instances, we eventually manage to obtain a semi-free-start collision attack for 42 steps of the RIPEMD-160 compression function, while the previously best know result reached 36 steps. In addition, we also describe a 36-step semi-free-start collision attack which starts from the first step.

Keywords: RIPEMD-160, semi-free-start collision, compression function, hash function

1 Introduction

Due to their widespread use in many applications and protocols, hash functions are among the most important primitives in cryptography. A hash function H is a function that takes an arbitrarily long message M as input and outputs a fixed-length hash value of size n bits. Cryptographic hash functions have the extra requirement that some security properties, such as collision resistance and (second)-preimage resistance, must be fulfilled. More precisely, it should be impossible for an adversary to find a collision (two distinct messages that lead to the same hash value) in less than $2^{n/2}$ hash computations, or a (second)-preimage (a message hashing to a given challenge) in less than 2^n hash computations. Most standardized hash functions are based upon the Merkle-Damgård paradigm [13,3] and iterate a compression function h with fixed input size to handle arbitrarily long messages. The compression function itself should ensure equivalent security properties in order for the hash function to inherit from them.

The cryptographic community have seen very impressive advances in hash functions cryptanalysis in the recent years [20,18,19,17], with weaknesses or even sometimes collisions exhibited for many standards such as MD4, MD5, SHA-0 and SHA-1. These functions have in common their design strategy, based on the utilization of additions, rotations, xors and boolean functions in an unbalanced Feistel network. In order to diversify the panel of standardized hash functions and make a backup plan available in case the last survivors of this MD-SHA family gets broken as well, NIST organized a 4-year SHA-3 competition which led to the selection of Keccak [1] as new standardized primitive. The move of the industry towards SHA-3 will take a lot of time and even broken functions such as MD5 or SHA-1 remain widely used. Among the MD-SHA family, only SHA-2 and RIPEMD-160 compression functions are still unbroken, although practical collisions on the SHA-2 compression function have been improved from 24 to 38 steps recently [11,12]. The compression function used in RIPEMD-128 was recently shown not to be collision resistant [8].

RIPEMD can be considered as a subfamily of the MD-SHA-family as its first representative, RIPEMD-0 [2], basically consists in two MD4-like [15] functions computed in parallel (but with different constant additions for the two branches), with 48 steps in total. Even though RIPEMD-0 was recommended by the European *RACE Integrity Primitives Evaluation* (RIPE) consortium, its security was put into question with the early work from Dobbertin [5] and the practical collision attack from Wang *et al.* [17]. Meanwhile, in 1996, Dobbertin, Bosselaers and Preneel [6] proposed two strengthened versions of the original RIPEMD-0, called RIPEMD-128 and RIPEMD-160, with 128/160-bit output and 64/80 steps respectively. RIPEMD-0 main flaw was that its two computation branches were too much similar and this issue was patched in RIPEMD-128 and RIPEMD-160 by using not only different constants, but also different rotation values, boolean functions and message insertion schedules in the two branches. This two-branch structure in RIPEMD family is a good method to reduce the ability of the attacker to properly use the available freedom degrees and to find good differential paths for the entire scheme. RIPEMD-160 is a worldwide ISO/IEC standards [7] that is yet unbroken and is present in many implementations of security protocols.

As of today, the best results on RIPEMD-160 are a very costly 31-step preimage attack [14], a practical 36-step semi-free-start collision attack [10] on the compression function (not starting from the first step), and a distinguisher on up to 51 steps of the compression function with a very high complexity [16].

Our contributions. In this article, we improve the best know results on RIPEMD-160, proposing a semi-free-start collision attack on 42 steps of its compression function and a semi-free-start collision attack on 36 steps starting from the first step. Our differential paths were crafted thanks to a very efficient non-linear path search tool (Section 3) and by inserting a difference only in a single message word, in a hope for a sparse difference trail. We then explain in Section 4 why we believe the 8^{th} message input word (M_7) is the best candidate for that matter. Once the differential paths settled, we leverage in Section 5 the freedom degree utilization technique introduced by Landelle and Peyrin [8] for RIPEMD-128 that merges two branch instances together in order to obtain a semi-free-start collision. It is to be noted that the step function of RIPEMD-160 makes it much more difficult to find a collision attack compared to RIPEMD-128. This is mainly due to the fact that the diffusion is better, but also because even though differences might be absorbed in the boolean function, they will propagate anyway at least once through a free term (which was not the case in RIPEMD-128). We give a description of RIPEMD-160 in Section 2 and summarize our results in Section 6.

2 Description of RIPEMD-160

RIPEMD-160 [6] 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 that takes as input a 512-bit message block m_i and a 160-bit chaining variable cv_i:

$$cv_{i+1} = h(cv_i, m_i)$$

where the message m to hash is padded beforehand to a multiple of 512 bits[3] and the first chaining variable is set to a predetermined initial value $cv_0 = IV$ (given in Table 3 of Appendix A).

We refer to [6] for a complete description of RIPEMD-160. In the rest of this article, we denote by $[Z]_i$ the i-th bit of a word Z, starting the counting from 0. $\boxplus$ and $\boxminus$ represent the modular addition and subtraction on 32 bits, and $\oplus$, $\vee$, $\wedge$, the bitwise "exclusive or", the bitwise "or", and the bitwise "and" function respectively.

2.1 RIPEMD-160 compression function

The RIPEMD-160 compression function is a wider version of RIPEMD-128, which is in turn based on MD4, but with the particularity that it uses two parallel instances of it. We differentiate these two computation branches by left and right branch and we denote by X_i (resp. Y_i) the 32-bit word of left branch (resp. right branch) that will be updated during step i of the compression function. The compression function process is composed 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 5 words h_i of 32 bits each, that will be used to initialize the left and right branch 160-bit internal state:

$$\begin{array}{lllll} X_{-4} = (h_0)^{\ggg 10} & X_{-3} = (h_4)^{\ggg 10} & X_{-2} = (h_3)^{\ggg 10} & X_{-1} = h_2 & X_0 = h_1 \\ Y_{-4} = (h_0)^{\ggg 10} & Y_{-3} = (h_4)^{\ggg 10} & Y_{-2} = (h_3)^{\ggg 10} & Y_{-1} = h_2 & Y_0 = h_1 \ . \end{array}$$

[3] The padding is the same as for MD4: a "1" is first appended to the message, then x "0" bits (with $x = 512 - (|m| + 1 + 64 \pmod{512})$) are added, and finally the message length $|m|$ coded on 64 bits is appended as well.

The message expansion. The 512-bit input message block is divided into 16 words M_i of 32 bits each. Each word M_i will be used once in every round in a permuted order (similarly to MD4 and RIPEMD-128) and for both branches. We denote by W_i^l (resp. W_i^r) the 32-bit expanded message word that will be used to update the left branch (resp. right branch) during step i. We have for $0 \leq j \leq 4$ and $0 \leq k \leq 15$:

$$W^l_{j\cdot 16+k} = M_{\pi^l_j(k)} \quad \text{and} \quad W^r_{j\cdot 16+k} = M_{\pi^r_j(k)}$$

where permutations π^l_j and π^r_j are given in Table 4 of Appendix A.

The step function. At every step i, the registers X_{i+1} and Y_{i+1} are updated with functions f^l_j and f^r_j that depend on the round j in which i belongs:

$$\begin{aligned} X_{i+1} &= (X_{i-3})^{\lll 10} \boxplus ((X_{i-4})^{\lll 10} \boxplus \Phi^l_j(X_i, X_{i-1}, (X_{i-2})^{\lll 10}) \boxplus W^l_i \boxplus K^l_j)^{\lll s^l_i}, \\ Y_{i+1} &= (Y_{i-3})^{\lll 10} \boxplus ((Y_{i-4})^{\lll 10} \boxplus \Phi^r_j(Y_i, Y_{i-1}, (Y_{i-2})^{\lll 10}) \boxplus W^r_i \boxplus K^r_j)^{\lll s^r_i}, \end{aligned}$$

where K^l_j, K^r_j are 32-bit constants defined for every round j and every branch, s^l_i, s^r_i are rotation constants defined for every step i and every branch, Φ^l_j, Φ^r_j are 32-bit boolean functions defined for every round j and every branch. All these constants and functions are given in Appendix A.

The finalization. A finalization and a feed-forward is applied when all 80 steps have been computed in both branches. The four 32-bit words h'_i composing the output chaining variable are finally obtained by:

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

3 Non-linear path search

To find a non-linear differential path in RIPEMD-160, we use the techniques developed by Mendel et al. [11,12]. This automated search algorithm can be used to find both, differential characteristics and conforming message pairs (note that we will use it only for the differential characteristics part in this article). We briefly describe the tool in Section 3.1, and the new improvements and specific configuration for RIPEMD-160 and our attack in Section 3.2.

3.1 Automated search for differential characteristics

The basic idea of the search algorithm is to pick and guess previously unrestricted bits. After each guess, the information due to these restrictions is propagated to other bits. If an inconsistency occurs, the algorithm backtracks to an earlier state of the search and tries to correct it. Similar to [11,12], we denote these three parts of the search by decision (guessing), deduction (propagation), and backtracking (correction). Then, the search algorithm proceeds as follows:

Decision (Guessing)
1. Pick randomly (or according to some heuristic) an unrestricted decision bit.
2. Impose new constraints on this decision bit.

Deduction (Propagation)
3. Propagate the new information to other variables and equations as described in [11,12].
4. If an inconsistency is detected start backtracking, else continue with step 1.

Backtracking (Correction)
5. Try a different choice for the decision bit.
6. If all choices result in an inconsistency, mark the bit as critical.
7. Jump back until the critical bit can be resolved.
8. Continue with step 1.

During the search, we mainly use generalized conditions [4] to store, restrict and propagate information (see Table 7 of Appendix B). The decision of choosing which bits to guess depends strongly on the specific attack, hash function and preferred resulting path. E.g. if some parts of the non-linear path should be especially sparse, we guess the corresponding state words first.

Similar to [11,12], new restrictions are propagated using brute-force propagation within bitslices for each Boolean function and modular addition. In the backtracking, we remember a small set of critical bits and repeatedly check if all of them can be resolved. This way, we leave dead search branches faster. Additionally, we restart the search after a certain number of inconsistencies occur.

The main difficulty in finding a long differential characteristic lies in the fine-tuning of the search algorithm. There are a lot of variations possible which can decide whether the search eventually succeeds or fails. We describe the specific improvements for `RIPEMD-160` in the next section.

3.2 Improvements for `RIPEMD-160`

To efficiently find non-linear differential paths and message pairs for a larger number of steps than in previous attacks [10], we had to improve the search in several ways. Especially finding a non-linear path for the XOR-round of `RIPEMD-160` was quite challenging.

In order to improve the propagation of information, we have combined the bitslices of the two modular additions in each step of `RIPEMD-160` into a single bitslice. The two carries of the first and second modular addition are computed and stored together within a generalized 3-bit condition, which is defined similarly as the 2.5-bit condition of [9]. Without this combination, many contradictions would be detected very late during the search, and therefore reduce the overall performance.

To find sparser paths at the beginning or end of the non-linear path, we first propagate the single bit condition in the message word backward and forward more or less linearly and by hand. Then, the automatic search tool is used to connect the paths. Note that due to the additional modular addition in `RIPEMD-160`, we can stay sparse longer in forward direction than in backward direction. This can be observed when looking at our resulting differential paths in Appendix D.

Once we have found a candidate for a differential path, we immediately continue the search for partial confirming message pairs, similar as in [11,12]. We first pick decision bits '-' which are constraint by linear two-bit conditions of the form ($X_{i,j} = X_{k,l}$ or $X_{i,j} \neq X_{k,l}$). This ensures that those bits which influence a lot of other bits are guessed first. This way, inconsistent characteristics are found faster and can also be corrected by the backtracking step of the path search.

4 Differential paths

The previous semi-free-start collision attacks on `RIPEMD` compression functions usually start by spending the available freedom degrees in the first steps in each branch, and then continue the computation in the forward direction, verifying the rest of the differential path in each branch probabilistically. With this attack strategy, the non-linear part of the differential paths for both branches should be located in the early steps. Indeed, the non-linear parts are usually the most costly part and therefore should be handle in priority by the attack with the available freedom degrees.

Since the compression functions belonging to the `RIPEMD` family use a two-branch parallel structure sharing the same initial chaining value, the left and right branches can be regarded as somehow connected in the first steps. With this observation, in [8] Landelle and Peyrin proposed a new method to find semi-free-start collisions for `RIPEMD-128`. Their method allows the attacker to use the message freedom degrees not necessarily in the early steps of each branch, and therefore relax a bit the constraint that the most costly parts (the non-linear chunks) must be located in the early steps as well. Consequently, the space of possible differential paths is increased and likely to contain better candidates since the probabilistic part in each branch is reduced.

Figure 1 shows the difference between the previous and the new strategies. The attack process proposed in [8] is made of three steps. Firstly, the attacker independently choose the internal states in both branches and start fixing some message words in order to handle the two non-linear parts. Then, he uses some of the remaining message words available to merge the two branches to the same chaining variable by computing backward from the middle. Finally, the rest of the differential path in both branches is verified probabilistically by computing forward from the middle.

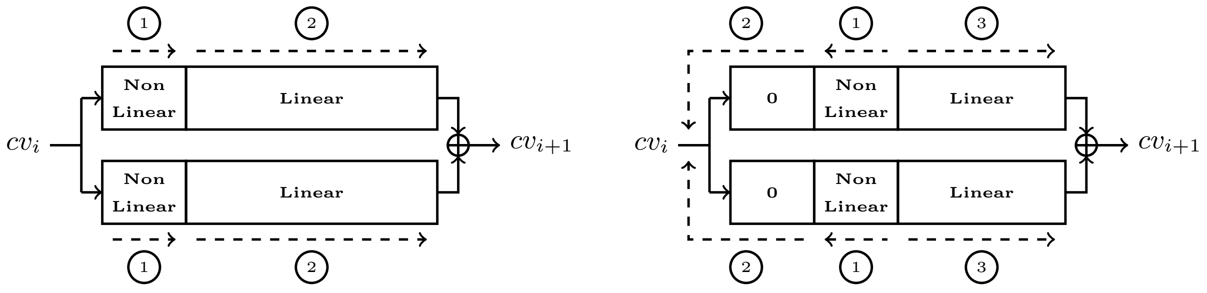

Fig. 1. The previous (left-hand side) and new (right-hand side) approach for collision search on double-branch compression functions introduced in [8].

4.1 On the choice of the message word

As in [8], in order to find a sparse differential path for a semi-free-start collision attack with the biggest number of steps, we chose to insert differences in only a single message word. Then, for all the 16 message words, we have analyzed how many steps can be potentially attacked. The results are summarized in Table 1. Note that the details of the attacks are not considered at this stage and the final complexity will highly depend on the merging process and the quality of the differential paths that can be found (both linear and non-linear parts). Yet, by guessing for each message insertion where would be the best location for the two non-linear parts, this preliminary analysis gives a rough estimation of how many steps can be reached potentially. The overall attack being quite complex and time consuming to settle, this will help us to focus directly on good candidates.

We found that message words M_7 and M_{14} both seem to be rather good choices when trying to verify the following criteria:

1. the non-linear part in both branches should be short, in order to consume less freedom degrees.
2. the early steps of the two non-linear parts should be rather close to each other, which will help the merging.
3. the late steps of the non-linear parts should be as sparse as possible, since after the merging comes the probabilistic phase and ensuring a sparse incoming difference mask would guarantee a rather high differential probability when computing forward.
4. some message word difference injections allow the differences injected in very late steps in the two branches to cancel each other through the final feed-forward operation. If this trick is applicable, one can usually get 4 to 6 extra steps for the collision attack with a relatively low cost.

Once M_7 and M_{14} identified as good candidates, we tried to design the entire differential path and establish the merging phase. During our search for the linear part of the differential path, we found it much harder to find good ones for `RIPEMD-160` compared to `RIPEMD-128`. The reason is that the diffusion of the step function of `RIPEMD-160` is much better than `RIPEMD-128` as it prevents from fully and directly absorbing all the differences. For example, a step in an IF round in `RIPEMD-128` can be fully controlled by the attacker such that no difference diffusion occurs. However, in `RIPEMD-160`, one extra free term appears in the addition of the step function formula and this forces at least a diffusion of a factor two (that cannot be absorbed by the IF function). As a consequence, we were not able to find differential paths as sparse as in [8] and the number of attacked steps is also much lower.

4.2 Difficulty of calculating the probability

Another important difference between `RIPEMD-128` and `RIPEMD-160` is the step differential probability calculation. While it is easy to calculate the differential probability for each step of a given differential path of `RIPEMD-128`, it is not the case for `RIPEMD-160`. The reason is that the step function in `RIPEMD-160` is no longer a S-function (a function for which the i-th output bit depends only on the i first lower bits of all input words), and therefore the accurate calculation of the differential probability is very hard. Yet, one can write the step function as two S-functions by introducing a temporary state that we denote Q_i.

Table 1. Rough estimation of the number of attackable steps for various choices of message words differences injection (in parenthesis are given the steps window)

Message Word	M_0	M_1	M_2	M_3
Attackable Steps	51 (26-76)	46 (2-47)	52 (6-57)	48 (4-51)

Message Word	M_4	M_5	M_6	M_7
Attackable Steps	42 (8-49)	50 (6-55)	39 (10-48)	56 (8-63)

Message Word	M_8	M_9	M_{10}	M_{11}
Attackable Steps	36 (12-47)	39 (10-48)	37 (14-50)	38 (12-49)

Message Word	M_{12}	M_{13}	M_{14}	M_{15}
Attackable Steps	38 (16-53)	34 (41-74)	58 (2-59)	43 (11-53)

We use the step function of the left branch as an example:

$$\begin{aligned} Q_i &= (X_{i-4})^{\lll 10} \boxplus \Phi^l_j(X_i, X_{i-1}, (X_{i-2})^{\lll 10}) \boxplus W^l_i \boxplus K^l_j, \\ X_{i+1} &= (X_{i-3})^{\lll 10} \boxplus Q_i^{\lll s^l_i}. \end{aligned}$$

Now the probability of the sub-steps can be calculated precisely. One possible way to calculate the probability of the step function is to specify the conditions on Q_i and obtain $Pr[X_i \to Q_i] \cdot Pr[Q_i \to X_{i+1}]$ as the step probability.

In fact, this estimation of the probability is not correct. First, there is no freedom degree injected in the step $Q_i \to X_{i+1}$, which means it is not independent from the step of $X_i \to Q_i$. Thus their probability can not be calculated as a simple multiplication. Even if this estimation is accurate, it will only represent a lower bound of the real probability, since there could be a lot of possible equivalent characteristics on Q_i and only one is taken in account here. We used experiments to estimate the real probability and found that the probabilities obtained using the first method is much lower than the real probability observed when running the attack.

We then tried to come up with another way to calculate the step differential probability. We summed the probabilities of the two sub-steps for all possible characteristics on Q_i, i.e. we used $\Sigma_{Q_i}(Pr[X_i \to Q_i] \cdot Pr[Q_i \to X_{i+1}])$ as differential probability for a step. The calculated probability turned out to be much higher than the real one. This is explained by the fact that characteristics on Q_i in different steps will sometimes introduce conditions on X_i and there could be contradictions between some of the conditions. In the calculation, we did not consider the compatibility between the Q_i in different steps. It is therefore not surprising that the calculated probability is much higher.

In the following sections, all the probabilities given were obtained by experiments while testing random samples. We leave the problem of theoretically calculating the real step differential probability as an open problem.

4.3 48-step semi-free-start collision path

We eventually chose M_7 as message word for the single difference insertion and the shape of the differential path that we will use can be found in Figure 2. The non-linear parts are located between steps 16-41 and 19-36 for left the right branch respectively. In steps 58-64, after a linear propagation of the difference injected by M_7, the differences in the output internal state are suitable to apply the feed-forward trick that allows us to get a collision on the output of the compression function (at the end of step 64). The complete differential path is displayed in Figure 6 in Appendix.

Note that this differential path does not necessarily require to be followed until step 64 to find a collision (thanks to the feed-forward trick). Indeed, by stopping 6 steps before (step 58), the last difference insertions from M_7 will be removed and no difference will be present in the internal states in both branches (therefore leading directly to a collision, without even using the feed-forward trick). We did a measurement and found that the collision probability for the feed-forward trick (from step 58 to 64) is about $2^{-11.3}$. However, our attacks requiring already a lot of operations, we have to remove these extra 6 steps and aim for a 42-step semi-free-start collision attack instead. Yet, one should keep in mind that a rather small improvement with regards to the attack complexity would probably lead to the direct obtaining of a 48-step semi-free-start collision attack by putting back the 6 extra steps. The details of the attack will be given in the next section.

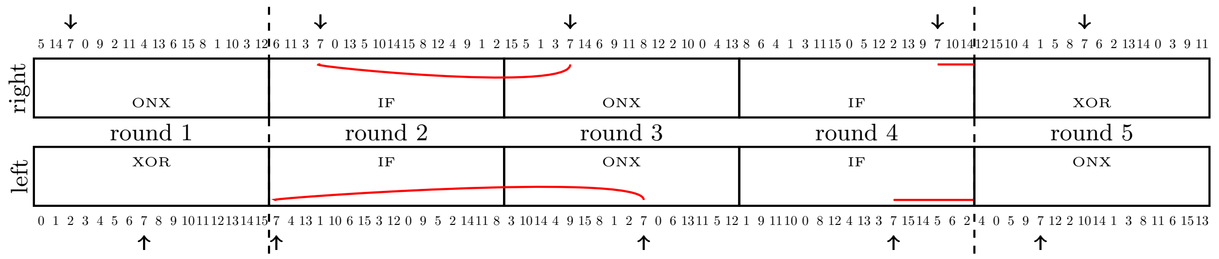

Fig. 2. The shape of our 48-step differential path for the semi-free-start collision attack on the `RIPEMD-160` compression function. The numbers represent the message words inserted at each step and the red curves represent the rough amount of differences in the internal state during each step. The arrows show where the bit differences are injected with M_7. The dashed lines represent the limits of the steps attacked.

4.4 36-step semi-free-start collision path from the first step

Besides the 48-step path, we also exhibit a semi-free-start collision path starting from the first step, which also use message word M_7 to introduce differences. Since the boolean function in the first round of the left branch is XOR, it is quite hard to find a non-linear differential path. As a consequence, the path we were able to find turns out to have three bits of differences in M_7 instead of a single one. The local collisions are located between the first two injections of M_7 in both branches, thus one can directly derive a 36-step collision path starting from the very first step of `RIPEMD-160`. Figure 3 shows the shape of this differential path and the detailed path is given in Figure 7 in Appendix.

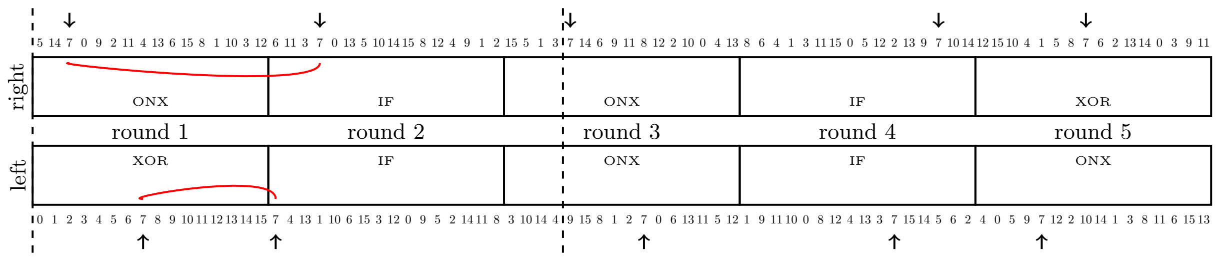

Fig. 3. The shape of our differential path for the 36-step semi-free-start collision attack on the `RIPEMD-160` from the first step. The numbers are the message words inserted at each step and the red curves represent the rough amount differences in the internal state during each step. The arrows show where the bit differences are injected with M_7. The dashed lines represent the limits of the steps attacked.

5 Merging the two branches

Once the differential path is set, we need not only to find conforming pairs for both branches, but also to merge the two branches in order to make sure that they will reach the same chaining variables on their input. Note that for a semi-free-start collision, one only needs to ensure that the input chaining variables for both branches are the same and the attacker can actually choose this value freely. In contrary, for a hash collision, the attacker would have to merge both branches to the same chaining variable, fixed to a certain predefined value.

5.1 Semi-free-start collision

As explained in previous sections, even though a interesting 48-step differential path has been found (Figure 6), we will only look for a 42-step semi-free-start collision attack on `RIPEMD-160`, since the feed-forward collision trick would increase the attack complexity beyond the birthday bound. Our algorithm to find a semi-free-start collision is separated in three phases, which we quickly describe here as a high-level view:

- Phase 1: fix some bits of the message words and the internal states in both branches as preparation for the next phases of the attack. This will allow us to fulfill in advance some conditions of the differential path.
- Phase 2: fix the internal state variables $X_{26}, X_{27}, X_{28}, X_{29}, X_{30}$ of the left branch and $Y_{21}, Y_{22}, Y_{23}, Y_{24}, Y_{25}$ of the right branch. Then, iteratively fix message words M_{11}, M_{15}, M_8, M_3, M_{12}, M_{14}, M_{10}, M_2, M_5, M_9, M_0 and M_6 in this particular order, so as to fulfill the conditions located inside or close to the non-linear parts. Once these internal state variables or message words are successfully fixed, we call this candidate at the end of phase 2 a *starting point* for the merging.
- Phase 3: use the remaining free message words M_1, M_4, M_7 and M_{13} to merge the internal states of both branches to the same input chaining value. Since every value is fixed at this point, check if the rest of the differential path is fulfilled as well (the uncontrolled part).

Phase 1: preparation. Before finding a starting point for the merging, we can prepare the differential path by introducing certain conditions on the internal states in both branches in order to increase the probability of the uncontrolled part of the differential path.

The condition that we will force is that bits 16 to 25 of X_{35} must be equal to `0n00n00000`. The effect of this condition is that when a starting point will be generated, we will be able to directly deduce the 8 lowest bits of X_{37} only by fixing bits 16 to 25 of M_9. In order to explain this, note that calculating X_{37} during the step function in the forward direction gives:

$$X_{37} = X_{33}^{\lll 10} \boxplus (X_{32}^{\lll 10} \boxplus \texttt{ONZ}(X_{36}, X_{35}, X_{34}) \boxplus M_9 \boxplus K_{36}^{l})^{\lll 14}$$

Since $\texttt{ONZ}(X_{36}, X_{35}, X_{34}) = (X_{36} \vee \overline{X_{35}}) \oplus X_{34}$, bits 16 to 25 of X_{36} will have no influence on the output of the boolean function `ONZ` if the corresponding X_{35} bits are set to zero (in a starting point, X_{32}, X_{33}, X_{34} and X_{35} are already fully known). Then, we can choose M_9 such that bit 16 of $X_{32}^{\lll 10} \boxplus (((X_{36} \vee \overline{X_{35}}) \oplus X_{34})\&\texttt{3ff}) \boxplus (M_9\&\texttt{3ff}) \boxplus K_{36}$ equals zero, which will stop the carry coming from the lower bits. As a result, the 8 lowest bits of X_{37} will not depend on X_{36} anymore (and thus neither on M_4 when computing forward, since X_{36} directly depends on M_4) .

One example of our generated starting points is shown in Figure 4, in which we applied our preparation trick. Before generating this starting point, we forced the additional conditions on X_{35}, and once the starting point found, fixing bits 16 to 25 of M_9 to `0110100010` will make sure that the last 8 bits of X_{37} will be equal to `11111010`. Note that the 26-th bit of M_9 and 9-th bit of X_{37} are deduced from the known conditions.

Applying this trick is interesting for the attacker because the uncontrolled probability (steps 35-58) of the left branch is increased.

Phase 2: finding a starting point. Given the differential path from Figure 6, we can use the freedom degrees available in both left and right branches internal states (320 bits) and in the message words (512 bits) to fulfill as much differential conditions as possible. To make the attacker easier, we chose to fix first the five consecutive internal states words that contain the most differential conditions ($X_{26}, X_{27}, X_{28}, X_{29}, X_{30}$ in the left branch and $Y_{21}, Y_{22}, Y_{23}, Y_{24}, Y_{25}$ in the right branch). Then, we fix a few message words one by one in the given order and continue the computation of the internal states by computing forward and backward in both branches (from $X_{26}, X_{27}, X_{28}, X_{29}, X_{30}$ and $Y_{21}, Y_{22}, Y_{23}, Y_{24}, Y_{25}$)

Step	X_i	W_i^l	Π_i^l	Step	Y_i	W_i^r	Π_i^r
12	-------- -------- -------- --------			12	-------- -------- -------- --------		
13	-------- -------- -------- --------			13	-------- -------- -------- --------		
14	-------- -------- -------- --------			14	-------- -------- -------- --------		
15	-------- -------- -------- --------			15	-------- -------- -------- --------		
16	-------- -------- -------- --------	x------- -------- -------- --------	7	16	-------- -------- -------- --------	-------- -------- -------- --------	6
17	-------- -------- -------- -n------	-------- -------- -------- --------	4	17	-------- -------- -------- --------	01100111 01010111 01001110 11101100	11
18	-------- -------- -------- -0------	-------- -------- -------- --------	13	18	10011001 01001100 00000011 10111000	01001111 11011100 00000100 11100000	3
19	-------- -------1 -------- -1------	-------- -------- -------- --------	1	19	10011110 00000100 10101111 00011011	x------- -------- -------- --------	7
20	-------- -------0 -------- --------	00000011 00100101 10100111 00001111	10	20	11001011 10111100 00011101 1u111001	-------- -------- -------- --------	0
21	-------- -------n -------- --------	-------- -------- -------- --------	6	21	11001011 10101u10 11111111 10001111	-------- -------- -------- --------	13
22	-------- -------0 00111100 0-----00	11110110 11100100 10110100 00010001	15	22	001unn01 00011101 011nuuuu 0110010u	01100000 11111110 10100110 01000000	5
23	01101110 00100100 11111101 10111000	01001111 11011100 00000100 11100000	3	23	00000un0 0110uuu1 01001010 1un01001	00000011 00100101 10100111 00001111	10
24	11010101 00011001 01001010 10110101	11101100 10100110 10100100 11100111	12	24	u11n1001 11111111 1000u010 11nn0010	11000000 00110011 00110000 01100000	14
25	11101n11 11110111 01100001 10110000	-------- -------- -------- --------	0	25	00u11000 n1010111 1111u100 1110n1nu	11110110 11100100 10110100 00010001	15
26	00110001 001001un nnnn00nu u1010001	-----011 01000010 -------- --------	9	26	01111001 11110111 001n1nnn 0nnnu000	10001110 10111001 11000010 10010100	8
27	0n00n1un 0u110100 00001111 10101010	01100000 11111110 10100110 01000000	5	27	10001u11 01uu011n 11110001 10100unn	11101100 10100110 10100100 11100111	12
28	10100010 101100nu 00n00nu1 u0un0110	01101100 10101011 01110010 00010011	2	28	1100u001 01n10100 01101101 11n000n1	-------- -------- -------- --------	4
29	01nun101 110u110u 11010101 11u0nnu1	11000000 00110011 00110000 01100000	14	29	00---0-- ---u--1- -u---u-- --------	-----011 01000010 -------- --------	9
30	00001111 uu010n10 1u11nu1n 000nn000	01100111 01010111 01001110 11101100	11	30	00--n--- --n----- -------- ---10--0	-------- -------- -------- --------	1
31	1n00110u 0uu11001 000011n0 0001n000	10001110 10111001 11000010 10010100	8	31	-1------ -------- -----u-- --1-----	01101100 10101011 01110010 00010011	2
32	11011010 1010011u 01101u10 00010111	01001111 11011100 00000100 11100000	3	32	-------- --u----- -----111 000-----	11110110 11100100 10110100 00010001	15
33	1n01101u 10u01101 11100110 11111001	00000011 00100101 10100111 00001111	10	33	-------- -------- -------- --------	01100000 11111110 10100110 01000000	5
34	11100110 10010010 01101000 u1100011	11000000 00110011 00110000 01100000	14	34	-------- -------- -------- --------	-------- -------- -------- --------	1
35	0111100n 00n00000 11110101 10110010	-------- -------- -------- --------	4	35	-------- -------- -------- --------	01001111 11011100 00000100 11100000	3
36	-------1 --0----- -------- u----1--	-----011 01000010 -------- --------	9	36	-------- -------- -------- --------	x------- -------- -------- --------	7
37	-------- --n---1- -------1 11111010	11110110 11100100 10110100 00010001	15	37	-------- -------- -------- --------	11000000 00110011 00110000 01100000	14
38	-------- --0---00 00000--- --------	10001110 10111001 11000010 10010100	8	38	-------- -------- -------- --------	-------- -------- -------- --------	6
39	-------- -------- -------- --------	-------- -------- -------- --------	1	39	-------- -------- -------- --------	-----011 01000010 -------- --------	9
40	-------- -------- -------- --------	01101100 10101011 01110010 00010011	2	40	-------- -------- -------- --------	01100111 01010111 01001110 11101100	11
41	-------- -------- -------- --------	x------- -------- -------- --------	7	41	-------- -------- -------- --------	10001110 10111001 11000010 10010100	8
42	-------- -------- -------- --------	-------- -------- -------- --------	0	42	-------- -------- -------- --------	11101100 10100110 10100100 11100111	12
43	-------- -------- -------- --------	-------- -------- -------- --------	6	43	-------- -------- -------- --------	01101100 10101011 01110010 00010011	2
44	-------- -------- -------- --------	-------- -------- -------- --------	13	44	-------- -------- -------- --------	00000011 00100101 10100111 00001111	10
45	-------- -------- -------- --------	01100111 01010111 01001110 11101100	11	45	-------- -------- -------- --------	-------- -------- -------- --------	0
46	-------- -------- -------- --------	01100000 11111110 10100110 01000000	5	46	-------- -------- -------- --------	-------- -------- -------- --------	4
47	-------- -------- -------- --------	11101100 10100110 10100100 11100111	12	47	-------- -------- -------- --------	-------- -------- -------- --------	13
48	-------- -------- -------- --------	-------- -------- -------- --------	1	48	-------- -------- -------- --------	10001110 10111001 11000010 10010100	8
49	-------- -------- -------- --------	-----011 01000010 -------- --------	9	49	-------- -------- -------- --------	-------- -------- -------- --------	6
50	-------- -------- -------- --------	01100111 01010111 01001110 11101100	11	50	-------- -------- -------- --------	-------- -------- -------- --------	4
51	-------- -------- -------- --------	00000011 00100101 10100111 00001111	10	51	-------- -------- -------- --------	-------- -------- -------- --------	1
52	-------- -------- -------- --------	-------- -------- -------- --------	0	52	-------- -------- -------- --------	01001111 11011100 00000100 11100000	3
53	-------- -------- -------- --------	10001110 10111001 11000010 10010100	8	53	-------- -------- -------- --------	01100111 01010111 01001110 11101100	11
54	-------- -------- -------- --------	11101100 10100110 10100100 11100111	12	54	-------- -------- -------- --------	11110110 11100100 10110100 00010001	15
55	-------- -------- -------- --------	-------- -------- -------- --------	4	55	-------- -------- -------- --------	-------- -------- -------- --------	0
56	-------- -------- -------- --------	-------- -------- -------- --------	13	56	-------- -------- -------- --------	01100000 11111110 10100110 01000000	5
57	-----0-- -------- -------- --------	01001111 11011100 00000100 11100000	3	57	-------- -------- -------- --------	11101100 10100110 10100100 11100111	12
58	-----1-- -------- -------- --------	x------- -------- -------- --------	7	58	-------- -------- -------- --------	01101100 10101011 01110010 00010011	2
59	-------- -------- -------- ---n----	11110110 11100100 10110100 00010001	15	59	-------- -------- -------- --------	-------- -------- -------- --------	13
60	-------- -------- -------- --------	11000000 00110011 00110000 01100000	14	60	-------- -------- -------- --------	-----011 01000010 -------- --------	9
61	-------- -------- -------- --01----	01100000 11111110 10100110 01000000	5	61	-------- -------- -------- --------	x------- -------- -------- --------	7
62	-------- -------- -------- --------	-------- -------- -------- --------	6	62	-------- -------- -------- ---u----	00000011 00100101 10100111 00001111	10
63	-------- -------- nu------ --------	01101100 10101011 01110010 00010011	2	63	-------- -------- -------- ---0----	11000000 00110011 00110000 01100000	14
64	-------- -------- -------- --------			64	-------- -------- -------- --------		

Fig. 4. Starting point for the 48-step differential path, on which the preparation trick was applied and the last 8 bits of X_{37} are fixed in advance by choosing several bits of M_9. At this point the remaining free message words are M_9, M_0, M_6, M_{13}, M_1, M_7 and M_4, which will be used during the merging phase.

Fixing the first 5 chaining values ($X_{26}, X_{27}, X_{28}, X_{29}, X_{30}$ of the left branch and $Y_{21}, Y_{22}, Y_{23}, Y_{24}, Y_{25}$ of the right branch) is quite an easy task. Note that the two branches can be fixed independently at this stage. We used algorithms similar to the ones searching for a differential path: we just guess the unrestricted bits – from lower step to higher step, lower bit to higher bit and check if any inconsistency occurs. If both 0 and 1 selection of one bit lead to an inconsistency, we apply the backtracking in the search tree by one level and guess the same bit again. The guessing continues until all bits are fixed. If after a predefined number of backtracking events (chosen according to the performance of the search) no solutions are found, we can restart the whole search in order to avoid being trapped in a bad subspace with no solution at all.

Concerning the fixing of the message words, we used a different approach. Here, our search was applied word by word. Following this message words ordering M_{11}, M_{15}, M_8, M_3, M_{12}, M_{14}, M_{10}, M_2, M_5, M_9, M_0 and M_6, we guess the free bits, and some internal states values will directly be deduced by computing in both forward and backward directions from the already known internal state values in both branches. Note that the two branches are not independent anymore at this stage (since all message words are added several times in both branches), so it is important to check often for any inconsistency that could be detected. The backtracking and restarting options are also helpful here. We can use an extra trick to get a performance improvement of the search by pre-fixing the value of the word with the biggest number conditions in it (either message word or internal state word), and then deduce the value from all the words involved in this computation.

Our tool can find a starting point in a couple of minutes, with a program not really optimized. We will discuss about the complexity to generate the starting points in the next section.

Phase 3: merging both branches with M_1, M_4, M_7 and M_{13}. A starting point example is given in Figure 4. Our target is to use the remaining free message words M_1, M_4, M_7 and M_{13} to make sure that we have a perfect match on the values of the five initial chaining words of both branches, i.e. $X_i = Y_i$ for $i \in \{12, 13, 14, 15, 16\}$ (the indexes started at 12 because we are not attacking from the first step here). The merging consists of four phases and in order to ease the reading we marked the free message words with colors in each phase. Once their values are fixed, we use black color for them.

- Step 1: Use M_{13} to ensure $X_{16} = Y_{16}$. As one can see, the value of X_{16} is already fixed at this point. Now, observe the two backward step functions of Y_{17} and Y_{16}:

$$Y_{17}^{\lll 10} = (Y_{22} \boxminus Y_{18}^{\lll 10})^{\ggg 8} \boxminus \texttt{IFZ}(Y_{21}, Y_{20}, Y_{19}^{\lll 10}) \boxminus K^r_{21} \boxminus M_{13}$$
$$Y_{16}^{\lll 10} = (Y_{21} \boxminus Y_{17}^{\lll 10})^{\ggg 12} \boxminus \texttt{IFZ}(Y_{20}, Y_{19}, Y_{18}^{\lll 10}) \boxminus K^r_{20} \boxminus M_0$$

Incorporating the equation $X_{16} = Y_{16}$, we can direcly calculate the value of M_{13} from the known ones:

$$M_{13} = (X_{16}^{\lll 10} \boxplus \texttt{IFZ}(Y_{20}, Y_{19}, Y_{18}^{\lll 10}) \boxplus K^r_{20} \boxplus M_0)^{\lll 12}$$
$$\boxminus Y_{21} \boxplus (Y_{22} \boxminus Y_{18}^{\lll 10})^{\ggg 8} \boxminus \texttt{IFZ}(Y_{21}, Y_{20}, Y_{19}^{\lll 10}) \boxminus K^r_{21}.$$

- Step 2: Similarly, use M_1 and M_7 to ensure conditions $X_{15} = Y_{15}$ and $X_{14} = Y_{14}$. Observing the step functions:

$$X_{15}^{\lll 10} = (X_{20} \boxminus X_{16}^{\lll 10})^{\ggg 13} \boxminus \texttt{IFX}(X_{19}, X_{18}, X_{17}^{\lll 10}) \boxminus K^l_{19} \boxminus M_1$$
$$= Y_{15}^{\lll 10} = (Y_{20} \boxminus Y_{16}^{\lll 10})^{\ggg 7} \boxminus \texttt{IFZ}(Y_{19}, Y_{18}, Y_{17}^{\lll 10}) \boxminus K^r_{19} \boxminus M_7$$

$$X_{14}^{\lll 10} = (X_{19} \boxminus X_{15}^{\lll 10})^{\ggg 8} \boxminus \texttt{IFX}(X_{18}, X_{17}, X_{16}^{\lll 10}) \boxminus K^l_{18} \boxminus M_{13}$$
$$= Y_{14}^{\lll 10} = (Y_{19} \boxminus Y_{15}^{\lll 10})^{\ggg 15} \boxminus \texttt{IFZ}(Y_{18}, Y_{17}, Y_{16}^{\lll 10}) \boxminus K^r_{18} \boxminus M_3$$

and introducing notations for the constants, the equations above are simplified to

$$A \boxminus M_1 = B \boxminus M_7$$
$$(X_{19} \boxminus (A \boxminus M_1))^{\ggg 8} \boxminus D = (Y_{19} \boxminus (B \boxminus M_7))^{\ggg 15} \boxminus E$$

Let $X = (X_{19} \boxminus (A \boxminus M_1))^{\ggg 8}$, $C_0 = E \boxminus D$ and $C_1 = Y_{19} \boxminus X_{19}$. The above equations become one:

$$X \boxplus C_0 = (C_1 \boxplus X^{\lll 8})^{\ggg 15} \qquad (1)$$

where C_0 and C_1 are constants. The problem of finding the value of M_1 and M_7 is equivalent to solving this equation. We claim that we can solve this equation with 2^9 computations (see explanation in Appendix): we can solve this equation for all 2^{64} possible values of C_0 and C_1 and store the solutions (M_1 and M_7) in a big look-up table. Building and storing this table requires 2^{73} time and 2^{64} memory.

- Step 3: Use M_4 to ensure $X_{13} = Y_{13}$. After step 2, Y_{13} is already fixed. Thus we can use a simple calculation to get the value of M_4:

$$M_4 = (X_{18} \boxminus X_{14}^{\lll 10})^{\ggg 6} \boxminus \texttt{IFX}(X_{17}, X_{16}, X_{15}^{\lll 10}) \boxminus Y_{13}$$

- Step 4: The uncontrolled part of the merging. At this point, all freedom degrees have been used and the last equation on the internal state $X_{12} = Y_{12}$ will be fulfilled with a probability of 2^{-32}.

Uncontrolled probability. After the merging, steps 36-58 of the left branch and steps 29-58 of the right branch are still uncontrolled. Due to the difficulty of calculating the probability, we used experiments to evaluate these probabilities. Starting from a generated starting point, e.g. in Figure 4, we randomly choose values of message words $M_5, M_9, M_0, M_6, M_{13}, M_1, M_7$ and M_4. Then we compute forward to check if the differences are all canceled after the last injection of M_7. Note that if there are conditions on message words M_1, M_4, M_7 and M_{13}, they should be fulfilled probabilistically and included in the probability estimation, since the freedom degrees of these message words are used to match the initial internal states values. For the other free message words M_5, M_9, M_0 and M_6, we do not need to consider their conditions in the probability, because we can freely choose their values to fulfill these conditions.

We measured the probability of both branches separately. After applying the preparation trick, the uncontrolled probability of the left branch is $2^{-8.8}$. The uncontrolled probability of the right branch is $2^{-36.6}$. Moreover, during the merging phase, we could not control the value matching on the first IV word, and this adds another factor of 2^{-32}.

In total, the uncontrolled probability is $2^{-32} \cdot 2^{-8.8} \cdot 2^{-36.6} = 2^{-77.4}$. Since this probability is too low and already close to the birthday bound for `RIPEMD-160`, we are not able to afford the feed-forward tricks in steps 58-64.

Complexity evaluation. First we calculate the complexity to generate the starting points. Since the uncontrolled probability is $2^{-77.4}$, we need to generate $2^{77.4}$ starting points. However, we do not need to restart the generation from the beginning. Indeed, every time we need a new starting point, we can randomize M_6 to get a new one. Once all possible choices of M_6 have been used, we can still use freedom degrees of M_0, M_9 and M_5 to generate all the required starting points. Though there are many constraints on these four message words, luckily the number of conditions on M_6 is only two bits (one on X_{18} and one on X_{17}). We can randomly choose value for X_{18} fulfilling the known conditions and check if the one-bit condition on X_{17} is fulfilled. Thus, we can find a new starting point from a known one with a complexity of 4 step functions, which is equivalent to $4/(42 * 2) \approx 2^{-4.4}$ calls of the 42-step compression function of `RIPEMD-160`. For the other message words, we do not need to go into the details of the complexity, since the number of times we have to regenerate them is quite small and it is not the bottleneck of our attack complexity. From the reasoning above, we can conclude that the average complexity to generate a starting point is $2^{-4.4}$. The complexity of generating all the required starting points is then 2^{73}.

Now, we need to consider the complexity of the merging phase. In order to evaluate this cost, we implemented the merging of the last four initial internal states. The table lookup in second phase is estimated using a RAM access (since the table will be very bog). In total, our implementation of the merging takes about 145 cycles. The OPENSSL implementation of `RIPEMD-160` compression function on the same computer takes about 1040 cycles. Thus, 42 steps of the compression function takes about $1040 * 42/80 = 546$ cycles. Then we can say that our merging costs $145/546 \approx 2^{-1.9}$ calls of the 42-step compression function.

Finally, we can calculate the complexity of the semi-free start collision attack on 42-step `RIPEMD-160`: $2^{73} + 2^{77.4-1.9} \approx 2^{75.5}$.

5.2 First step semi-free-start collision

This section discusses about the merging phase of the two branches to get a semi-free-start collision attack on the the first 36 steps. The idea of the merging is similar to the merging for the 42-step attack and we describe it briefly.

We start with generating a starting point, and one example is provided in Figure 8. The path in the left branch has been satisfied until step 14, and the remaining uncontrolled probability amounts to $2^{-4.6}$. The path in the right branch has been fully satisfied. After that, there are free bits left in message words M_0,

M_2, M_5, M_7, M_9 and M_{14}. Next, we show these free bits are enough to generate semi-free-start collisions, and thus we only need to generate a single starting point.

The procedure of merging is detailed as below.

1. Set random values to M_9 and the free bits of M_7, and then compute until X_2 in the left branch.
2. Set $M_5 = M_5 \boxplus 1$ (initialize M_5 as 0), and compute until X_{-1} in the left branch. If M_5 becomes 0 again, goto Step 1.
3. Compute the values of M_2 and M_0 that make $Y_0 = X_0$ and $Y_{-1} = X_{-1}$.
4. Compute X_{-2} and Y_{-2}, and check if $X_{-2} = Y_{-2}$ holds. In case of $X_{-2} \neq Y_{-2}$, goto Step 2.
5. Compute X_{-3}, and then compute the value of M_{14} that makes $Y_{-3} = X_{-3}$. Check if the conditions on M_{14} are satisfied. If the conditions are not satisfied, goto step 1.
6. Compute X_{-4} and Y_{-4}, and check if $X_{-4} = Y_{-4}$ holds. In case of $X_{-4} \neq Y_{-4}$, goto Step 2.

Both $X_{-2} = Y_{-2}$ and $X_{-4} = Y_{-4}$ are satisfied with a probability 2^{-32}, and four bit conditions are set on M_{14}. Thus we have to try 2^{68} random values of M_7, M_9, and M_5 to succeed in merging the two branches once. Recall that the uncontrolled probability is $2^{-4.6}$. So we need to merge the two branches $2^{4.6}$ times. Thus, the total complexity of the attack is $2^{68+4.6} \times 16/72 \approx 2^{70.4}$.

6 Results

We give in Table 2 a comparison of our attacks to previous results on `RIPEMD-160`. Compared to the previous best semi-free-start collision attack on `RIPEMD-160` (36 middle steps), we have increased the number of attackable steps by 6 and proposed a 36-step semi-free-start collision attack that starts from the first step.

Table 2. Summary of known and new preimage and collision attacks on `RIPEMD-160` hash and compression function

Function	Size	Target	Attack Type	#Steps	Complexity	Ref.
`RIPEMD-160`	160	comp. function	preimage	31	2^{148}	[14]
`RIPEMD-160`	160	hash function	preimage	31	2^{155}	[14]
`RIPEMD-160`	160	comp. function	semi-free-start collision	36	low	[10]
`RIPEMD-160`	**160**	**comp. function**	**semi-free-start collision**	**42**	$2^{75.5}$	**new**
`RIPEMD-160`	**160**	**comp. function**	**semi-free-start collision**	**36**	$2^{70.4}$	**new**
`RIPEMD-160`	160	comp. function	non-randomness	48	low	[10]
`RIPEMD-160`	160	comp. function	non-randomness	51	2^{158}	[16]

Conclusion

In this article, we have proposed an improved cryptanalysis of the hash function `RIPEMD-160`, which is an ISO/IEC standard. We have found a 42-step semi-free-start collision attack on `RIPEMD-160` starting from the second step and a 36-step semi-free-start collision attack starting from the first step. Compared to previous results, we have two improvements. First the number of attacked steps is increased from 36 to 42, and secondly, for the same number of attacked steps, we propose an attack that starts from the first step. Moreover, our semi-free-start collision attacks give a positive answer to the open problem raised in [10], in which the authors were not able to find any non-linear differential path in the first step, due to the XOR function that makes the non-linear part search much harder.

Our 42-step semi-free-start attack is obtained from a 48-step differential path. Unfortunately, we couldn't add these extra 6 steps to our attack without reaching a complexity beyond the birthday bound (this extra part would be verified with probability $2^{-11.3}$). Future works might include improving the probabilistic part even further. If one can improve this part by a factor of about 2^7, a 48-step semi-free-start collision attack would then be obtained directly with our proposed differential path. Another possible improvement would be that if one can find a better non-linear differential path in the first round, it might be possible to merge both branches at the same time to a given IV and eventually obtain a hash function collision.

Acknowledgments

The authors would like to thank the anonymous referees for their helpful comments. Thomas Peyrin, Wang Lei and Wu Shuang are supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak reference. Submission to NIST (Round 3) (January 2011), available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html`
2. Bosselaers, A., Preneel, B. (eds.): Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evaluation RIPE-RACE 1040, LNCS, vol. 1007. Springer (1995)
3. Damgård, I.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO. LNCS, vol. 435, pp. 416–427. Springer (1989)
4. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT. LNCS, vol. 4284, pp. 1–20. Springer (2006)
5. Dobbertin, H.: RIPEMD with Two-Round Compress Function is Not Collision-Free. J. Cryptology 10(1), 51–70 (1997)
6. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A Strengthened Version of RIPEMD. In: Gollmann, D. (ed.) FSE. LNCS, vol. 1039, pp. 71–82. Springer (1996)
7. International Organization for Standardization: Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions. ISO/IEC 10118-3:2004 (2004)
8. Landelle, F., Peyrin, T.: Cryptanalysis of Full RIPEMD-128. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT. LNCS, vol. 7881, pp. 228–244. Springer (2013)
9. Leurent, G.: Analysis of Differential Attacks in ARX Constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT. LNCS, vol. 7658, pp. 226–243. Springer (2012)
10. Mendel, F., Nad, T., Scherz, S., Schläffer, M.: Differential Attacks on Reduced RIPEMD-160. In: Gollmann, D., Freiling, F.C. (eds.) ISC. LNCS, vol. 7483, pp. 23–38. Springer (2012)
11. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT. LNCS, vol. 7073, pp. 288–307. Springer (2011)
12. Mendel, F., Nad, T., Schläffer, M.: Improving Local Collisions: New Attacks on Reduced SHA-256. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT. LNCS, vol. 7881, pp. 262–278. Springer (2013)
13. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) CRYPTO. LNCS, vol. 435, pp. 428–446. Springer (1989)
14. Ohtahara, C., Sasaki, Y., Shimoyama, T.: Preimage Attacks on Step-Reduced RIPEMD-128 and RIPEMD-160. In: Lai, X., Yung, M., Lin, D. (eds.) Inscrypt. LNCS, vol. 6584, pp. 169–186. Springer (2010)
15. Rivest, R.L.: The MD4 Message-Digest Algorithm. IETF Request for Comments (RFC) 1320 (1992), available online at `http://www.ietf.org/rfc/rfc1320.html`
16. Sasaki, Y., Wang, L.: Distinguishers beyond Three Rounds of the RIPEMD-128/-160 Compression Functions. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS. LNCS, vol. 7341, pp. 275–292. Springer (2012)
17. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Cramer, R. (ed.) EUROCRYPT. LNCS, vol. 3494, pp. 1–18. Springer (2005)
18. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO. LNCS, vol. 3621, pp. 17–36. Springer (2005)
19. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT. LNCS, vol. 3494, pp. 19–35. Springer (2005)
20. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO. LNCS, vol. 3621, pp. 1–16. Springer (2005)

A Tables and constants for `RIPEMD-160` compression function

Table 3. Initialization values in `RIPEMD-160`

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

Table 4. Word permutations for the message expansion in `RIPEMD-160`

round j	$\pi^l_j(k)$																$\pi^r_j(k)$															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
1	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
2	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
3	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14
4	4	0	5	9	7	12	2	10	14	1	3	8	11	6	15	13	12	15	10	4	1	5	8	7	6	2	13	14	0	3	9	11

Table 5. Boolean functions and round constants in `RIPEMD-160`, with $\mathrm{XOR}(x, y, z) := x \oplus y \oplus z$, $\mathrm{IFX}(x, y, z) := (x \wedge y) \oplus (\bar{x} \wedge z)$, $\mathrm{IFZ}(x, y, z) := (x \wedge z) \oplus (y \wedge \bar{z})$, $\mathrm{ONX}(x, y, z) := x \oplus (y \vee \bar{z})$ and $\mathrm{ONZ}(x, y, z) := (x \vee \bar{y}) \oplus z$

round j	Φ^l_j	Φ^r_j	K^l_j	K^r_j
0	`XOR`	`ONX`	`0x00000000`	`0x50a28be6`
1	`IFX`	`IFZ`	`0x5a827999`	`0x5c4dd124`
2	`ONZ`	`ONZ`	`0x6ed9eba1`	`0x6d703ef3`
3	`IFZ`	`IFX`	`0x8f1bbcdc`	`0x7a6d76e9`
4	`ONX`	`XOR`	`0xa953fd4e`	`0x00000000`

Table 6. Rotation constants in `RIPEMD-160`

round j	$s^l_{16\cdot j+k}$																$s^r_{16\cdot j+k}$															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
1	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
2	11	13	6	7	14	9	13	15	14	8	13	6	5	12	7	5	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
3	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8
4	9	15	5	11	6	8	13	12	5	12	13	14	11	8	5	6	8	5	12	9	12	5	14	6	8	13	6	5	15	13	11	11

B Notations for the differential paths

(x, x^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
`?`	✓	✓	✓	✓
`-`	✓	-	-	✓
`x`	-	✓	✓	-
`0`	✓	-	-	-
`u`	-	✓	-	-
`n`	-	-	✓	-
`1`	-	-	-	✓
`#`	-	-	-	-

(x, x^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
`3`	✓	✓	-	-
`5`	✓	-	✓	-
`7`	✓	✓	✓	-
`A`	-	✓	-	✓
`B`	✓	✓	-	✓
`C`	-	-	✓	✓
`D`	✓	-	✓	✓
`E`	-	✓	✓	✓

Table 7. Notations used in [4] for a differential path: x represents a bit of the first message and x^* stands for the same bit of the second message.

C Solving $X \boxplus C_0 = (C_1 \boxplus X^{\lll 8})^{\ggg 15}$

Before solving this equation, we need to simplify it. First we consider the relation between $A^{\lll r} \boxplus B^{\lll r}$ and $(A \boxplus B)^{\lll r}$. Let $A = a_0||a_1$ and $B = b_0||b_1$, where A and B are n-bit values, a_0 and a_1 are r-bit values, b_0 and b_1 are $(n-r)$-bit values. We can write:

$$\begin{aligned} A \boxplus B &= (a_0||a_1) \boxplus (b_0||b_1) = (a_0 \boxplus b_0 \boxplus carry_1)_r||(a_1 \boxplus b_1)_{n-r} \\ A^{\lll r} \boxplus B^{\lll r} &= (a_1||a_0) \boxplus (b_1||b_0) = (a_1 \boxplus b_1 \boxplus carry_0)_{n-r}||(a_0 \boxplus b_0)_r \end{aligned}$$

where $carry_0 = (a_0 \boxplus b_0)^{\ggg r}$ and $carry_1 = (a_1 \boxplus b_1)^{\ggg (n-r)}$ are the carry bits which can only take the values of 0 or 1. Notation $(x)_k$ stands for the lower k bits of x. Thus we can write equation (1) as:

$$X \boxplus C_0 = C_1^{\ggg 15} \boxplus X^{\ggg 7} \boxplus y_1 \boxminus y_0 \cdot 2^{15}$$

where y_0 and y_1 are the carry bits calculated from the higher 17 and lower 15 bits of $C_1 \boxplus X^{\lll 8}$, which can be determined by the values of X.

Let $C = C_0 \boxminus C_1^{\lll 15} \boxplus y_0 \cdot 2^{15} \boxminus y_1$, the equation is finally simplified to

$$X \boxplus C = X^{\ggg 7} \tag{2}$$

Note that there are four possible values for C.

This equation can be solved with a Guess-and-Determine approach. Let $X = x_4||x_3||x_2||x_1||x_0$ and $C = c_4||c_3||c_2||c_1||c_0$, where x_i and c_i are 7-bit values for $i \in \{0,1,2,3\}$ and x_4 and c_4 are 4-bit values. Then equation (2) is shown in details in Figure 5. Guess the value of x_0 and we can deduce the value of

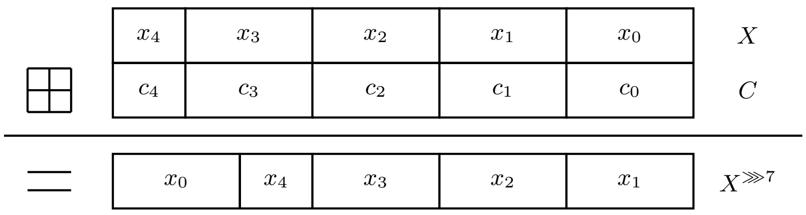

Fig. 5. Demonstration of solving equation (2)

x_1, x_2, x_3 and x_4 one by one and check if the original equation holds. One guess succeeds with probability of 2^{-7} and it takes 2^7 computations to solve equation (2) Considering the guessing of y_0 and y_1, the total complexity to solve equation (1) is 2^9.

D The differential paths and starting points

```
Step Xi                                                                                       Step Yi
 12  -------- -------- -------- --------                                                       12  -------- -------- -------- --------
 13  -------- -------- -------- --------                                                       13  -------- -------- -------- --------
 14  -------- -------- -------- --------                                                       14  -------- -------- -------- --------
 15  -------- -------- -------- --------  Wi^l                                       Πi^l      15  -------- -------- -------- --------  Wi^r                                       Πi^r
 16  -------- -------- -------- --------  x------- -------- -------- --------   7    16  -------- -------- -------- --------  -------- -------- -------- --------   6
 17  -------- -------- -------- -n------  -------- -------- -------- --------   4    17  -------- -------- -------- --------  -------- -------- -------- --------  11
 18  -------- -------- -------- -0------  -------- -------- -------- --------  13    18  --01---- 01------ -------- --------  -------- -------- -------- --------   3
 19  -------- -------1 -------- -1------  -------- -------- -------- --------   1    19  ---1---- -------- -------1 --------  x------- -------- -------- --------   7
 20  -------- -------0 -------- --------  -------- -------- -------- --------  10    20  11------ -----100 -------1 -u---001  -------- -------- -------- --------   0
 21  -------- -------n -------- --------  -------- -------- -------- --------   6    21  11-01--- -----u10 1-----11 1----111  -------- -------- -------- --------  13
 22  -------- -------0 -------- --------  -------- -------- -------- --------  15    22  ---unn-- --0--1-1 0--nuuuu 01----0u  -------- -------- -------- --------   5
 23  -----1-- -------0 -------- --------  -------- -------- -------- --------   3    23  ---00un- 011-uuu- --0----- -un0----  -------- -------- -------- --------  10
 24  1----1-- -------- -------- --------  -------- -------- -------- --------  12    24  u11n1--- ----11-1 1-0-u--- -1nn0010  -------- -------- -------- --------  14
 25  -----n-- ---1--1- ------0- --------  -------- -------- -------- --------   0    25  0-u110-- n-0----1 11--u10- -1-0n1nu  -------- -------- -------- --------  15
 26  001-00-- ------un nnnn--nu u1--0---  -------- -------- -------- --------   9    26  ----1--1 -11---1- 00-n1nnn -nnnu---  -------- -------- -------- --------   8
 27  -n--n-un -u--0100 0000--11 1010----  -------- -------- -------- --------   5    27  1---1u-- 0-uu--1n 1---00-- ---0-unn  -------- -------- -------- --------  12
 28  1010001- ----00nu --n0-nu- u-un0110  -------- -------- -------- --------   2    28  ----u--- --n1--00 0--011-- --n0--n1  -------- -------- -------- --------   4
 29  0-nun-0- 11-u--0u -1010101 1-u0nnu-  -------- -------- -------- --------  14    29  -------- ---u--1- 0u-11u-- ---0---0  -------- -------- -------- --------   9
 30  -000-111 uu010n10 1u-1nu1n -00nn000  -------- -------- -------- --------  11    30  -0--n--- --n----- 1---1--- --------  -------- -------- -------- --------   1
 31  -n---1-u 0uu1-0-1 000011n0 0001n---  -------- -------- -------- --------   8    31  -1------ --1----- -----u-- --1-----  -------- -------- -------- --------   2
 32  -10----0 -010---u -1--1u10 ----01--  -------- -------- -------- --------   3    32  -------- --u----- -----1-- --0-----  -------- -------- -------- --------  15
 33  -n-----u --u----1 -----1-- 1----0--  -------- -------- -------- --------  10    33  -------- -------- -------- --------  -------- -------- -------- --------   5
 34  -11----0 --0----- ----1--- u-------  -------- -------- -------- --------  14    34  -------- -------- -------- --------  -------- -------- -------- --------   1
 35  -------n --n----- -------- 1--1----  -------- -------- -------- --------   4    35  -------- -------- -------- --------  -------- -------- -------- --------   3
 36  -------1 --0----- ----0--- u----1--  -------- -------- -------- --------   9    36  -------- -------- -------- --------  x------- -------- -------- --------   7
 37  -------- --n---1- -------- 1----0--  -------- -------- -------- --------  15    37  -------- -------- -------- --------  -------- -------- -------- --------  14
 38  -------- --0---0- -------- --------  -------- -------- -------- --------   8    38  -------- -------- -------- --------  -------- -------- -------- --------   6
 39  -------- -------- -------- --------  -------- -------- -------- --------   1    39  -------- -------- -------- --------  -------- -------- -------- --------   9
 40  -------- -------- -------- --------  -------- -------- -------- --------   2    40  -------- -------- -------- --------  -------- -------- -------- --------  11
 41  -------- -------- -------- --------  x------- -------- -------- --------   7    41  -------- -------- -------- --------  -------- -------- -------- --------   8
 42  -------- -------- -------- --------  -------- -------- -------- --------   0    42  -------- -------- -------- --------  -------- -------- -------- --------  12
 43  -------- -------- -------- --------  -------- -------- -------- --------   6    43  -------- -------- -------- --------  -------- -------- -------- --------   2
 44  -------- -------- -------- --------  -------- -------- -------- --------  13    44  -------- -------- -------- --------  -------- -------- -------- --------  10
 45  -------- -------- -------- --------  -------- -------- -------- --------  11    45  -------- -------- -------- --------  -------- -------- -------- --------   0
 46  -------- -------- -------- --------  -------- -------- -------- --------   5    46  -------- -------- -------- --------  -------- -------- -------- --------   4
 47  -------- -------- -------- --------  -------- -------- -------- --------  12    47  -------- -------- -------- --------  -------- -------- -------- --------  13
 48  -------- -------- -------- --------  -------- -------- -------- --------   1    48  -------- -------- -------- --------  -------- -------- -------- --------   8
 49  -------- -------- -------- --------  -------- -------- -------- --------   9    49  -------- -------- -------- --------  -------- -------- -------- --------   6
 50  -------- -------- -------- --------  -------- -------- -------- --------  11    50  -------- -------- -------- --------  -------- -------- -------- --------   4
 51  -------- -------- -------- --------  -------- -------- -------- --------  10    51  -------- -------- -------- --------  -------- -------- -------- --------   1
 52  -------- -------- -------- --------  -------- -------- -------- --------   0    52  -------- -------- -------- --------  -------- -------- -------- --------   3
 53  -------- -------- -------- --------  -------- -------- -------- --------   8    53  -------- -------- -------- --------  -------- -------- -------- --------  11
 54  -------- -------- -------- --------  -------- -------- -------- --------  12    54  -------- -------- -------- --------  -------- -------- -------- --------  15
 55  -------- -------- -------- --------  -------- -------- -------- --------   4    55  -------- -------- -------- --------  -------- -------- -------- --------   0
 56  -------- -------- -------- --------  -------- -------- -------- --------  13    56  -------- -------- -------- --------  -------- -------- -------- --------   5
 57  -----0-- -------- -------- --------  -------- -------- -------- --------   3    57  -------- -------- -------- --------  -------- -------- -------- --------  12
 58  -----1-- -------- -------- --------  x------- -------- -------- --------   7    58  -------- -------- -------- --------  -------- -------- -------- --------   2
 59  -------- -------- -------- ---n----  -------- -------- -------- --------  15    59  -------- -------- -------- --------  -------- -------- -------- --------  13
 60  -------- -------- -------- --------  -------- -------- -------- --------  14    60  -------- -------- -------- --------  -------- -------- -------- --------   9
 61  -------- -------- -------- --01----  -------- -------- -------- --------   5    61  -------- -------- -------- --------  x------- -------- -------- --------   7
 62  -------- -------- -------- --------  -------- -------- -------- --------   6    62  -------- -------- -------- ---u----  -------- -------- -------- --------  10
 63  -------- -------- nu------ --------  -------- -------- -------- --------   2    63  -------- -------- -------- ---0----  -------- -------- -------- --------  14
 64  -------- -------- -------- --------                                                       64  -------- -------- -------- --------
```

Fig. 6. The 48-step differential path

Step	X_i	W_i^l	Π_i^l	Step	Y_i	W_i^r	Π_i^r
-4	-------- -------- -------- --------			-4	-------- -------- -------- --------		
-3	-------- -------- -------- --------			-3	-------- -------- -------- --------		
-2	-------- -------- -------- --------			-2	-------- -------- -------- --------		
-1	-------- -------- -------- --------			-1	-------- -------- -------- --------		
00	-------- -------- -------- --------	-------- -------- -------- --------	0	00	-------- -------- -------- --------	-------- -------- -------- --------	5
01	-------- -------- -------- --------	-------- -------- -------- --------	1	01	--1----- -------- -1------ ----1---	-------- -------- -------- --------	14
02	-------- -------- -------- --------	-------- -------- -------- --------	2	02	--0---10 -------- -10----- 0---0---	-n------ -------- n------- ---n----	7
03	-------- -------- -------- --------	-------- -------- -------- --------	3	03	------1n -------- -1n----0 n---0---	-------- -------- -------- --------	0
04	-------- -------- -------- --------	-------- -------- -------- --------	4	04	100----n 0-11-n0- -------1 10--n0--	-------- -------- -------- --------	9
05	-------- -------- -------- --------	-------- -------- -------- --------	5	05	n1-1---0 0----00n 0-1---00 0---1101	-------- -------- -------- --------	2
06	-------- -------- -------- --------	-------- -------- ---11--- --------	6	06	n--10001 11110un- --uuuuuu uu11-1u-	-------- -------- -------- --------	11
07	-------- -------- -------- --------	-n------ -------- n------- ---n----	7	07	uuu00un- n1u011nn 00000110 1-01-n00	-------- -------- -------- --------	4
08	------n- uuuuuuuu uuu-nuuu -uuuuuuu	-------- -------- -------- --------	8	08	11u0uu-- 1-u1u0n1 nn-nu-10 0---000u	-------- -------- -------- --------	13
09	--uun-nn -n---nnu nnuu---- ---n--nn	-------- -------- -------- --------	9	09	0-uu011- -01-u000 11000n-n n01--111	-------- -------- ---11--- --------	6
10	-----n-- unuun-u- u-----nn ----u---	-------- -------- -------- --------	10	10	--1u1nnn nnn01100 10-0-0-0 0100u1-0	-------- -------- -------- --------	15
11	--n---nu uu--nu-- un-n---- --------	-------- -------- -------- --------	11	11	0--u-n1n ---1--1u n--n-n-- 100001n-	-------- -------- -------- --------	8
12	-----u-- --n-nnnn nnnnnnnn nnn-----	-------- -------- -------- --------	12	12	10-110-- --n0---0 u1--1--- -0-111-n	-------- -------- -------- --------	1
13	-------- -------- -------- --------	-------- -------- -------- --------	13	13	1u--0n-u -------- ---1-1u- -10---1-	-------- -------- -------- --------	10
14	-------- -------- -------- --------	-------- -------- -------- --------	14	14	----1--- -n-00--0 ----n111 --n1-0--	-------- -------- -------- --------	3
15	-------- -------- -------- --------	-------- -------- -------- --------	15	15	-----u-- -1-u0--- ----1--- --u--1--	-------- -------- -------- --------	12
16	-------- -------- -------- --------	-n------ -------- n------- ---n----	7	16	-------- -------- -------- ---0----	-------- -------- ---11--- --------	6
17	-------- -------- -------- --------	-------- -------- -------- --------	4	17	-------- -------- -------- ---1----	-------- -------- -------- --------	11
18	-------- -------- -------- --------	-------- -------- -------- --------	13	18	-------- -------- -------- --------	-------- -------- -------- --------	3
19	-------- -------- -------- --------	-------- -------- -------- --------	1	19	-------- -------- -------- --------	-n------ -------- n------- ---n----	7
20	-------- -------- -------- --------	-------- -------- -------- --------	10	20	-------- -------- -------- --------	-------- -------- -------- --------	0
21	-------- -------- -------- --------	-------- -------- ---11--- --------	6	21	-------- -------- -------- --------	-------- -------- -------- --------	13
22	-------- -------- -------- --------	-------- -------- -------- --------	15	22	-------- -------- -------- --------	-------- -------- -------- --------	5
23	-------- -------- -------- --------	-------- -------- -------- --------	3	23	-------- -------- -------- --------	-------- -------- -------- --------	10
24	-------- -------- -------- --------	-------- -------- -------- --------	12	24	-------- -------- -------- --------	-------- -------- -------- --------	14
25	-------- -------- -------- --------	-------- -------- -------- --------	0	25	-------- -------- -------- --------	-------- -------- -------- --------	15
26	-------- -------- -------- --------	-------- -------- -------- --------	9	26	-------- -------- -------- --------	-------- -------- -------- --------	8
27	-------- -------- -------- --------	-------- -------- -------- --------	5	27	-------- -------- -------- --------	-------- -------- -------- --------	12
28	-------- -------- -------- --------	-------- -------- -------- --------	2	28	-------- -------- -------- --------	-------- -------- -------- --------	4
29	-------- -------- -------- --------	-------- -------- -------- --------	14	29	-------- -------- -------- --------	-------- -------- -------- --------	9
30	-------- -------- -------- --------	-------- -------- -------- --------	11	30	-------- -------- -------- --------	-------- -------- -------- --------	1
31	-------- -------- -------- --------	-------- -------- -------- --------	8	31	-------- -------- -------- --------	-------- -------- -------- --------	2
32	-------- -------- -------- --------	-------- -------- -------- --------	3	32	-------- -------- -------- --------	-------- -------- -------- --------	15
33	-------- -------- -------- --------	-------- -------- -------- --------	10	33	-------- -------- -------- --------	-------- -------- -------- --------	5
34	-------- -------- -------- --------	-------- -------- -------- --------	14	34	-------- -------- -------- --------	-------- -------- -------- --------	1
35	-------- -------- -------- --------	-------- -------- -------- --------	4	35	-------- -------- -------- --------	-------- -------- -------- --------	3
36	-------- -------- -------- --------			36	-------- -------- -------- --------		

Step	X_i	W_i^l	Π_i^l
-4	-------- -------- -------- --------		
-3	-------- -------- -------- --------		
-2	-------- -------- -------- --------		
-1	-------- -------- -------- --------		
00	-------- -------- -------- --------	-------- -------- -------- --------	0
01	-------- -------- -------- --------	01101110 01011111 10010010 11001000	1
02	-------- -------- -------- --------	-------- -------- -------- --------	2
03	-------- -------- -------- --------	10000000 00011010 10011001 10100011	3
04	-------- --100111 1------- --------	00010101 10111110 01111000 11000100	4
05	011101-- -------- -------- -------0	-------- -------- -------- --------	5
06	00011010 10010010 01101000 10101101	10111110 10000001 11011011 01100010	6
07	11110001 00001111 00000000 11110011	-n------ -------- n------- ---n----	7
08	100111n1 uuuuuuuu uuu1nuuu 1uuuuuuu	00111001 01010101 01101010 01011100	8
09	11uun0nn 1n111nnu nnuu1001 110n10nn	-------- -------- -----100 101-----	9
10	01001n00 unuun0u0 u11100nn 0011u111	01001110 00100010 01011011 11011100	10
11	10n011nu uu11nu01 un0n0111 10101010	10100101 10011100 00101010 11000101	11
12	00100u11 01n1nnnn nnnnnnnn nnn10111	01001111 00111010 10101100 11001010	12
13	00000100 11110000 10011011 11101110	10100001 10100010 10011010 10001000	13
14	11101111 10111001 00001001 01100000	1------- -1------ -------- --------	14
15	1011---- -------- -------0 --------	10000011 11010101 00000100 11011110	15
16	00------ -------- -------- 100-----	-n------ -------- n------- ---n----	7
17	-------- -------- -------- --------	00010101 10111110 01111000 11000100	4
18	-------- -------- -------- --------	10100001 10100010 10011010 10001000	13
19	-------- -------- -------- --------	01101110 01011111 10010010 11001000	1
20	-------- -------- -------- --------	01001110 00100010 01011011 11011100	10
21	-------- -------- -------- --------	10111110 10000001 11011011 01100010	6
22	-------- -------- -------- --------	10000011 11010101 00000100 11011110	15
23	-------- -------- -------- --------	10000000 00011010 10011001 10100011	3
24	-------- -------- -------- --------	01001111 00111010 10101100 11001010	12
25	-------- -------- -------- --------	-------- -------- -------- --------	0
26	-------- -------- -------- --------	-------- -------- -----100 101-----	9
27	-------- -------- -------- --------	-------- -------- -------- --------	5
28	-------- -------- -------- --------	-------- -------- -------- --------	2
29	-------- -------- -------- --------	1------- -1------ -------- --------	14
30	-------- -------- -------- --------	10100101 10011100 00101010 11000101	11
31	-------- -------- -------- --------	00111001 01010101 01101010 01011100	8
32	-------- -------- -------- --------	10000000 00011010 10011001 10100011	3
33	-------- -------- -------- --------	01001110 00100010 01011011 11011100	10
34	-------- -------- -------- --------	1------- -1------ -------- --------	14
35	-------- -------- -------- --------	00010101 10111110 01111000 11000100	4
36	-------- -------- -------- --------		

Step	Y_i	W_i^r	Π_i^r
-4	-------- -------- -------- --------		
-3	-------- -------- -------- --------		
-2	-------- -------- -------- --------		
-1	-------- -------- -------- --------		
00	-------- -------- -------- --------	-------- -------- -------- --------	5
01	--1----- -------- -1------ ----1---	1------- -1------ -------- --------	14
02	11011010 11010111 01010001 01010100	-n------ -------- n------- ---n----	7
03	1001001n 00111111 11n10100 n0010101	-------- -------- -------- --------	0
04	1000100n 01111n01 11000011 1010n000	-------- -------- -----100 101-----	9
05	n1110100 0101100n 01111000 00001101	-------- -------- -------- --------	2
06	n1110001 11110un0 10uuuuuu uu1111u0	10100101 10011100 00101010 11000101	11
07	uuu00un1 n1u011nn 00000110 10010n00	00010101 10111110 01111000 11000100	4
08	11u0uu11 10u1u0n1 nn0nu110 0011000u	10100001 10100010 10011010 10001000	13
09	01uu0110 0011u000 11000n1n n0111111	10111110 10000001 11011011 01100010	6
10	001u1nnn nnn01100 10101000 0100u100	10000011 11010101 00000100 11011110	15
11	000u1n1n 1111011u n00n0n10 100001n0	00111001 01010101 01101010 01011100	8
12	10111011 01n01110 u1011010 1011111n	01101110 01011111 10010010 11001000	1
13	1u100n0u 11101100 001111u1 11010010	01001110 00100010 01011011 11011100	10
14	10011100 0n100010 1110n111 11n11000	10000000 00011010 10011001 10100011	3
15	10111u10 111u0001 01001000 00u11110	01001111 00111010 10101100 11001010	12
16	10101010 00101101 00110011 01100100	10111110 10000001 11011011 01100010	6
17	00000000 10100011 00001101 10110000	10100101 10011100 00101010 11000101	11
18	01010101 11000011 11111110 00010010	10000000 00011010 10011001 10100011	3
19	01111011 01000100 10111001 01110000	-n------ -------- n------- ---n----	7
20	-------- -------- -------- --------	-------- -------- -------- --------	0
21	-------- -------- -------- --------	10100001 10100010 10011010 10001000	13
22	-------- -------- -------- --------	-------- -------- -------- --------	5
23	-------- -------- -------- --------	01001110 00100010 01011011 11011100	10
24	-------- -------- -------- --------	1------- -1------ -------- --------	14
25	-------- -------- -------- --------	10000011 11010101 00000100 11011110	15
26	-------- -------- -------- --------	00111001 01010101 01101010 01011100	8
27	-------- -------- -------- --------	01001111 00111010 10101100 11001010	12
28	-------- -------- -------- --------	00010101 10111110 01111000 11000100	4
29	-------- -------- -------- --------	-------- -------- -----100 101-----	9
30	-------- -------- -------- --------	01101110 01011111 10010010 11001000	1
31	-------- -------- -------- --------	-------- -------- -------- --------	2
32	-------- -------- -------- --------	10000011 11010101 00000100 11011110	15
33	-------- -------- -------- --------	-------- -------- -------- --------	5
34	-------- -------- -------- --------	01101110 01011111 10010010 11001000	1
35	-------- -------- -------- --------	10000000 00011010 10011001 10100011	3
36	-------- -------- -------- --------		