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

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

[1]Tokyo Institute of Technology, Tokyo, Japan
[2]University of Hyogo, Hyogo, Japan
[3]East China Normal University, Shanghai, China
[4]State Key Laboratory of Cryptology, Beijing, China
[5]Indian Institute of Technology Madras, Chennai, India
[6]FHNW, Windisch, Switzerland
[7]NICT, Tokyo, Japan

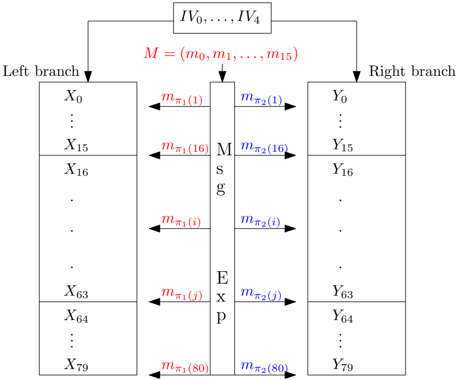
EUROCRYPT 2023

Overview

1. Background
   - RIPEMD-160
   - Difficulty to Analyze RIPEMD-160
   - Finding Trails
2. Our MILP-based Method
   - Problem Analysis
   - Modelling
   - Application
3. Summary and Open Questions

RIPEMD-160

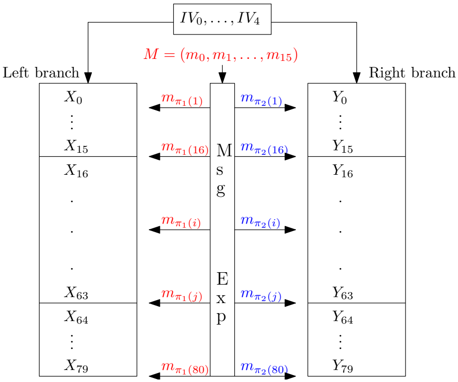
- FSE 1996 by Dobbertin et al.
- Strengthens MD5 (double branches, complex round function)
- Bitcoin address (with SHA-256)
- ISO/IEC standard



Round Function of RIPEMD-160

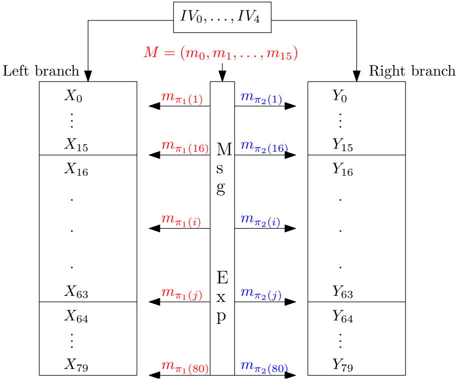
- Round function (left branch as an example)

$$\begin{aligned} Q_i &= X_{i-5} \lll 10 \boxplus F_i(X_{i-1}, X_{i-2}, X_{i-3} \lll 10) \boxplus m_{\pi(i)} \boxplus K_i, \\ X_i &= X_{i-4} \lll 10 \boxplus Q_i \lll s_i. \end{aligned}$$



Difficulty to Analyze RIPEMD-160

- Difficulty to analyze RIPEMD-160
  - Finding valid differential trails
  - Finding conforming message pairs
  - Constructing better local collisions



Finding Trails for RIPEMD-160

- Finding valid differential trails
  - Bit conditions by the Boolean functions
  - Extra conditions on Q_i

$$(Q_i \boxplus \alpha_i) \lll s_i = Q_i \lll s_i \boxplus \beta_i,$$

where α_i, β_i are constants derived from the differential trail.

Finding Trails for RIPEMD-160

- Where are the contradictions?
  - Boolean functions[1]:

$$F_{i+1}(X_i, X_{i-1}, X_{i-2} \lll 10), \quad (1)$$
$$F_{i+2}(X_{i+1}, X_i, X_{i-1} \lll 10), \quad (2)$$
$$F_{i+3}(X_{i+2}, X_{i+1}, X_i \lll 10). \quad (3)$$

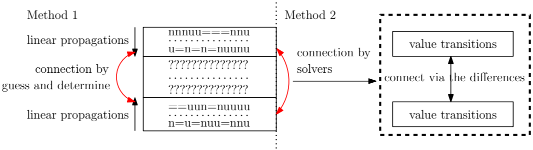
  - Extra conditions on Q_i

$$(Q_i \boxplus \alpha_i) \lll s_i = Q_i \lll s_i \boxplus \beta_i, \quad (4)$$
$$X_i = Q_i \lll s_i \boxplus X_{i-4} \lll 10 \quad (5)$$

[1] e.g., $F(x, y, z) = ONX(x, y, z) = (x \vee \overline{y}) \oplus z$.

Finding Trails for RIPEMD-160

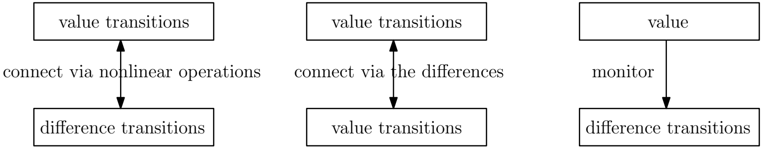
- Finding differential trails: guess-and-determine technique
  - Linearly propagate the difference: sparse part
  - Connect two sparse parts: dense part (difficult!!!)



- Not open-sourced (Method 1)
  1. But, we can still access the tools by contacting the authors.
  2. But, can we have an alternative choice (maybe a different approach with new potential?) e.g. MILP/SAT/SMT solvers?
  3. I want to have my own tool (personal interest)

Finding Trails for RIPEMD-160

- Our method (and comparison with other methods)
  - left: CRYPTO 2020 (Gimli)
  - middle: SAT 2006, CRYPTO 2017 (MD4, SHA-1)
  - right: ours



- Highlight the difference
  - Encode the difference transitions into the model
  - Remove the expensive value transitions
  - A light method (monitoring variables) to detect contradictions

Our Method

- Analysis of the target problem
  - How to describe the following difference transitions[2]?

$$(\nabla d_i, \nabla d_{i+1}, \nabla d_{i+2}, \nabla d_{i+3}, \nabla d_{i+4}, \nabla m) \to \nabla d_{i+5}$$

where

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

[2] ∇a: signed difference, δa: modular difference

Our Method

- Equivalent problem (bit-level)
  - How to describe the following difference transitions?

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

where

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

Our Method

- Introduce intermediate variables

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

  - Note 1: basically, we do not need b_0 because $\nabla b_0 = \nabla m$.
  - Note 2: m is a free variable

Our Method

- Main Idea:
  - Deterministically compute the signed difference transitions for $b_0 = m \boxplus c$, $b_2 = b_0 \boxplus b_1$ and $b_3 = b_2 \boxplus a_0$. Specifically, for each given $(\nabla x, \nabla y)$, uniquely compute one ∇z such that $\delta z = \delta x \boxplus \delta y$, even though there are many such possible ∇z.
  - Compute the signed difference transitions for $b_1 = F(a_4, a_3, a_2)$, where F is a boolean function.
  - Compute a possible value of ∇b_5 for $b_4 = b_3 \lll s$ and $b_5 = a_1 \boxplus b_4$.
  - Expand ∇b_5 to get all possible ∇a_5 such that the modular differences satisfy $\delta b_5 = \delta a_5$.

Our Method

- The problems to address:
  - Model $\nabla z = \nabla x \boxplus \nabla y$ where we only need to ensure $\delta z = \delta x \boxplus \delta y$.
  - Model $\nabla b_1 = F(\nabla a_4, \nabla a_3, \nabla a_2)$.
  - Model how to expand ∇b_5 to get all possible ∇a_5 such that the modular differences satisfy $\delta b_5 = \delta a_5$.

Describing the Signed Difference

- Three status: $\{\mathtt{n}, \mathtt{u}, \mathtt{=}\}$
  - Method 1: use one ternary variable (inefficient for F^3)

$$v \in \{-1, 0, 1\}$$

  - Method 2: use two binary variables (v, d)

$$(v, d) \in \{(0,1), (1,1), (0,0)\},$$

    where $(1,1)$ is not allowed.

[3] e.g., $F(x, y, z) = ONX(x, y, z) = (x \vee \overline{y}) \oplus z$.

Modelling the Modular Addition

- Target: $\nabla z = \nabla x \boxplus \nabla y$
  - Use an intermediate variable c to represent the carry
  - No branches

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

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

Modelling the Modular Addition

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

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

Linear inequalities (with LogicFriday):

$$\mathcal{H}_{\mathtt{ADD}} \cdot V_{\mathtt{ADD}}^{T} \geq \mathcal{C}_{\mathtt{ADD}},$$

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

Modelling the Modular Addition

$$
\mathcal{H}_{\mathtt{ADD}}|\mathcal{C}_{\mathtt{ADD}} = \left[\begin{array}{cccccccccc|c}
0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\
0 & -1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 0 & -2 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \\
1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & -1 \\
0 & 0 & 1 & -1 & 1 & 0 & -1 & 0 & 0 & 0 & -1 \\
1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & -1 \\
0 & -1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 0 & -2 \\
0 & -1 & -1 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -2 \\
0 & 1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & -2 \\
0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \\
0 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & -1 \\
1 & -1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & -1 \\
1 & 0 & 1 & 0 & 1 & -1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & -1 \\
-1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -1 \\
0 & 1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & -1 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & -1 \\
-1 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & -1 \\
0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\
-1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & -2
\end{array}\right],
$$

Modelling the Expansion

- Get all $\nabla\xi$ from ∇z such that $\delta\xi = \delta z$:
  - Use an intermediate variable c to represent the carry
  - Branches!!! (tree structure)

Table: Propagation rules for $(\nabla z[i], \nabla c[i]) \rightarrow (\nabla\xi[i], \nabla c[i+1])$

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

Modelling the Boolean Function

- Example: $\nabla w = ONX(\nabla x, \nabla y, \nabla z)$ where

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

  - List all possible cases!!!

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

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

Detecting Contradictions in F

- Example: $\nabla w = ONX(\nabla x, \nabla y, \nabla z)$
  - Introduce variables (x, y, z) representing the values into the model
  - Each possible $(\nabla x, \nabla y, \nabla z, \nabla w)$ will impose certain bit conditions on (x, y, z).
  - List all possible cases.

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

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

Detecting Contradictions in Q

- How to avoid contradictions between a_5 and a_1?
  - Modelling the following 3 conditions is sufficient ($\delta q = \delta b_4$)

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

Other Details

- Many other minor details (see the paper)
  - Model $b_4 = b_3 \lll s$ and $b_5 = a_1 \boxplus b_4$
  - A different way to model the expansion
  - Many optional parameters to control the searching strategies (e.g. where to detect the contradictions)

The Differential Trail for 36-Round RIPEMD-160

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

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

The Differential Trail for 36-Round RIPEMD-160

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

i	∇X_i	$\pi_l(i)$	i	∇Y_i	$\pi_r(i)$
-5	10100101010010101101011111001000		-5	10100101010010101101011111001000	
-4	11101110001000000011110110000011		-4	11101110001000000011110110000011	
-3	11111010101100010100111101100010		-3	11111010101100010100111101100010	
-2	00011100010010000100111100010010		-2	00011100010010000100111100010010	
-1	00111011110101101010010000011111		-1	00111011110101101010010000011111	
0	nuuuuuuuuuuuuuuuu1nuuuuuuuuuuuu1	0	0	10111000110000010010000010111011	5
1	n010u0u11n1un0100uuu1u1nn000u1uu	1	1	11111010100111011001011011100001	14
2	1nun1u0n10n00nn10u10uun01nnu1un1	2	2	01101001000000101010110011010110	7
3	1011nu11111110010nu1110011100011	3	3	00100111110111101n001101000010n1	0
4	nnnnnnnn000unnnnnnnnnnnnnnnunnnn	4	4	0101n000010010011n10101010110010	9
5	11111110100000100000101011100010	5	5	010010100100011101000n0001110010	2
6	00110001111011101111011010111010	6	6	10001nuunnnnnnnnnnnnnn0un1101110	11
7	01101000000111101011001111000001	7	7	0u0n1uun00n10nu01nnun0nuuuuuuuuu	4
8	10011000010111000010010111111011	8	8	n1un0nuuuu110u0un0unnnn1nn0nunuu	13
9	01111100011111000110101010010100	9	9	110010u1000n00u01uu010n1010n100n	6
10	10000100100000000111100110011011	10	10	u100u0110uu0u0111001101u10100111	15
11	00000110100010001011100111011111	11	11	111n110011110n000110110100n00010	8
12	10100111100100110101011100111110	12	12	0000010100101011100010n0111001111	1
13	10011000000000010001000010001011	13	13	10001011u11111010n010001u011u010	10
14	00011010101011010110100101110110	14	14	01u000011n010n01111001101n010010	3
15	00100001000111110011110010111100	15	15	101110u110100001101uu110010n0010	12
16	================================	7	16	11010010110011 1n0101101000001011	6
17	================================	4	17	0001101u110111011110111101011110	11
18	================================	13	18	10100001110000011100110111111110	3
19	================================	1	19	0111011100n1111001010011n0111111	7
20	================================	10	20	010nu010011000010100110010100101	0
21	=======u==================u=====	6	21	00001110010001011100110110011010	13
22	=======0==================0=====	15	22	00011011111010011010u11101001000	5
23	================1============1==	3	23	n===1=======nu===1==============	10
24	================================	12	24	=======u==============0===u=====	14
25	================================	0	25	======================1======0==	15
26	================================	9	26	=============================1==	8
27	================================	5	27	================================	12
28	================================	2	28	================================	4
29	================================	14	29	================================	9
30	================================	11	30	================================	1
31	================================	8	31	================================	2
32	================================	3	32	================================	15
33	================================	10	33	================================	5
34	================================	14	34	================================	1
35	================================	4	35	================================	3
m_0	1111101nuu1111010101110111 00n000		m_8	10001000100110111010111000011100	
m_1	11100010100010101000011010001010		m_9	01101001001110 1nu110001110001n01	
m_2	01110100001111011001110110000001		m_{10}	10011100001110000100101111001101	
m_3	01100101011111001001111010101101		m_{11}	00100100001110000001100010011 1110	
m_4	01011010111100011001011001010001		m_{12}	10000110011010100101011001001110	
m_5	10000010000110010100000110001110		m_{13}	00100011101101110011111000101001	
m_6	00un10111111111 0u01110000000100101		m_{14}	10100010111011001101010111011101	
m_7	11011100001000100110001010001000		m_{15}	11010001001001110100011001001011	

New Progress

Questions quickly arising:

- Is the technique really useful or efficient? (Other targets like SHA-2?)
- Can it beat ad-hoc dedicated tools? (evidence?)

Our new work[4]:

- The first practical collisions for 40-round RIPEMD-160
- The first practical SFS collisions for 39-round SHA-256 (previous record: 38 rounds at EC 2013 by Mendel et al.)

[4] New Records in Collision Attacks on RIPEMD-160 and SHA-256 (eprint 2023/285)

Summary

- Alternative methods to find trails for the MD-SHA hash family
- Further improve the efficiency?
- More applications? (SHA-256, SHA-512...)

`https://github.com/LFKOKAMI/Find_RIPEMD_Trail`