Cryptanalysis of RIPEMD-128/160

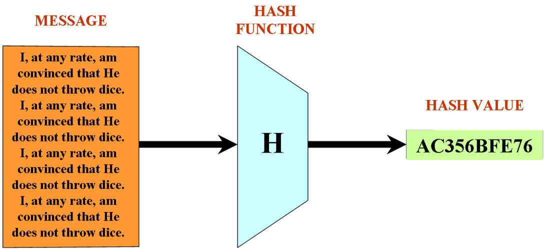
Thomas Peyrin

NTU - Singapore

ChinaCrypt 2013

Fuzhou, China - October 25, 2013

What is a Hash Function ?



- H maps an **arbitrary length input** (the message M) to a **fixed length output** (typically $n = 128$, $n = 160$ or $n = 256$).
- no secret parameter.
- H must be easy to compute.

The security goals

pre-image resistance:

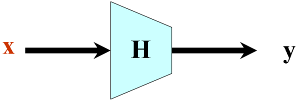
given an output challenge y, the attacker can not find a message x such that $H(x) = y$, in less than $\theta(2^n)$ operations.

2nd pre-image resistance:

given a challenge (x, y) so that $H(x) = y$, the attacker can not find a message $x' \neq x$ such that $H(x') = y$, in less than $\theta(2^n)$ operations.

collision resistance:

the attacker can not find two messages (x, x') such that $H(x) = H(x')$, in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79]).



The security goals

pre-image resistance:

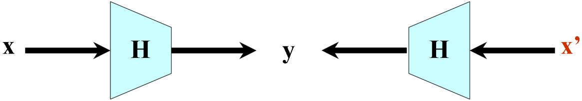
given an output challenge y, the attacker can not find a message x such that $H(x) = y$, in less than $\theta(2^n)$ operations.

2nd pre-image resistance:

given a challenge (x, y) so that $H(x) = y$, the attacker can not find a message $x' \neq x$ such that $H(x') = y$, in less than $\theta(2^n)$ operations.

collision resistance:

the attacker can not find two messages (x, x') such that $H(x) = H(x')$, in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79]).



The security goals

pre-image resistance:

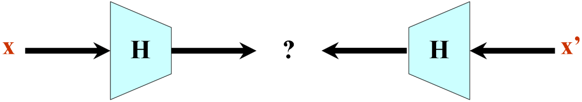
given an output challenge y, the attacker can not find a message x such that $H(x) = y$, in less than $\theta(2^n)$ operations.

2nd pre-image resistance:

given a challenge (x, y) so that $H(x) = y$, the attacker can not find a message $x' \neq x$ such that $H(x') = y$, in less than $\theta(2^n)$ operations.

collision resistance:

the attacker can not find two messages (x, x') such that $H(x) = H(x')$, in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79]).



The security goals

pre-image resistance:

given an output challenge y, the attacker can not find a message x such that $H(x) = y$, in less than $\theta(2^n)$ operations.

2nd pre-image resistance:

given a challenge (x, y) so that $H(x) = y$, the attacker can not find a message $x' \neq x$ such that $H(x') = y$, in less than $\theta(2^n)$ operations.

collision resistance:

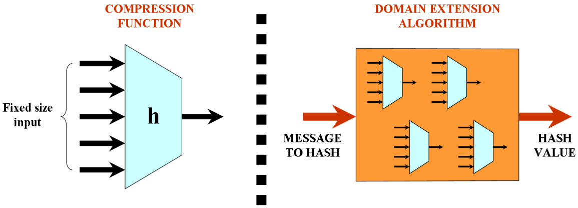
the attacker can not find two messages (x, x') such that $H(x) = H(x')$, in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79]).

And other ones: near collisions, multicollisions, random oracle look-alike, ...

General construction

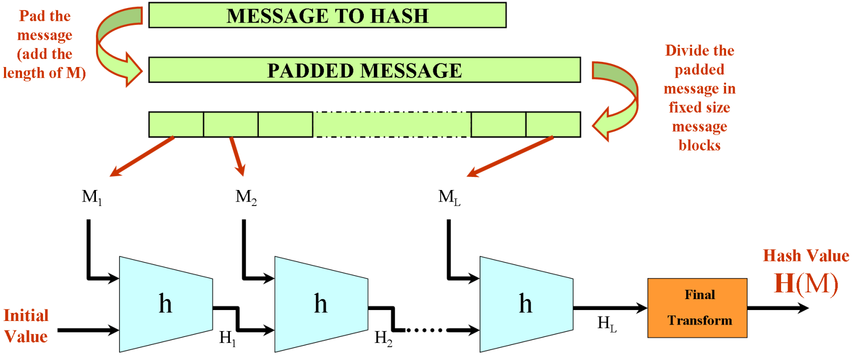
For historical reasons, most hash functions are composed of two elements:

- **a compression function** h**:** a function for which the input and output size is fixed.
- **a domain extension algorithm:** an iterative process that uses the compression function h so that the hash function H can handle inputs of arbitrary length.



The Merkle-Damgård domain extension algorithm

The most famous domain extension algorithm used is called the **Merkle-Damgård** [Merkle Damgård-89] iterative algorithm.



General design and security notions

- A collision on an iterated hash function $\mathcal{H}$ always comes from a collision on the compression function h:

$$\mathcal{H}(M) = \mathcal{H}(M^*) \Longrightarrow h(cv, m) = h(cv^*, m^*)$$

The conditions on *cv* and *m* give different kind of attacks :

Collision $cv = cv^*$ fixed and $m \neq m^*$ free.

Semi-free-start Collision $cv = cv^*$ and $m \neq m^*$ are free.

Free-start Collision $(cv, m) \neq (cv^*, m^*)$ are free.

The cryptanalysis history of MD5 is a good example of why **(semi)-free-start collisions are a serious warning.**

Motivations to study RIPEMD

- MDx-like hash function is a very frequent design :

  1990' MDx (MD4, MD5, SHA-1, HAVAL, RIPEMD)
  2002 SHA-2 (SHA-224, ..., SHA-512)

- Some old hash functions are still unbroken :

  Broken MD4, MD5, RIPEMD-0
  Broken HAVAL
  Broken SHA-1
  Unbroken RIPEMD-128, RIPEMD-160
  Unbroken SHA-2

- RIPEMD-128/ RIPEMD-160

  Design 15 years old.
  unbroken 9 years after Wang's attacks [WLF+05].

Cryptanalysis of RIPEMD-128

Thomas Peyrin

joint work with Franck Landelle

(accepted at Eurocrypt 2013)

ChinaCrypt 2013

Fuzhou, China - October 25, 2013

NANYANG TECHNOLOGICAL UNIVERSITY

Results on RIPEMD-128 compression function

RIPEMD-128 parameters :

Digest 128 bits

Steps 64 steps (4 rounds of 16 steps each)

Known and new results on RIPEMD-128 compression function:

Target	#Steps	Complexity	Ref.
collision	48	2^{40}	[MNS12]
collision	**60**	$2^{57.57}$	**new**
collision	**63**	$2^{59.91}$	**new**
collision	**Full**	$2^{61.57}$	**new**
non-randomness	52	2^{107}	[SW12]
non-randomness	**Full**	$2^{59.57}$	**new**

In this talk

Function RIPEMD-128 compression function

Attack a semi-free-start collision

$$\text{Find } cv, m \neq m^* \,/\, h(cv, m) = h(cv, m^*).$$

Strategy

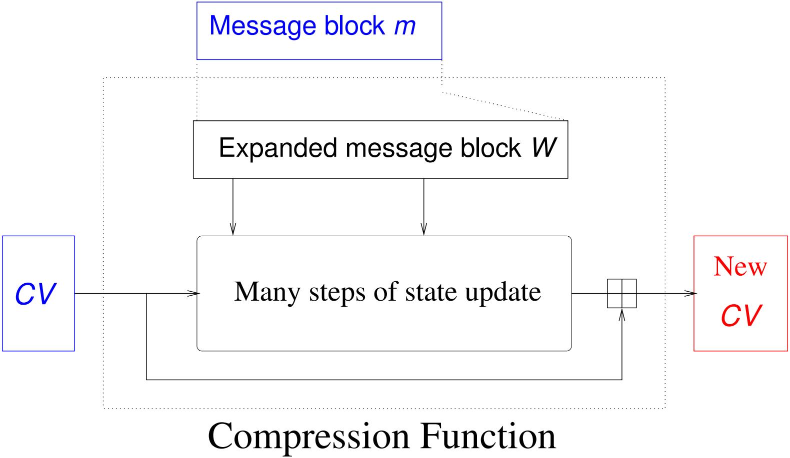
- Choose a message difference $\delta_m = m \oplus m^*$
  $\rightarrow$ new message difference used
- Find a differential path on all intermediate state variables
  $\rightarrow$ new type of differential path with two non-linear parts
- Find conforming *cv* and *m*
  $\rightarrow$ new branch merging technique for collision search

Outline

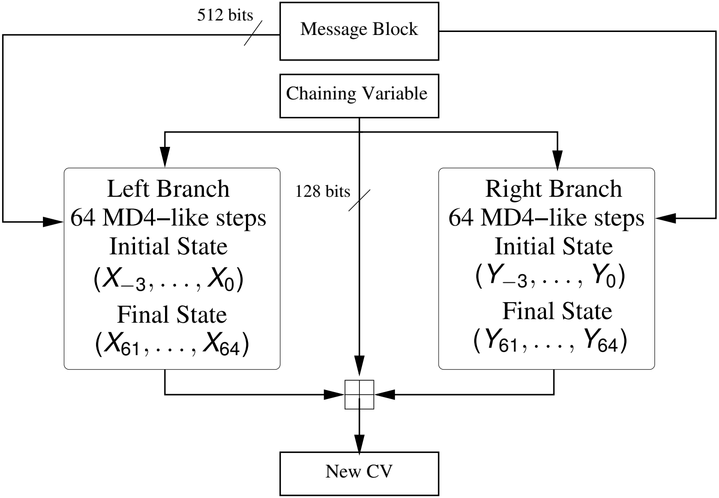
Outline

A compression function

$$m = m_0 || m_1 || \cdots || m_{15}$$

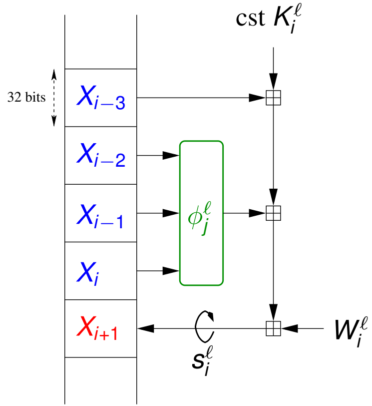


Overview of RIPEMD−128 compression function

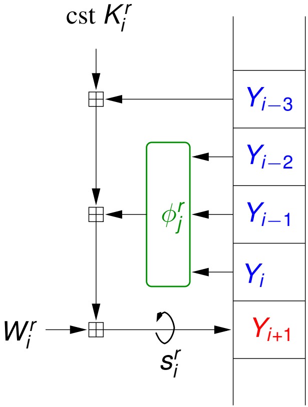


The step function

$$W_i^r = m_{\pi_j^r(i)} \qquad W_i^\ell = m_{\pi_j^\ell(i)}$$



Left Branch - step i, round j



Right Branch - step i, round j

The boolean functions

Boolean functions in RIPEMD-128:

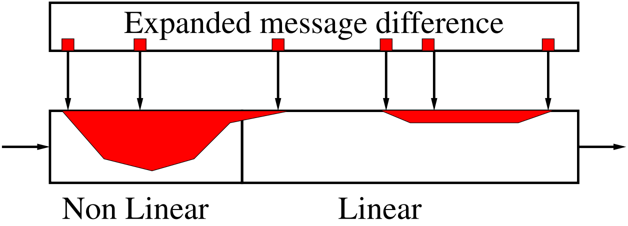
- $\text{XOR}(x, y, z) := x \oplus y \oplus z$,
- $\text{IF}(x, y, z) := x \wedge y \oplus \bar{x} \wedge z$
- $\text{ONX}(x, y, z) := (x \vee \bar{y}) \oplus z$

Steps i	Round j	$\phi_j^\ell(x, y, z)$	$\phi_j^r(x, y, z)$
0 to 15	0	$\texttt{XOR}(x, y, z)$	$\texttt{IF}(z, x, y)$
16 to 31	1	$\texttt{IF}(x, y, z)$	$\texttt{ONX}(x, y, z)$
32 to 47	2	$\texttt{ONX}(x, y, z)$	$\texttt{IF}(x, y, z)$
48 to 63	3	$\texttt{IF}(z, x, y)$	$\texttt{XOR}(x, y, z)$

Outline

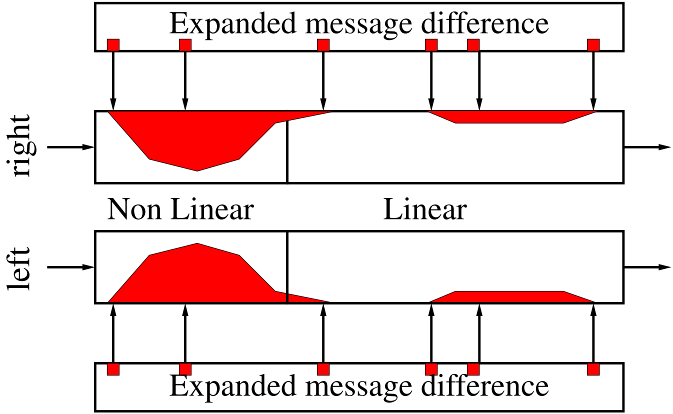
The classical strategy (example SHA-1)

1. Find a message difference δ_m and a differential path with high probability on the middle and last steps (ideally after the first round).
2. Find a “realistic” non-linear differential path on the first steps (ideally on the first round for a semi-free-start collision).
3. Find a chaining variable *cv* and a message *m* such that the state differential path is followed (use special freedom degrees tricks like neutral bits, message modification, boomerangs, etc.).



The classical strategy (example RIPEMD-128)

1. Find a message difference δ_m and a differential path with high probability on the middle and last steps for both branches.
2. Find a "realistic" non-linear differential path on the first steps.
3. Find a conforming chaining variable *cv* and a message *m*.



What shape should have the differential path ?

Boolean functions can help to control the diff. propagation.

Properties of the boolean functions:

- XOR : no control of differential propagation
- ONX: some control of differential propagation and permits low diffusion.
- IF : a good control of differential propagation and permits **no** diffusion.

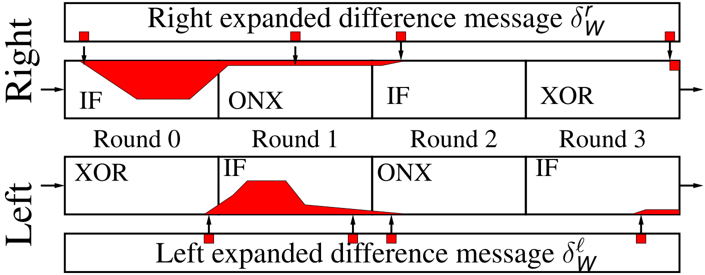
Steps i	Round j	$\phi_j^l(x, y, z)$	$\phi_j^r(x, y, z)$
0 to 15	0	$\mathtt{XOR}(x, y, z)$	$\mathtt{IF}(z, x, y)$
16 to 31	1	$\mathtt{IF}(x, y, z)$	$\mathtt{ONX}(x, y, z)$
32 to 47	2	$\mathtt{ONX}(x, y, z)$	$\mathtt{IF}(x, y, z)$
48 to 63	3	$\mathtt{IF}(z, x, y)$	$\mathtt{XOR}(x, y, z)$

Outline

Choosing the message block difference

Goals keep low ham. weight on the expanded message block

Choice Put a difference on a single word of message



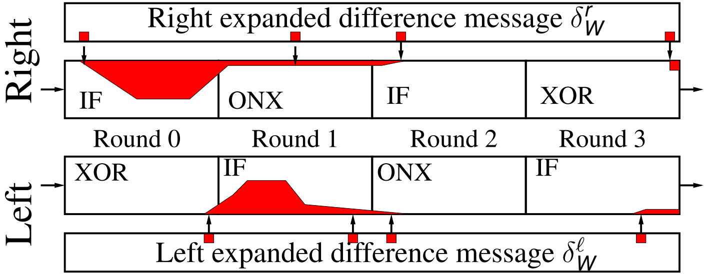
With the message block difference on m_{14}:

- "no difference" on rounds with XOR function.
- Non-linear differential paths are in the round with IF

Choosing the message block difference

m_{14} is really "**magic**" with regards to our criteria.

However, **how to handle these** **two** **non-linear parts which are in different branches, and not in the first round** ?



Outline

Description of RIPEMD-128

Finding a differential path

- Finding a message difference
- Finding the non-linear part

Finding a conforming pair

- Generating a starting point
- Merging the 2 branches

Conclusion

Automatic tool on generalized conditions

We implemented a tool similar to [CR06] for `SHA-1` that uses generalized conditions.

Hexa	(b, b^*) Notation	$(0, 0)$	$(1, 0)$	$(0, 1)$	$(1, 1)$
0xF	?	✓	✓	✓	✓
0x9	-	✓			✓
0x6	x		✓	✓	
0x1	0	✓			
0x2	u		✓		
0x4	n			✓	
0x8	1				✓

Where

- b: a bit during the treatment the message m
- b^*: the same bit for the second message m^*.

Left branch

```
Step           Xi                                       Wi                      Πi
13: -------------------------------- | -------------------------------- 13
14: -------------------------------- | x------------------------------- 14
15: ???????????????????????????????? | -------------------------------- 15
16: ???????????????????????????????? | --------------------------------  7
17: ???????????????????????????????? | --------------------------------  4
18: ???????????????????????????????? | -------------------------------- 13
19: ???????????????????????????????? | --------------------------------  1
20: ???????????????????????????????? | -------------------------------- 10
21: ???????????????????????????????? | --------------------------------  6
22: ???????????????????????????????? | -------------------------------- 15
23: ???????????????????????????????? | --------------------------------  3
24: ???????????????????????????????? | -------------------------------- 12
25: ???????????????????????????????? | --------------------------------  0
26: -------u------------------------ | --------------------------------  9
27: 1------0-----u------------------ | --------------------------------  5
28: 0------1-----0------------------ | --------------------------------  2
29: n------------1------------------ | x------------------------------- 14
30: u------------------------------- | -------------------------------- 11
31: u------------------------------- | --------------------------------  8
32: 1------------------------------- | --------------------------------  3
33: -------------------------------- | -------------------------------- 10
34: -------------------------------- | x------------------------------- 14
35: -------------------------------- | --------------------------------  4
```

Left branch

```
Step             Xi                                        Wi                   Πi
13: -------------------------------- | -------------------------------- 13
14: -------------------------------- | x------------------------------- 14
15: -----------------------n-------- | -------------------------------- 15
16: ----------unnnn--------0-------- | --------------------------------  7
17: -------n---00000-------1-------- | --1-----------------------------  4
18: -------0---01111---------------- | -------------------------------- 13
19: ---u---1-------n------------1--- | --------------------------------  1
20: ---0-----------0------------0--- | -------------------------------- 10
21: ---1-----------1------------n--- | --------------------------------  6
22: ---------------unnnn--------0--- | -------------------------------- 15
23: ---------------00000--------u--- | --------------------------------  3
24: -------------n-11101--------1--- | -------------------------------- 12
25: -----------n-0--------------1--- | --------------------------------  0
26: -------u---0-1------------------ | --------------------------------  9
27: 1------0---1-u------------------ | --------------------------------  5
28: 0------1-----0------------------ | --------------------------------  2
29: n------------1------------------ | x------------------------------- 14
30: u------------------------------- | -------------------------------- 11
31: u------------------------------- | --------------------------------  8
32: 1------------------------------- | --------------------------------  3
33: -------------------------------- | -------------------------------- 10
34: -------------------------------- | x------------------------------- 14
35: -------------------------------- | --1-----------------------------  4
```

Right branch

```
Step              Yi                                  Wi                         πi
   : --------------------------------
   : --------------------------------
   : --------------------------------
   : -------------------------------- | --------------------------------  5
01:  -------------------------------- | x------------------------------- 14
02:  ???????????????????????????????? | --------------------------------  7
03:  ???????????????????????????????? | --------------------------------  0
04:  ???????????????????????????????? | --------------------------------  9
05:  ???????????????????????????????? | --------------------------------  2
06:  ???????????????????????????????? | -------------------------------- 11
07:  ???????????????????????????????? | --------------------------------  4
08:  ???????????????????????????????? | -------------------------------- 13
09:  ???????????????????????????????? | --------------------------------  6
10:  ???????????????????????????????? | -------------------------------- 15
11:  ???????????????????????????????? | --------------------------------  8
12:  ???????????????????????????????? | --------------------------------  1
13:  ???????????????????????????????? | -------------------------------- 10
14:  ???????????????????????????????? | --------------------------------  3
15:  -------u------------------------ | -------------------------------- 12
16:  -------u----u------------------- | --------------------------------  6
17:  -----u-0----u------------------- | -------------------------------- 11
18:  -----u------0------------------- | --------------------------------  3
19:  0----0-------------------------- | --------------------------------  7
20:  u------------------------------- | --------------------------------  0
```

Right branch

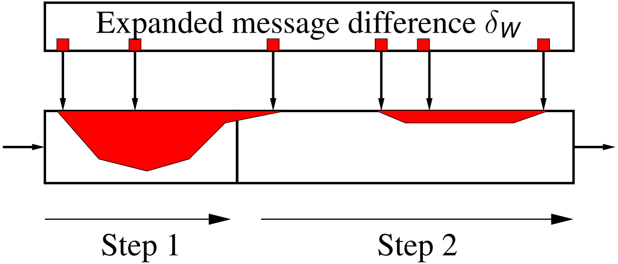
```
Step               Yi                                                   Wi                         πi
   : --------------------------------
   : --------------------------------
   : --------------------------------
   : -----------------------0-------- | --------------------------------  5
01:  -----------------------1-------- | x------------------------------- 14
02:  -----------------------n-------- | --------------------------------  7
03:  -------------------------------- | --------------------------------  0
04:  --0000000----------------------- | --------------------------------  9
05:  --1111111----------------------- | --------------------------------  2
06:  --nuuuuuu----------------------- | -------------------------------- 11
07:  --01-----------------------0-000 | --1-----------------------------  4
08:  -01------------------------0-011 | -------------------------------- 13
09:  -1----------------10-0-----n-nnn | --------------------------------  6
10:  1n010000----------11-1---------- | -------------------------------- 15
11:  00111111-----00--0nu-n---------- | --------------------------------  8
12:  nuuuuuuu-----11--11--0---------- | --------------------------------  1
13:  -------1-----nn--un--u---------- | -------------------------------- 10
14:  -------1----01----u------------- | --------------------------------  3
15:  -------u----10----0------------- | -------------------------------- 12
16:  -----0-u----u------------------- | --------------------------------  6
17:  -----u-0----u------------------- | -------------------------------- 11
18:  -----u------0------------------- | --------------------------------  3
19:  0----0-------------------------- | --------------------------------  7
20:  u------------------------------- | --------------------------------  0
```

Outline

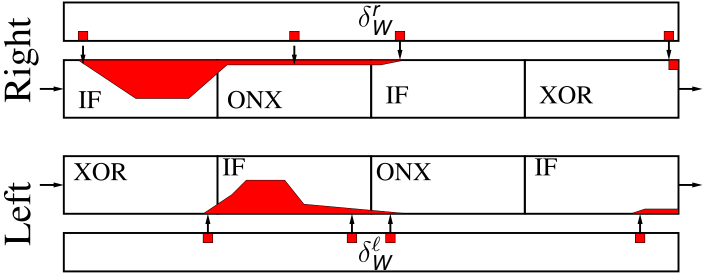
Following a classical differential path

A classical collision search is composed of two subparts:

- **step 1** handling the low-probability non-linear parts using the message block freedom
- **step 2** the remaining steps in both branches are verified probabilistically



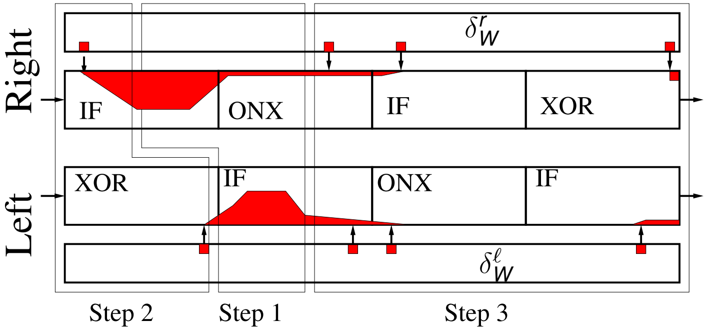
Finding a conforming pair



Our collision search is composed of three subparts:

- step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words
- step 2 From this **starting point**, merge the two branches using some remaining free message words
- step 3 Handle probabilistically the linear part in both branches

Finding a conforming pair

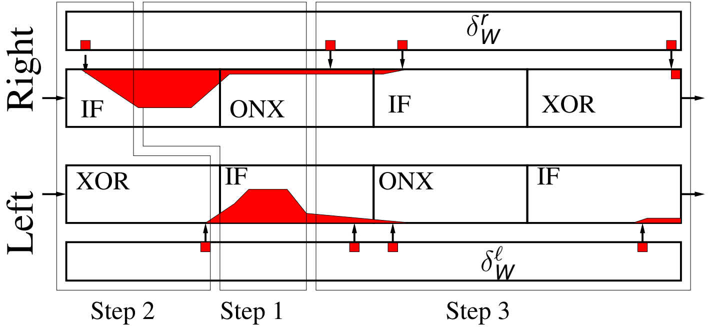


Our collision search is composed of three subparts:

- step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words
- step 2 From this **starting point**, merge the two branches using some remaining free message words
- step 3 Handle probabilistically the linear part in both branches

Outline

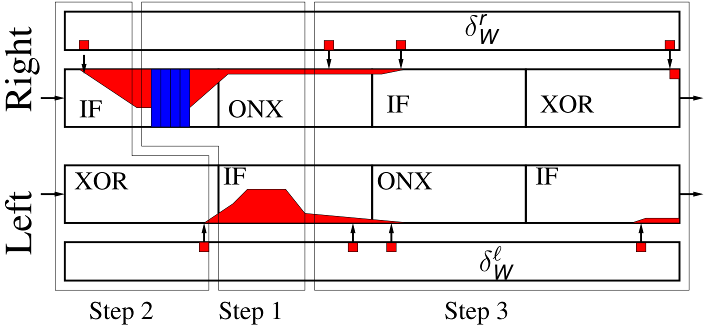
Satisfying the two non-linear parts simultaneously (step 1)



Our collision search is composed of three subparts:

- step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words
- step 2 From this **starting point**, merge the two branches using some remaining free message words
- step 3 Handle probabilistically the linear part in both branches

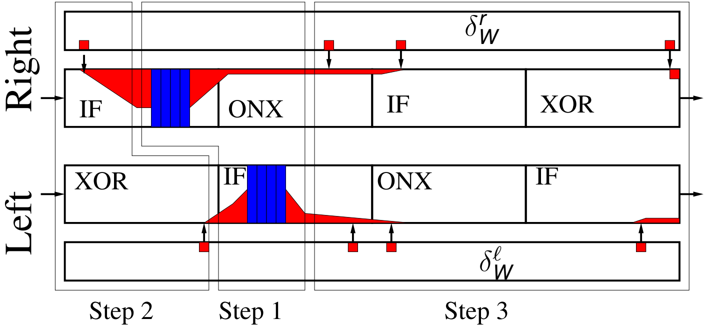
Satisfying the two non-linear parts simultaneously (step 1)



Our collision search is composed of three subparts:

- step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words
- step 2 From this **starting point**, merge the two branches using some remaining free message words
- step 3 Handle probabilistically the linear part in both branches

Satisfying the two non-linear parts simultaneously (step 1)



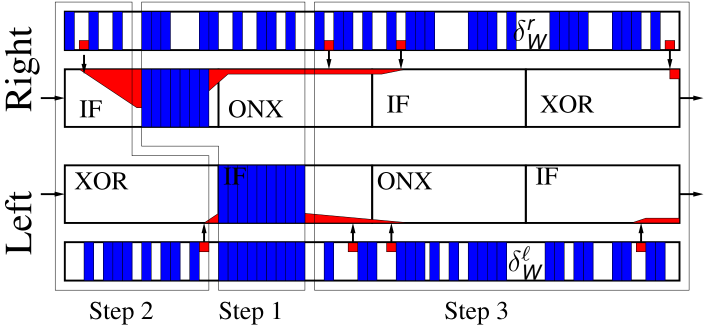
Our collision search is composed of three subparts:

step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words

step 2 From this **starting point**, merge the two branches using some remaining free message words

step 3 Handle probabilistically the linear part in both branches

Satisfying the two non-linear parts simultaneously (step 1)



Our collision search is composed of three subparts:

- step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words
- step 2 From this **starting point**, merge the two branches using some remaining free message words
- step 3 Handle probabilistically the linear part in both branches

Handling probabilistically the linear parts (step 3)

Probabilities of the linear parts are fixed after the first step:

- The probability of the left branch is 2^{-15}.
- The probability of the right branch is $2^{-14.32}$.
- one extra bit condition in order to get a collision when adding the two branches
- $\rightarrow$ The overall probability for collision is $2^{-30.32}$.

(these probabilities have been verified experimentally)

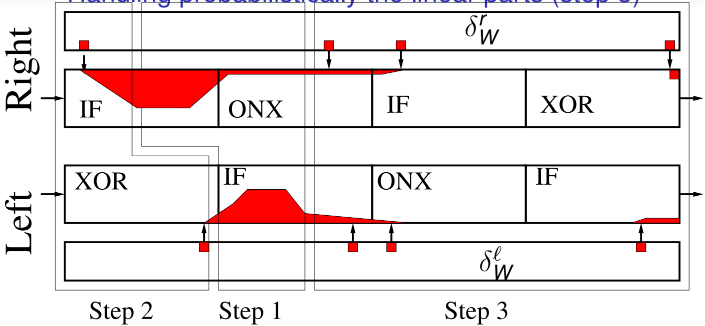
Our collision search is composed of three subparts:

step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words

step 2 From this **starting point**, merge the two branches using some remaining free message words

step 3 Handle probabilistically the linear part in both branches

Handling probabilistically the linear parts (step 3)



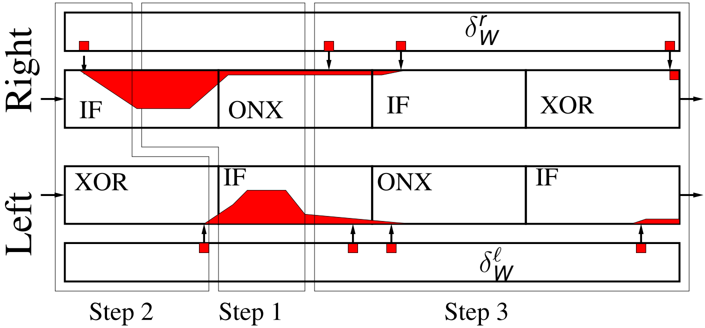
→ we need to obtain $2^{30.32}$ solutions of the merging system

Our collision search is composed of three subparts:

- step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words
- step 2 From this **starting point**, merge the two branches using some remaining free message words
- step 3 Handle probabilistically the linear part in both branches

Outline

Merging the two branches (step 2)



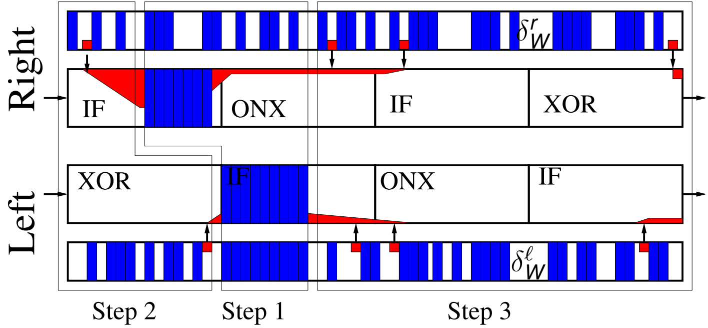
Our collision search is composed of three subparts:

step 1 Satisfy the two non-linear parts using the freedom from both branches internal states and a few message words

step 2 From this **starting point**, merge the two branches using some remaining free message words

step 3 Handle probabilistically the linear part in both branches

The starting point



What is fixed ?

Message $m_{12}, m_3, m_{10}, m_1, m_8, m_{15}, m_6, m_{13}, m_4, m_{11}, m_7$.

Left State $(X_{12}, \ldots, X_{24})$

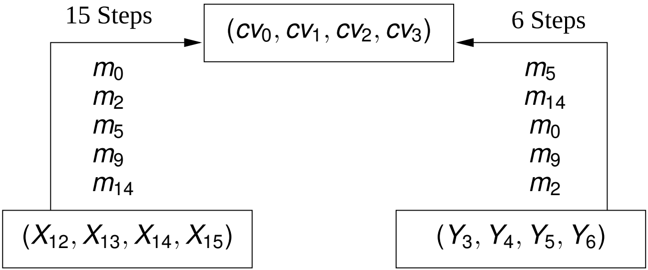
Right State $(Y_3, Y_4, \ldots, Y_{14})$.

What is free ?

Message $m_0, m_2, m_5, m_9, m_{14}$.

Prepare the merging system

The system is quite complex:



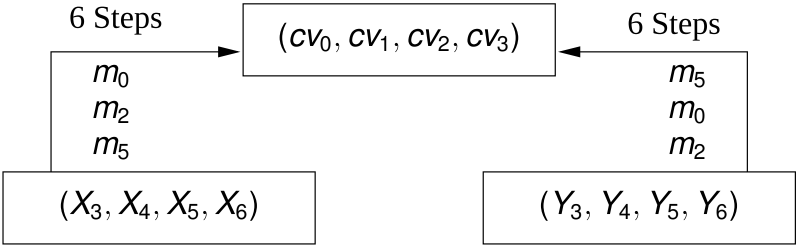
The probability that a random choice of $m_0, m_2, m_5, m_9, m_{14}$ gives a solution is

$$2^{-128}$$

Reducing the merging system

- in the search for a starting point (step 1), we chose m_{11} such that: $Y_3 = Y_4$
- randomly chose a m_{14} value and deduce m_9 such that: $X_5^{\ggg 5} \boxminus m_4 = \texttt{0xffffffff}$

$\rightarrow$ **the system becomes much simpler and represents less steps of the compression function.**



Solving the merging system

The goal now is to find m_0, m_2, m_5 such that

$$X_i = Y_i \text{ for } i \in \{-3, -2, -1, 0\}$$

	X_0	Y_0	X_{-1}	Y_{-1}	X_{-2}	Y_{-2}	X_{-3}	Y_{-3}
m_2		✓	✓	✓	✓	✓	✓	✓
m_0		✓					✓	
m_5					✓		✓	✓

To solve the merging system:

1. find a value of m_2 that verifies $X_{-1} = Y_{-1}$
2. deduce m_0 to fulfill $X_0 = Y_0$
3. obtain m_5 to satisfy a combination of $X_{-2} = Y_{-2}$ and $X_{-3} = Y_{-3}$
4. finally the 4^{th} equation is verified with probability 2^{-32}

Solving the merging system

The goal now is to find m_0, m_2, m_5 such that

$$X_i = Y_i \text{ for } i \in \{-3, -2, -1, 0\}$$

	X_0	Y_0	X_{-1}	Y_{-1}	X_{-2}	Y_{-2}	X_{-3}	Y_{-3}
m_2		✓	✓	✓	✓	✓	✓	✓
m_0		✓					✓	
m_5					✓		✓	✓

To solve the merging system:

1. find a value of m_2 that verifies $X_{-1} = Y_{-1}$
2. deduce m_0 to fulfill $X_0 = Y_0$
3. obtain m_5 to satisfy a combination of $X_{-2} = Y_{-2}$ and $X_{-3} = Y_{-3}$
4. finally the 4^{th} equation is verified with probability 2^{-32}

Solving the merging system

The goal now is to find m_0, m_2, m_5 such that

$$X_i = Y_i \text{ for } i \in \{-3, -2, -1, 0\}$$

	X_0	Y_0	X_{-1}	Y_{-1}	X_{-2}	Y_{-2}	X_{-3}	Y_{-3}
m_2		✓	✓	✓	✓	✓	✓	✓
m_0		✓					✓	
m_5					✓		✓	✓

To solve the merging system:

1. find a value of m_2 that verifies $X_{-1} = Y_{-1}$
2. deduce m_0 to fulfill $X_0 = Y_0$
3. obtain m_5 to satisfy a combination of $X_{-2} = Y_{-2}$ and $X_{-3} = Y_{-3}$
4. finally the 4^{th} equation is verified with probability 2^{-32}

Solving the merging system

The goal now is to find m_0, m_2, m_5 such that

$$X_i = Y_i \text{ for } i \in \{-3, -2, -1, 0\}$$

	X_0	Y_0	X_{-1}	Y_{-1}	X_{-2}	Y_{-2}	X_{-3}	Y_{-3}
m_2		✓	✓	✓	✓	✓	✓	✓
m_0		✓					✓	
m_5					✓		✓	✓

To solve the merging system:

1. find a value of m_2 that verifies $X_{-1} = Y_{-1}$
2. deduce m_0 to fulfill $X_0 = Y_0$
3. obtain m_5 to satisfy a combination of $X_{-2} = Y_{-2}$ and $X_{-3} = Y_{-3}$
4. finally the 4^{th} equation is verified with probability 2^{-32}

Solving the merging system

The goal now is to find m_0, m_2, m_5 such that

$$X_i = Y_i \text{ for } i \in \{-3, -2, -1, 0\}$$

	X_0	Y_0	X_{-1}	Y_{-1}	X_{-2}	Y_{-2}	X_{-3}	Y_{-3}
m_2		✓	✓	✓	✓	✓	✓	✓
m_0		✓					✓	
m_5					✓		✓	✓

To solve the merging system:

1. find a value of m_2 that verifies $X_{-1} = Y_{-1}$
2. deduce m_0 to fulfill $X_0 = Y_0$
3. obtain m_5 to satisfy a combination of $X_{-2} = Y_{-2}$ and $X_{-3} = Y_{-3}$
4. finally the 4^{th} equation is verified with probability 2^{-32}

Complexity of the semi-free-start collision attack

- Solving the merging system costs 19 RIPEMD-128 step computations ($19/128$ of the compression function cost).
- The probability of success of the merging is 2^{-34} (because of 4^{th} equation and 2 extra hidden bit conditions)
- We need to find $2^{30.32}$ solutions of the merging system.

The **total complexity** is therefore

$$19/128 \times 2^{34} \times 2^{30.32} \simeq 2^{61.57}$$

calls to the compression function.

Outline

Conclusion

This work:

- a new cryptanalysis technique for parallel branches based functions
- a collision attack on the full compression function of RIPEMD-128
- a distinguisher on the hash function of RIPEMD-128
- a LOT of details (many not described here)

Perspectives:

- improvements of this technique
- an example of collision for RIPEMD-128?
- apply to other 2-branch hash functions
- what about RIPEMD-160?

Cryptanalysis of RIPEMD-160

Thomas Peyrin

joint work with F. Mendel, M. Schläffer, L. Wang and S. Wu

(accepted at Asiacrypt 2013)

ChinaCrypt 2013

Fuzhou, China - October 25, 2013

NANYANG TECHNOLOGICAL UNIVERSITY

Results on RIPEMD-160 compression function

RIPEMD-160 parameters :

- **Digest** 160 bits
- **Steps** 80 steps (5 rounds of 16 steps each)

Known and new results on RIPEMD-160 compression function:

Target	#Steps	Complexity	Ref.
semi-free-start collision	36	low (practical)	[MNS12]
1st round			
semi-free-start collision	**36**	$2^{70.4}$	**new**
semi-free-start collision	**42**	$2^{75.5}$	**new**

RIPEMD-160 >> RIPEMD-128

Why are the improvements far less impressive for RIPEMD-160?

The technique we applied on RIPEMD-128 is much harder to apply on RIPEMD-160:

- finding non-linear parts is more difficult than for RIPEMD-128
- evaluating the probability of a differential path is hard (because two additions are interlinked)
- ... so more complicated to have a global view of what will and what won't work when trying to organize the attack

On top of that, RIPEMD-160 has

- better diffusion (impossible to force no diffusion, even in IF rounds)
- more steps ...

Thank you for your attention !

We are looking for good PhD students
in symmetric key crypto.

If interested, please contact me at:
thomas.peyrin@ntu.edu.sg

NANYANG TECHNOLOGICAL UNIVERSITY