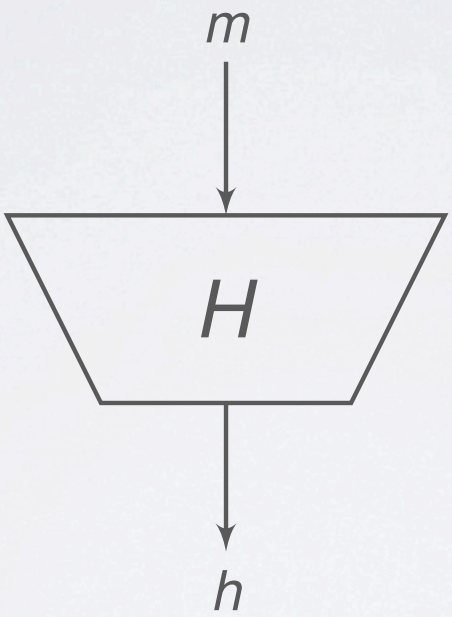
Tools in Cryptanalysis

Florian Mendel - Tomislav Nad - Martin Schläffer
Christoph Dobraunig - Maria Eichlseder

TU Graz

Hash Functions

A cryptographic hash function produces cryptographic checksums or fingerprints



- Fast
- Secure

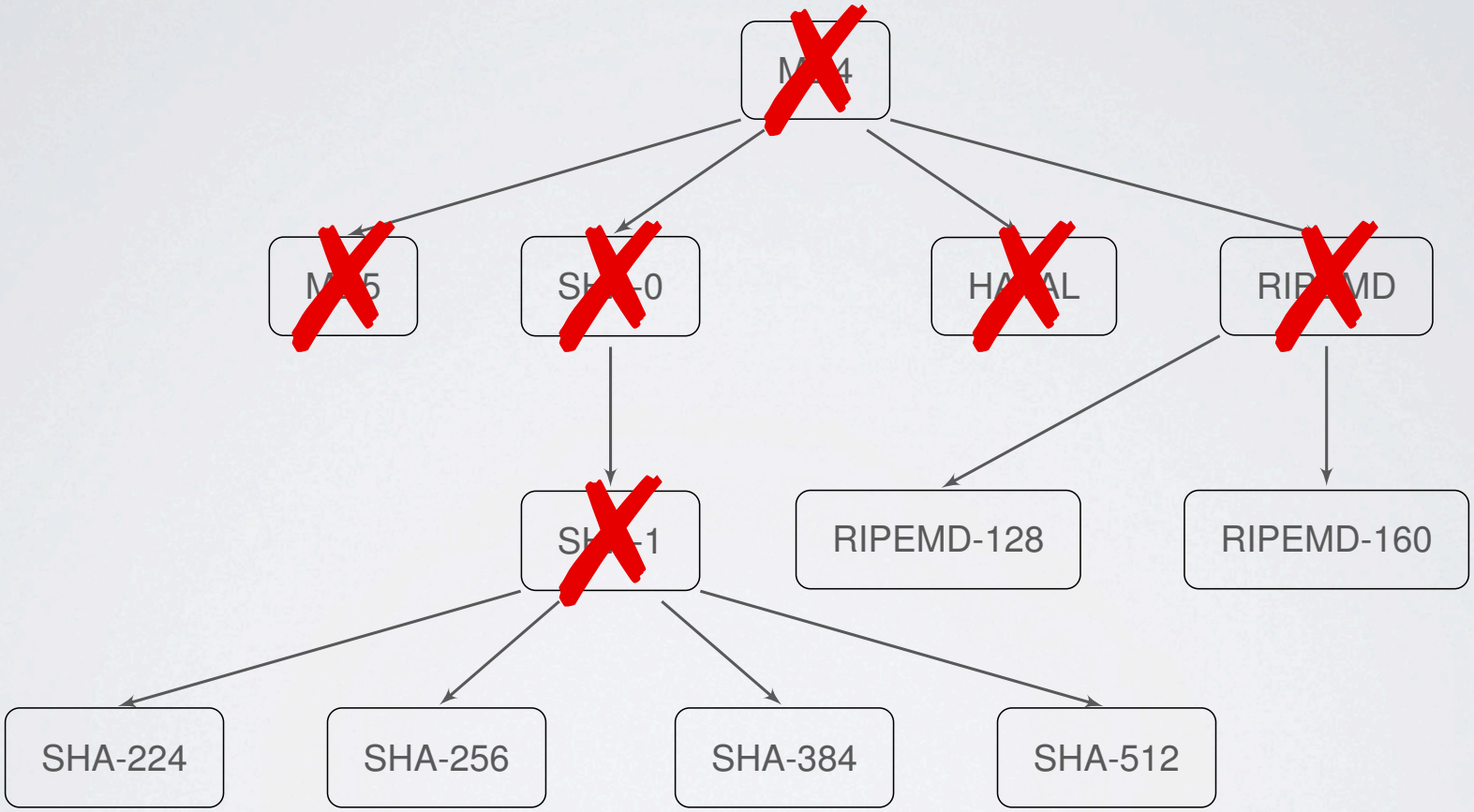
Security properties

Preimage resistance:
Given $H(m)$, difficult to find m

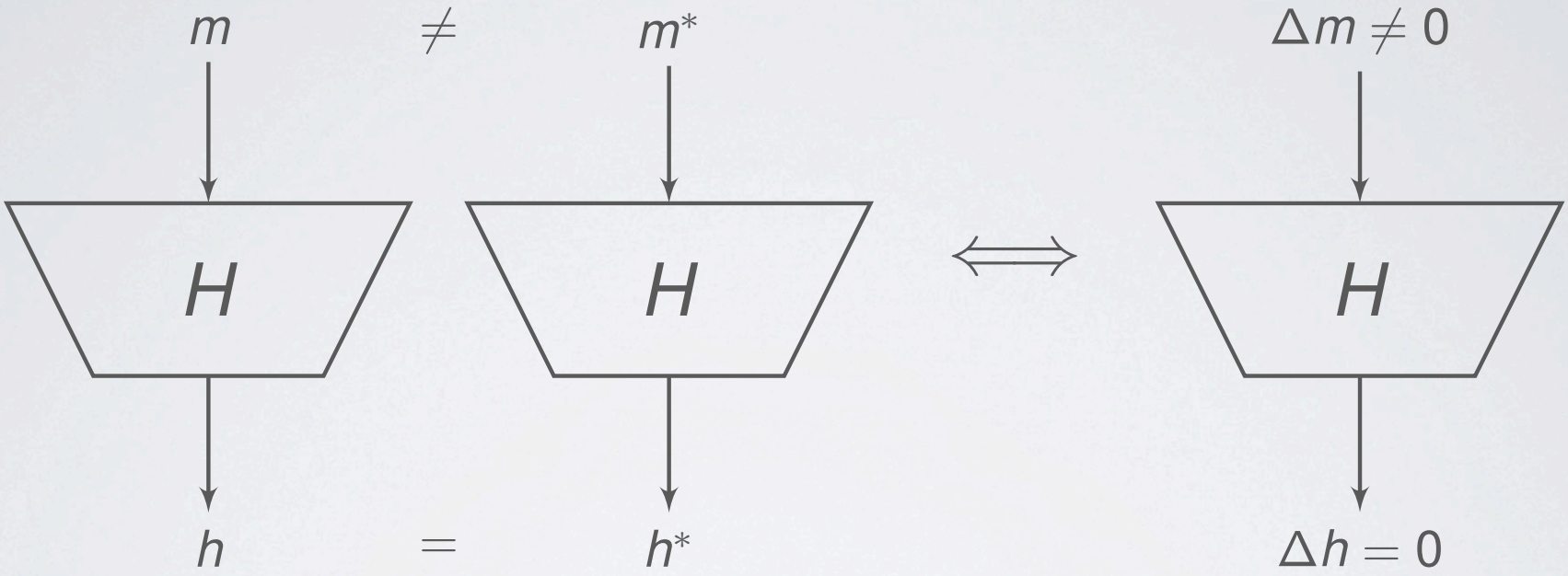
Second preimage resistance:
Given $m, H(m)$, difficult to find m^* such that $H(m^*) = H(m)$

Collision resistance:
Difficult to find m, m^* such that $H(m^*) = H(m)$

Hash Function Crisis



Collision Attacks



- Find a differential characteristic
- Find a message m following the differential characteristic

Basic Attack Strategy

- **Construct differential characteristic**
  - Find a characteristic (collision) for the last rounds (high probability)
  - Find a characteristic (not necessary with high probability) for the first rounds
- **Find message following the characteristic**
  - Use message modification techniques to fulfill conditions imposed by the characteristic in the first two round
  - Use random trials to find values for the remaining free message bits such that the message follows the characteristic

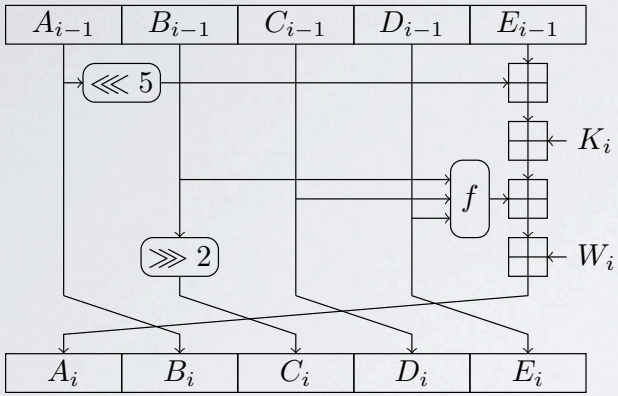
$\Rightarrow$ The attack complexity is dominated by the last step

How to Construct Differential Characteristics

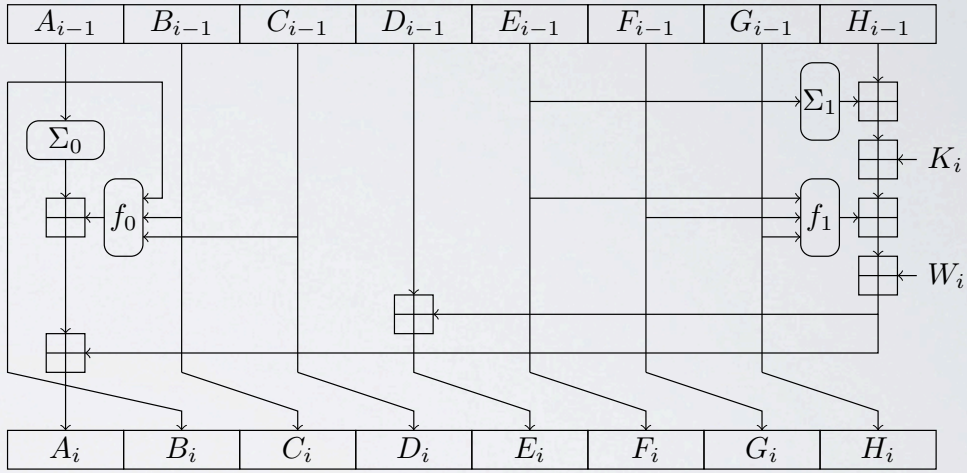
- Wang's Approach: by hand
- Gröbner Basis, SAT solvers, . . .
- Guess-and-Determine Approach

Increased Complexity of SHA-2

SHA-1



SHA-2



Design Complexity

How to overcome the problems?

- **Problem description**
  - Starting point, high-level strategy
  - Hash function description
- **Guessing strategy, branching rules**
  - Which variable to pick first?
  - Which value to guess first for this variable?
- **Propagation**
  - How to detect contradictions?
  - How to determine implications of a guess?
- **Backtracking**
  - How many guesses to undo?
  - Restart?

How to overcome the problems?

C. Dobraunig, M. Eichlseder, and F. Mendel:
Analysis of SHA-512/224 and SHA-512/256
ASIACRYPT 2015

M. Eichlseder, F. Mendel, and M. Schläffer:
Branching Heuristics in Differential Collision Search with Applications to SHA-512
FSE 2014

M. Eichlseder, F. Mendel, T. Nad, V. Rijmen, and M. Schläffer:
Linear Propagation in Efficient Guess-and-Determine Attacks
WCC 2013

F. Mendel, T. Nad, and M. Schläffer:
Improving Local Collisions: New Attacks on Reduced SHA-256
EUROCRYPT 2013

F. Mendel, T. Nad, and M. Schläffer:
Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions
ASIACRYPT 2011

Results for SHA-2

- Collision attacks for the SHA-2 family

	component	attack	steps	complexity
SHA-256	compression	collision	38	example
	hash	collision	28	example
	hash	collision	31	$2^{65.5}$
SHA-512	compression	collision	39	example
	hash	collision	27	example

RIPEMD-128/160

- Designed by Dobbertin, Bosselaers and Preneel in 1996
- ISO/IEC 10118-3 standard on dedicated hash function
- Similar design principle as MD5 and SHA-1

	component	attack	steps	complexity
RIPEMD-128	compression	collision	48	example
	hash	collision	38	example
	hash	near-collision	44	example
RIPEMD-160	compression	collision	48	example

$\Rightarrow$ Theoretical attacks on full RIPEMD-128 [LP13]

Other Applications

HAS-160

- Standardized by the Korean government
- Similar design principle as SHA-1

component	attack	steps	complexity
compression	collision	65	example

SM3

- Standardized by the Chinese government
- Similar design principle as SHA-256

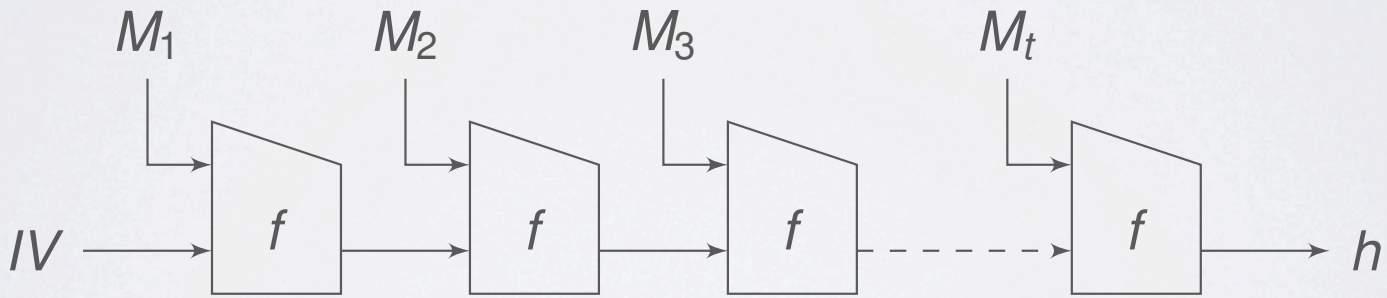
component	attack	steps	complexity
compression	collision	24	example
hash	collision	20	example

How does it work?

Application to MD4

Description of MD4

- Iterated hash function processing message blocks of 512 bits and producing a hash value of 128 bits.
- Compression function *f* consists of 2 parts:
  - Message Expansion
  - State Update (48 steps)

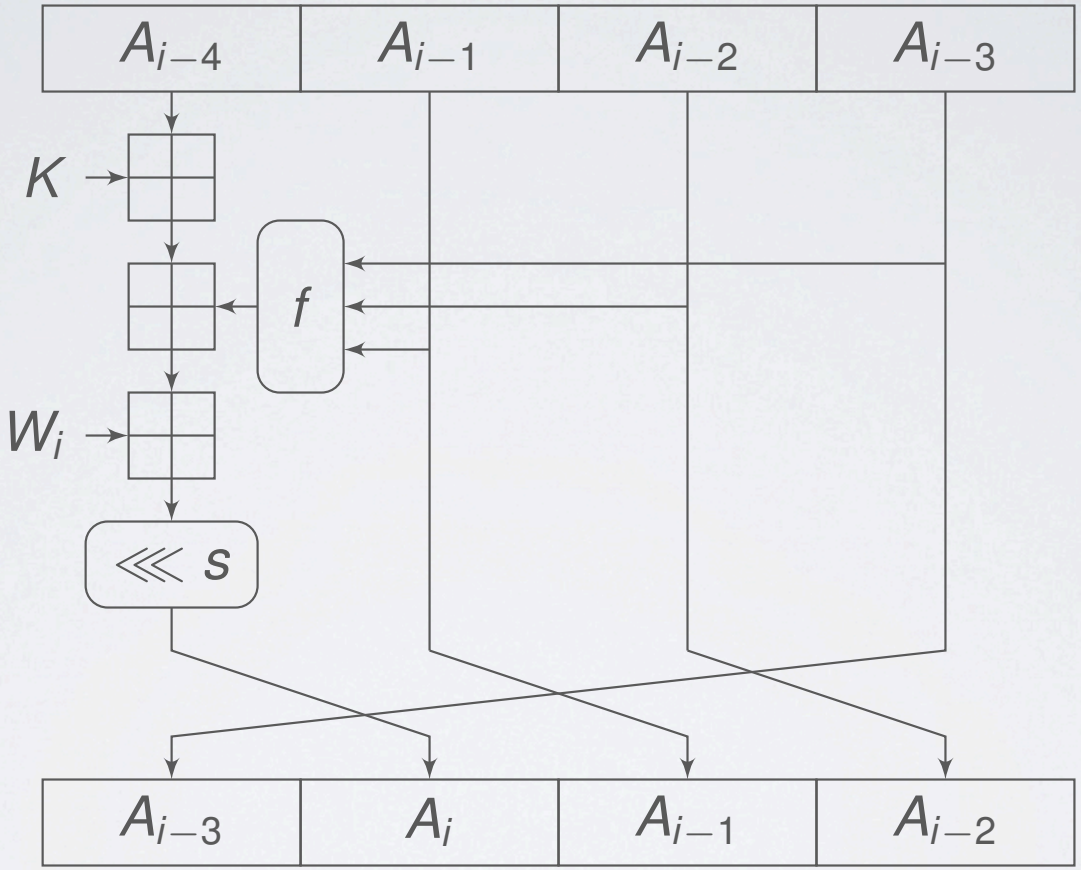


Message Expansion

- Permutation of the 16 message words in each round (16 steps)

steps	message word															
0–15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16–31	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15
32–47	0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15

Step function



Constructing a Differential Characteristic

Guess-and-Determine Attack

On a high level, a guess-and-determine attack can be described as a repetition of the following two steps

- guess the value of some unknowns
- determine the value of as many unknowns as is possible

until all unknowns have been determined.

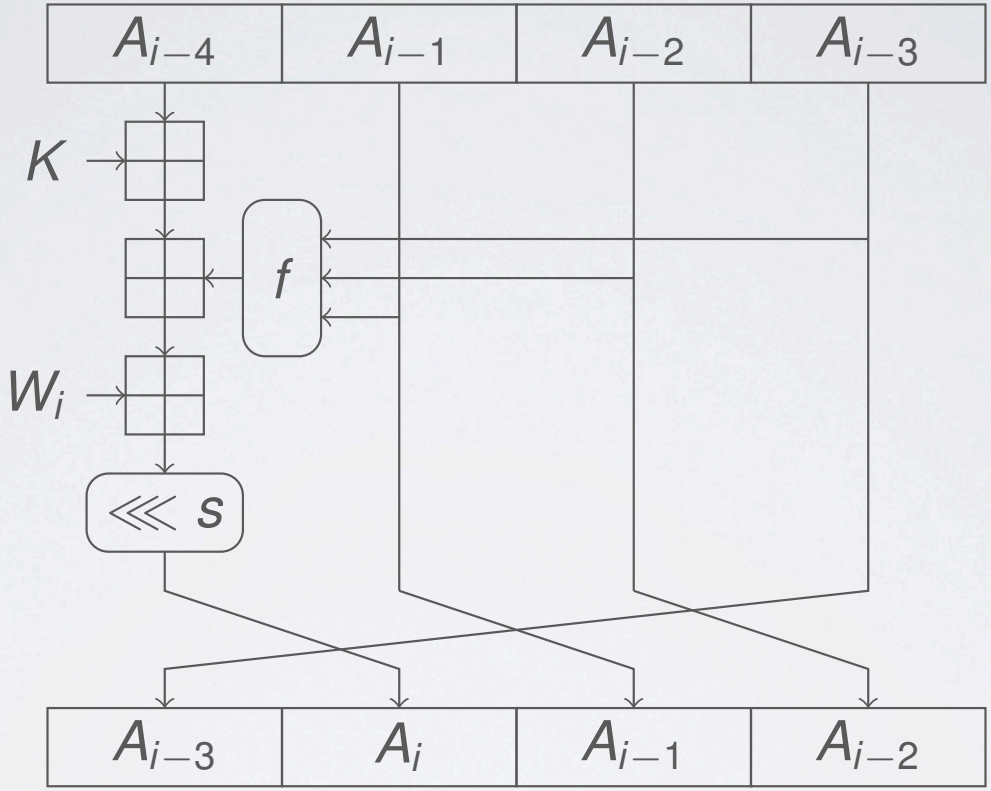
Guess-and-Determine Attack

A guess-and-determine attack works specially well if there are

- many sparse equations
- the set of equations can be split into a number of subsets with very few variables occurring in more than one subset

$\Rightarrow$ A successful attack employs a strategy to convert the complex and dense equations into a form that is more amenable to attack

Choice of the Intermediate Variables



- $A_i = (A_{i-4} + K + F_i + W_i) \lll s$
- $F_i = f(A_{i-1}, A_{i-2}, A_{i-3})$

Choice of the Information to Store

- All 16 possible conditions on a pair of bits are taken into account.

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

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

This is ideal for bitslice functions, but less ideal for functions that mix bits from different slices.

Search Algorithm

(1) Start with an unrestricted characteristic (only ‘?’)

(2) Successively impose new conditions on the characteristic (replace ‘?’ by ‘-’ and ‘x’ by ‘n’ or ‘u’)

(3) Propagate the conditions in a bitslice manner and check for consistency

- If a contradiction occurs then backtrack
- else proceed with step 2

(4) Repeat steps 2 and 3 until all bits of the characteristic are determined

Example

How to use the tool?

Getting Started

- Building the tool

```
cp local.cmake.template local.cmake
#modify local.cmake (select hash functions, etc.)
mkdir build && cd build
cmake ..
make
cp ../printconfig-example.xml printconfig.xml
#modify printconfig
```

- Running the tool (e.g. with config for MD4)

```
./tool ../hash/md4/chars/eurocryptWangLFCY05.xml
```

Directory Structure

```
hash
 |---md4
      |---chars
      |    |-----(some startpoints)
      |
      |---includes
      |    |-----md4.h
      |
      |---src
      |    |-----md4.cpp
      |
      |---testvectors
      |    |-----(automated tests)
      |
      |---sources.cmake
```

Startpoints

```
<config>
  <parameters>
    <parameter name="f" value="md4"/>
    <parameter name="s" value="48"/>
    <parameter name="w" value="32"/>
    <parameter name="z" value="main"/>
  </parameters>
  <char value="
    ...
  "/>
</config>
```

Y. Sasaki, L. Wang, K. Ohta, and N. Kunihiro:
New Message Difference for MD4
FSE 2007

X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu:
Cryptanalysis of the Hash Functions MD4 and RIPEMD
EUROCRYPT 2005

Search Strategy

```
<search credits="5000">
  <phase twobit_complete="1">
    <setting prob="1">
      <mask word="A"/>
      <mask word="W"/>
      <guess condition="?" choice_prob="1"/>
      <guess condition="x" choice_prob="0.01"/>
    </setting>
  </phase>
</search>
```

Search Strategy II

```
<search credits="5000">
  <phase twobit_complete="1">
    <setting prob="1">
      <mask word="A"/>
      <mask word="W"/>
      <guess condition="?" choice_prob="1"/>
      <guess condition="x" choice_prob="0.01"/>
    </setting>
    <setting prob="100">
      <mask word="A" steps="32-47"/>
      <guess condition="?" choice_prob="1"/>
      <guess condition="x" choice_prob="0.01"/>
    </setting>
  </phase>
</search>
```

Search Strategy III

```
<search credits="5000">
  <phase twobit_complete="1">
    <setting prob="1">
      <mask word="A"/>
      <mask word="W"/>
      <guess condition="?" choice_prob="1"/>
      <guess condition="x" choice_prob="0.01"/>
    </setting>
    <setting prob="100">
      <mask word="A" steps="32-47"/>
      <guess condition="?" choice_prob="1"/>
      <guess condition="x" choice_prob="0.01"/>
    </setting>
  </phase>
  <phase twobit_complete="1">
    <setting prob="1" ordered_guesses="1">
      <mask word="W"/>
      <guess condition="-" choice_prob="0.5"/>
    </setting>
  </phase>
</search>
```

How to add a new function?

Directory Structure

```
hash
 |---md4
      |---chars
      |    |-----(some startpoints)
      |
      |---includes
      |    |-----md4.h
      |
      |---src
      |    |-----md4.cpp
      |
      |---testvectors
      |    |-----(automated tests)
      |
      |---sources.cmake
```

Hash Function Description

```
#include "hash.h"

class Md4 : public Hash {

  public:
    Md4(int steps, int N = 32);

  ...

  protected:
    ConditionWord W[16];
    ConditionWord A[48 + 4];
    ConditionWord F[48];

   ...

};
```

```
Md4::Md4(int steps, int N) : Hash(N) {

  for (int i = -4; i < 0; i++)
    A[i] = AddConditionWord("A", i, 4 + i, 0);
  for (int i = 0; i < min(16, steps); i++)
    W[i] = AddConditionWord("W", i, 4 + i * 2 + 1, 1);

  for (int i = 0; i < steps; i++) {
    F[i] = AddConditionWord("F", i, 4 + i * 2 + 0, 1, SUBWORD);
    A[i] = AddConditionWord("A", i, 4 + i * 2 + 1, 0);

    if (i < 16)
      Add(new BitsliceStep<IF>(N, A[i-1], A[i-2], A[i-3], F[i]));
    else if (i < 32)
      Add(new BitsliceStep<MAJ>(N, A[i-1], A[i-2], A[i-3], F[i]));
    else
      Add(new BitsliceStep<XOR3>(N, A[i-1], A[i-2], A[i-3], F[i]));

    const int p = P[i]; const int s =  S[(i / 16) * 4 + i % 4];
    ConditionWord k(new ConditionWordImpl(K[i / 16]));
    Add(new CarryStep<ADD4>(N, A[i - 4], F[i], W[p], k, A[i]->Rotr(s)));
  }
}
```

Adding a bitslice function

```
class MAJ: public F {
public:
  static const int IN = 3;
  static const int OUT = 1;
  static const int NUM = IN + OUT;
  static constexpr char NAME[] = "MAJ";
  template<class T> static inline void f(T x[NUM]) {
    T a = x[0];
    T b = x[1];
    T c = x[2];
    T r = x[3];
    r = (a & b) ^ (b & c) ^ (c & a);
  }
};
```

$$MAJ : (a \wedge b) \oplus (b \wedge c) \oplus (c \wedge a)$$

Adding a bitslice function

```
class SBOX: public F {
public:
  static const int IN = 5;
  static const int OUT = 5;
  static const int NUM = IN + OUT;
  static constexpr char NAME[] = "SBOX";
  template<class T> static inline void f(T x[IN + OUT]) {
    T r0 = x[0]; T r1 = x[1]; T r2 = x[2]; T r3 = x[3]; T r4 = x[4];
    T t0, t1, t2, t3, t4;

    t0  = r0;    t1  = r1;    t2  = r2;    t3  = r3;    t4  = r4;
    t0 =! t0;    t1 =! t1;    t2 =! t2;    t3 =! t3;    t4 =! t4;
    t0 &= r1;    t1 &= r2;    t2 &= r3;    t3 &= r4;    t4 &= r0;
    r0 ^= t1;    r1 ^= t2;    r2 ^= t3;    r3 ^= t4;    r4 ^= t0;

    x[5 + 0] = r0; x[5 + 1] = r1; x[5 + 2] = r2; x[5 + 3] = r3; x[5 + 4] = r4;
  }
};
```

Adding a linear function

```
template<int R0, int R1, int R2>
class Sigma: public F {
public:
 static const int IN = 1;
 static const int OUT = 1;
 static const int NUM = IN + OUT;
 static constexpr char NAME[] = "SIGMA";
 static void f(int N, uint64 x[IN + OUT]) {
   x[1] = Rotr(x[0], R0, N) ^ Rotr(x[0], R1, N) ^ Rotr(x[0], R2, N);
 }
};
```

$$\Sigma : (x \ggg r_0) \oplus (x \ggg r_1) \oplus (x \ggg r_2)$$

Live Demo

Summary and Future Work

Summary

- Sophisticated tool to find differential characteristics
- Attacks on several popular hash functions
    - SHA-256/512, RIPEMD-128/160, SM3, ...

Other Applications

- Analysis of CAESAR candidates
- Analysis of block ciphers
- . . .

THANK YOU!

Questions?