



Università degli Studi “*Roma Tre*”

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Magistrale in Matematica

Cryptographic hash functions: algorithms for collision searching and lambda-calculus optimal reduction

Tesi di Laurea Magistrale
Anno Accademico 2007/2008

Laureanda
Simona Giovannetti
Matricola: 269957

Relatore
Marco Pedicini

KeyWords: Hash functions, Collision Attacks, Lambda-calculus, Optimal Reduction.
MSC AMS: 06e30, 94a60, 94a62, 03b40, 68n18.

1 Hash Functions and MD4

Since the early development of internet as a data communication system, there is a remarkable diffusion of data exchange even when they are private and official. This led to the study of a way of assuring the authentication of the sent document. This is due to the possible interceptions and potential alterations of the data, that can occur during the path between the sender and the receiver. A way of verifying this authenticity is the use of *digital signature*. The digital signature is an authentication of the digital documents which is similar to the traditional handwritten signature. A pair (document, signature) represents a signed document-record, or, to be more clear, a document to which has been attached a signature. The verifying algorithm can be used by anyone at any time in order to establish the authenticity of a document digital signature.

Hash functions are creating and verifying algorithms of the digital signature. We give their formal definition.

Definition 1. A *hash function* (or *unkeyed hash function*) is a function $h : \mathcal{X} \rightarrow \mathcal{Y}$, where \mathcal{X} is the set of possible messages and \mathcal{Y} is a finite set of possible message digests or authentication tags.

In an abstract level, a hash function is a function that compress a binary sequence of arbitrary length (generally very high) to another one with a fixed low length (see Figure 1). Therefore, because the common and typical use of hash functions, we give two additional important properties to be added to their definition.

Property 1 (Compression). h maps an input x of arbitrary finite bitlength, to an output $h(x)$ of fixed bitlength n .

Property 2 (Ease of computation). Given h and an input x , $h(x)$ is easy¹ to compute.

Henceforth, we use notation

$$h : \{0, 1\}^* \longrightarrow \{0, 1\}^n$$

for unkeyed hash functions that satisfy the two properties above. Moreover, we generally use m to identify input message.

Evidently, because the set length, a hash function could not be injective, or better still, it is strongly not injective. This is not enough: given an initial value, it has to be computationally hard compute one or more images

¹The term "easy" (like "computational infeasible") is intentionally left without definition: it is intended it be interpreted relative to an understood frame of reference. *Easy* might mean polynomial time and space, or within a certain number of machine operations or time units.

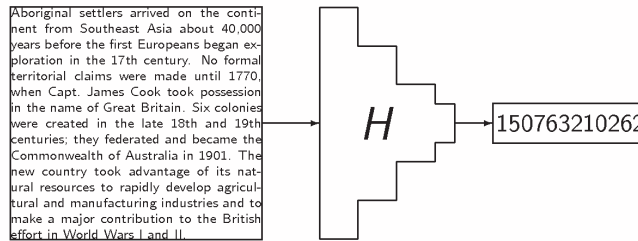


Figure 1: General idea for hash functions

that have the same hash value. This security concept is expressed in these three problems:

Definition 2. Given a hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ and a message digest y , the **Preimage** problem consists in finding an initial message m such that $h(m) = y$.

If Preimage can be solved for a given y , then (m, y) is a valid pair. A hash function for which Preimage cannot be efficiently solved is said to be *one-way* (**OWHF**)² or *preimage resistant*.

Definition 3. Given a hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ and an initial message m , the **Second Preimage** problem consists in finding m' such that $m' \neq m$ and $h(m') = h(m)$.

If Second Preimage can be solved, given $m \in \{0, 1\}^*$, $(m', h(m))$ is a valid pair. A hash function for which Second Preimage cannot be efficiently solved is said to be *second preimage resistant* (**CRHF**)³ or alternatively *weak collision resistant*.

Definition 4. Given a hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$, the **Collision** problem consists in finding $m, m' \in \{0, 1\}^*$ such that $m' \neq m$ and $h(m') = h(m)$.

If Collision can be solved, if $y = h(m) = h(m')$, the two pairs (m, y) and (m', y) are valid pairs. A hash function for which Collision cannot be efficiently solved is said to be *collision resistant* (**CRHF**)³ or alternatively *strong collision resistant*.

An "ideal" hash function is such that the only efficient way to determine the value of $h(m)$ for a given m is to actually evaluate the function h at the

²OWHF = One-Way Hash Function

³CRHF = Collision Resistant Hash Function. It is the same for second-preimage and collision case: we will demonstrate that they are equivalent

value m . It is important that this should remain true even if many other values $h(m_1), h(m_2), \dots$ have already been computed.

The *random oracle model* provides a mathematical model of an ideal hash function: in this model, a hash function h is chosen randomly, and we can access to h only through "the oracle". This means that we have not a formula or an algorithm to compute values of the function h : we can, just, query the oracle to compute the value $h(m)$.

A consequence of the definition of a random oracle model, is the following independence property.

Theorem 1. *Let h be an hash function chosen randomly. Let us suppose that the values $h(m_i)$ have been determined (by querying an oracle for h) for all $1 \leq i \leq k$. Then $\Pr[h(x) = y \mid h(m_1) \dots h(m_k)] = \frac{1}{2^n}$ for all $m \neq m_i \forall i$ and $\forall y \in \{0, 1\}^n$.*

Therefore, whatever is the number of evaluated terms, probability to know which will be hash evaluation of initial message m , is $\frac{1}{2^n}$. A true random oracle does not exist, but a good hash function "behave" like a random oracle.

For what we said, an algorithm in the random oracle model can be applied to any hash function; for this reason we study the complexity of the three problems defined above in this model. The algorithms we analyse are *randomized algorithms*: they can make random choices during their execution.

Preimage Following algorithm is a brute-force attack against preimage problem.

Algorithm 1: FIND-PREIMAGE (h, y, Q)

```

choose any  $m_1, \dots, m_Q \in 0, 1^*$ ,
for each  $m_i$  with  $1 \leq i \leq Q$ 
  do { if  $h(m_i) = y$ 
        then return ( $m_i$ )
      }
return (failure)

```

Theorem 2. *For any m_1, \dots, m_Q , the average-case success probability of Algorithm 1 is $\epsilon = 1 - (1 - \frac{1}{2^n})^Q$.*

Second Preimage Next algorithm tries solve the **Second Preimage problem**.

Algorithm 2: FIND-SECOND-PREIMAGE (h, m, Q)

```

 $y \leftarrow h(m)$ 
choose  $m_1, \dots, m_{Q-1} \in 0, 1^*$ 
for each  $m_i$  with  $1 \leq i \leq Q - 1$ 
  do  $\left\{ \begin{array}{l} \text{if } h(m_i) = y \\ \text{then return } (m_i) \end{array} \right.$ 
return (failure)

```

It is similar to the previous algorithm, in the steps and in the analysis.

Theorem 3. For any $\{m_1, \dots, m_{Q-1}\} \in 0, 1^*$, the success probability of Algorithm 2 is $\epsilon = 1 - (1 - \frac{1}{2^n})^{Q-1}$.

Collision Following algorithm is a brute-force attack against collision problem.

Algorithm 3: FIND-COLLISION (h, Q)

```

 $y \leftarrow h(m)$ 
choose  $\{m_1, \dots, m_Q\} \in 0, 1^*$ 
for each  $m_i$  with  $1 \leq i \leq Q$ 
  do  $y_{m_i} \leftarrow h(m_i)$ 
if  $y_{m_i} = y_{m_j}$  for some  $m_i \neq m_j$ 
  then return  $(m_i, m_j)$ 
  else return (failure)

```

This algorithm has a high base in the **birthday paradox**: it says that in a group of 23 people (obviously randomly chosen), at least two will celebrate birthday in the same day with probability at least 1/2. This fact allows to reduce complexity.

To compare rough complexities of above algorithms for a typical hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$, we can see the following table.

attack	rough complexity
preimage	2^n
2nd preimage	2^n
collision	$\sqrt{2^n} = 2^{\frac{n}{2}}$

For Preimage and Second Preimage algorithms we have to compute all variables m_i . In case of collision algorithm we have base of birthday paradox, and complexity go down to value $2^{\frac{n}{2}}$: for this reasons sometimes, to indicate collision brutal attack we speak of *birthday attack*.

A particular construction for unkeyed hash functions is *Iteration of hash functions*. It is composed by three steps:

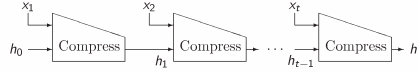


Figure 2: Compression by compression, every single part of padding product x_i takes part in the process in order to product an unique output h_t with length n .

preprocessing step: we fix a length n . Let x be an input string with $|x| \geq m + n + 1$. Using a *public algorithm* we construct a new string x' such that $|x'| \equiv 0 \pmod{n}$. Therefore, we have

$$x' = x_1 || x_2 || \dots || x_t$$

where $|x_i| = n$ for $1 \leq i \leq t$;

processing step: let IV be a *public initial value*, that is a bitstring of length n . We make the following computation (resumed in Figure 2):

$$\begin{array}{rcl} IV & \longrightarrow & h_0 \\ \text{compress}(h_0 || x_1) & \longrightarrow & h_1 \\ \text{compress}(h_1 || x_2) & \longrightarrow & h_2 \\ & \vdots & \vdots \\ \text{compress}(h_{t-1} || x_t) & \longrightarrow & h_t \end{array}$$

output transformation: we consider *public function* $g : \{0, 1\}^n \rightarrow \{0, 1\}^l$. We can define $h(x)$ output as $g(h_t)$.

A particular family that has this construction is the **MD-Family**. MDx-family consists of the most useful hash functions (*i.e.* MD4, MD5, SHA-1 etc.). In 1990, Rivest developed a new hash function of this family, MD4[1], for use in message integrity checks. It is again in use in computer world, for example, in the ed2k URI scheme to provide a unique identifier for a file in the popular eDonkey2000 / eMule P2P networks. Anyway, it was the base for the most popular and used hash function of this family MD5[2], developed in 1991. It has been employed in many security applications, and is also commonly used to check the file integrity especially for online softwre. It is even used by the Nevada State Gaming Authority to ensure slot-machine ROMs have not been tampered with.

MD4

Here we present **MD4** principal steps. With the **preprocessing step**, input message m is transformed in M that is

$$M = m_0 \parallel m_1 \parallel \dots \parallel m_{N-1}$$

where every m_i is a 32-bitstring, and N is multiple of 16. We can write M in a different way:

$$M = M[0] \parallel \dots \parallel M[d] \quad \text{where} \quad d = \frac{N}{16} - 1$$

in which $M[l] = [m_0^l, \dots, m_{15}^l]$, so M is composed by 16 words that we can recall with $M[l][j] = m_j^l$. For each $M[l]$, we apply MD4 **processing step**.

We denote with r_i a 32-bit variable that records the MD4-algorithm value in i -th step, where $0 \leq i \leq 47$. It is called *state variable*, because contains the state we have in step i . Fixed an initial value $IV = (r_{-4}, r_{-3}, r_{-2}, r_{-1})$, to evaluate state variables, we apply rule

$$r_i = (r_{i-4} + f_i(r_{i-1}, r_{i-2}, r_{i-3}) + m_{w_i} + k_i) \lll_{s_i}, \quad \text{for } 0 \leq i \leq 47$$

where

- w_i, k_i and s_i are step constants;
- f_i is a non-linear boolean function;
- \lll_{s_i} is the left circular rotation of s_i bits.

After last step, we define a new initial value $IV = (r_{44} + r_{-4}, r_{45} + r_{-3}, r_{46} + r_{-2}, r_{47} + r_{-1})$, and restart.

After d repetition of processing step, the **output** is given by the concatenation of the results of the last four process steps:

$$MD4(m) = r_{44} \parallel r_{45} \parallel r_{46} \parallel r_{47}$$

2 Attacks on MD-family and Optimal Reduction

Much of the studied hash function attacks have MD-family as target. One of the attacks which most influenced the MDx-family cryptography history, is Wang *et al* differential attack [3]. It gives a new beginning to go on with studies, because it is not automated: in its first presentation, the algorithm had some part in which intuition was used. One of the most significant result of automation research is given by Schl affer and Oswald

[4]: they present an algorithm on MD4 for searching differential path using signed difference and carry expansions.

Definition 5. Let x and x' be in $\{0, 1\}^{32}$. The signed difference Δx between x and x' , is defined bitwise by

$$\Delta x = x - x' = (\delta x_{31}, \dots, \delta x_0) \text{ with } \delta x_j = x'_j - x_j \in \{-1, 0, 1\}, 0 \leq j \leq 32$$

An abbreviation form of Δx is

$$\Delta x = \Delta[d_1, d_2, \dots, d_w] \quad \text{where } d_i = \begin{cases} j & \text{if } \delta x_j = 1 \\ -j & \text{if } \delta x_j = -1 \end{cases}$$

If we have a modular difference, we can obtain diverse signed difference, each one connected to the other by a specific rule called **Carry Expansion**.

Definition 6. Carry expansion on a signed difference $\Delta[d_1, \dots, d_w]$, is given by the rule

$$\begin{aligned} & \Delta[d_1, \dots, d_i, \dots, d_w] = \\ & = \begin{cases} \Delta[d_1, \dots, -d_i, \dots, d_w] + \Delta[d_i + 1] & \text{if } \text{sign}(d_i) = 1 \\ \Delta[d_1, \dots, -d_i, \dots, d_w] + \Delta[-(|d_i| + 1)] & \text{if } \text{sign}(d_i) = -1 \end{cases} \quad (1) \end{aligned}$$

Schl affer and Oswald find relations between input differences and propagations of them during the computation. Their algorithm is composed by three steps. We report their general aspects.

target differences computation: for every step we determine the target output difference Δt_i for the function f_i , that are the desired difference we want to obtain as output of function f_i at step i . To obtain target differences, the message differences are computed backward and forward in the MD4 algorithm.

cancellation search: in each step i , we try to achieve differences in relation to the target difference t_i , using function f_i and carry expansions. In this step we found several conditions to evaluate differential path.

correction step: often, in previous step we obtain *impossible paths*, that are differential paths with at least one contradiction. In such impossible paths a specific target difference cannot be met in some step or a zero output difference of the function f_i cannot be achieved. As a consequence, these additional (disturbance) differences induced by the contradictions, need to be cancelled in some other step.


From automation of Schl affer and Oswald, we thought of trying something even more general: we reread cryptographic hash functions with the more abstract λ -calculus. Schl affer and Oswald try to find collision on MD4, researching a differential path. Using graph representation of λ -terms, we

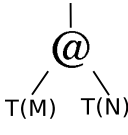
try to find "true" path on a graph that represent operations of the algorithm. To do this, we use optimal reduction technique on graphs: if a λ -term or a sub- λ -term is repeated, it is written only once. Shared parts correspond to partial collision of Schl affer and Oswald attack.

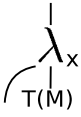
The graph representation of λ -terms are *Syntactic trees*.

Definition 7. Let M be in Λ . A syntactic tree of λ -term M is a labelled tree⁴ that represent the λ -term M . \mathcal{G} is the syntactic trees set.

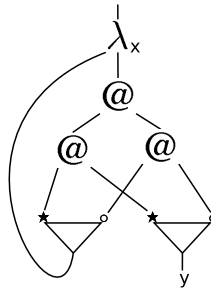
Definition 8. We define $T : \Lambda \rightarrow \mathcal{G}$ application from λ -term set to syntactic trees set. It is defined recursively:

1. let x be in V ; $T(x)$ is  that is called variable node;

2. let M, N be Λ ; then $T((M)N)$ is  that is called application node;

3. let M be in Λ and x be in V ; then $T(\lambda x M)$ is  that is called λ node: left edge of λ node is connected with each occurrence of bound x variable.

Example 1. Let M e λ -term $\lambda x((x)y)(x)y$. Variables x and y are twice used. The corresponding syntactic tree $T(M)$ is



Optimal reduction was introduced by L evy [5] and developed by Lamping [6]. It consists of modifying graphs without losing of sharing (if it is possible).

Using this technique, we tried to rewrite collision problem on MD4. This approach has been harder than we thought and we have not been able to

⁴A tree is a connected and acyclic graph.

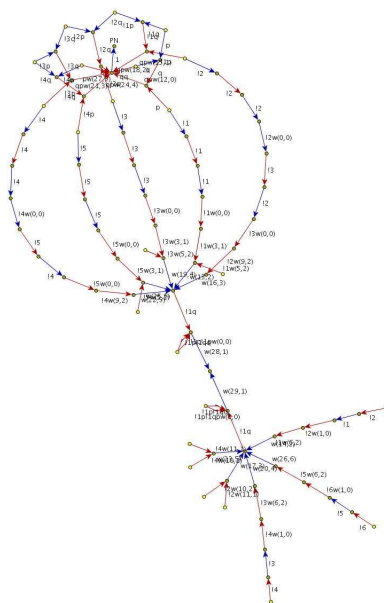


Figure 3: This is a 5-list of true results of functions AND and OR

bring it to an end. In any case we found a sharing propriety of the function results which constitutes a preliminary step for the study of new methodology for automatic research of collisions in hash functions.

Proposition 1. *Given two functions F_1 and F_2 so that $F_i : \{0, 1\}^n \rightarrow \{0, 1\}^m$, there exist two λ -terms, M_{F_1} and M_{F_2} , so that if $(M_{F_1})N = (M_{F_2})P$, the two results are shared in corresponding Lamping's graph.*

This kind of approach is an exhaustive search of function inputs. Its complexity is about $O(k2^n)$, where n is the input length and k function complexity. However, if we want to evaluate the function more times, we have to compute this search once. Therefore, if iteration is used, this approach can be advantageous. Moreover, we can choose to control either input or output: for example in both case Figure 3 and example above, we shared results, but in first case we check similar output; instead in second case, we check input sequences, so that we can have same result not shared: in Figure 4, we have eight different result for function f ; actually, they are **True** and **False**. We can share again, obtaining just two different result, but in this way we lose informations. Therefore, we separate input cases.

3 Thesis Conclusion

As in all cryptography, also in this thesis algebra, probability and computational theory are involved. This last has a particular importance, because

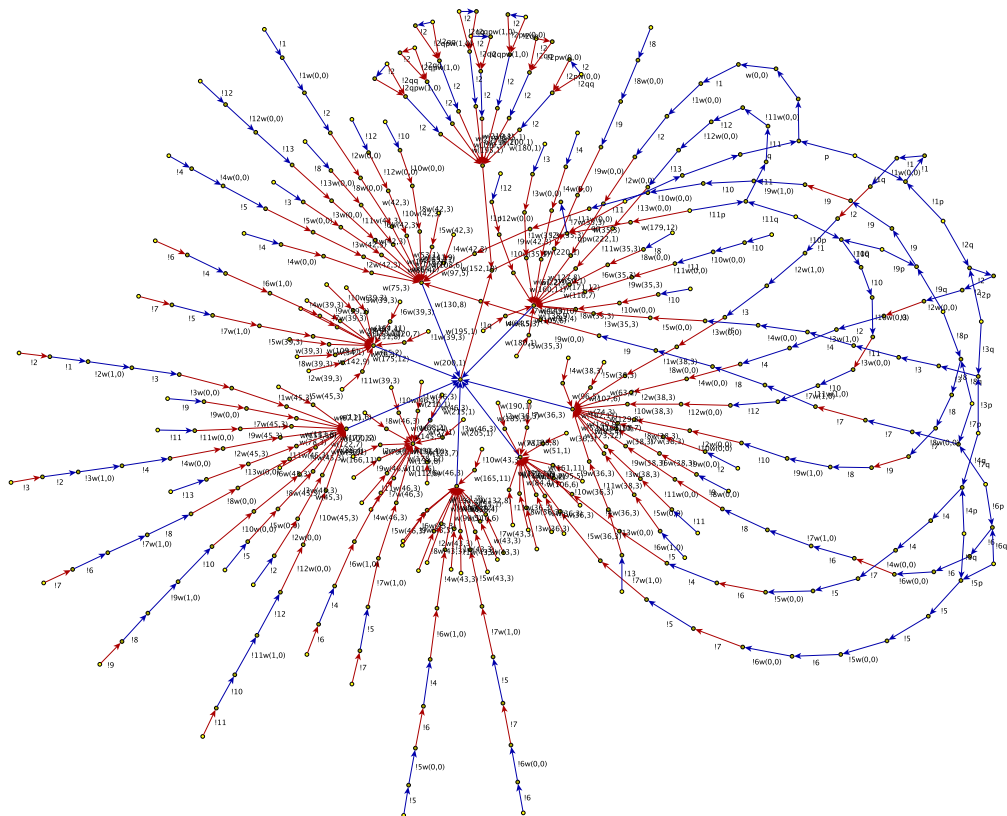


Figure 4: Here we present an example in which eight shared part (eight little triangles at the top of the graph) are achieved by twelve functions.

we use λ -calculus to introduce a new point of view for hash functions. In geometry of interactions, this graphs are represented as matrices: edge labels solve relations that define a particular algebra. Ultimate aim is to develop tools based on mathematics properties for finding hash function collisions. We think that this method with the property described in Proposition 1 could be useful for differential path search.

We were not able to find an attack algorithm, also because sometimes graph read back is not easy enough. Anyway, in this thesis we begin analysing hash function attacks from a new point of view.

References

- [1] R. Rivest. The MD4 Message-Digest Algorithm. RFC 1320 (Informational), April 1992.
- [2] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992.
- [3] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology – EURO-CRYPT 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.
- [4] Martin Schl affer and Elisabeth Oswald. Searching for Differential Paths in MD4. In Matthew J. B. Robshaw, editor, *Fast Software Encryption 2006, Proceedings*, volume 4047 of *Lecture Notes in Computer Science*, pages 242–261. Springer, 2006.
- [5] Jean-Jacques L evy. Optimal reduction on lambda-calculus. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 159–191, 1980.
- [6] J. Lamping. An Algorithm for Optimal Lambda Calculus Reduction. 1990.
- [7] Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45. 1998.
- [8] J. L. Krivine. *“Lambda-calculus, Types and Models”*. 1993.
- [9] Douglas R. Stinson. *Cryptography: Theory and Practice, Second Edition*. Chapman & Hall/CRC, February 2002.
- [10] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, 2004. <http://eprint.iacr.org/>.