MD5 message-digest algorithm

Franck Jeannot

Montréal, Canada, Septembre 2018, S562, v1.2

Abstract

A review of the **MD5, Message-Digest algorithm 5** by Ronald Rivest, with a focus on MD5 algorithm graphical representation (based on TikZ Latex module) and code implementations in various languages. This is a known vulnerable algorithm but still very interesting for study and historic review.

Keywords: APR1-MD5, collisions, MD5, RFC1321, Rivest, TikZ, SHA-1,2,3

1. Introduction

L'algorithme **MD5** (Message Digest 5) a été inventé par **Ronald Rivest** en 1991 au sein de *RSA Data Security, Inc* et a été spécifié via la **RFC 1321** [1] en 1992. C'est une fonction de hachage cryptographique produisant en sortie un « hash » de 128 bits (32 caractères de type Hexadécimal en sortie). Cet algorithme est essentiellement basé sur des entiers de 32 bits non signés (appelés *mots*) et des opérations de type XOR, OR, AND et rotations de bits.

ATTENTION, l'algorithme MD5 est **vulnérable**, voire section Vulnérabilités et collisions avec MD5. Il n'est donc plus recommandé de l'implémenter ou utiliser. Il n'est plus non plus recommandé d'utiliser **SHA-1** [2]. Il est recommandé d'utiliser en lieu et place **SHA-2** ou SHA-3 [3] [4]. Cet article fait un focus sur le fonctionnement de l'**algorithme MD5**.

1. https://tools.ietf.org/html/rfc1321
2. https://shattered.it/
3. https://en.wikipedia.org/wiki/SHA-3
4. https://github.com/rhash/RHash/blob/master/librhash/sha3.c

2. Préparation du message (ou *Préprocessing*)

Étape 1 : Ajout de bits de padding. Durant cette étape, le message d'origine, dont on veut obtenir le hash, est étendu ou découpé de telle manière que sa longueur totale en bits soit congruente à 448 *modulo* 512. Cette opération est toujours réalisée, même si la taille du message d'origine est déjà congruente à 448 *modulo* 512. Le "*padding*" est fait par l'ajout d'un simple bit "1" suivi par le nombre nécessaire de "0" bit de telle manière que la longueur en bits du message modifié devienne congruent à 448 modulo 512, cela signifie qu'il y aura toujours entre 1 et 512 bits ajoutés dans cette étape.

Étape 2 : Ajout de longueur au message. Une représentation de la longueur en bits du message d'origine M, sur de 64 bits, avant *padding*, est ajoutée au résultat de l' étape 1. Ainsi cette dernière partie contient la longueur du message d'origine M modulo 2^{64} soit $b \ mod \ 2^{64}$. Ces bits sont ajoutés comme deux mots de 32 bits (mot le moins significatif en premier).

Étape 3 : initialisation des buffers. Quatre buffers de 32 bits (A, B, C, D) sont utilisés pour un total de 128 bits pour les **V**aleurs d'**I**nitialisation (**IV** : Initialisation Values) avec les valeurs de 32 bits suivantes, stockées en **Little endian** [5] :

```
var int a0 := 0x67452301   //A
var int b0 := 0xefcdab89   //B
var int c0 := 0x98badcfe   //C
var int d0 := 0x10325476   //D
```

Une représentation de l'algorithme dans sa globalité est la suivante :

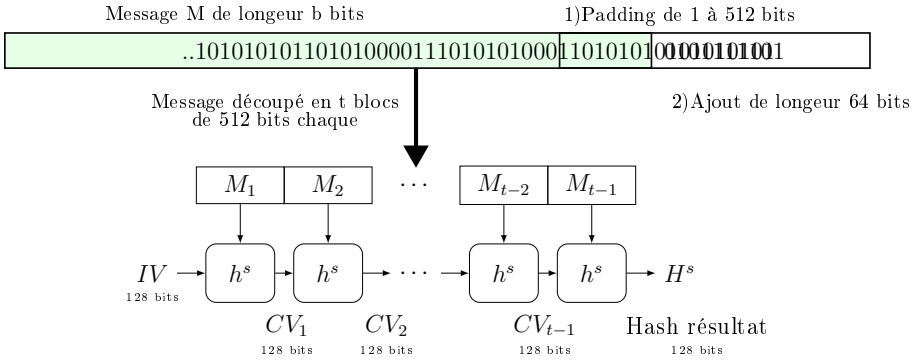

FIGURE *(1): Vue globale de l'algorithme MD5*

5. https://fr.wikipedia.org/wiki/Endianness#Little_endian

3. Représentation graphique de l'algorithme général MD5 après prépocessing

3.1. Algorithme général

L'algorithme général (parfois appelé "*compression function*") du MD5, après le préprocessing, consiste en 4 rondes, et chaque ronde dispose de 16 **étapes** (voir sous-section 3.2 Fonction étape de compression) pour un total de 64 opérations au total.

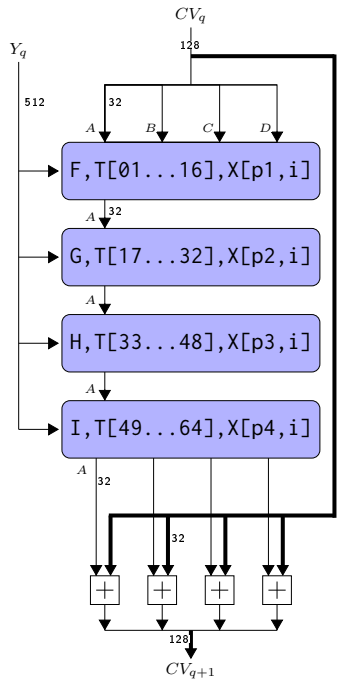

FIGURE *(2): Algorithme général MD5. Nota : par simplification du schéma, les flèches représentatives pour B,C,D ne sont pas positionnées entre chaque fonction, noter quelles existent. Crédit Franck Jeannot, Tikz, avec des bases inspirées de William Stallings, Cryptography and Network Security : Principles and Practice*

Y_q : le q^{eme} bloc de 512 bits (du message d'entrée)
$IV = CV_0$: Valeur d'initialisation (***I**nitialization **V**alue*) stockée dans les buffers ABCD
CV_q : variable de chainage (***C**haining **V**ariable*) calculée avec le q^{eme} bloc
RF_x : fonction de ronde (Round Function) utilisant la fonction logique primitive x
$X[k]$: $M[q * 16 + k]$ le k^{eme} mot de 32 bits dans le q^{eme} bloc de 512 bits du message
$T[i]$: $2^{32} * abs(sin(i))$ avec i en radians
CV_{L-1} : MD= "Message Digest value" ou valeur finale du hash sur 128 bits pour un message de L blocs
CV_{q+1} : $SUM_{32}[CV_q, RF_I(Y_q, RF_H(Y_q, RF_G(Y_q, RF_F(Y_q, CV_q))))]$

3.2. Fonction étape de compression

C'est la structure de calcul de chaque ronde dite "*step function*" ou "*computational structure*" :

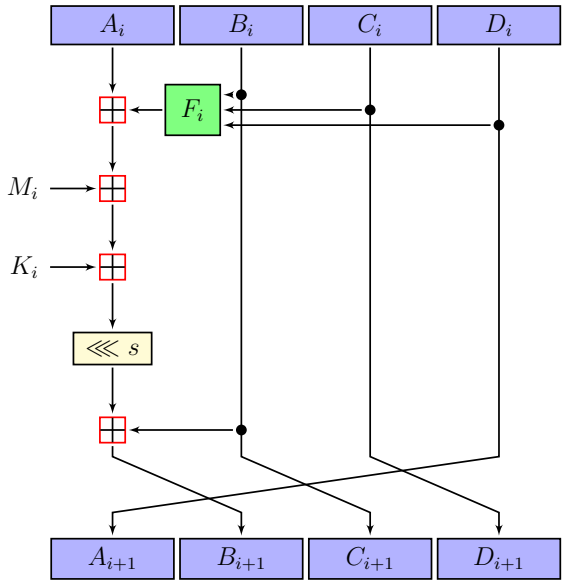

FIGURE *(3): Fonction étape de compression du MD5. L'algorithme complet fait appel à 64 de ces opérations, groupées en 4 rondes de 16 opérations.*[6]

$A_i \dots D_i$: 4 mots de 32 bits, l'algorithme opère sur un état de 128 bits
F_i : une fonction non-linéaire, une fonction est utilisée à chaque ronde
M_i : dénote un bloc de 32 bits du message d'entrée
K_i : dénote une constante de 32 bits, différente à chaque opération
$\boxplus$: dénote une addition modulo 2^{32}
$\lll s$: dénote une rotation de bits à gauche de s places, s variant à chaque opération

6. Adapté de https://www.iacr.org/authors/tikz/tikz/Hash%20Functions/md5.tex

Les constantes K_i de l' étape de compression MD5 sont issues des parties entières de la fonction sinus (radians), comme on peut le représenter ci-dessous :

```
#Pseudocode:
for i from 0 to 63
K[i] := floor(2^32 × abs(sin(i + 1)))
end for

latex-->K{[i]} = 2^32*abs(sin(i))}$
PHP  -->K[$i] = floor(abs(sin($i + 1)) * (pow(2, 32))) & 0xffffffff;
Java -->K[i]  = (int)(long)((1L << 32) * Math.abs(Math.sin(i + 1)));
python->K[i]  = [int(math.floor(abs(math.sin(i + 1)) * (2 ** 32))) for i in range(64)]
```

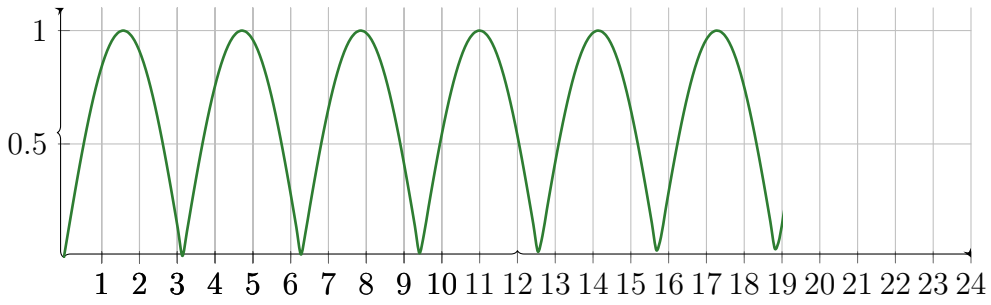

FIGURE *(4): La fonction* $f(x) = |sin(x)|$

Après formatage et calcul final, les constantes des 64 étapes deviennent :

```
K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
K[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
K[32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
K[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }
```

4. Des implémentations en différents langages de MD5

Une source très complète est le site `rossettacode.org`[7] qui couvre 36 langages différents d'implémentation de MD5 !

4.1. Des exemples d'implémentations en Lua de md5

Librairie pour Lua 5.0 (Roberto Ierusalimschy - 2003), c'est une encapsulation de librairie compilée en C (gcc pour linux)[8]

```
...
local function invert (s)
return md5.exor(s, string.rep('\255', string.len(s)))
end
x = string.rep('0123456789', 1000)
assert(md5.exor(x,x) == string.rep('\0', 10000))
assert(md5.exor(x,invert(x)) == string.rep('\255', 10000))
assert(invert(invert('alo alo')) == 'alo alo')
assert(invert(invert(invert('alo\0\255alo'))) == invert('alo\0\255alo')
    )
-- test some known sums
assert(md5.sumhexa("") == "d41d8cd98f00b204e9800998ecf8427e")
assert(md5.sumhexa("a") == "0cc175b9c0f1b6a831c399e269772661")
assert(md5.sumhexa("abc") == "900150983cd24fb0d6963f7d28e17f72")
...
```

Listing 1: Extrait d'exemple de code Lua de l'algorithme MD5

4.2. Des exemples d'implémentations en C et C++ de md5

1. **md5sum GNU Core utils**[9] (Ulrich Drepper - 1995)
2. Fourmilab[10] (John Walker - 2012) (Base code Colin Plumb, 1993 sans prototypes)
3. Le projet **Apache** et son implémentation APR1-MD5 (Poul-Henning Kamp)[11]
4. HashCheck Shell Extension DLL (Kai Liu, 2009)[12]

7. https://rosettacode.org/wiki/MD5/Implementation
8. http://www.inf.puc-rio.br/~roberto/md5/md5-5/md5.html
9. https://github.com/coreutils/coreutils/blob/master/src/md5sum.c
10. https://www.fourmilab.ch/md5/md5.zip
11. https://svn.apache.org/viewvc/apr/apr-util/branches/1.7.x/crypto/apr_md5.c?view=markup
12. http://code.kliu.org/hashcheck/

5. Version de **Colin Plumb (1993 !)**, avec Fix Apple "*Fixed endian-dependent cast in MD5Final; byteReverse() tolerant*" [13], une version très similaire est utilisée pour **isomd5sum** [14]
6. **WinMD5sum** [15] (GUI de *pidalu* [16], fonctionnel, cependant commentaires en Mandarin dans le code C++...)

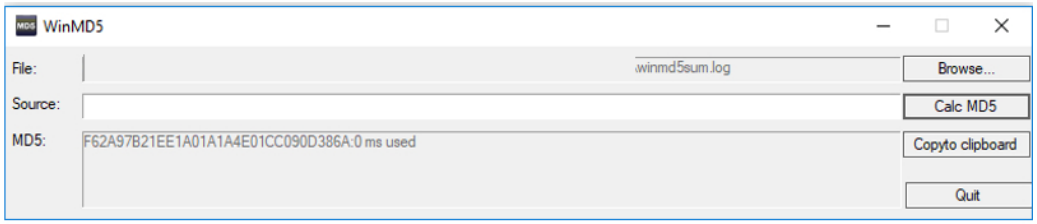

FIGURE *(5): Exemple de l'interface basique de winmMD5sum*

4.3. Des implémentations en javascript de md5

4.3.1. Extrait Code de "Dustin C. Fineout " (2009)

Voir extrait avec mise en avant des fonctions F,G,H,I... etc. [17] [18]

```
/**
* Usage example:
* var md5 = new MD5();
* alert(md5.hash('Dustin Fineout')); // 8844
    be37f4e8b3973b48b95b0c69f0b1
**/
function MD5()
{
this.F = function(x,y,z) { return (x & y) | ((~x) & z); };
this.G = function(x,y,z) { return (x & z) | (y & (~z)); };
this.H = function(x,y,z) { return (x ^ y ^ z); };
this.I = function(x,y,z) { return (y ^ (x | (~z))); };
this.C = function(q,a,b,x,s,ac) { return this.addu(this.rol(this.
    addu(this.addu(a,q),this.addu(x,ac)),s),b); };
```

13. https://opensource.apple.com/source/Security/Security-28/AppleCSP/MiscCSPAlgs/MD5.c
14. https://github.com/rhinstaller/isomd5sum/blob/2f0df17f636232178072ec02522e3c3ca6e6dbde/md5.c
15. https://sourceforge.net/projects/winmd5sum/files/winmd5en-src.rar/download
16. https://sourceforge.net/u/pidalu/profile/
17. https://stackoverflow.com/questions/997284/how-does-md5sum-algorithm-work
18. https://stackoverflow.com/users/120990/defines

```
    this.FF = function(a,b,c,d,x,s,ac) { return this.C((b & c) | ((~b)
        & d),a,b,x,s,ac); };
    this.GG = function(a,b,c,d,x,s,ac) { return this.C((b & d) | (c &
       (~d)),a,b,x,s,ac); };
    this.HH = function(a,b,c,d,x,s,ac) { return this.C(b ^ c ^ d,a,b,x
       ,s,ac); };
...
    this.rol = function(v, s)
    {
    return (v << s) | (v >>> (32 - s));
    };
    .......
```

Listing 2: Extrait d'exemple de code Javascript de l'algorithme MD5

4.3.2. Code de Sebastian Tschan "blueimp "

Voir page Github de "blueimp " [19].

```
/*
* Bitwise rotate a 32-bit number to the left.
*/
function bitRotateLeft (num, cnt) {
    return (num << cnt) | (num >>> (32 - cnt))
}
```

Listing 3: Extrait d'exemple de code Javascript de l'algorithme MD5 et la rotation de bits à gauche

4.4. Des implémentations en PHP de md5

4.4.1. Typage PHP, opérateur de comparaisons "==" et "==="

1. Un problème connu en PHP vient du typage, des opérateurs de comparaisons qui peuvent être utilisés et ceux qui doivent être utilisés, un mauvaise usage et cela rajoute des vulnérabilités d'implémentation de l'algorithme MD5 [20] [21]

4.4.2. APR1-MD5 et implémentations PHP

Il faut noter que certaines variantes du MD5 consistent à multiplier le nombre d'itérations utilisant l'algorithme MD5 X fois. Le projet Apache utilise l'algorithme **APR1-MD5**, c'est une fonction de hachage qui **utilise MD5 comme base** avec 1000

19. https://github.com/blueimp/JavaScript-MD5
20. https://cryptologie.net/article/268/how-to-compare-password-hashes-in-php/
21. https://d7x.promiselabs.net/2018/02/01/md5-collisions-and-the-way-php-interprets-types/

itérations[22][23]. **APR1-MD5** est par exemple utilisé pour les mots de passe dans les fichiers *.htaccess*. Dans l'exemple plus bas on affiche les hash respectifs pour les variantes indiquées de MD5[24].

```
pass        : blahblah
MD5x2       = md5(md5($pass))
MD5x3       = md5(md5(md5($pass)))
MD5         : 42d388f8b1db997faaf7dab487f11290
MD5x2       : e9ac1bcecd4b466b58330717d92b1367
MD5x3       : 36ec04164cc7e6bac91e3e16cb22a2a9
MD5x4       : 681c8e572b95b85a01259736882adb3b
MD5x5       : f35255c899028797f542e27d8de46be7
MD5(Reverse): 9012f187b4daf7aa7f99dbb1f888d342
MD5(Unicode): 3081dc08a22c7774af6dfe538b3f8ac5
```

4.5. Des implémentations en python de md5

1. Générations de collisions MD5[25]

4.6. Des implémentations en Latex de md5

La commande **pdfmdfivesum** est intégrée avec **pdfTEX** :

```
1 \texttt{\pdfmdfivesum{Hello World}}
```

Listing 4: Exemple de commande latex (pdfTEX)

Résultat : B10A8DB164E0754105B7A99BE72E3FE5

22. https://fr.wikipedia.org/wiki/APR1
23. https://github.com/whitehat101/apr1-md5/blob/master/src/APR1_MD5.php
24. Source : https://hashkiller.co.uk/hash-a-password.aspx
25. https://github.com/thereal1024/python-md5-collision

5. Vulnérabilités et collisions avec MD5

Dés 1993 après la publication de 1991 de Ronald Rivest de MD5 **B. den Boer and A. Bosselaers** [1] ont montré des faiblesses dans MD5, établissant alors des « pseudo-collisions ». Parmi la littérature importante sur le sujet des vulnérabilités et des collisions MD5, on notera notamment :

5.1. 2005

Collisions de certificats X.509 basées sur des collisions MD5 par **Wang et al.** [2] [26]

5.2. 2006

— Tunnels in Hash Functions : MD5 Collisions Within a Minute [3]
— Target Collisions for MD5 and Colliding X.509 Certificates for Different Identities [4]
— Fast Collision Attack on MD5 [5]
— Page de Peter Selinger sur les collisions MD5 : "MD5 Collision Demo" [6]

5.3. 2007

— 2007 : On collisions for MD5 (Master thesis de Peter Selinger) [7]

5.4. 2009

— Finding Preimages in Full MD5 Faster than Exhaustive Search [8]

5.5. 2010

— Collisions MD5 sur un seul bloc de 512 bits (Marc Stevens) [27]
— Collisions par Tao Xie et al. [9]

5.6. 2012-2013

— 2012 : Annonce de fin de support de MD5 *Crypt* par Poul-Henning Kamp [28] [29]
— 2013 : **Fast Collision Attack on MD5** [10]
— Decrypters : [30] [31] [32] [33]

26. https://www.win.tue.nl/~bdeweger/CollidingCertificates/ddl-full.pdf
27. https://eprint.iacr.org/2010/643.pdf
28. http://phk.freebsd.dk/sagas/md5crypt_eol.html
29. https://www.systutorials.com/docs/linux/man/n-md5crypt/
30. https://hashkiller.co.uk/md5-decrypter.aspx
31. http://md5decrypt.net/en/
32. https://hashkiller.co.uk/db-info.aspx
33. http://www.nitrxgen.net/md5db_info/#api

— Visualisations de collisions MD5[34]
— Patrick Stach : code C de générations de collisions MD5 (Algo de Wang)[35]
— Evilize (Peter Selinger) : Création de paires d'executables avec le même hash MD5[36]
— Projet **HasClash**[37][38][39][40][41]
— Articles de "Zoltan" sur la nécessité d'arrêter d'utiliser MD5[42]
— 2014 : Nat McHugh two PHP files with the same MD5 hash[43]
— 2015 : Collisions par Nat McHugh[44][45]

34. https://www.links.org/?p=6
35. http://www.securiteam.com/tools/6O00E1FEKO.html
36. https://github.com/silentsignal/sheep-wolf/tree/master/evilize
37. https://marc-stevens.nl/p/hashclash/
38. https://github.com/cr-marcstevens/hashclash
39. https://github.com/cr-marcstevens/hashclash
40. https://www.win.tue.nl/hashclash/
41. https://marc-stevens.nl/research/
42. http://www.stopusingmd5now.com
43. https://natmchugh.blogspot.com/2014/10/how-i-made-two-php-files-with-same-md5.html
44. https://natmchugh.blogspot.com/2015/05/how-to-make-two-binaries-with-same-md5.html
45. https://github.com/natmchugh/longEgg

Références

[1] den Boer, Bert and Bosselaers, Antoon, Collisions for the compression function of MD5 (1994) 293–304.
URL https://www.esat.kuleuven.be/cosic/publications/article-143.pdf

[2] Arjen Lenstra and Xiaoyun Wang and Benne de Weger, Colliding x.509 certificates.
URL https://eprint.iacr.org/2005/067

[3] Vlastimil Klima, Tunnels in hash functions : Md5 collisions within a minute.
URL CryptologyePrintArchive,Report2006/105

[4] Marc Stevens and Arjen K. Lenstra and Benne de Weger, Target Collisions for MD5 and Colliding X.509 Certificates for Different Identities, IACR Cryptology ePrint Archive 2006 (2006) 360.
URL https://eprint.iacr.org/2006/360.pdf

[5] M. Stevens, Fast Collision Attack on MD5, IACR Cryptology ePrint Archive 2006 (2006) 104.
URL https://homepages.cwi.nl/~stevens/papers/eprint2006-104%20-%20Fast%20Collision%20Attack%20on%20MD5.pdf

[6] Peter Selinger, MD5 Collision Demo, Dalhousie University.
URL https://www.mscs.dal.ca/~selinger/md5collision

[7] M. Stevens, On collisions for MD5, Eindhoven University of Technology.
URL https://www.win.tue.nl/hashclash/On%20Collisions%20for%20MD5%20-%20M.M.J.%20Stevens.pdf

[8] Sasaki, Yu and Aoki, Kazumaro, Finding Preimages in Full MD5 Faster Than Exhaustive Search (2009) 134–152.
URL https://iacr.org/archive/eurocrypt2009/54790136/54790136.pdf

[9] Xie, Tao and Feng, Dengguo, Construct MD5 collisions using just A single block of message, IACR Cryptology ePrint Archive 2010 (2010) 643.
URL https://eprint.iacr.org/2010/643.pdf

[10] Tao Xie and Fanbao Liu and Dengguo Feng, Fast Collision Attack on MD5, Cryptology ePrint Archive, Report 2013/170, https://eprint.iacr.org/2013/170 (2013).