KU LEUVEN

FACULTEIT
INGENIEURSWETENSCHAPPEN

Quasidifferential cryptanalysis of the MD5 hash function

Honours programme

Petar Vitorac

Supervisors:
Prof. dr. ir. Vincent Rijmen
Dr. ir. Tim Beyne

15 May 2024

Contents

Part A

Administrative information

1 Student

First name: Petar
Last name: Vitorac
Student number: r0924724
Study programme: *Bachelor in de ingenieurswetenschappen (Leuven)*
Stage: 2

2 Project

Project name: Quasidifferential cryptanalysis of the MD5 hash function
Course: *Onderzoeksgebonden track, optie A*, 9 ECTS (H0T78A)
Period: November 2023 - May 2024
Supervisors: prof. dr. ir. Vincent Rijmen, dr. ir. Tim Beyne

Part B

Project

3 Introduction

A cryptographic hash function such as MD5 is a one-way function where it should be impossible to find two different inputs that produce the same output (a so-called collision) within a reasonable timeframe. Cryptographic hash functions are used, among other things, to verify the authenticity of files and perform digital signatures. A hash collision can have serious consequences, as it can bypass the verification of the authenticity of a file or digital signature - hence the importance of researching them.

The goal of the project is to apply quasidifferential cryptanalysis, a new method introduced by Beyne and Rijmen [1], to the MD5 hash function. Collisions for MD5 have already been found by Wang and Yu [2], but this further research can contribute to speeding up collision searching (for example, by finding conditions on the input message that will reduce the amount of necessary calls to MD5) and can provide more insights for analysing more modern function (such as those from the SHA family) since they are based on MD hash functions.

This part of the report starts with a brief description of the MD5 function. It is followed by the quasidifferential analysis of an example cipher and then MD5 itself. Finally, possible further works are presented. Part C includes a reflection about my personal experience, and part D formulates a conclusion. The appendix is in part E.

4 Description of the MD5 hash function

MD5 [3] is a function which receives an input message of arbitrary length and outputs the 128-bit "hash" of the message. After the necessary padding, the message is split in multiple 512-bit blocks. This section (and the rest of the project) describes the behaviour of the MD5 function for a single 512-bit input block.

64 steps, split in four rounds of 16 steps, are iteratively applied. Each one modifies the four 32-bit state registers ($A[i], B[i], C[i], D[i]$ represent the states of the registers at the beginning of step $0 \leq i < 64$). Each round $0 \leq j < 4$ is characterised by a function $F_j(x, y, z)$ and a function $g_j(i)$. For the first round:

$$F_0(x, y, z) = (x \wedge y) \vee (x' \wedge z)$$
$$g_0(i) = i$$

For the second round:

$$F_1(x, y, z) = (x \wedge z) \vee (y \wedge z')$$
$$g_1(i) = (5i + 1) \pmod{16}$$

For the third round:

$$F_2(x, y, z) = x \oplus y \oplus z$$
$$g_2(i) = (3i + 5) \pmod{16}$$

For the fourth round:

$$F_3(x, y, z) = y \oplus (x \vee z')$$
$$g_3(i) = 7i \pmod{16}$$

With these functions defined, a general expression for the modification of the state during step i can be formulated:

$$A[i + 1] = B[i]$$
$$B[i + 1] = C[i]$$
$$C[i + 1] = D[i]$$
$$D[i + 1] = ((A[i] + F_j(B[i], C[i], D[i]) + M[g_j(i)] + K[i]) \lll S[i]) + B[i]$$

with $M[x]$ being the x-th 32-bit word of the message block, $S[i]$ and $K[i]$ being predefined constants, $\lll$ being the left rotate operation, and $+$ representing addition modulo 2^{32}.

5 Analysis of an example cipher

As an illustration of the applications of quasidifferential trails, an example cipher is briefly analysed.

5.1 Description

This cipher takes an 9-bit input and outputs 9 bits, depending on the four chosen 9-bit round keys. It consists of three layers and finishes with an XOR with a round key. Each layer performs a XOR with a round key, passes the data (separated in 3-bit blocks) through the same S-Box, and finally permutes the 9 bits as shown on figure 1.

5.2 Probability of a characteristic

Given the characteristic shown on figure 1 (a bold red line represents a difference in that bit), an average probability of 2^{-6} can be calculated for that characteristic (2^{-2} for each

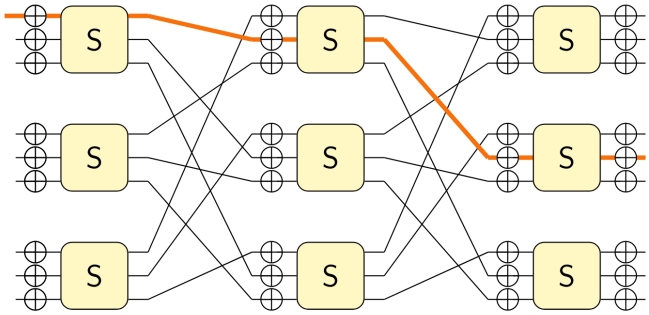

Figure 1: Visual representation of the example cipher, with a characteristic drawn (a red line means a difference on that bit). Image source: [4]

layer). Note that this does not factor in the choice of the keys. To obtain an exact, key-dependent probability, quasidifferential trails can be applied.[1]

Using a divide-and-conquer algorithm [1], the quasidifferential transition matrix can be computed. As for the XOR operations, they can flip the sign of the trail depending on the key, as later explained in section 6.2.1. By implementing an algorithm resembling path searching, it is possible to list the quasidifferential trails and their corresponding correlation, for a given key. By summing the correlations of the different trails, the total probability that the characteristic occurs is obtained. For example, if all bits of the round keys are zero, a probability of $9 \cdot 2^{-8}$ is obtained.

For each characteristic, it is also possible to build a general expression for the probability in function of the keys (without immediately substituting). For the characteristic on figure 1, the result is $2^{-6}+(-1)^{k_{2,8}+k_{3,4}}2^{-7}+(-1)^{k_{2,5}+k_{3,6}}2^{-7}+(-1)^{k_{2,8}+k_{3,4}+k_{2,5}+k_{3,6}}2^{-8}$, with $k_{i,j}$ the j-th bit of the key in the i-th layer. This is particularly useful: by analysing this expression, it is possible to control the probability of the characteristic by modifying the keys. This will be the goal of the analysis on MD5 (with the keys here being analogous to the message in MD5).

6 Analysis of the MD5 hash function

6.1 Characteristics

To perform a quasidifferential analysis, a characteristic of the function is needed. This project is based on the characteristic discovered by Wang and Yu [2]. However, that characteristic only fixes the differences on the states at the beginning of each step, not the intermediary differences (those after each operation inside a step). For a quasidifferential

[1]The source code is available here: `https://github.com/petar-vitorac/md5-quasidifferential/tree/main/example-cipher`

analysis, all the intermediary differences would be needed.

Results[2] show that the attack described by Wang and Yu does implicitly fix the difference at the output of the F function, but not the subsequent differences (at the output of each modular addition). [5] Hence, the analysis is performed on an arbitrarily chosen characteristic. Tables 1 and 2 in the appendix contain two different computed characteristics to illustrate this issue.

6.2 Modelling quasidifferential trails

This section describes the quasidifferential transition matrices of different individual operations used in MD5.

6.2.1 XOR and branching

The effect of XOR operations and branching for quasidifferential trails is analogous to that for linear trails (masks) and characteristics (differences).

XOR with a constant As shown on figure 2, the difference and mask at the input is equal to those at the output. However, the sign of the trail is flipped if $c^\top u$ is odd, with c the constant, and u the mask at the input.

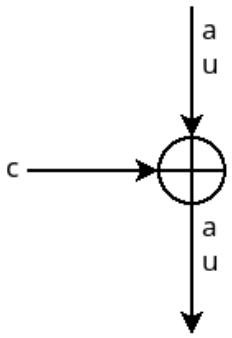

Figure 2: Differences and masks through an XOR between an input and a constant. A top line of text represents the difference at a certain place in the function, while the bottom line represents the mask.

XOR with two inputs As shown on figure 3, the difference at the output is equal to the XOR of the differences at the input. The masks at both inputs and at the output must be equal.

[2]The source code is available here: `https://github.com/petar-vitorac/md5-quasidifferential/tree/main/md5/differential`

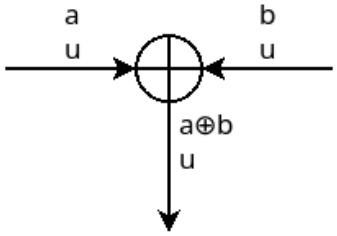

Figure 3: Differences and masks through an XOR between two inputs. A top line of text represents the difference at a certain place in the function, while the bottom line represents the mask.

Branching As shown on figure 4, the differences before and after the branching are equal. The mask at the input branch is equal to the XOR of the masks at the output branches.

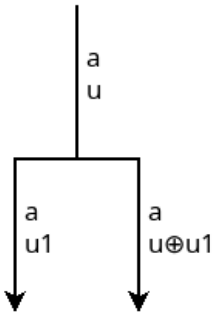

Figure 4: Differences and masks through a branching. A top line of text represents the difference at a certain place in the function, while the bottom line represents the mask.

6.2.2 F-functions

It is possible to formulate the F functions of rounds 1, 2 and 4 of MD5 as $F_0 = f(x, y \oplus z, z)$, $F_1 = f(x \oplus y, z, y)$ and $F_3 = f(x, z, y \oplus z) \oplus 1$ for the respective rounds, with $f(x, y, z) = (x \wedge y) \oplus z$. Given input masks and differences respectively u, v, w and a, b, c, and output mask and difference t and d, it can be shown [6] that the quasidifferential transition matrix of f is

$$D^f_{(t,d),(u\|v\|w,a\|b\|c)} = (-1)^{(a\wedge b\oplus a'\wedge b\wedge u)^T v\oplus(a\wedge v\oplus a'\wedge b\wedge u)^T(c\oplus d)} 2^{-wt(t\vee a\vee b)}$$

if and only if $t = w$, $a \wedge u \oplus b \wedge v = t \wedge (c \oplus d)$, $c \oplus d \leq a \vee b$ and $u \wedge v \leq t \vee a \vee b$. Otherwise, $D^f_{(t,d),(u\|v\|w,a\|b\|c)} = 0$.

Now, the input masks of F are defined as uB_1, uC_1, uD_1 and the output mask as v_1. The input differences are labelled aB, aC, aD and the output difference is b_1. The reason for this naming will become clear in section 6.2.3.

Then, the quasidifferential transition matrix can be expressed for the F-function in the first round, based on the propagation of quasidifferential trails through the XOR operation. A visual representation of the propagation of intermediary differences and masks are shown visually in figure 5.

$$D^{F_0}_{(v_1,b_1),(uB_1\|uC_1\|uD_1,aB\|aC\|aD)} = D^{f}_{(v_1,b_1),(uB_1\|uC_1\|uC_1\oplus uD_1,aB\|aC\oplus aD\|aD)}$$

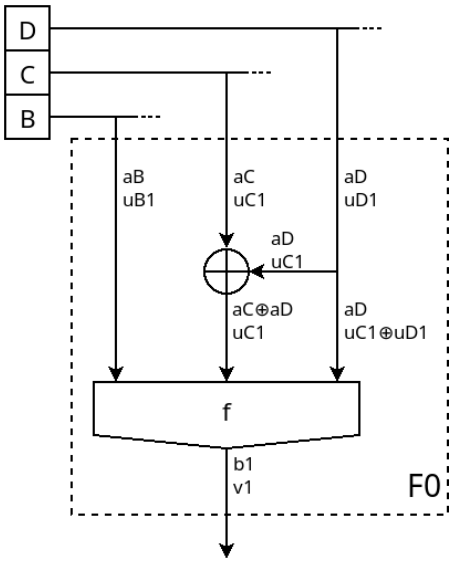

Figure 5: Differences and masks through F_0. A top line of text represents the difference at a certain place in the function, while the bottom line represents the mask.

Similarly, applying it to the F-functions of the second and fourth round respectively yields

$$D^{F_1}_{(v_1,b_1),(uB_1\|uC_1\|uD_1,aB\|aC\|aD)} = D^{f}_{(v_1,b_1),(uB_1\|uD_1\|uB_1\oplus uC_1,aB\oplus aC\|aD\|aC)}$$

and

$$D^{F_3}_{(v_1,b_1),(uB_1\|uC_1\|uD_1,aB\|aC\|aD)} = D^{f}_{(v_1,b_1),(uB_1\|uC_1\oplus uD_1\|uC_1,aB\|aD\|aC\oplus aD)}$$

Figures 6 and 7 display the visual representation.

On the other hand, F_2 simply consists of two consecutive XORs, and can be modelled as described in section 6.2.1.

6.2.3 A step of MD5

Figure 8 gives an overview of a step of MD5, whose notation this section uses.

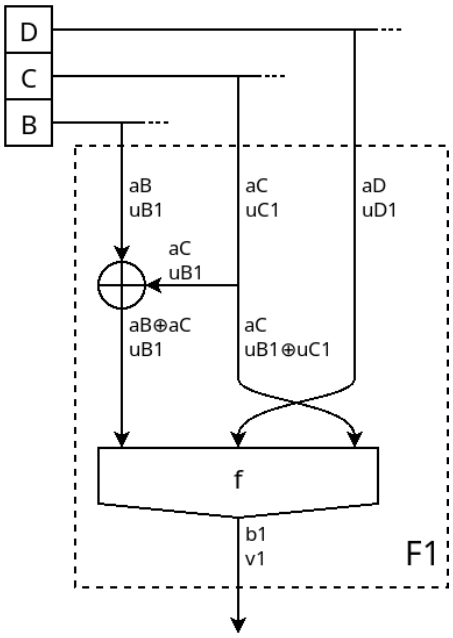

Figure 6: Differences and masks through F_1. A top line of text represents the difference at a certain place in the function, while the bottom line represents the mask.

In each step, an F-function, branching, modular addition, and bit rotation is used. The quasidifferential transition matrix for F has already been calculated and establishes relations between $uB_1[i], uC_1[i], uD_1[i]$ and $v_1[i]$. Branching rules yield assertions for the message masks (since the same message is used in four different steps) and transitions to the next step (as visible on figure 8):

$$\begin{aligned} uA[i+1] &= uB[i] \oplus uB_1[i] \oplus uB_2[i] \\ uB[i+1] &= uC[i] \oplus uC_1[i] \\ uC[i+1] &= uD[i] \oplus uD_1[i] \end{aligned}$$

As bit rotations simply permute the order of bits, it holds that

$$v_5[i] = v_4[i] \lll S[i]$$

Finally, it is possible to write a formula for modular addition [1].

6.3 Extracting conditions on the message

One way to improve the attack presented by Wang [2], is to reduce the necessary number of calls to MD5 by placing some conditions on the input message. Messages that do not meet this conditions will not need to be tested, thereby speeding up the attack.

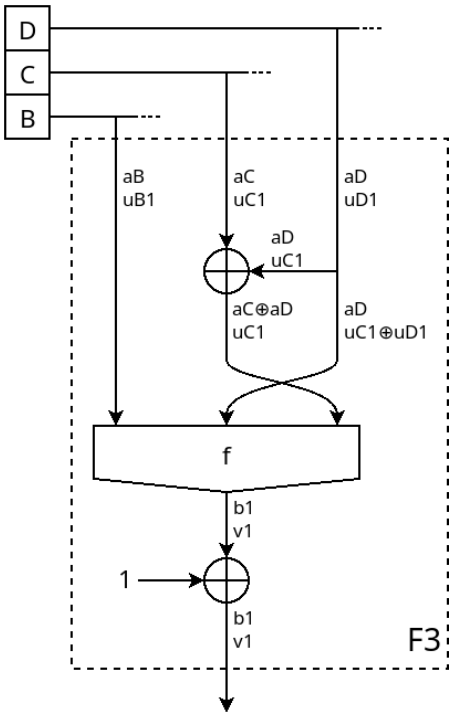

Figure 7: Differences and masks through F_3. A top line of text represents the difference at a certain place in the function, while the bottom line represents the mask.

Given a trail with sign $(-1)^s$, the following probabilistic relation can be established when the characteristic holds: $u^\top x = s$ with u the concatenation of all input and output masks, and x their actual values. [1] For MD5, this can be converted to an equation containing only the message and its mask by simply substituting the known values of K and the initial values of the A, B, C, D state registers. Finally, it is possible to enforce a null mask on the output (last state) so that it doesn't impact this calculation.

Applying this for multiple trails will create a system of probabilistic relations which are conditions on the message. Although it is probabilistic, section 6.5.3 shows how to get a deterministic system (which can then be solved).

6.4 Computing quasidifferential trails

In general, a "Satisfiability Modulo Theories" (SMT) solver can be used to determine whether a set of assertions on some unknowns is satisfiable. In this project, *Boolector* [7], an SMT solver for bit vectors, is used to compute possible values for the masks, thereby deducing the possible quasidifferential trails.[3]

[3]The source code is available here: `https://github.com/petar-vitorac/md5-quasidifferential/tree/main/md5/quasidifferential`

For MD5, all intermediary masks (as shown on figure 8) are represented as unknown 32-bit vectors, for each step. All differences (the characteristic) are chosen in advance and are treated as constants. By asserting the relations deduced in section 6.2.3, it is possible to generate masks belonging to a quasidifferential trail with non-zero correlation.

Evidently, trails having a higher probability are more relevant. Therefore, an expression representing the weight of a trail in function of the masks is constructed. Then, it is asserted that this expression equals a desired weight w. w starts at 0 (searching for trails with correlation $2^{-0} = 1$) and gets progressively increased until a trail of correlation 2^{-w} is found.

6.5 Filtering solutions

An SMT solver can only determine whether a solution to the given problem exists, and provide *one* such solution, which in this context translates to only one trail. Thus, it is necessary to use additional tricks to obtain all possible solutions.

6.5.1 Enforcing uniqueness

A naive approach is to keep track of all solutions found so far, and add assertions to the SMT model enforcing that the masks are different from any previously found solution. However, a trail that is a linear combination of previously found trails will not give any new condition on the input. Therfore, a problem arises when a lot of linearly dependent trails satisfy the model: it will not be possible to enumerate them all, and most of them will not provide any extra information.

6.5.2 Enforcing linear independence

Let u be the concatenation of all bit vector unknowns in the SMT model. Let n be the number of linearly independent solutions found by the SMT solver so far and u_i $(0 \leq i < n)$ the concatenation of all bit vector masks of the i-th linearly independent solution found by the SMT solver.

The goal is to express an assertion on u that will guarantee the next solution will not be a linear combination of the existing ones. Formally, given $A = [u_0^\top | u_1^\top | \dots | u_{n-1}^\top]$ an $m \times n$ matrix of full rank, the aim is to express conditions on u such that $[A|u]$ is also of full rank.

Consider T an $m \times m$ matrix such that $C = TA$ is in row echelon form. Then, $[A|u]$ will be of full rank if and only if Tu has a non-null value in the last $m - n$ rows: this can easily be asserted in the SMT model. Considering $[C|T] = \text{REF}([A|I_{m\times m}])$ with $I_{m\times m}$ the $m \times m$ identity matrix, it is easy to compute T by performing a row echelon reduction. [8]

6.5.3 Optimisation for deterministic trails

A "deterministic" quasidifferential trail is one whose absolute correlation is the same as that of the differential characteristic. [1] In this case, the linear relations determined by the trail (including those on the input) always hold. Hence, none will ever be conflicting in a characteristic with non-zero probability. Based on the observation that MD5 has a lot of deterministic trails, the analysis can be limited to those, which can lead to some optimisations.

In this scenario, all trails with the same mask on the message will yield the same conditions on the input. Hence, it is not necessary to consider all of them - one suffices. This means that when enforcing uniqueness and linear independence, not all masks need to be compared - only those on the message. This drastically reduces the size of the matrix A and the computation speed.

6.6 Results

By applying this methodology, it was possible to extract conditions on the input message. For the characteristics from tables 1 and 2, a basis for the message masks of deterministic trails has been computed and is included in tables 3 and 4 in the appendix. For each mask, s (0 or 1) is given such that $u^\top m = s$, with u the mask on the message and m the message itself. This respectively yields a total of 36 and 30 linearly independent conditions on the message for the characteristics in tables 1 and 2.

7 Future work

A considerable challenge that remains is reconciling the different characteristics of MD5 (formed by all possible combinations of intermediary differences) and deduce common conditions. A possible way to achieve is would be to model the quasidifferential transition matrix of an entire MD5 step (excluding the F-function) as a whole. This way, the same characteristic would be valid overall, as there would be no intermediary differences (except the one at the output of F, but it is implicitly determined by the differences on the state registers).

Part C

Personal experience

8 Self-reflection

At the start of the project, my main motivation was to challenge myself by learning more about a domain that interests me a lot, but which I have not had the chance to explore within my regular courses. This was perfectly accomplished, as I learned many new topics in cryptography, linked directly and indirectly with my research. For this, I am deeply grateful to my supervisors, Vincent Rijmen and Tim Beyne, who devoted much of their time to guide me through this project by offering their invaluable advice and insights.

In addition, my hope was to develop research-related skills, such as being able to read scientific literature and answering novel questions. This was successful: at the start of my project, I spent a considerable amount of time on reading research papers about various hashing algorithms and techniques applied to find weaknesses in them. Moreover, through the entire project, I adopted a researcher's way of thinking by performing hands-on experiments myself, which is new to me.

As I expected, this was also an excellent opportunity to perfect engineering competences. In fact, the conducted research was not purely theoretical - quite the contrary. It was focused on applying novel theory and performing practical experiments for solving problems: precisely what an engineer should be able to do.

Of course, my soft skills were also significantly improved. This includes but is not limited to creativity and communication.

During the year, I have also had interesting experiences which did not expect at the start. Among others, I had the chance to attend the Fast Software Encryption Conference in the last week of March, for which I wholeheartedly thank Siemen Dhooghe. Hearing many experts talk about their new research was a unique first experience that - in addition to providing many insights - definitely inspired me.

Finally, I would like to reflect on the mistakes I have made. In my opinion, the biggest one was being too focused on finding the issues in the solver that I wrote, instead of keeping the big picture in mind. This caused me spend weeks, if not months, dissecting the output and recalculating results by hand to find the origin of the inconsistencies. In the end, I found the few mistakes that were present in the calculation of the sign of a trail through the F-functions. The lost time meant I didn't have much time in the end to explore everything that I wanted. If I could start from the beginning, I would make sure to try working on other things while searching for the mistake, to clear my mind and improve my efficiency.

9 Link with study programme

For this project, skills and knowledge from various courses of the *Bachelor in de ingenieurswetenschappen* were used. Firstly, mathematical concepts from *Togepaste Algebra* and *Toegepaste Discrete Algebra* were very useful. Next, the programming skills from *Methodiek van de Informatica* helped in the implementation of the solver. Finally, the experience from *Probleemoplossen en Ontwerpen, deel 1/2/3* was applicable for problem-solving, planning, organising ideas, writing a report and preparing the presentation in this project.

10 Societal relevance

Researching the collision resistance of hash functions such as MD5 is of great societal relevance, namely for the preservation of data integrity and the security of digital signatures. The latter relies on a hash function to produce an output of fixed length, which can then be digitally signed instead of the initial message. Finding a hash collision would make it possible to falsify one's digital signature.

Moreover, techniques like quasidifferential cryptanalysis have much broader applications in cryptography: for instance, they can be applied on ciphers. Those also have a lot of relevance in society, ranging from private communication to secure data storage.

Part D

Conclusion

Overall, the goal of the project is reached: applying the quasidifferential methodology to MD5 to find ways to speed up the collision search and providing insights for analysing modern hash functions. It was possible to express conditions on the message input that are always true for a given characteristic, thereby reducing the search space. Moreover, parts of this analysis could also be applied to SHA functions - thereby completing the second goal.

However, there are still improvements to be made. The main one is being able to reconcile the conditions obtained for different characteristics and formulate more general conclusions.

On a personal note, this project has been very fulfilling. Both in terms of acquired knowledge in the domain of cryptography, and in terms of acquired competences related to research, engineering, and soft skills, I am extremely happy with my progress.

References

[1] T. Beyne and V. Rijmen, "Differential cryptanalysis in the fixed-key model," 2022. [Online]. Available: https://eprint.iacr.org/2022/837.

[2] X. Wang and H. Yu, "How to break MD5 and other hash functions," R. Cramer, Ed., pp. 19–35, 2005.

[3] R. L. Rivest, *The MD5 message digest algorithm*, Internet RFC 1321, Apr. 1992.

[4] T. Beyne, "Example cipher," unpublished.

[5] P. Stach, *Md5 collision generator*, 2005. [Online]. Available: https://packetstormsecurity.com/files/41548/md5coll.c.html.

[6] T. Beyne, "MD5 F-functies," unpublished.

[7] A. Niemetz, M. Preiner, and A. Biere, "Boolector 2.0," *J. Satisf. Boolean Model. Comput.*, vol. 9, no. 1, pp. 53–58, 2014. DOI: 10.3233/sat190101. [Online]. Available: https://doi.org/10.3233/sat190101.

[8] D. Lay, S. Lay, and J. McDonald, *Linear Algebra and Its Applications*. Pearson Education, 2021.

Part E

Appendix

11 Characteristics with intermediary differences in MD5

Table 1: Intermediary differences for a message that follows the characteristic discovered by Wang: M = [0x2dd31d1, 0xc4eee6c5, 0x69a3d69, 0x5cf9af98, 0x87b5ca2f, 0xab7e4612, 0x3e580440, 0x897ffbb8, 0x634ad55, 0x2b3f409, 0x8388e483, 0x5a417125, 0xe8255108, 0x9fc9cdf7, 0xf2bd1dd9, 0x5b3c3780]. The naming is consistent with that of figure 8.

Step	b_1	b_2	b_3	b_4
0	0x0	0x0	0x0	0x0
1	0x0	0x0	0x0	0x0
2	0x0	0x0	0x0	0x0
3	0x0	0x0	0x0	0x0
4	0x0	0x0	0x80000000	0x80000000
5	0x80800	0x80800	0x783800	0x80800
6	0x7fbc00	0x1c400	0x1cc00	0xbc00
7	0xa010424	0x76030464	0xa031de4	0x76010be4
8	0x81010540	0x87010500	0x83070f00	0x8106fd00
9	0x84900041	0xc100003	0xc100001	0x4100003
10	0x802041	0x18002000	0x80fe000	0x8002000
11	0x101	0x838300	0x860100	0x1fe0100
12	0x800e0080	0x200bf	0x2004f	0x20043
13	0x800fe000	0x1000	0x7000	0x1000
14	0x80040000	0x403c0000	0xc0040000	0x400c0000
15	0x82000000	0x607e180	0x1e006080	0xe006080
16	0x80000000	0x1000000	0x7000000	0x1000000
17	0x80000000	0x0	0x0	0x0
18	0x80000000	0x8008	0x8	0x78
19	0x80000000	0x20000000	0x20000000	0xe0000000
20	0x80000000	0x0	0x0	0x0
21	0x80000000	0x0	0x0	0x0
22	0x80000000	0xe0000	0x60000	0x20000
23	0x0	0x80000000	0x0	0x0
24	0x80000000	0x0	0x0	0x0
25	0x0	0x80000000	0x0	0x0
26	0x0	0x0	0x0	0x0

27	0x0	0x0	0x0	0x0
28	0x0	0x0	0x0	0x0
29	0x0	0x0	0x0	0x0
30	0x0	0x0	0x0	0x0
31	0x0	0x0	0x0	0x0
32	0x0	0x0	0x0	0x0
33	0x0	0x0	0x0	0x0
34	0x0	0x0	0x8000	0x8000
35	0x80000000	0x80000000	0x0	0x0
36	0x0	0x0	0x0	0x0
37	0x80000000	0x80000000	0x0	0x0
38	0x80000000	0x0	0x0	0x0
39	0x80000000	0x0	0x0	0x0
40	0x80000000	0x0	0x0	0x0
41	0x80000000	0x0	0x0	0x0
42	0x80000000	0x0	0x0	0x0
43	0x80000000	0x0	0x0	0x0
44	0x80000000	0x0	0x0	0x0
45	0x80000000	0x0	0x0	0x0
46	0x80000000	0x0	0x0	0x0
47	0x80000000	0x0	0x0	0x0
48	0x80000000	0x0	0x0	0x0
49	0x80000000	0x0	0x0	0x0
50	0x0	0x80000000	0x0	0x0
51	0x80000000	0x0	0x0	0x0
52	0x80000000	0x0	0x0	0x0
53	0x80000000	0x0	0x0	0x0
54	0x80000000	0x0	0x0	0x0
55	0x80000000	0x0	0x0	0x0
56	0x80000000	0x0	0x0	0x0
57	0x80000000	0x0	0x0	0x0
58	0x80000000	0x0	0x0	0x0
59	0x80000000	0x0	0x0	0x0
60	0x0	0x80000000	0x0	0x0
61	0x80000000	0x0	0x8000	0x8000
62	0x80000000	0x0	0x0	0x0
63	0x80000000	0x0	0x0	0x0

Table 2: Intermediary differences for a message that follows the characteristic discovered by Wang: $M = [$0xea2dc786, 0xc087389d, 0xc5f86464, 0xba921814, 0x14497da7, 0x5ecaf6d5, 0xe135826b, 0xd76678a6, 0x06352db1, 0x8473adf3, 0x8ba6b173, 0xd2b11966, 0xdff13cc3, 0x592f2f0d, 0x63768582, 0xf614e4a7$]$. The naming is consistent with that of figure 8.

Step	b_1	b_2	b_3	b_4
0	0x0	0x0	0x0	0x0
1	0x0	0x0	0x0	0x0
2	0x0	0x0	0x0	0x0
3	0x0	0x0	0x0	0x0
4	0x0	0x0	0x80000000	0x80000000
5	0x80800	0x80800	0x783800	0x80800
6	0x7fbc00	0xcc00	0x3cc00	0xbc00
7	0xa010424	0x1a0f0c24	0xa031de4	0x76010be4
8	0x81010540	0x87010500	0x83070f00	0x8101fd00
9	0x84900041	0xc100007	0x4100001	0x4100003
10	0x802041	0x8002000	0x803e000	0x8002000
11	0x101	0x1838300	0x860f00	0x19e0100
12	0x800e0080	0x20043	0x200c1	0x20041
13	0x800fe000	0x1000	0x1f000	0x1000
14	0x80040000	0xc03c0000	0x400c0000	0x401c0000
15	0x82000000	0x2002780	0x6006080	0x2002080
16	0x80000000	0x1000000	0xf000000	0x3000000
17	0x80000000	0x0	0x0	0x0
18	0x80000000	0x18008	0x8	0x18
19	0x80000000	0x20000000	0x60000000	0x20000000
20	0x80000000	0x0	0x0	0x0
21	0x80000000	0x0	0x0	0x0
22	0x80000000	0x20000	0x60000	0x20000
23	0x0	0x80000000	0x0	0x0
24	0x80000000	0x0	0x0	0x0
25	0x0	0x80000000	0x0	0x0
26	0x0	0x0	0x0	0x0
27	0x0	0x0	0x0	0x0
28	0x0	0x0	0x0	0x0
29	0x0	0x0	0x0	0x0
30	0x0	0x0	0x0	0x0
31	0x0	0x0	0x0	0x0
32	0x0	0x0	0x0	0x0
33	0x0	0x0	0x0	0x0
34	0x0	0x0	0x18000	0x8000

35	0x80000000	0x80000000	0x0	0x0
36	0x0	0x0	0x0	0x0
37	0x80000000	0x80000000	0x0	0x0
38	0x80000000	0x0	0x0	0x0
39	0x80000000	0x0	0x0	0x0
40	0x80000000	0x0	0x0	0x0
41	0x80000000	0x0	0x0	0x0
42	0x80000000	0x0	0x0	0x0
43	0x80000000	0x0	0x0	0x0
44	0x80000000	0x0	0x0	0x0
45	0x80000000	0x0	0x0	0x0
46	0x80000000	0x0	0x0	0x0
47	0x80000000	0x0	0x0	0x0
48	0x80000000	0x0	0x0	0x0
49	0x80000000	0x0	0x0	0x0
50	0x0	0x80000000	0x0	0x0
51	0x80000000	0x0	0x0	0x0
52	0x80000000	0x0	0x0	0x0
53	0x80000000	0x0	0x0	0x0
54	0x80000000	0x0	0x0	0x0
55	0x80000000	0x0	0x0	0x0
56	0x80000000	0x0	0x0	0x0
57	0x80000000	0x0	0x0	0x0
58	0x80000000	0x0	0x0	0x0
59	0x80000000	0x0	0x0	0x0
60	0x0	0x80000000	0x0	0x0
61	0x80000000	0x0	0x78000	0x8000
62	0x80000000	0x0	0x0	0x0
63	0x80000000	0x0	0x0	0x0

12 Conditions on the input message of MD5

Table 3: Conditions on the input message for the characteristic in table 1.

Mask on the message	s
0x0 0x0 0x0 0x0 0x0 0x0 0x28000 0x60030060 0x30e0600 0xc000000 0x10000000 0x200 0x0 0x4000 0x0 0x6060000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x3c800 0x60 0x60a00 0x3 0x10000000 0x0 0x8f 0x0 0x80000 0x6006000	0
0x0 0x0 0x0 0x0 0x0 0x0 0x24800 0x34000000 0x7040801	

0x14000001 0x0 0x30200 0x8f 0x4000 0x0 0x6000	0
0x0 0x0 0x0 0x0 0x0 0x0 0x18000 0x54030060 0x7020200 0x3 0x10000000 0x200 0x0 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x60030000 0x3080c00 0xc000003 0x0 0x200 0x8f 0x4000 0x80000 0x6000000	0
0x0 0x0 0x0 0x0 0x0 0x0 0x28000 0x60030060 0x30e0600 0xc000000 0x10000000 0x200 0x0 0x4000 0x0 0x6000000	0
0x0 0x0 0x0 0x0 0x0 0x0 0x18000 0x54030060 0x7020200 0x3 0x10000000 0x30200 0x0 0x4000 0x0 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x30000 0x34000000 0x40c0400 0xc000003 0x0 0x0 0x0 0x0 0x80000 0x6000000	0
0x0 0x0 0x0 0x0 0x0 0x0 0x18000 0x54030060 0x7020200 0x3 0x10000000 0x200 0x5 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x18000 0x54030060 0x7020200 0x3 0x10000000 0x200 0x4 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x60030000 0x3080c00 0xc000003 0x0 0x200 0x84 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0xc800 0x54030060 0x40a0a00 0x3 0x10000000 0x0 0x84 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x54030060 0x40a0e00 0x3 0x10000000 0x0 0x84 0x0 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x14030060 0x40a0e00 0x3 0x10000000 0x0 0x84 0x0 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x74000060 0x40a0a00 0x3 0x10000000 0x0 0x84 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0xc000 0x74000060 0x40a0a00 0x3 0x10000000 0x0 0x84 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x74000060 0x40a0e00 0x3 0x10000000 0x0 0x84 0x0 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x18800 0x24030000 0x30e0200 0x3 0x10000000 0x0 0x84 0x0 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14000 0x44000000 0x3060600 0x3 0x10000000 0x0 0x84 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0xc800 0x54000000 0x40e0600 0x3 0x10000000 0x0 0x84 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14000 0x44000000 0x30e0200 0x3 0x10000000 0x0 0x84 0x0 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x24000 0x74030000 0x4060200 0x3 0x10000000 0x0 0x84 0x0 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x44000000 0x4060600 0x3 0x10000000 0x0 0x84 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x44000000 0x40e0600	

0x3 0x10000000 0x0 0x4 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x30800 0x44000060 0x40e0200 0x3 0x10000000 0x200 0x84 0x0 0x80000 0x6000000	0
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x40000000 0x40e0600 0x3 0x10000000 0x200 0x84 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x40000000 0x40e0e00 0x3 0x10000000 0x200 0x84 0x4000 0x80000 0x6000000	0
0x0 0x0 0x0 0x0 0x0 0x0 0x14800 0x44000000 0x70a0e00 0x3 0x10000000 0x200 0x84 0x0 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14000 0x20000000 0x40e0c00 0x3 0x10000000 0x200 0x84 0x4000 0x80000 0x6006000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x30800 0x54030060 0x3060c00 0x3 0x10000000 0x0 0x4 0x0 0x80000 0x6006000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x14000 0x10030060 0x3080c00 0x3 0x10000000 0x30200 0x84 0x0 0x0 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x30800 0x10030000 0x30e0a00 0x14000001 0x0 0x200 0x4 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0xc000 0x54030000 0x3000800 0xc000001 0x0 0x0 0x84 0x4000 0x80000 0x6000000	0
0x0 0x0 0x0 0x0 0x0 0x0 0x30800 0x50030060 0x3000800 0x18000001 0x0 0x200 0x4 0x4000 0x80000 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x30000 0x54030060 0x30e0a00 0x14000001 0x10000000 0x0 0x4 0x4000 0x0 0x6000000	1
0x0 0x0 0x0 0x0 0x0 0x0 0x3c000 0x54030060 0x30e0a00 0x14000001 0x10000000 0x0 0xc 0x4000 0x0 0x0	1

Table 4: Conditions on the input message for the characteristic in table 2.

Mask on the message	s
0x0 0x0 0x0 0x0 0x0 0x0 0x1400 0x71000 0x7020001 0x8000000 0x0 0x30300 0x180 0x1c000 0xc0000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x21400 0x10041c00 0x3060c00 0x8000003 0x0 0x1000300 0x3 0x18000 0x1c0000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x21400 0x10071c00 0x40e01 0x3 0x0 0x1030000 0x183 0x4000 0x100000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x21400 0x10071c00 0x40e01 0x3 0x0 0x1030000 0x181 0x4000 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x21400 0x10071c00 0x40e01 0x3 0x0 0x1030000 0x180 0x4000 0x100000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x21400 0x10071c00 0x40e01 0x3 0x0 0x1030000 0x180 0x14000 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x21400 0x71c00 0x40e01	

0x3 0x0 0x1030000 0x180 0x14000 0x100000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x1400 0x10071c00 0x40a01 0x3 0x0 0x1030000 0x180 0x14000 0x100000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x1800 0x10031c00 0x4000801 0x3 0x0 0x1030000 0x180 0x14000 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x21400 0x10031c00 0x7060a01 0x8000003 0x0 0x1030300 0x183 0x1c000 0x1c0000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0xc00 0x30c00 0x7020e01 0x3 0x0 0x1030000 0x182 0x4000 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0xc00 0xc00 0x4020c00 0x3 0x0 0x1000000 0x1 0x0 0x100000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0xc000 0x30c00 0x4060801 0x3 0x0 0x1030000 0x180 0x4000 0x100000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x10070c00 0x3000c00 0x3 0x0 0x1030000 0x180 0x4000 0x100000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x10030c00 0x4060c01 0x3 0x0 0x1030000 0x180 0x4000 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x21800 0x71c00 0x4020c01 0x3 0x0 0x1030000 0x180 0x4000 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x21800 0x71c00 0x4020c01 0x3 0x0 0x1030000 0x180 0x4000 0x0 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x10000c00 0x3040c01 0x3 0x0 0x1000000 0x1 0x0 0x0 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x2cc00 0x10040c00 0x4060801 0x3 0x0 0x1030000 0x180 0x4000 0x0 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x2cc00 0x10040c00 0x4020801 0x3 0x0 0x1030000 0x180 0x4000 0x0 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x2d800 0x10031000 0x4020801 0x3 0x0 0x1030000 0x180 0x4000 0x0 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x20c00 0x0 0x4060801 0x8000003 0x0 0x1030000 0x1 0x0 0x0 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x2c000 0x0 0x4060801 0x8000003 0x0 0x1000000 0x1 0x0 0x0 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x20000 0x40c00 0x4020801 0x2 0x0 0x0 0x181 0x0 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x2c000 0x40c00 0x4020801 0x3 0x0 0x1000000 0x181 0x0 0x0 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x2cc00 0x70c00 0x4020801 0x3 0x0 0x0 0x181 0x0 0x100000 0x0	0
0x0 0x0 0x0 0x0 0x0 0x0 0x2cc00 0x70c00 0x4020801 0x3 0x0 0x300 0x0 0x4000 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x2cc00 0x70c00 0x4020801	

0x3 0x0 0x300 0x0 0xc000 0x100000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x2cc00 0x70000 0x3040c01 0x1 0x0 0x30000 0x0 0x10000 0x1c0000 0x0	1
0x0 0x0 0x0 0x0 0x0 0x0 0x2d800 0x71c00 0x4060401 0x8000003 0x0 0x0 0x0 0xc000 0x100000 0x0	0

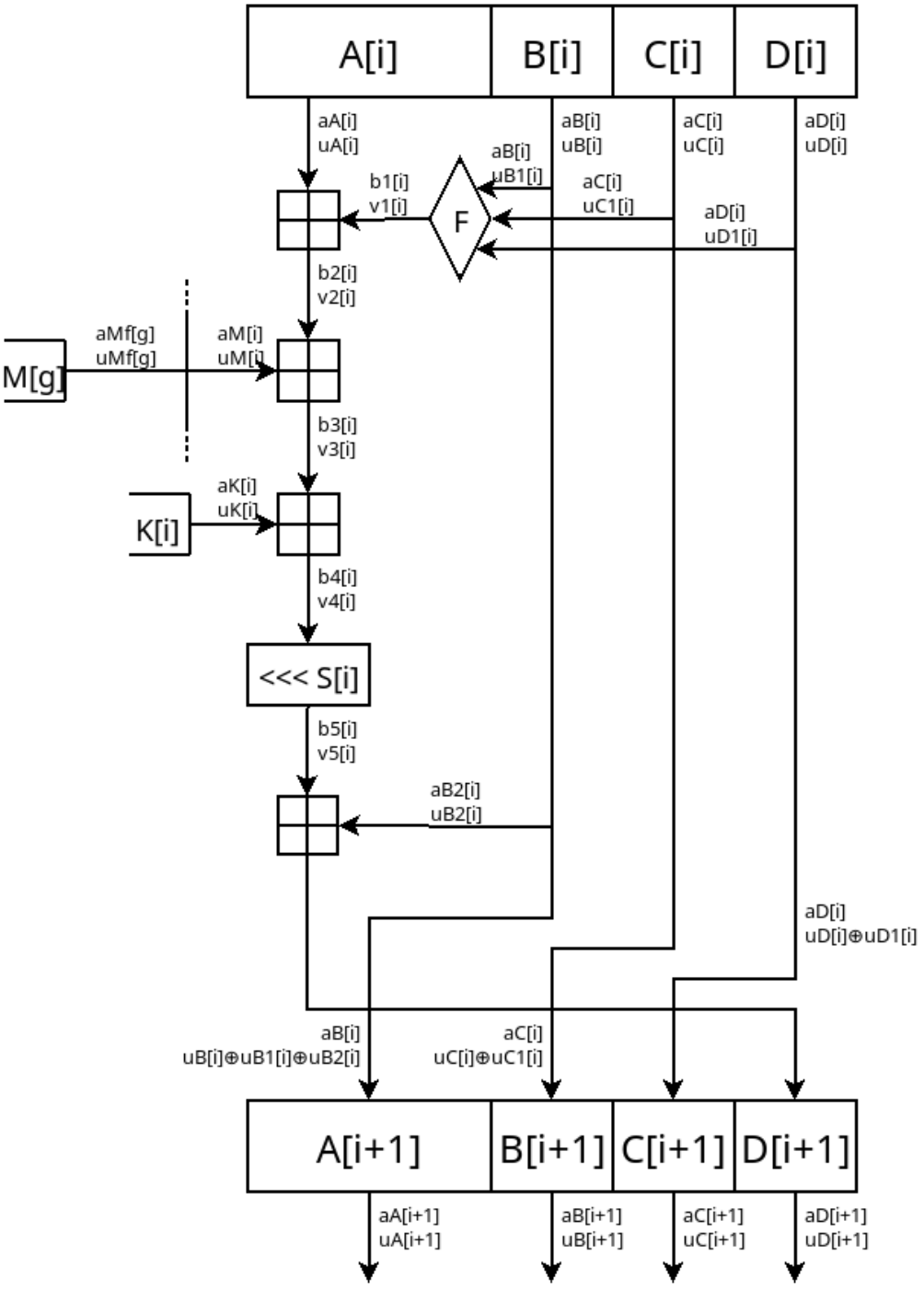

Figure 8: Differences and masks through a step of MD5. A top line of text represents the difference at a certain place in the function, while the bottom line represents the mask.