Abstract

This project includes two main components. First part mainly includes theoretical research. It includes analysis of MD6 and differential security of MD6. The analysis of MD6 includes analysis of its compulsory and optional parameters, mode, structures, compression functions and its security properties. The analysis of differential security of MD6 refers to birthday attack and differential attack. It also introduces some concepts like differential path and differential weight hamming pattern. In the analysis of differential security it mainly use Ethan Heilman's method.

The second parts refers to my contribution. It analyses the differential security of three smaller MD6 hash functions, MD6-17-8, MD6-35-8 and MD6-35-16. It provides the lower bounds of differential security of those three smaller versions of MD6, using Ethan Heilman's method. However we found that it has a bug in his code which he posted on his github. I solve that bug and sent email to him. He agreed with me and changed his code on his github. The bug took me more than 2 weeks to solve, but it is worth and I am proud of that. We discussed my result of smaller MD6-35-8 and parallelism for his method. He gave suggestion about how to make my result on smaller versions of MD6 more reasonable. MD6 is a new algorithm. It need time to prove the correctness of Ethan Heiman's method and my results about lower bound of differential security in smaller versions of MD6. This project has the following contributions:

- Find a bug at Ethan Heilman's code and solves it.
- Provide the lower bound of differential security for smaller versions of MD6 and analyse the results.
- Analyse the parallelism about Ethan Heilman's code and tried to use OpenMp on it.

Future work will be to prove the correctness of my results for smaller for the smaller versions of MD6, so making Ethan Heilman's code parallel will be a good choice.

Contents

1 Introduction

1.1 Background

A cryptographic hash function h is to map an arbitrary M-bit string to a fixed bit-length output $h(M)$ d string. The string to be inputed is often called "message", the output string is often called the message digest. They have many practical applications, for example they are used in time-stamping methods, digital signatures and message authentication codes (MACs).
The cryptographic hash function also satisfy some cryptographic properties: **One-wayness, Collision-Resistance, Second Pre-image Resistance** and **Pseudo-randomness**[17]. In this paper the property **Collision-Resistace** is used, because differential resistance this paper discussed belongs to collision-resistance. Here give its decryption: If a function is collision-resistance, it is infeasible to find distinct values M, M', $h(M) = h(M')$[17]. the descriptions of the cryptographic properties of others can be find in Menezes et al[17]. they are beyond of the scope of this paper.
Differential cryptanalysis was first proposed by Biham Shamir [3]. It was the first analysis of attacking on block ciphers including the Data Encryption Standard. Latter, Differential cryptanalysis of hash functions based on block ciphers was pioneered by Preneel et al.[22].
SHA family hash functions: MD4, RIPEMD[25, 29], SHA-0[1, 2, 4], SHA-1[2, 26] and MD5[27] have been proved differential collisions. However, the MD6 hash functions of the participant of SHA-3 competition is provable of differential security with repeat to differential cryptanalysis.We are interested about this, but researching the standard MD6 is a big challenging for me.
The MD6 team provided a proof of resistance against differential security. It provided the lower bound for preventing the differential attack using computional-aided method. However the NIST committee reported a bug in its result and they think the number of lower bound, the MD6 team provided, is not enough for differential resistance. They found it had some problem when computing because of long the running. Later Ethan Heilman restored it. He found a novel method to reestablish the differential security of MD6 [7] and modifies the original source code provides by the MD6 team. Also he provided his new results. It looks very interesting. That's also why we chose this project.

1.2 Project Overview

This project mainly has two scope. One is to analyse the MD6. Having a deep understanding at MD6 must be required. In my research skill i already know about MD6 but not very deep. When I did my project, I spent one more month to analyse it and its code. Compression functions are the spirit of MD6 hash functions. If compression functions are safe, Md6 hash function are safe. So compression functions stands some pages in this paper. The input size, output size and rounds number should mainly be considered when analysing compression functions. Also it refers to other factors for example tap positions, shift amounts and mode structure. All of them are analysed in this paper.
The second scope is differential cryptanalysis. The previous hash functions are fallen facing differential attack. In other words MD6 is provided specially for preventing differential attack. It has two methods to analyse the differential security of MD6. One is the original method and the other is Ethan Heilman's method. However both of these two method need large time for months or years. Because of long time computation the original happened a problem, but Ethan Heilman improved the method based on the original code. He divide the differential path into trial and non-trial path for saving time of searching valid differential path. Both of the two methods tries to find all the valid the differential path in some certain round and get the one which generated the least AAG in the certain round, so this least AAG number is the certain round can create.
When I did this project, I contacted with Ethan Heilman. He took one month to get the results and he tries to run his code in multi-core machine with 200 cores. But most of the results are reasonable, some are strange. This is challenging for me. So I did not research the standard MD6. I chose three smaller versions of MD6, MD6-17-8, MD6-35-8, and MD6-35-16, where the two numbers means the size of each rotation and each round.
When I tried to analyse the tree smaller versions of hash function using Ethan Heilman method, I found a bug and solved it. This bug will be discussed at page. Ehtan Heilman agree with my solutions. He changed his code on his github. Also I posted at the appendix.
It provided our results about lower bounds of the differential security for the three smaller MD6 hash functions. Some results seems strange and it provided the analysis of these strange results. It did not provide method for checking the correctness of the results. However we provided analysis of the results. Our analysis is reasonable.
The last part discuses parallelism strategy. We tried to use OpenMP at

Ethan Heilman's code and we provide the analysis of possibility for paralleling his code. But we did not success.

1.3 Organization

This paper is organised as follows.

Section 2 describes the MD6 hash function and Differential Cryptanalysis. In this section it provides briefly technics of MD6 hash function and the model of compression function. It represents basic definitions and assumptions about differential cryptanalysis. This part is mainly theoretical part of this project. It includes some what I got in my research skill and some new what I enlarged the research review according to my feedback.

Section 3 provide analysis's of the compression function combining differential cryptanalysis. It analyses the three important operation : Xor, And and the $g(x)$ provide their effect in preventing differential attacks. Compression functions are the spirit of MD6 hah functions. In this part we provided the detail things of it and enlarge the content of research review.

Section 4 analyses two approaches about lower bound proof. The original method is from the NITS MD6 hash function Report, and the other improved method is from professor Ethan Heilman. It compare these two methods from their correctness and cost time. It includes some content of my research review and also some new analysis.

Section 5 discuses the main contribution of this project. It discussed the bug we find in Ethan Heilman's code and analyse the differential security of the smaller versions of MD6 hash functions, MD6-17-8, MD6-35-8 and MD6-35-16. It discussed why chose these kinds of MD6,how to choose and what kinds of results we want to get. It provides the lower bound of differential security for the three versions of MD6.

Section 6 discusses parallelism on both MD6 algorithm and Ethan Heilman's method. We tried to use OpenMP on them, and analyse the possibility of using OpenMP on them.

Section 7 discusses future work. We did not provided the exact method to prove the correctness of my result and even if Ethan Heilman did not provide method to prove correctness method. Because proving the method need large time and large computational ability. Making the method parallel will be a good choice, so I think in my future work I will focus on that.

2 Analysis of MD6 and Differential Cryptanalysis

In this section it will briefly describe the relevant details of hash function, MD6 hash function and differential cryptanalysis which are needed for our analysis.

2.1 Notation

Let w denote the word size in bits. the size of word in MD6 by default is 64 bits. Let $\mathbf{W}$ denote the set$\{0,1\}^w$ of all w-bit words.In this paper, a word always refers to 64-bit word.
A denotes an array, its lower letter a denotes its length. $A[i] or A_i$ represent the element of array A. $A[i..j] or A_{i..j}$ denotes a subarray of A from A_i to A_j.
It may uses the other standard notations include:
$\oplus$: the bitwise "XOR" operator on words.
$\wedge$: the bitwise "AND" operator on words.
$x \ll b$: x left-shifted by b bits (zeros padding)
$x \gg b$: x right-shifting by b bits (zeros padding)
$x \lll b$: x rotated left by b bits.
$x \ggg b$ x rotated right by bits.

2.2 MD6 Overview

This paper dose not provide descriptions of full MD6 hash functions, it is not necessary . It covers relevant details for my analysis instead.
MD6 hash function has different versions according to different digest size d. MD6-224, MD6-256, MD6-384, and MD6-512 are most relevant for SHA-3. This paper use MD6-160 for our experiment. It have 5 input parameters $\{M, d, K, L, r\}$. M denote the messages to be hashed in word. its length m is finite in bits, where $0 \leq m < 2^{64}$. d is 160 bits. K, L is **nil** and 64 respectively by default, and a number r of rounds that is the default for that digest size. It is 80 for digest size 160 provided by The MD6 SHA-3 NIST report. This paper mainly focus on the round number, which refers to the MD6 compression function f.
The compression function of MD6 is feedback function. It has an input array $N[0..n-1]$ of n=89 words, which typically consists of 25-word "auxiliary information" block , followed 64-word block. It also has input parameter r rounds, where each round include 16 steps denoted by c ($c = 16$). In computation loop, each step get a word $A[i]$, finally it gets $A[0, 1..t + n - 1]$

of t+n words($t = rc$). The first n words of $A[0, 1..n + t - 1]$ was initialised by $N[0, 1..n]$. The compression function outputs the last 16 words, which means output $A[t + n - c..t + n - 1]$.
The feedback function calculate $A[i]$from $i = 89$($A[0]$ to $A[88]$ is initialised). It takes six "feedback tap positions" t_0, t_1, t_2, t_3, t_4 to index words $A[i - t_0]$, $A[i - t_1]$, $A[i - t_2]$, $A[i - t_3]$, $A[i - t_4]$ as input. Tap positions are constants as figure 2.1:

t_0	t_1	t_2	t_3	t_4
17	18	21	31	67

Figure 1: Tap positions

The nature of tap positions is beyond the scope of this paper, it will not be discussed in this paper. For our analyses it provide a simply description of the compress function. It remove other features which are not relevant of our analysis, so it is easy for our analysis.

```
**Algorithm 1** Compress(A[], ROUNDS)
**Intput:**
**A**: An input array $A[0..n - 1..n + r * 16 - 1]$, $n = 89$. $A[0]$ to $A[0..n - 1]$ is initialized.
**r** : A non-negative number of rounds.
**Output:**
**C:** An output array $C[0..c - 1]$ of length $c = 16$ words.
**Main for loop:**
for $i = 89$ to $n + r * 16$ **do**
$x \leftarrow feedback(A[i - t_0], A[i - t_1], A[i - t_2], A[i - t_3], A[i - t_4], A[i - t_5])$
$A[i] \leftarrow x \oplus g(x)$
**end for**
```

Table 1: Compression function

The feedback function is described as:

$$feedback(X_0, X_1, X_2, X_3, X_4, X_5) : X_0 \oplus (X_1 \wedge X_2) \oplus (X_3 \wedge X_4) \oplus X_5.$$

The Intra-word diffusion operation $g(x)$ is described as:

$$g(x) : (x \gg R) \oplus (x \ll L)$$

Where R and L are constants , they are indexed by i. The discussion of $g(x)$ is beyond the scope of our analysis. the full decryption can be found in the MD6 SHA-3 NIST report [23]. the operation of "xor" and "and" in the feedback function generate differential path, latter it will be represented. It is also the purpose the compress function is introduced here.

2.3 Differential Cryptanalysis

Differential cryptanalysis was first pioneered by Biham and Shamir [3] , which analyse attacks on block ciphers including The Data Encryption Standard. Subsequently, Preneel et al.[22] provide the deferential attack on hash functions.
MD5, SHA-0, and SHA-1 have be proved they are not against differential attack by Professor Xiaoyun Wang. our goals is to analyse the MD6 is security under differential attack. to fulfil this goal need to find two factor: upper bound and lower bound. This paper mainly focus on lower bound, which means finding a lower bound represents there is not a standard differential attack more efficient than birthday attack.
The rest of this section provides definitions and assumptions which are relevant for analysis of the MD6 against differential attack.

2.3.1 Definitions and assumptions

Differential attack is that giving two different plaintext inputs, the outputs can get two ciphertexts which have a special difference with a high probability. For keyless hash function the differential collision attack is to give two different plain texts and can get same output with a non-negligible probability. Differential cryptanalysis should analyse many of plaintext pairs with differences. It is belong to chosen plaintext attacks[11, 5].
The definition is provided in research skill. Here it adds more detail research. Differential attack in keyless MD6 hash functions is not completely same with the traditional differential attack on block cipher text. Differential collision attack in MD6 means it can find the same output from different input, whereas differential attack in block cipher text means it belongs to chosen plaintext attacks, so it can choose different particular inputs and get particular different outputs with high probability.These particular different outputs can reveal the key. For example it has two different n-bits length inputs X_n and X'_n with 1 bit difference, so the probability of this difference is 2^{-n} and it can get a particular different pair of outputs with much more

than 2^{-n} probability. From this it can conclude the key. However differential collision attack on hash function just need find the same output. Table 2 gives a collision on SHA-0 hash functions[28]

M_1 :	474204bb 3b30a3ff f17e9b08 3ffa0874 6b26377a 18abdc01 d320eb93 b341ebe9
	13480f5c ca5d3aa6 b9f3bd88 21921a2d 4085fca1 eb65e659 51ac570c 54e8aae5
M'_1 :	c74204f9 3b30a3ff 717e9b4a 3ffa0834 6b26373a 18abdc43 5320eb91 3341ebeb
	13480f1c 4a5d3aa6 39f3bdc8 a1921a2f 4085fca3 6b65e619 d1ac570c d4e8aaa5
h_1 :	2af8aee6 ed1e8411 62c2f3f7 3761d197 0437669d

Table 2: A collision of 80-step SHA-0

Firstly differential cryptanalysis is to find measures of difference.It usually depends on the mathematical operations. In hash function exclusive-or is usually is used measure of difference. For example, A and A' be a pair of values, $\triangle A = A \oplus A'$ denotes their difference. exclusive-or can specify each bits difference of the pair of values.
A **differential path** is a output set of differences between two different pair inputs for hash functions. So for the MD6 compression function, the differential path is denoted as:

$$\{\triangle A_i\} \; for \; i = 0, ...t + n - 1.$$

If $\triangle A_i = 0$ for $i = t + n - c, ...t + n - 1$, it is called a **collision differential path** for the compression function[7].
The probability of differential path in step i of the compression functions is an important property, which denoted by p_i. It is to define that the output pairs satisfy the differences given by the differential path from step i. The differential path is given by the pair of inputs. p_i is expressed as:

$$p_i(\triangle A_i) = prob(\triangle A_{i-t_0}, .. \triangle A_{i-t_5}).$$

$$P = \prod_{i=0}^{t+n-1} p_i \; .$$

Where P is the total probability of the differential path. It is commonly assumed that each step output appears random and independent after a certain number of rounds [22] So the probability of each step is randomness

because its inputs are from previous step outputs and each step output is randomness. So P can be computed by multiplying $p_i s$.
Another useful notation is **differential path weight pattern**. Let $\{\triangle A_i\}$ be a differential path. the corresponding differential path weight is the set $\{D_i\}$, where $D_i = |\triangle A_i|$, which means D_i equals the number of different bits between A_i and A'_i. For example, a differential path weight pattern is $[4.1, 39.24]$ represents it has two different position $A[4]$ and $A[39]$, the hamming weight of D_4 and D_{39} are 1 and 24 respectively.

2.4 Summary

This part introduced some notations it required in this paper. It introduced MD6 algorithm and differential cryptanalysis and also some definitions of differential path, collision differential path and differential path weight pattern. These definitions refer to MD6 and differential cryptanalysis. The next part will discuss compression functions.

3 Compression function

3.1 Introduction

In my research skill, I just have an simple overview about compression function, however compression function is the most important part in hash function of course including MD6 hash function and also is the important part in this project. It will deeply analyse the compression functions.
Compression functions are defined in terms of families. Using mathematic symbol to express as $f : \mathcal{K}_f$ x $\{0,1\}^m$ x $\{0,1\}^k \to \{0,1\}^k$ [25] Sometimes the output length is not fixed k. For example in standard MD6 hash function, the input length is 89 64-bit words and the output length is 16-bit words. If this compression function in MD6 is differential resistant, MD6 hash function is differential resistant according to the MD6 team research[23]. It is similar with the conclusion Merkle-Damgard construction is collision-resistant as long as the compression function is collision-resistant[29, 4]. In Md6 hash function collision-resistant is same with differential -resistant, but people don't say that. These thesis are not belong to this area of paper, so here it does not provide the proof, just use it.

3.2 Analization in MD6

From table 1, we can see the algorithm of compress function in MD6. It mainly refers to three parameters, L, n, c, which represent respectively Mode, input length, output length of MD6. MD6 is tree based structure, which is to get one aim of NIST competition: efficiency, because it can work parallel well. It obviously uses many storage because of tree basic structure. Md6 is very flexible, which provides different kinds of mode for small and large storage respectively.
Referring to the mode of MD6, it must discus the parameter $L(\ 0 \leq L \geq)$. In the code of MD6 it actually represents the level of structure. The standard one is 4-to-1 block tree, which the level of root is not more than 64, but it can be smaller than 64. It also has differential versions. $L = 0$ is the standard Merkle-Damgard structure as said before it will save the storage, because the total number of compression functions decrease as parameter L decreases.
The main technical of compress function is that it has one 89 initial array as input and every time it will get a new item of the array, which calls one step. 16 steps consistent into one round and 89 steps consistent into one rotation. The definition of round is impotent to this project. The lower bound means the lower bound of round. It is also why it here discuses the compression

function. In the 89 initialise items, it has 4 blocks about 64 words message and 25 optional and mandatory parameters. If it is not enough for 89 words, it uses 0 padded. MD6 determine how many compress functions should be used according to the size of input. An extreme example the input size is empty, it just has one compression function with the 64 0 words padded.
In below paragraphs it will analyse the operations of compression function. Some are derived from research skill and some are new added.
When we discuss the operation, it refers to some thesis. It refers to differential paths, differential positions and differential path weight pattern. They are introduced in the previous chapter. Here we will highlight the difference between differential positions and differential path weight pattern. The former described the position of input, this will be complicated, because every item of the array will have probably w different positions, where the w means the length of one item or the length of one word. While differential path weight pattern simplifies this question, it just consider how many there are differences of every item of the input array.
If a function is secure against differential attacks, any differential path should has a probability P no more than $2^{-d/2}$, where d is the length of hash output in bits.It means differential attack is not more efficient than “birthday attack” ,which require effort $O(d/2)$ [23]. Recall the definition of differential collision. If it has a pair of different inputs , it can get the same output without difference. Let A is a arbitrary hash output, a is its length d. Its probability is 2^{-d}. Let A_i and A_j be the pair of different inputs, P_i and P_j are probabilities of the differential path of A_i and A_j respectively. A_i and A_j hash to the same output A. So

$$P_i * P_j = 2^{-d}$$

each P is less than $2^{-d/2}(\sqrt[2]{2^{-d}})$, it can ensure any $P_i * P_j \leq 2^{-d}$. Thus this function is secure against differential attack.
Differential paths refers to initial, intermedium and output values in compress functions. Actually it hopes every bit difference of input generate at least one difference of intermedium output. That means any one item of the input array express the differences between the different previous 89 items, because every item is determined by its previous 89 items. This will give answer about someone maybe ask the last 16 outputs don't have differences, but before the last 16 they have difference. This case is collision or not. So that analysis answer this question. For example, for a input initialise array $A[0, ..88]$, the output array should be $A[0, ...n + 88]$. One item $A[i]$ will express the difference from item $A[i - 89]$ to $A[i - 1]$. If there are two

input pair A_1 and A_2, they just have one bit different in there initialisation values, the last 16 output at least will have one word different, and the actually number of the differences of output is discussed later. Why it has that conclusion, because it has g operation, which is right and left shift, it propagate the difference. However usually the example will generate much more than 1 difference in the output, It is because it has AND and Xor operation. Which one is the determined factor. It is discussed below.
For easy discussion, we denote some symbols. X, Y, Z denotes the w-bit input and output. $\triangle X, \triangle Y \triangle Z$ denote the differences. D_X, D_Y, D_Z are the hamming weight of $\triangle X, \triangle Y \triangle Z$. The lower letters $\triangle x, \triangle y, \triangle z$ denotes the position $\triangle x, \triangle y, \triangle z$ bit

3.3 XOR gate

XOR gate can keep differences but it can not propagate differences. It can be proved using the equation $\triangle Z = \triangle X \oplus \triangle Y$. If $\triangle X$ and $\triangle Y$ are different in bit a, $\triangle Z$ keeps this difference in its bit a. If If $\triangle X$ and $\triangle Y$ are same in bit a, $\triangle Z$ keeps same in its bit a. In other words,XOR gate makes $p_i = 1$. So we can straightforwardly get the relationship:

$$max(D_X, D_Y) - min(D_X, D_Y) \leq D_z \leq D_X + D_Y$$

.
Conbine this with the two sub-functions of the compression function:

$$Z \leftarrow A_{i-t_0} \oplus A_{i-t_5} \oplus (A_{i-t1} \wedge A_{i-t2}) \oplus (A_{i-t3} \wedge A_{i-t4}),$$

$$A_i \leftarrow g(Z).$$

If A_{i-t_0} and A_{i-t_5} have differences, assuming 5 differences, so its intermedium result also keeps 5 differences. The differences don't increase, so this operation does not have effect on the differential resistant.

3.4 AND gate

AND gate is different with XOR gate. AND gate could propagate differences. Consider the following cases:

- If $\triangle x = 0$ and $\triangle y = 0$, then $pro(\triangle z = 0) = 1$. This AND gate is "inactive".

- If $\triangle x = 1$ or $\triangle y = 1$, then $pro(\triangle z = 0) = pro(\triangle x = 1) * pro(\triangle y = 0) + pro(\triangle x = 0) * pro(\triangle y = 1) = 1/2 * 1/2 + 1/2 * 1/2 = 1/2$; it can also get $pro(\triangle z = 1) = 1/2$. This gate is called AND gate "active".

Combine this with the sub-two functions. Here $\triangle x$ is the difference between A_{i-t_1} and A'_{i-t_1}, and $\triangle y$ is the difference between A_{i-t_2} and A'_{i-t_2}, every $\triangle x$ or $\triangle y$ generate a new difference with the probability 1/2 . This and gate is Active And Gate (AAG). So if the hamming weight of A_{i-t_1} is I_1 and the hamming weight of A_{i-t_2} is I_2, then this operation have $max(I_1, I_2)$ AAGs.
So a "active" AND gate can generate one 1/2 of the number $d/2$ of 1/2, it is active when computing the probability of the differences. It plays an important role of proving lower bound. It also can get the bounds of D_Z as the following:

$$0 \leq D_Z \leq D_X + D_Y.$$

We always use this equation to check if the differential path is valid or not, this save time of finding the lower bound using computational aided method.

3.5 g operator

The g operator is same with Xor gate, which means if the input changes, the output changes with probability 1. Actually one bit change in input will at most leads to from 2 to 4 bits change. It can get from its operation:

$$y = x \oplus (x \gg R), g(x) = y \oplus (y \ll L)$$

One bit difference in Xor and Shift operations at most create 1 or 2 differences, so g operator can create from 2 to 4 difference because it has double that operation. The key thing is if its input is different, its output is also different. So it does not have effect on the probability $2^{-d/2}$. Also we analyse its upper and lower bound of D_Z. It is easy to know the upper bound of D_Z using the upper bound of D_Z in Xor gate. So

$$D_Z \leq 4D_X.$$

The lower bound of D_Z depends on the shift amounts. Specifically if $D_X \leq 4$, $D_Z \geq 2$. It prevent from generating differential path with lower hamming weight at each step. constructing differential path with hamming weight 1 need at least 5 input differences. When $D_X > 4$, non-zero input differences will get non-zero output differences, so $D_Z \geq 1$. The values of L and R can make that. Every round shares one l_i and one r_i, where l_i and r_i are the values of L and R respectively. These two values have some rules as below:

- l_i and r_i are nonzero and not more than $w/2$. They are not multiples of each other.
- If D_X is not more than 4, D_Z must not be 1.

i mod 16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
r_i	10	5	13	10	11	12	2	7	14	15	7	13	11	7	6	12
l_i	11	24	9	16	15	9	27	15	6	2	29	8	15	5	31	9

Figure 2: Standard $w = 64$ the suggested l_i and r_i

This values related tap positions, especially t_0, t_5,$(t_5 - t_0)$. In Figure 2 t_0= 17, t_5=89 and $t_5 - t_0$= 72. These values will module c. These shift values can be constant, but its diffusion rate within a word is not better than the uncertain values.

3.6 Summary

Through the above analysis, we get the probability of generating output differences and the lower bound and upper bound of the hamming weight of differential path in the MD6 each step functions .The results are summarised as Figure 3 and Figure 4.

operations	output differences	probability
$\oplus$	$\triangle Z = \triangle Y \oplus \triangle X$	1
$\wedge$	if $\triangle X = 0$ and $\triangle Y = 0$, then $\triangle Z = 0$	1/2
	if $\triangle X = 1$ or $\triangle Y = 1$, then $\triangle Z = 1$.	1/2
$g(x)$	$\triangle Z = g(x)$	1

Figure 3: Differential characteristics for $\oplus$,$\wedge$ and g.

operations	Lower bound(LB)	Upper bound(UB)
$\oplus$	$D_Z \geq max(D_X, D_Y) - min(D_X, D_Y)$	$D_Z \leq D_X + D_Y$
$\wedge$	$D_Z \geq 0$	$D_Z \leq D_X + D_Y$
$g(x)$	if D_X $leq4$, $D_Z \geq 2$	$D_Z \leq 4D_X$
	if $D_X > 4$, $D_Z \geq 1$	

Figure 4: Hamming weight

According to Figure 4, We can get Lower bound and Upper bound of D_X.

$D_X \leq \sum_{j=0}^{j\leq 5} D_{i-t_j}$.

$D_X \geq max(D_i - D_{i-t_0}, D_{i-t_5}) - min(D_{i-t_0}, D_{i-t_5}) - \sum_{j=1}^{j\leq 4} D_{i-t_j}$.

Given D_X, The hamming weight of output difference in step i, D_i=$\mid \triangle A_i \mid$ using Figure 4. Latter it will uses these to check if the differential path is valid or not.

4 Lower Bound Proof

The goal of our analysis is to prove a lower bound on workload of any standard differential collision attack. As decried previously, If the MD6 compression function is resistant of differential attack, the probability of any differential path of occurring is less than $2^{-d/2}$. From section 3, we know each an Activate AND gate generate one 1/2. So this problem changes into find how many rounds could generate $d/2$ activate AND gates(AAG). First,this part provide a briefly description of prior work on Lower Bound Proof. It includes the approaches provided by the MD6 team[7] and Ethan Heilman[23]. Second, It combines the two approaches to provide the lower bound of proof.
The before paragraph is my research skill. In this project I will described more detailed things and provide my understanding.

4.1 Prior Work on Lower Bound proof

The method in MD6 NIST Proposal calculates the number of AAGs generated by lower round s to extend to a higher r rounds[23]. using the following algorithm:

$$AAG_r \geq AAG_s \times \lfloor r/s \rfloor \tag{1}$$

It is straightforward. Consider a differential path weight pattern D_i, It generate $AAGs$ in the first s rounds. These $AAGs$ will generate at least another $AAGs$ in the next s rounds, and the other differences also may generates some active AND gates in the s rounds. So the lower bound of $AAGr$ is $AAGs \times \lfloor r/s \rfloor$. It also can get proof from the source code proposed by the MD6 team. It is impractical to compute the number of AAGs of r rounds. Its computation consumption is expensive. So they use calculate lower s rounds to reason the higher r rounds. It find all valid differential paths of s rounds. Any differential path has no more than maxAAGs. Therefore they find the lower bound maxAAG. Obviously,the consumption of finding all differential paths of lower s rounds is cheaper. The lower s rounds can create the number of AAGs as the following figure described [7].

s	≤ 5	6	7	8	9	10	11	12	13	14	15
LB on $AAGs$	0	3	4	4	4	4	7	13	19	20	26

Figure 5: AAG lower bounds for s rounds

If hash output size $d = 512$, it need 256 $AAGs$ against differential attack. $AAG_{153} \geq AAG_{15} \times \lfloor 153/15 \rfloor = 26 \times 10 = 260$, so the first lower bound($153 + 15$, including the security margin of 15 rounds) proposed by the MD6 NITS report is secure under differential attack. However, latter an Official Comment on MD6 noted a gap in the differential resistance proof [7]. The bug is search through all valid differential paths to construct the number of $AAGs$ of lower s rounds. The 153 rounds is not sufficient to resistant differential attack when the length of hash output is 512 bits. The problem of creating this bug is not the method per se, but rather the method of searching the all valid paths. Latter, the MD6 team construct a new number of $AAGs$ of lower s rounds as the following figure.

s	$6-8$	$9-10$	11	12	13	14
LB on $AAGs$	2	4	5	11	13	≥ 15

Figure 6: New version of AAG lower bound for s rounds[7]

According to Figure 6, generating 256 AAGs needs at least 259 rounds including 15 security margin rounds. $AAG_{238} \geq AAG_{15} \times \lfloor 238/14 \rfloor \geq 15 \times 17 = 255$, which is less 1. It need more 6 rounds , it can produce $AAG_{238+6} \geq (255 + 2)$. Thus MD6 -512 need 259 (including 15 security margin rounds) rounds for resistanting differential attacks.

4.2 Original method of Lower Bound proof

This section gives the proof proposed in MD6 NITS report. as we described before, the proof rather than the method of searching all valid differential path is correct.
First, It is discussed how to count the number of active AND gate, it will divided to three cases to discuss. In the second part, it provides the differences between valid and invalid differential path weight pattern and how to search for valid differential path weight patterns. Finnally, it provides the computer-aided method to searching lower bounds.

4.2.1 Computing the number of Active AND gate

Given a differential path weight pattern up to s rounds, consider how to calculate the number of active number of active AND gate. Recall the feedback function, if a different bit input A_{ij}(the j bit of A_i is different) want to activate one AND gate, it should be one of the four inputs $A_{i-t_1}, A_{i-t_2}, A_{i-t_3}$

and A_{i-t_4}. These four inputs refer to the AND operation. If A_{ij} belongs to the four inputs, it will activate one AND gate each step and it can activate at most four AND gate. Because it can be arbitrary one of the four inputs and the four inputs are used once at s rounds numbers of computations.
In most case a bit difference indeed activate 4 AND gate. However there are three special cases which should be considered when counting the number of Active AND gate within an s-round segment.

Case 1: Two bit differences activate the same AND gate

Consider two bit differences inputs A_{uj} and A_{vj}. the j bit of A_{uj} has difference in step i ($i - t_1 = u$) and also the j bit of A_{vj} has difference in step i' ($i' - t_1 = v$). If $u - v = 3$ or $u - v = 36$, the AND gates they activate at step i and i' are same, which means the two AND gates equals one active AND gate.

Case 2: Activated two AND gate in the same step

Recall the feedback function, it has two AND operations. Let X and Y be the outputs of the first and the second AND operation in step i. Let α be $\alpha = \triangle X \oplus \triangle Y$. If X and Y activate AND gates at sept i, the $(\triangle X, \triangle Y)$ is $(0, \alpha)$ and $(1, \alpha \oplus 1)$ each with 1/4 probability. Thus two active AND gate get together to contribute one 1/2. This two active AND gate are counted as one active AND gate.

Case 3: Going across the round boundary

An AND gate does not counted when it go across the round boundary. The inputs and outputs of the active AND gate must be within s-round segment.

4.2.2 Searching for differential path weight patterns

How to seach for differential path weight patterns is discussed in this section. Within s rounds, D_i can be from 0 to w ($w = 64$) in a differential pattern, where i is from 0 to $s \times 16$(one rounds has 16 steps).Searching all differential path weight patterns. are hard even through s is small. Here can give a roughly number $2^{64} \times 64^{s \times 16}$.. Therefor when we search for, we eliminate the invalid differential path weight patterns. Recall the $g(X)$ operator, $Z = g(X)$. D_X lies in the range $[LB_X, UB_X]$. Let D_i be the differential path hamming weight of step i .Combining D_X, the following values of D_i is invalid:

1. $D_i = 0$ but $LB_X > 0$. Different inputs get same output, it leads to collision.

2. $D_i > 0$ and $D_i > 4UB_X$. the upper boundary of $g(x)$ is $4UB_X$. $D_i =$

D_Z.

3. $D_i = 1$ and $UB_X < 5$. if $D_Z = 1$,$UB_X \geq 4$.

It uses a function called SearchDiff() to search for valid differential path weight patterns. Latter this function will be demonstrated .Consider if a invalid differential path weight pattern participate the calculation of the valid differential path weight patterns, which dose not effect the lower bound. We can get the proof from technic of Searchdiff() function.

4.2.3 Analysing lower bounds through computer-aided search

The algorithm of searching for differential path weight patterns and the number of active AND gates is described as Table 3. Through this algorithm we can get all the valid differential path weight patterns.

Algorithm 2 SeachDiff($i, s, maxAAG$**)**

```
$i = s \times 16$ ;
$sumAAG = 0$;
Computing $UB_X$ and $LB_X$ for $D_X$ at step $i$;
do $k = 0$ to $s$;
    do $D_i = 0, 1, ....w - 1$;
    if( $D_i$ is valid) do next; else continue;
        Compute the new active AND gates given $D_i$,let $AAG_{D_i}$ denotes
        $sumAAG+ = AAG_{D_i}$
        if($sumAAG > maxAAG$) write down $D_i$ ; continue;
        else do **SearchDiff(**$i + +, s, maxAAG$**)**
```

Output:
The differential path weight pattern D_i and the number of active AND gates.

Table 3: SearchDiff algorithm

4.3 A improved method of Lower bound proof

Ethan Heilman provided a improved a method to derive the lower bound based on the original method. This method includes mainly three steps. First, it divide the differential path weight pattern into trivial and nontrivial patterns. It use prior method to calculate the trivial patterns lower bound, but it use a new method to get lower bound of nontrivial patterns. Then it combines the trivial and nontrivial lower bound into a general lower bound.

4.3.1 Types of Differential weight patterns

According to the consumption of computation it classify differential weight patterns into two class:trivial and nontrivial. It is easy to calculate lower bound of trivial patterns while the lower bound of nontrivial patterns is hard. It also has a standard definition to classify trivial and nontrivial patterns.

A Trivial Pattern is differential path weight pattern which has three or less different positions in its first rotation, where one rotation is 89 steps.

A Non-trivial pattern is a differential path weight pattern which has at least four different positions in its first rotation.

From the definition, we know trivial patterns and nontrivial patterns just have difference in their first rotation. In the first rotation, they are exclusive,but after they do not have difference. In other words a nontrivial patterns can be a sub pattern of a trivial pattern. In Figure 7, we give some example of these two types of patterns.

Trivial	$[73.4]$ $[16.3, 54.1, 88.2]$ $[4.1, 13.4, 34.4, 56.2, 78.5, 134.34, 213.5, 223.12]$
Non-Trivial	$[1.2, 4.5, 34.3, 35.2]$ $[2.3, 5.3, 34.5, 56.12, 72.7]$ $[3.1, 4.1, 5.1, 15.2, 17.1, 23.4, 56.1, 76.1, 134.1]$

Figure 7: Examples of trivial and nontrivial patterns

We have this claim: a pattern has less different positions in its first rotation and it also reduces the number of different positions in its sequential rotations. There are at most $\sum_{k=1}^{a}\binom{89}{k}$ different positions under *a AAG*. So the trivial patterns at most have:

$$\sum_{k=1}^{3}\binom{89}{k} = 117,569 \tag{2}$$

different positions in its first positions. According to the claim, the trivial patterns are more likely to have fewer different positions in its subsequent rotations. So trivial patterns can be computed the lower bound using the original method, which means it can use Searchdiff (a function of searching differential path weight patterns) method to search for differential path weight patterns and then get the lower bound. However , the lower bound of nontrivial patterns take expensive computations. Its first rotation has at least $\binom{89}{4} = 2,441,626$ different positions. In next part it will introduce a method to calculate non-trivial patterns.

4.3.2 Caculating Non-Trivial Patterns

It is found that a difference in the first 48 steps of the first rotation is unlikely to lead to *AAGs*. From this observation we can reduce some important points. For a same differential pattern it generates different number of AAGs between its first rotation and subsequent rotation. For example, a difference within the 3 first words up to s rounds just activate one AND gates, whereas after the 36 positions, a difference activate 4 AND gates up to s rounds.
This strategy is to start from the second rotation skipping the first rotation. When we get the lower bounds of AAGs, we will add 4 to the final result. It is likely to undercount the first rotation, so it may calculate more rounds until it gets our requirement. This strategy also run the prover proposed in the NIST MD6 Report with a simple modification such that it can skip the first rotation and add 4 AAGs to the final result.

4.3.3 Combining Trivial and Non-Trivial Patterns

When we get *AAGs* lower bounds of trivial and non-trivial patterns, they are combined together into a general *AAGs* lower bounds. The assembling method is fairly strait forward. It has two steps: one put the two list of the lower bounds together and then choose the minimum number of *AAGs* lower bounds as the *AAGs* lower bound corresponding the rounds. We do this at least up to s rounds which we want use s rounds to extend lower bound we need.

4.3.4 Result

Implementing the improved algorithm, it can get the following result described in Figure 8 and Figure 9 [7].

Rounds	6	7	8	9	10	11	12	13	14	15	16	17
LB on $AAGs$	4	4	6	7	7	7	14	16	16	17	≥ 18	≥ 35

Figure 8: Non-Trivial Differential Weight Patterns Lower Bound[7]

Combining Figure 7 and Figure 8, it gets that 137 rounds can activate 257 AND gates. So it is resistant for differential attack for MD6-512 hash function which previously provided 168 rounds. It is to reestablish the MD6 hash function against differential attacks.

Rounds	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
LB on $AAGs$	2	2	2	4	4	5	11	13	19	22	22	30	34	35	38	39	47	55	60

Figure 9: Trivial Differential Weight Patterns Lower Bound[7]

This method provides the reduced 152 round of also can ensure the MD6 hash functions differential resistant collisions. So we can see the 168 rounds proposed by the MD6 team has a more than twice security margin as they claimed originally.

4.4 Summary

This part mainly discussed two method for lower bound proof. Here the lower bound means the lower number of rounds. This number refers to the AAG number and the differential security. It uses two methods to get this number, the MD6 team original method and Ethan Heilaman's method. It discussed how Ethan Heilman improved the original method and main idea of Ethan Heilman's method. However using both of these methods on standard MD6 hash functions need large time and large computational ability. The next part will discuss the smaller versions of MD6 using the same method with some changes.

5 Analysing smaller MD6-like functions

This part is my own work. This part mainly discusses my own contribution during this three months. It discusses from these aspects. Why we chose smaller version MD6 hash function to analyse, what are our aims, How to choose a smaller MD6 hash function, what is the lower bound of differential resistance of the smaller MD6 hash function, How to prove its correctness, and what is my result? These questions mainly include what I got in these months.

5.1 Versions of MD6 hash functions

NIST requires MD6 meets these versions:

- MD6-244
- MD6-256
- MD6-384
- MD6-512

However, MD6 hash functions don't just have these versions. Those standard versions are mainly different at those output length (bits). The number of the above MD6 hash is the output length.

I am a student of university of bristol. This is my project.
MD6-244: 56d2642f30bbf4c15dd781cb98533c4fbae470195f75e368f45a9a61e98b2
MD6-256: 677f5d7ea846684a1498adf87aa9a0615ee0bb46595cc96eddc1741f6ed0dd05
MD6-384: 95e2847f546d6a51a732f9b122809b3704a9858fedd147c2eb44a462284f271b61faf0f83368fc6d871e1704a2dd2b61
MD6-512: 9d5e002db8dc6f84934a899423af0aee94e2f068da25a927e5b92b699bf0ea1bec97ca0233d7073871746fa1f12da8ab1f36e8059de0a79255c1870f977d3cc3

Figure 10: Outputs of different versions with same inputs

The versions in Figure 10 have default values $K = Nil($ empty $)$, $r = 40+d/4$ $L = 64$, $w = 64$, $n = 89$ and $c = 16$. In this project the parameters n, c, w

are important.
n and c are different. they lead to different tap positions. Tap positions are used in compression functions. They determined the inputs of compression function. Choosing tap positions has some rules as following:

- $g(t_0, n) = gcd(t_1, n) = gcd(t_2, n) = gcd(t_2 - t_1, n) = gcd(t_4 - t_3, n) = 1$. Here we do not provide the reason(it can be found on the MD6 NITS report [23]).

- t_0 should be relatively prime to a round. As MD6 should be parallelism in one round, t_0 should be at lead equal $r + 1$. In practice we find $t_0 = r + 1$ is good, not too small(so it can compute a full round in parallel) and prime to the length of a round(different shift amounts work well). In the standard MD6 hash function $t_0 = 17$.

- A tap position should ensure the feedback operation to be invertible, which means we can calculate A_{i-n} from $A_{i-n+1...i}$.

- All the tap position can not be the factor of c, which means it modular c, the result is nonzero. $t_4 - t_3 \neq t_2 - t_1$, so the two pair of "and gate" have different distances, which makes sure the possibilities that A[i] relates to the previous 89 array items from $A[i - 89]$ to $A[i - 1]$ are equal.

- The positions $t_1, .., t_4$ can be searched by using a computer. It is a simple brute force search.

If we follow the above rules, they are the best choice. Other choices sometimes are also right, but maybe attackers can grab the chance to break the hash functions.We use the above rules, always we get one result($n \leq 256$). $n > 256$ will has possibility that the result is more than one. Table 4 provides some examples.

n	c	t_0	t_1	t_2	t_3	t_4	t_5
89	16	17	18	21	31	67	89
35	8	9	12	13	19	34	35
35	16	17	18	22	24	31	35
17	8	9	10	12	14	15	17
45	8	11	13	14	17	31	45

Table 4: Tap positions for different n, c

From Table 4, t_0 is not always $c + 1$,but we calculate it from $c + 1$. t_5 should equal n. We will use these examples as smaller MD6 hash functions in this project. When we choose these parameters, these parameters will effect other parameters. We will analyse that bellow.

5.1.1 Shift amounts r_i and l_i

In the MD6 report they provided the shift r_i and l_i as described Table 5:

i mod 16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$w = 32$,r_i	5	3	6	5	4	6	7	3	5	6	5	5	4	6	7	5
l_i	4	7	7	9	13	8	4	14	7	4	8	11	5	8	2	11
$w = 16$,r_i	5	4	3	5	7	5	5	2	4	3	4	3	4	7	7	2
l_i	6	7	2	4	2	6	3	7	5	7	6	5	5	6	4	3
$w = 8$,r_i	3	3	3	4	3	3	3	3	2	2	3	2	2	3	2	3
l_i	2	4	2	3	2	2	2	4	3	3	3	3	3	4	3	4

Table 5: Shift amounts for $w = 32, 16, 8$

These shift amounts are the best choices, which means they have not just one choice. The main idea of the best choice is maximise the rate at which input changes through the computation. It provides some choices of shift amounts for MD6-35-8 and the word size is 16 as below:

i mod 8	0	1	2	3	4	5	6	7
R_i	5	2	6	2	5	2	6	5
L_i	7	5	5	3	4	3	7	3
R_i	4	3	4	4	3	2	5	7
L_i	3	7	5	6	5	5	4	4
R_i	4	5	3	2	8	4	8	5
L_i	3	7	4	7	6	3	6	6

Figure 11: Different pairs of shift amounts for MD6-35-8

The last pair is the best choice. It used 10000 samples to get this best choice and took 3 hours.

5.1.2 Constant Q

Q is a constant number. In standard MD6, each compression function has 15 constant numbers. These 15 numbers are the first 960 bits of $\sqrt{6}$. These 15 numbers are from $A[0]$ to $A[14]$, where $A[]$ is the initialise array with 89 items. The initialise array in includes 15 constant numbers, 8 key words, $1U$ word, 1 V word and 64 message words.
The values of Q refer to this equation: $(2 + Q)^2 = 6$, so $Q = \sqrt{6} - 2$, that means the fractional part of $\sqrt{6}$. For example:
$A[0] = Q_0 = 0x7311c2812425cfa0$
$A[1] = Q_1 = 0x643228643411c8e7$
This is for $w = 64$. When $w = 32$, $A[0] = Q_0 = 0x7311c281, A[1] = Q_1 = 0x2425cfa0$. The 15 constant words are the first 480 bits of $\sqrt{6}$ fractional part. For $w = 16$ $A[0] = Q_0 = 0x7311, A[1] = Q_1 = 0xc281, A[2] = Q_3 = 0x2425, A[3] = Q_4 = 0xcfa0$.
Consider this case: it does not have constant words. In one compression function an attacker uses just the last 16 same outputs to invert the input, because compression function is invertible. We can see it is easy to find collision. However if it have the constant, it is hard to find the collision, even if it does the same attack, because use the last 16 same outputs to invert the inputs is hard to meet the inputs should have 15 constants. Also someone maybe ask why use 6, because this hash function is MD6. So Q must not be 6, or it can be random, but it should be non-zero because of some of the algebraic attacks. Here we do not do more research about algebraic attacks.

5.1.3 Round Constants

The round constants S' provides differences between round, which strength its security[16, 21]. This constant uses in the first step of compress functions. Its values also refer to the word size as described Table 6:

w	$S*$	S'_0
64	0x7311c2812425cfa0	0x0123456789$abcdef$
32	0x7311c281	0x01234567
16	0x7311	0x0123
8	0x73	0x01

Table 6: $S*$, S'_0 for different word-size w

The algorithm is the below equation:

$$S'_{j+1} = (S'_j \lll 1) \oplus (S'_j \wedge S*)$$

So we can calculate arbitrary round constant. This is not a important constant for this project. It is automatically computed in the compression function.

5.1.4 Round number r

Round number determines the iterations that compress function run. The number of rounds refers to differential security of MD6. The compression function do same thing r times to make weaken function strong. Of course the inputs for every iteration are not same. This round number refer to the number of "AAG" and "AAG" provides differences. We use computation-aided method to search this number, but it takes large time. From we can know Getting figure 12 took 3 weeks. The original round 153 provided by MD6 team is not trusted by NIST, because it need large time to get the result and is too complicated to have some computational problem.
So this project will research smaller version of MD6 instead of standard MD6. The rules of choosing smaller versions don't have standard rules. We just choose 3 three versions for convenience.

- $n = 35, c = 8, w = 64, t_0 = 9, t_1 = 12, t_2 = 13, t - 3 = 19, t_4 = 34, t_5 = 35$. Here we call MD6-35-8.
- $n = 35, c = 16, w = 64, t_0 = 17, t_1 = 18, t_2 = 22, t_3 = 24, t_4 = 31, t_5 = 35$. Here we call MD6-35-16.
- $n = 17, c = 8, w = 64, t_0 = 9, t - 1 = 10, t - 2 = 12, t_3 = 14, t_4 = 15, t_5 = 17$. Here we cal MD6-17-8.

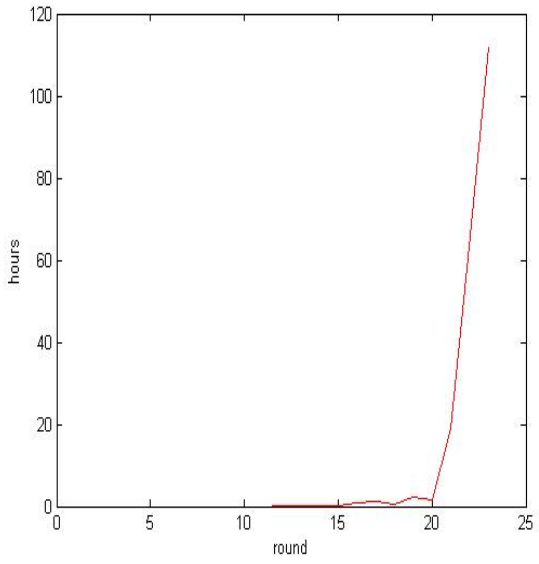


Figure 12: Spending time of first 22 rounds

We mainly discuss these three versions. When discuss these, they refer to another thesis wide-pip strategy. Stefan Lucks provides that the size of intermediate chaining variables have a significant effect on the security of overall shame[14, 15]. Actually it focuses on the rate of chaining variables and final hash output. In standard MD6 hash function, the chaining variable is the final compression output with 16 words (1024-bit) and the standard output is 512 bits, which meets double-pipe strategy. It prevent "multi collision attacks" studied by Joux[10], Nadi et al. [18, 19] Houch et al. [8], and Yu et al. [29]. MD6 adopt a wipe-piped strategy: all internal chaining variables length *cw* is at least as twice as final output length.
For MD6-35-8 and MD6-17-8, the final hash output length is less than or equal 64. The final hash output length of MD6-35-16 is less than or equal 128. Their structure will be different. The standard MD6 is 4-to-1 tree based structure as figure 13. But we think 4-to-1 tree based structure is not suit for our versions of MD6. In the structure every node is a compression function. A leaf is a message block and the white and grey leaf is padded some zeros.
For MD6-35-8 it can use the 4-to-1 compression function. If like that, its blog message length is 8 words. U and V together is one words. Key is one

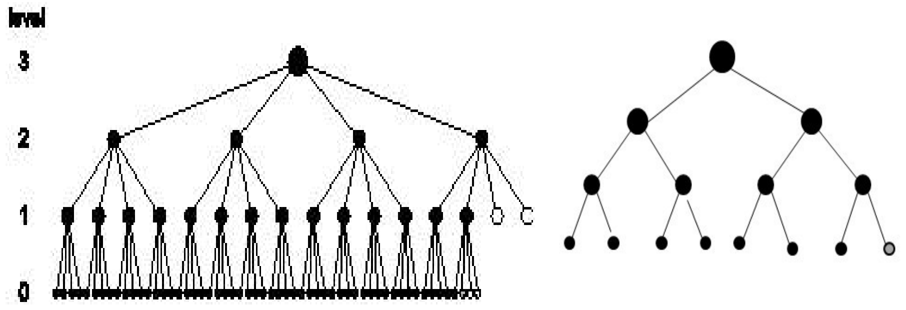


Figure 13: Structures of 4-to-1 and 2-to-1 compression functions

words and constant Q is one word. We choose its output length is 128 bits. We encrypt the same plaintext:

I am a student of university of bristol. This is my project
5e463aecb6cf70879c3410bc0afed58d9

The other two versions of MD6 we did not get the result. Because it maybe need change the whole structure as the 2-to-1 compression function of figure 12 and also need consider U and V. However these parameters don't effect this project. This project mainly focus on n and c and it does not focus on how to implement MD6. Or maybe $n = 17, c = 8$ is not practical, but when we analyse the lower bound of differential resistance, we consider the compression function, not the whole MD6, so we don't care the structure[13].

5.2 Time of finding lower bound of differential security

From Figure 11, we know it took almost 3 weeks to search lower bound at the first 24 rounds, so we use the equation $AAG_r \geq AAG_s \times \lfloor r/s \rfloor$. This equation make sure we get the AAG number of r rounds is the least in those rounds can be differential resistant, because differences are more and they would generate more and more differences. For example
Consider why it took so long. Figure provided Ethan heilman method for finding different weight pattern path and AAG.

Ehtan Heilman's mehod:

```
Initialise: $AAG[], D[], maxAAG$
  for $i$=$minsteps$ to $maxsteps$ do
    $if(non-trial)$
      $search(n)$
    $else$
      $search(0)$
    $maxagg++$
    $i+c$
    $maxagg > maxAAG$
```

```
$search()$
    for $D[i] = 0$ to $D[i] = 64$
    $if$(path is invalid)
      continue
    calculate $newaag$
    $if$ ($i < n$ and trial )
      search($i++$)
    $if(i > stepLimit)$
      return
```

From his method we can see indie $search()$ function for loop has recursion function, the time is mainly spent here. We provide the complexity of his method $O(wcn!)$, because w and c are constant, so the complexity of his method is $O(n!)$. Table provide the number of recursions at round 20 to 24. They are very large.

Round	Recursion
20	39766911864
21	19437792855
22	260232205617
23	613555007229
24	1279185322699

Table 7: Recursions at round 20 to 24

The recursions of round 24 is near to (16!). The main idea of this method is to search all valid path that all of them can provide enough "AAG". It always search from every word with smallest difference, which means $D[i] = 0$ to $D[i] = 64$. So it always find the differential path that generate the smaller "AAG", but this number is enough for differential security. So we can see his result about different weight pattern path like this 16.1 32.1 71.1 105.2 160.2 177.2 for round 23 where 16.1 means the 16th word has 1 different bit. This path generate 55 "AAG", when round is 23. Other valid differential paths of round 23 generate at least 55.
For me it is challenging to analyse this. We can't wait one month just for one result. However this method is through optimisation.

5.2.1 Optimization

As the previous part described , It divided the valid path into trial and non-trial path. For non-trial path it would skip the first rotation, it works well. From Figure 13 we can see, they process the same step 192, but the time is very different. The method check if the path is valid or not. From the

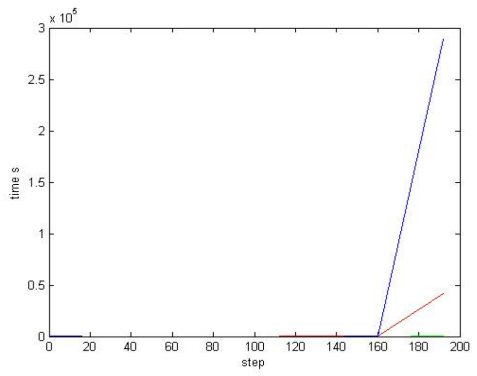


Figure 14: Time of processing 192 steps. Red line represent trial path, blue one means non-trial which skip the first rotation, and green one means non-trial pattern which does not skip the first rotation. Surely non-trial pattern took much more time than trial pattern. Also we can know from the Figure the tendency in cress quickly like $O(2^n)$ and actually it is $O(n!)$. So it will be hard to get the result after192 steps.

previous part's analysis, we already know three cases. Here it will summarise them

- $LB_i \geq 0$, $UB_i \leq w$, where LB_i is lower bound on input diffs and UB_i is upper bound on input diffs.
- if the possibility of input diff is not zero, the output diff can't get zero.
- the differences in $A[i]$ can't be more than 4 times UB_i, before we analyse this.
- if the difference in one item of $A[]$ equal 1, its $UB_i \geq 5$.
- Any difference is in same AAG. it just count one time, for example $(I_1 \wedge I_2) \oplus (I_3 \wedge I_4)$, $AAG_{I_1,I_2} = max(I_1, I_2), AAG_{I_3,I_4} = max(I_3, I_4)$
- Any difference is in the same step, the differences just count one time, so $AAG = max(AAG_{I_1,I_2}, AAG_{I_3,I_4})$.

- the all input of one compression function must not fall off the beginning of a segment.

These optimisations decrease the time a lot. However it is still for us too difficult to research. So we research the smaller versions would be good choices. We maybe can parallel his method, because MD6 works parallelism well. We tried to use OpenMP do it, finally we did not succeed. Latter we will analyse it more.

5.3 A bug at Ethan Heilman's code

As before we said Ethan Heilman restore the Differential resistance of MD6. He improved the original method and added his own values. He mainly provides these contributions. He divides the differential paths into trial and non-trial paths. He restored the round of differential resistance, gave his analysis of the restored round number and added the test file to check his method logic.
When I tried to use his method to search the smaller versions of MD6 I chosen. It has some errors, Which can't pass his test file. This test file is to check his method logic. If it can't pass through, that means his method is not correct. The main idea of the test file is to use some special example to test its result with using his method, because the test file provide the simple method to calculate AAG for special examples. Table give the simple logic.

test-epectedAAGs Algorithm
test zero to four differences
test four differences of different values
test four differences of very different values
$nAAGs+ = max(max(I_i, I_2), max(I_3, I_4))$, this $nAAGs$ should be same with his method
test-ComputeBound Algorithm()
test differences in the six input of compression functions.
The differences are may be in beginning or end or middle.
the main equation: sum $=D[n-t_0] + D[n-t_1] + D[n-t_2] + D[n-t_3] + D[n-t_4] + D[n-t_5]$

Table 8: test-epectedAAGs and test-ComputeBound Algorithm

If its code can pass the 11 steps test, it prove its method logic is correct. He uses his method inside the method of searching differential weight pattern

path to get results and then compare with method in Table 8. We tried to use this code with small change to find the lower bound of smaller versions of MD6 hash function, because its test file is logical. It randomly use one case of his method as test file. So it can be trusted, but it must not be say this method is completely correct. Because it does not test every cases.
When we tried to use it, we find the code has some errors in Ethan Heiman's github. Through my analysis, I think it is something wrong in his code, especially in the md6parts.c file. At first I think maybe its code is not compiled in unix and can't use GCC to compile. I ran it on Microsoft visual studio 2008. It also can't pass the test file. So I suppose the logical of his code is wrong. I asked my supervisor and my supervisor also confused about this errors. So I decided to email this error to Ehtan Heilman. He told me he would check this problem.
I tried to solve this problem, because actually I think the logic of the method is right. I found something strange, especially one array's value as Figure 15 described

Bugs on the code	
Original code	The code I changed
int DAMP[ ] = 0, 7, 12, 15, 16	int DAMP[ ] =0, 1, 2, 3, 4, 5, 6, 7, 8, 9
15, 12, 7, 1, 1, 1, 1, 1, 1, 1, 1	10, 11, 12, 13, 14, 15, 16, 17, 18
1, 1, 1, 1, 1, 1, 1, 1, 1, 1	19, 20, 21, 22, 23, 24, 25, 26, 27, 28
1, 1, 1, 1, 1, 1, 1, 1, 1, 1	29, 30, 31, 32, 33, 34, 35, 36, 37, 38
1, 1, 1, 1, 1, 1, 1, 1, 1, 1	39, 40, 41, 42, 43, 44, 45, 46, 47, 48
1, 1, 1, 1, 1, 1, 1, 1, 1,1	49, 50, 51, 52, 53, 54, 55, 56, 57, 58
1, 1, 1, 1, 1, 1	59, 60, 61, 62, 63, 64
ComputeBounds Bug	
if (TRUE)	if (FALSE)
$max\_t1 = D[i - t_1] * 7$	$max\_t1 = D[i - t_1] * 7$
$min\_t1 = D[i - t_1]$	$min\_t1 = D[i - t_1]$
else	else
$max\_t1 = D[i - t_1]$	$max\_t1 = D[i - t_1]$
$min\_t1 = D[i - t_1]$	$min\_t1 = D[i - t_1]$

Figure 15: Bugs I found

The array values of DAMP[ ] are not right, it means the i-th word different bits. DAMP[i] represent i-th word can have DAMP[i] different bits. So it can have 0 to 64 bits for word size 64. In compute bounds bug, max and min are used to calculate LB_in and UB_in. UB_in should be equal $min(w, D[i-t_0]+D[i-t_1]+D[i-t_2]+D[i-t_3]+D[i-t_4]+D[i-t_5])$.

$$LB\_in = (max(D[i-t_0], D[i-t_5]) - min(D[i-t_0], D[i-t_5])) - and\_diff$$

$$and\_diff = D[i-t_1]+D[i-t_2]+D[i-t_3]+D[i-t_4]$$

I changed the code and ran it on Unix. It pass the test file and ran it about one week. I get the same result that Ethan Heilman posted. I emailed it to Ethan heilman. He agree with what I changed and correct his gihub code. We discuss what I researched. He thought $n=35, c=8, w=64$ is a good choice to research.
However the logic of Ethan Heilman 's method is correct and the method is more efficient than the original method. From the analysis in the previous part it divide the differential path into trial and non-trial path. In the trial path it have less than 4 different positions. In the non-trial pattern it has 4 or more than 4 differential positions. one differential weight can at most activate 4 AAG. In the non-tiral pattern it will skip the first rotation and at last it will add extra 4 AAG. It assume that the first rotation generate 4 AAG. 4 AAG number must be less than the number of AAG it actually produced in the first rotation. But it does not have effect in differential security. Within 48 steps it has a lower limit of AAG, it always is 2 AAG. So it is reasonable that it add 4 extra AAG. From my result and Ethan Heilman's result can prove this. The Figure shows my result and Ethan Heilman's result.

	n=89,c=16,w=64 within 144 steps	n=35, c=8,w=64 within 48steps
AAG:	2	2

Figure 16: AAG number within 148 steps in standard MD6 and 48 steps in smaller MD6

If it adds 4 additional AAG, it will compute more rounds. So it maybe our result maybe is not very accuracy. However we compute also can compute the non-trial pattern which does not skip the first rotation, then compare the three result: the first one is the trial-pattern result, the second one is the non-trial pattern which skip the first rotation and add 4 extra AAG and the third one is non-trial pattern which does not skip the first rotation. The method is strait forward.

We tried to use 6 extra AAG numbers, but maybe the result is not very accuracy .

5.4 Aims

Until here we have already determine the smaller versions of MD6: MD6-35-8, MD6-35-16, MD6-17-8. The aims of this project is to get the lower bound of differential security. It will use Ethan Heilman's method to get those result, because at the previous part, we analyse that the method is logical and efficient. Also it will find the special one of all valid differential path. This special one is to generate the least AAG within all of valid differential path. We should analyse the result. Because it is smaller, maybe we don't need use $AAG_r \geq AAG_s \times \lfloor r/s \rfloor$ the s. We can get the actual rounds number. However so far we don't have good method to test the correctness of the result. We can just analyse its logic. Aslo we directly get the AAG number of r round and can test the correctness of the equation. For saving time we try to use OpenMP to parallel Ethan Heilman's method. There are lots method that can parallel it, because of the requirement of hardware we just use OpenMP, not MPI or OPenCL.
We don't have method to prove the correctness of the result, but we can check its relations within round, the tendency of the number of recursion and using comparative method to analyse them. Also we want to analyse the smaller versions of MD6 hash function is practical and security. We tried to provide the exact round number of smaller MD6-35-8 hash function, because we find the structures of MD6-17-8 and MD6-35-16 are different with the standard MD6 hash functions, so they are maybe not practical.

5.5 Analysis of Differential path

This part will combine experiment with the pervious theories to analyse differential path and also prove the correctness of the theory. At first we provide one pair of results as Figure 17 . From it we can see the differences generated in 5 rounds. The result is provided by standard MD6 hash function.

Hash message “abd”	Hash message “abe”
A[153] = 2b982d428e2c06d0	A[153] = 2b98cc24f3e6d6c0
A[154] = 61d59fd2b7310353	A[154] = 61d59fd2b7310353
A[155] = ea7da28dec708ec7	A[155] = ea7da28dec708ec7
A[156] = b73eccb034f50555	A[156] = b23f89b174f50555
A[157] = 290b4fabe80104c4	A[157] = 290b4fabe80104c4
A[158] = 8c6a3503cf881a99	A[158] = 8c6a3503cf881a99
A[159] = e370e23d1b700cc5	A[159] = e370e23d1b700cc5
A[160] = 4492e78e3fe42f13	A[160] = 4492e78e3fe42f13
A[161] = df6c91b7eaf3f088	A[161] = df6c91b7eaf3f088
A[162] = e0a57fe83bc2f822	A[162] = a5a7e1ca1392f80a
A[163] = c587b5fb3a6e104f	A[163] = c28db5f3023e104f
A[164] = aeae805de12b0d24	A[164] = aeae805de12b0d24
A[165] = 28867842491166fa	A[165] = fd2ef040119654c4
A[166] = a86fe7b42deacbe2	A[166] = 993755651922d73f
A[167] = 6bf0761b560bf001	A[167] = eb4ce6bd52cfc8eb
A[168] = 41204e9e3349c8c8	A[168] = 43b92c8959bca4cc

Figure 17: Message “and” and “abe” hash value in 5 rounds

These two messages just has one different bits and only need one compression function. So when we analyse this , it equally analyse one compression function. The inputs just at A[25] are different. The “abd” of A[25] is 6162640000000000 and the “abe” of A[25] is 6162650000000000. Other A[25-88] are padded by zero. Their intermediate value at A[114] start to have different. The inputs of A[114] is A[97], A[98], A[93], A[83], A[47] and A[25]. Because A[25] is not same. Within 5 rounds, 1 bit difference generate 8 different positions. A[114]=1040090486e3df85 for abd, and A[114]= 10400c048ce3df85 for abe. They have 4 bits differences. Because the A[25] does the $\oplus$ operation, it transfer the differences. From these result we also can discuss the differential weight hamming pattern and differential position. A[114] is a differential position and its weight hamming is 8. We usually express this like 114.8, which means the 114 item of the array have 8 weight hamming. When it introduces these two theories, we don’t need care which exact bit is different. it is convenient for us research and also we don’t need know the exact differential bit.

hash for abd within 150 rounds	hash for abe within 150 rounds
A[2473] = 44e56318c2de95ac	A[2473] = 461a1ccb3d821091
A[2474] = bb877a9cd0ec692b	A[2474] = bc31e5cc76012a35
A[2475] = 46fbd8357ee04b4b	A[2475] = 020c5ded52cfaf23
A[2476] = e624716720fcd743	A[2476] = 03217a411f7dcde9
A[2477] = 2985e14a066cd257	A[2477] = fc4b6b98ee02c9ec
A[2478] = 9e8a5d55f64a95ef	A[2478] = 181ab5a1e7cb189f
A[2479] = 920e0c156cf7ac48	A[2479] = 2cc33dc5eaf40991
A[2480] = 428c7f95a67ca35f	A[2480] = 47d39369a088b8e3
A[2481] = 567c39631cb1f075	A[2481] = 5c65eb5baec8a696
A[2482] = 759bf191a085dda8	A[2482] = becca3e4d5dad6c3
A[2483] = ddb11b051ccdda02	A[2483] = 8a2252a71ed24b89
A[2484] = ec5c190e5073aeb2	A[2484] = 06484721b3fb7653
A[2485] = ca913d3048d9b06f	A[2485] = 157afbdb3e25293b
A[2486] = 84e559ecf6ca0acc	A[2486] = de4bbc690203c872
A[2487] = 9d8e424fc98782e0	A[2487] = 4693d2cccf7f0ec1
A[2488] = cb51aa085571644c	A[2488] = 991fe9ca38816207

Figure 18: Message “abd” and “abe” hash value within 150 rounds

The last 16 pair outputs are totally different. Obviously it is more security than within 5 rounds, as the results in 5 rounds have 8 same. Iterated functions make the weaken compression function more security. From the previous analysis, if it want to resist differential attack, it refers to this round number and the digest length. Also we can see every item of the array is effected by its 6 inputs, which are within its previous n items. It tests the whether tap positions choose well or not. If the tap positions are not good, the diffusion of the array values are not good. Also tap positions $t_2 - t_1 = t_4 - t_3$ is not good, also because of the same reason.
Ethan Heilman's method aims to find the valid differential weight hamming pattern. The difference pattern from Figure 19 is finded within corresponding rounds in standard MD6 hash function.

Round	Difference pattern	AAG number
14	48.1 65.2 137.2	19
15	33.2 88.1 105.2 177.2	22
16	33.2 88.1 105.2 177.2	22
17	33.1 88.1 105.2 177.2 266.2	30
18	35.1 69.2 124.2 141.2 213.2	34
19	37.1 71.2 126.2 143.2 215.2	35
20	16.2 54.1 88.2 143.2 160.2 232.2	38
21	16.1 33.2 71.1 105.2 160.2 177.2 249.2	39
22	16.1 33.2 71.1 105.2 160.2 177.2 249.2 338.2	47

Figure 19: Difference pattern example

We get this result in standard MD6 hash function and the word size length is 64. The fragment of the number means the differential weight hamming, so it should be in the range of (0,64), and it can equal 64 but not 0. However we can see the differential weight hamming is very less. it is because that this is just one valid difference pattern. Also his method increase from 0 different weight hamming to 64, which make sure the first found difference pattern can generate lower bound of differential security.
In round 17 it get the difference pattern: 33.1 88.1 105.2 177.2 266.2. The total weight hamming is $1 + 1 + 2 + 2 + 2 = 8$, so it can produce $8 \times 4 = 32$ AAG theoretically, but it produce 30 practically. At the previous part, we discussed this. It has three case, the difference weight hamming can't generate the AAG.
Round 15 and 16 have the same difference pattern,but the AAG number they created also are the same. It also indirectly prove his method correctness because of the happiness of some special case. In round 21it have differences at A[16]. Here we should recall the structure of the input of compression functions. The constants Q always is from A[0] to one item, then it is key word, U,V and it is message block at last.
However we use the same default key word, which default key word is empty. But here the A[16] actually is the A[16+25]=A[41],because the array starts from message block. Here also it has one place to explain: the less difference, the less AAG. So for each round it wil add one new difference based on the previous differences. This also is the main part of Ethan heilman's code.

5.6 Lower bound of smaller MD6 for differential security

This part provides the exact lower bound for MD6-35-8, MD6-35-16 and MD6-17-8 smaller hash functions. We did not find a good method to prove the correctness of the number, but it provides our analysis.

5.6.1 MD6-35-8

We discuss about this version with Ethan Heilman. He also agreed that this is a good version to research the MD6 properties. It can meet the 4-to-1 compression function and wipe-pipe strategy. In this version some parameters should be changed. The input length can't be 2^{64}, because the input n and output length w is smaller[24, 20]. Also the length of U is smaller. In standard MD6 it is one-word length 64 bits, while it is half word length 32 bits. In the tree structure there will be at most 12 levels (since $12 \approx log_4(2^{32}/2^9)$. So L can be changed into 32 by default. Half word size 32 is enough for 12 levels and is enough for the level index. We discussed other parameters like tap positions and shift amounts at the previous part. Here we don't provide again.
We used Ehtan Heilman method to calculate the lower bound about this. I did a change at his code and get our result. Although we changed the size of MD6, it still took a long time to run it. We can't expect the code to run all the time until get more than 40 rounds. So we still have to use equation:

$$AAG_r \geq AAG_s \times \lfloor r/s \rfloor$$

This version of MD6 can't have the big output. 160 bits and 128 bits will be suit for. When we research the lower bound of differential security. We don't need consider parameter Q,U and L. Actually we just consider the tap positions, n, and c. Because when we consider the differential security of MD6, we should analyse the compression function. Most time Q, U and L are same and don't have differences. Ethan Heilman's method is independent with MD6 hash functions.
Our result provided as below. We took 2 days to get this results.

Round	Recursion	Time (s)	AAG
13	91048754	5	24
14	459870857	45	32
15	1278650643	71	36
16	1528976753	104	39
17	4708374131	400	44
18	17405341778	2031	52
19	21853325991	1300	54
20	82112464513	6920	58
21	525898766014	17707	62

Figure 20: Recursions , Times and AAGs within 21 rounds for MD6-35-8

Round	Difference pattern
13	17.1 18.1 34.1 43.2 69.2
14	17.1 18.1 34.1 43.2 69.2 104.2
15	13.2 20.1 39.2 55.2 64.2 90.2
16	18.2 25.1 44.2 60.2 69.2 95.2
17	14.2 31.1 40.2 65.2 66.2 75.2 101.2
18	14.2 31.1 40.2 65.2 66.2 75.2 101.2 136.2
19	12.5 14.2 21.1 40.2 47 66.2 82.2 91.2 117.2
20	13.2 20.1 39.2 55.2 64.2 89.2 99.2 125.2
21	5.1 14.1 30.1 40.2 65.2 75.2 100.2 110.2 135.2 145.2

Figure 21: Difference pattern within 21 round for MD6-35-8

From Figure 20 we can see that the time increases quickly, tend to 2^n. We know the method complexity is $O(n!)$, which is similar with $O(2^n)$. It looks a bit strange that the time of spending in round 18 is more than the time in round 19. Also in round 10 it took much less than them. We can analyse the case from Figure 21. Figure 201we Know at round 19 the first position is 12, and the different weight hamming is 5. It has the same 5 positions with round 8 and it end the last differential position earlier than round 18. So it is reasonable that it took less time than round 18. However the recursion is much more than in round 18. We can explain this because of the first different position which differential weight hamming is 5. And also its total differential weight hamming is bigger than round 18.

Round	Total differential weight hamming	practical AAG	theory AAG
13	1+1+1+2+2=7	24	28
14	1+1+1+2+2+2=9	32	36
15	2+1+2+2+2+2=11	36	44
16	2+1+2+2+2+2=11	39	44
17	2+1++2+2+2+2+2=13	44	52
18	2+1+2+2+2+2+2+2=15	52	60
19	5+2+1+2+1+2+2+2+2=19	54	76
20	2+1+2+2+2+2+2+2=15	58	60
21	1+1+1+2+2+2+2+2+2+2=17	62	68

Figure 22: Comparation of practical AAG and theory AAG

The differential weight hamming means the number of different bits. From the previous analyse 1 bit at most generate 4 AAG, which is the theory AAG. No practical AAG are bigger than theory AAG. Also it prove the correctness of the equation $AAG_r \geq AAG_s\lfloor r/s \rfloor$. However it is just the result of trial pattern. But their technicals are similar. We compare them at figure

Round	Trial pattern AAG	Non-trial pattern AAG
11	19	16
12	22	17
13	24	20
14	32	41
15	36	43
16	39	44
17	44	44

Figure 23: Comparation of Trial and Non-trial pattern

The result of Non-trial pattern AAG took 18 hours. The number of AAG at trial pattern and non-trial pattern don't have some certain relationship. It can find that the AAG number of round11 at trial pattern is bigger than at non-trial pattern. However the number of round 14 at trial pattern is smaller than in non-trial pattern. Also the number at 13 and 14 in non-trail pattern had a big gap. We consider the differential weight hamming pattern first as figure 24 described:

Round	Difference pattern
13	44.1 45.1 46.1 47.1 69.1
14	48.1 53.1 54.1 60.1 79.2 82.2 89.2

Figure 24: Difference pattern at round 13 and 14 in non-trial pattern

We can see the differential weight hamming increased obviously. Why this is happen maybe is because that the previous difference in round 13 all do $\oplus$ operations. Latter we will assemble the trial and non-trial pattern.

5.6.2 Results

We assemble the trial and non-trial patterns and get the final results as Figure show:

Resistant rounds	MD6 rounds	Security margins	Output size	AAGs
40	55	15	158 bits	79
60	80	20	284 bts	142
80	100	20	376 bits	188
100	120	20	482 bits	241
120	140	20	574 bits	287

Figure 25: Given the output size, the result of MD6 rounds, resistant rounds and security margins

We get this result using the trial and non-trial patterns as Figure 23 shows. We use the equation $AAG_r \geq AAG_s \times \lfloor r/s \rfloor$ get two list respectively for trial pattern and non-trial pattern. Then we choose the the smaller AAG numbers as the AAG number that the round generates. In Figure 22, 11 rounds generate 19 AAGs in trial pattern, but it generates 16 AAGs in non-trial pattern. So assembling them get that 11 rounds generate 16 AAGs at last. In Figure 24 the output size is 2 times resistant rounds. Security margins depend on designers and hardware. So in Figure 25 we can choose 15 or 20 as security margins. The number of security margins is more and it is more safe.
The size of output size in figure 26 is large for MD6-35-8. The below chart shows the number of round for output sizes we choose.

Output size	Resistant rounds	MD6 rounds	Security margins
96 bits	27	45	18
128 bits	30	50	20

Figure 26: Rounds number of differential security for MD6-35-8

5.6.3 MD6-17-8

This part discussed the smaller version MD6-17-8. Before we analysed its structure, tap positions and shift amounts constant. We are not sure if the structure is right or not. But we think the change of the structure won't change the property of MD6, because the 4-to-1 tree based structure is used for parallelism. It won't effect other properties of MD6. The technics of compression functions will be same. It also include feedback functions and $g(x)$ operation. Also it will process $r \times c$ step and finally truncate the last c output. So we still use this version as one we researched. We also analyse them from our result we got. These result took less than one day because of its smaller size. It obviously took less time than the bigger versions. Firstly we analyse its recursions, time and AAGs. We provided the results as below:

Round	Recursion	Time (s)	AAG
7	9427156	1	32
8	17709365	1	36
9	212770626	22	45
10	3714430165	283	51
11	219840003583	18688	59

Figure 27: Recursion, Time and AAG within 11 rounds for MD6-17-8

As we expected, the number of AAG, recursion and time increases as rounds increase. We also got the results of 0 to6 rounds. We did not list them. We got those of result fast, just 1 second. At round 4 it create 8 AAGs. According $AAG_r \geq AAG_s \times \lfloor r/s \rfloor$, at round 9, 10 and 11 they should be bigger than $2 \times 8 = 16$. they are correct. Compare with the result of MD6-35-8 and MD6-89-16. It is obviously that in the same round the smaller size of MD6 hash function create more AAGs. It is because that the round and the rotation are smaller and the diffusion of the words are more density. For example, in MD6-17-8, the best tap positions are 9,10, 12, 14, 15 and 17 respectively. if the 30th word(A[29]) is a differential position and it will

quickly effect A[38], while in MD6-89-16, it will effect the A[96] first. Also this happens because of the number of rotation.
Then we analyse its differential weight pattern as below:

Round	Total differential weight hamming	Practical AAGs	Theory AAGs
7	1+2+2+2+2=9	31	36
8	1+1+2+2+2+2=10	36	40
9	1+1+1+2+2+2+2+2=13	45	52
10	1+1+2+2+2+2+2+2=14	51	56
11	1+1+2+2+2+2+2+2+2=16	59	64

Figure 28: Comparation with practical AAGs and theory AAGs

The practical AAGs is smaller than theory AAGs, so these results are reasonable. The following will analyse the differential weight pattern from another angle. First we see one pair of figures.

Rounds	Difference pattern	AAGs
7	7.1 24.2 32.2 41.2 42.2	32
8	7.1 15.1 32.2 41.2 49.2 50.2	36
9	4.1 12.1 13.1 29.2 30.2 38.2 46.2 63.2	45
10	4.1 12.1 29.2 38.2 46.2 47.2 55.2 63.2	51
11	7.1 11.1 28.2 37.2 45.2 46.2 54.2 62.2 79.2	59
fixed first and second differential positions (7 and 15)		
7	7.1 15.1 16.1 31.2 32.2 33.2 40.2	35
8	7.1 15.1 32.2 41.2 49.2 50.2	36
9	7.1 15.116.1 32.2 33.2 41.2 49.2 66.2	46
10	7.1 15.1 32.2 41.2 49.2 50.2 58.2 66.2	52
11	7.1 15.1 32.2 41.2 49.2 50.2 58.2 66.2 83.2	60
fixed first and second differential positions (4 and 12)		
7	4.1 12.1 29.2 38.2 46.2 47.2	35
8	4.1 12.1 16.1 33.2 42.2 50.2 51.2	38
9	4.1 12.1 13.1 29.2 30.2 38.2 46.2 63.2	45
10	4.1 12.1 29.2 38.2 46.2 47.2 55.2 63.2	51
11	4.1 12.1 29.2 38.2 46.2 47.2 55.2 63.2 80.2	59

It tries to find the difference pattern that can meet the requirement of AAG, but it create smallest AAG in all the valid difference pattern. For example unfixed difference method created the difference pattern 7.1 24.2 32.2

Figure 29: Comparation of difference pattern with unfixed and unfixed first and second differential positions

41.2 42.2 with 32 AAGs, but the fixed difference method created the difference pattern 7.1 15.1 16.1 31.2 32.2 33.2 40.2 with 35 AAGs. The unfixed difference method always tries to find the appropriate one that not only requirement the AAGs and also the number of AAGs is the smallest one in all the valid difference pattern at certain round. The fixed differential positions use less time, because of they have two fixed values, the saved time depends on the two differential positions, of course the first two differential positions should be greater than 0 and less than n. if we want to find the all of the possible two fixed values, it need times and running all of them will spends large time. It need run on the multi-core mat chine, and can parallel run all the cases and then compare them. Because MD6-17-8 is very smaller, the output size also should be very smaller. 40 bits and 80 bits should be suit for MD6-17-8. It provided its result for the two output size with differential security as below:

Output size	Resistant rounds	MD6 rounds	Security margins
40 bits	6	12	6
80 bits	9	15	6

Figure 30: Result for MD6-17-8 hash function

5.6.4 MD6-35-16

We did not spend lots time on this version, because it is similar with MD6-17-8. $n/c \approx 2$. The structure of MD-35-16 is similar with MD6-17-8. They both are not standard 4-to-1 compression function tree-based structurer. It analyse it as before. First it provides its recursion, time and AAG for certain round.

Round	Recursion	Time (s)	AAGs
7	129993851	9	30
8	349883182	66	38
9	1486220358	212	45
10	13025447313	1038	53
11	45866196383	4532	61
12	558062670839	33426	69

Figure 31: Recursions, Times and AAGs for MD6-35-16

it took more than 2 days to get all the results about MD6-35-16. We also got its unfixed and fixed difference pattern. The first two differential position in 20 and 30 in that fixed difference pattern as below:

Rounds	Difference pattern	AAGs	Time (s)
6	12.1 30.1 65.2 82.2	23	3
7	8.1 26.1 61.2 78.2 95.2	30	9
8	8.1 26.1 61.2 78.2 95.2 112.2	38	66
9	6.1 24.1 59.2 76.2 93.2 110.2 127.2	45	212
10	6.1 24.1 59.1 76.2 93.2 110.2 127.2 144.2	53	1038
fixed first and second differential positions (20 and 30)			
6	20.1 30.1 37.2 47.2 54.2 64.2 72.2 81.2	51	0
7	20.1 30.1 37.2 47.2 54.2 60.2 65.2 78.2	58	2
8	20.1 30.1 37.2 47.2 54.2 60.2 65.2 78.2 58.2 113.2	65	19
9	20.1 30.1 37.2 47.2 54.2 60.2 73.2 91.2 109.2	69	44

6 Parallism

The important character of MD6 is that it works parallel well. It is tree based structure and tree based structure is the best parallel structure for hash functions[12].

In the MD6 report it have more detailed analysis about parallelism. Here we provide our understanding of parallelism for MD6 and our project. when analysing parallelism of MD6, it refers to hierarchical mode of operation. The mode parameter is L. It divided the structure to each layer. One level would be one barrier, which means all of the compression functions in the same level should be finished and then get into the upper level.

We consider the requirements of hardwares. We tried to use OpenMP(Open Multiprocessing) to implement parallel. OpenMP provides an API which can used in C, C++, Fortan[9, 6]. Here we simply discuss the technics of OpenMP. It is multithreading programming. It has a master thread and it can forks a number of threads working for one task as described in Figure 25. For us it must not need multi-core machine. It can use at single core machine. We just insert OpenMP language into our serial code and add its compiling directives (-fopenmp), our compiler would recognise the OpenMP language. So in this case we don't have GPU or multlicore machine, we

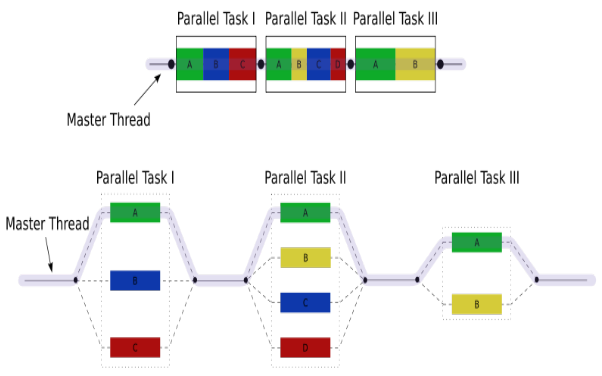


Figure 32: An illustration of multithreading[**?**]

can use OpenMP. Using OpenMP have two aims: make MD6 original code parallel and Ethan Heilman method parallel. Because in Ethan Heilman's code, it have lots of For loop, it easily make us expet to use OpenMP.

6.1 OpenMP for Ethan Heilman's code

We know Ethan Heilman's code took long time. Because its function is recursion function and it is in a for loop. OpenMp mainly works in for loop. However in Ethan heilman's code each item of for loops it will effect each other and the logic of the code can't be paralleled. But we think one for loop in main() function can be parallel. pseudo code in main () described in below:

For loop in main()

```
for ( stepLimit = arguments.minSteps; stepLimit $\leq$ argument.maxSteps;
    StepLimit += arguments.stepInc);
    initialise variables and calculate rounds and rotation;
    search();
    while( success )
        if()
            maxnagg++
```

OpenMP directive in this for loop

```
pragma mop parallel for schedule( static) private( stepLimit, success,
        rounds, rotations, numOfRecursions);
```

Other parameters are public by default

We set the stepLimit, success, rounds, rotations and numOfRecursions as private, so each thread will have their own these parameters. The arguments.stepInc means the number of step in one round. Our aims is to separate every rounds to calculate their own AAGs generated in its own rounds. Here it has one problem. We can't deal with the parameter *maxnagg* very well. If we set it as public, that means every round can access to this parameter. It exist this case that round 4 tries to change *maxnaag* and round 5 also tries to change *maxnagg* at the same time. Althrough OpenMP can make it synchronise, it effect round 5 *maxnagg*. Because the *maxnagg* of round 5 is based on round 4. It must wait that round 4 finish. If we set it as private, it looks like good. But it still has problem. It also needs the previous *maxnagg*. We tried to set a new variable, but it should wait for the finish of previous *maxnagg*. So the logic is still serial not parallel.
So we should change the whole logic of main() function, but we don't have time to do that, also it is not my main part of my project, so I did do it during the previous three month. This will be my future works.

6.2 OpenMP for MD6

This part discusses parallel for MD6 hash algorithm. It can mainly use parallelism in compression functions. In the compression function the steps of each round don't have effects. Because one item of one round is effected by its previous 89 items, actually it will be effected by 6 inputs of compression function, and the 6 inputs are not possible in the same round. Because the smallest tap position is t_0 and this number must be bigger than c. We can

use OpenMP, MPI and OpenCL to parallel it. We simply used OpenMP in the compression function. We must set barriers at the end of every level. Every compression function should wait for finishing all of the compression function at the same level. However I tried to change the code in the original code , it has one problem that the point of finishing all of the compression functions. I did not figure it out.
Here we provide a figure that shows parallelism works indeed well in MD6. It tends to a linear increase. When I do research we don't choose the large

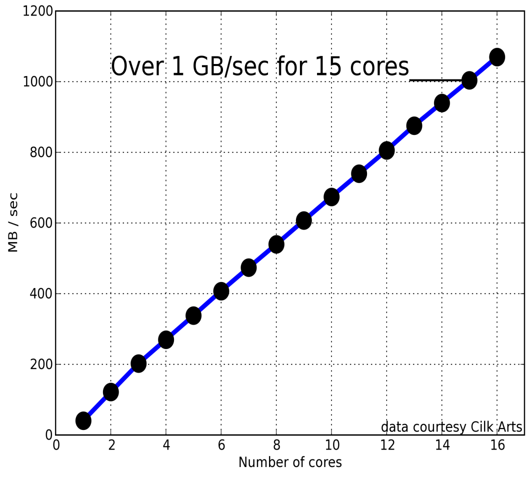


Figure 33: Chart of MD6 speed using various numbers of cores on a 16-core machine using CLIK[23]

input, so we don't need parallel the code. Also it won't effect my project, but it will be a good aspect for future works.

6.3 Future work

In this project we met some problem. We choose the three smaller versions of MD6, MD6-17-8, MD6-35-8 and MD6-35-16. We are not sure of the structure of MD6-17-8 and MD6-35-16. But the structure won't effect our project. To solve this problem need understand the relationship of the steps of one round, one block message size of compression function input and compression function output.
It provided the result of lower bound of differential security for smaller MD6, but we don't know whether these results are correct. We need super com-

puter to calculate all the AAG numbers that all the valid differential path create. We can use the first two fixed differential path to traversal all the valid differential paths. This paper use the first two differential path, whose the first two differential positions are 0, to traversal all the valid differential paths.
Ethan Heilman's code has many for loops and recursion function. We can transfer this recursion function into loop and then parallel them using OpenMP or OpenCL. This will help the research of differential security of MD6 hash functions and also will be good for other research of MD6.

7 Conclusions

In these month's researching I have a good understanding MD6 hash functions. When I started to do my project, I analyse my feedback and discussed my feedback with my supervisor. It said I did not have a good understanding MD6 from the feedback. My supervisor also thought doing my project required a good understanding of the whole MD6, like the input, output, compression function, tap positions , shift functions, mode structures and so on. I spent almost one month to analyse the MD6 report it includes 232 pages and then I combine its code to help me exactly understand MD6 hash functions. During this process I met some problem, some are about report and some are about codes. Because its codes are ran on certain platform(visual studio 2008) and I changed it into Linux. Also its operations are all xor, and, and shift operations based bit using c, so it is also a challenging for me.

When finished the analysis of MD6. I go forward to the research of differential security of MD6. From my research review, I know there are two method in prior work. I reviewed my restach review and then start to research Ethan Heilman's method, because his method is to improve the original method based on the original method and it is useless to analyse the original method again. Ethan Heilman's paper is complicated and not very clear when he explain his method, but from his code I know his main idea about traversal all the valid differential paths and the challenging of his code is to find the one, which creates the least AAGs numbers, then it stop running and get into next round to calculate the AAG numbers of next round. It is hard to judge the appropriate differential path which appropriately creates the least AAG number.

The most important thing is the code has problem. It is my project bottleneck and milestone, I must solve this problem. First I discussed with my supervisor, but unfortunately we did not find our problem, which means we don't think our operation on his code is wrong. So we suppose his code has some bugs. I tried to contact with Ethan Heilman. But it needs time for him to check his code and his method. So I tried to solve it. I think first the logic of his method looks reasonable and the bug is possible on his code.I spent three days to check the bug. Because I am not sure his method is correct or not, so I combine the method and the code again and again. Finally I solve that bugs. And I sent those bugs to Ethan Heilman and he agreed with my solutions. It is really worthful for my project, because I planned to use this method and do a little change and then use it on my chosen smaller versions of MD6. Because I think research differential security

don't have many good method, it need to search all the differential paths. If every differential path can prevent differential attack, the MD6 would be differential security. So all of the method need analyse of all the differential paths. Maybe the difference of the methods is just how to search all of these valid differential paths. Ethan Heilman did some change on searching these all of valid differential paths. it divided the differential paths into two kinds, trial and non-trial differential paths. How to combine these theoretical with lower number. It introduced the AAG. One AAG will generate one 1/2, so there are $d/2$ AAGs, it will have $2^{-d/2}$ possibility corresponding to birthday attack.
Then I moved to my mainly contributions. I chose smaller versions of MD6, MD6-17-8, MD6-35-8 and MD6-35-16. This will save my time and also gave my time to analyse the results. Because Ethan Heilman told me he get his result took more than one month and he ran his code on multi-core machine (200 cores). Some result are strange with him.
I spent more than 1 week to get all the results what I wanted, there are also some interested result. I gave my own analysis. At last I provided how many rounds the three versions of MD6 need for preventing differential attacks and also provided the lower bound of these round numbers. Of course for different output size it need different round number even if in the same version of MD6. The d of possibility $2^{-d/2}$ is the output size of MD6 hash functions. However I did not provide the exact method to prove my result correctness. But I have a idea that is to try to find the all valid differential path with the first two fixed differential paths and compare with my results. It sounds need large time and large computer with large computational ability.

References

[1] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. *IACR Cryptology ePrint Archive*, page 146, 2004.

[2] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. In Ronald Cramer, editor, *In EUROCRYPT,LNCS 3491*, pages 36–57. Springer.

[3] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *CRYPTO'91*, 1991.

[4] Florent Chabaud and Antoine Joux. Differential collisions in sha-0. In *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '98, pages 56–71, London, UK, UK, 1998. Springer-Verlag.

[5] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer Verlag, Berlin, Heidelberg, New York, 2002.

[6] Owen Harrison and John Waldron. Practical symmetric key cryptography on modern graphics hardware. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 195–209, Berkeley, CA, USA, 2008. USENIX Association.

[7] Ethan Heilman. Restoring the Differential Resistance of MD6 . *IACR Cryptology ePrint Archive*, page 374, 2011.

[8] Jonathan J. Hoch and Adi Shamir. Breaking the ice - finding multicollisions in iterated concatenated and expanded (ice) hash functions. In *In Proceedings of FSE 06*, pages 179–194, 2006.

[9] Jonathan J. Hoch and Adi Shamir. Breaking the ice - finding multicollisions in iterated concatenated and expanded (ice) hash functions. In *In Proceedings of FSE 06*, pages 179–194, 2006.

[10] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[11] Charanjit S. Jutla and Anindya C. Patthak. A simple and provably good code for sha message expansion, 2005.

[12] P. Kitsos and O. Koufopavlou. Efficient architecture and hardware implementation of the whirlpool hash function. *IEEE Trans. on Consum. Electron.*, 50(1):208–213, February 2004.

[13] Moses Liskov. Constructing secure hash functions from weak compression functions: The case for non-streamable hash functions, 2006.

[14] Stefan Lucks. Design principles for iterated hash functions, 2004.

[15] Stefan Lucks. A failure-friendly design principle for hash functions. pages 474–494. Springer, 2005.

[16] George Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 08(i14), 2003.

[17] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 2001.

[18] M. Nandi and D. R. Stinson. Multicollision attacks on a class of hash functions, 2005.

[19] Mridul Nandi and Douglas R. Stinson. Multicollision attacks on some generalized sequential hash functions. *IEEE Trans. Inf. Theor.*, 53(2):759–767, February 2007.

[20] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Topics in Cryptology - CT-RSA 2006, The Cryptographers Track at the RSA Conference 2006*, pages 1–20. Springer-Verlag, 2005.

[21] François Panneton and Pierre L'ecuyer. On the xorshift random number generators. *ACM Trans. Model. Comput. Simul.*, 15(4):346–361, October 2005.

[22] Bart Preneel, Ren Govaerts, and Joos Vandewalle. Differential Cryptanalysis of Hash Functions Based on Block Ciphers. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 183–188. ACM, 1993.

[23] Ronald L. Rivest, Benjamin Agre, Daniel V. Bailey, Christopher Crutchfield, Yevgeniy Dodis, Kermin Elliott, Fleming Asif Khan, Jayant Krishnamurthy, Yuncheng Lin, Leo Reyzin, Emily Shen, Jim Sukha, Drew Sutherland, Eran Tromer, and Yiqun Lisa Yin. The MD6 hash function A proposal to NIST for SHA-3, 2008.

[24] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, and Maki Shigeri. Cryptanalysis of des implemented on computers with cache. In *Proc. of CHES 2003, Springer LNCS*, pages 62–76. Springer-Verlag, 2003.

[25] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions md4 and ripemd. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

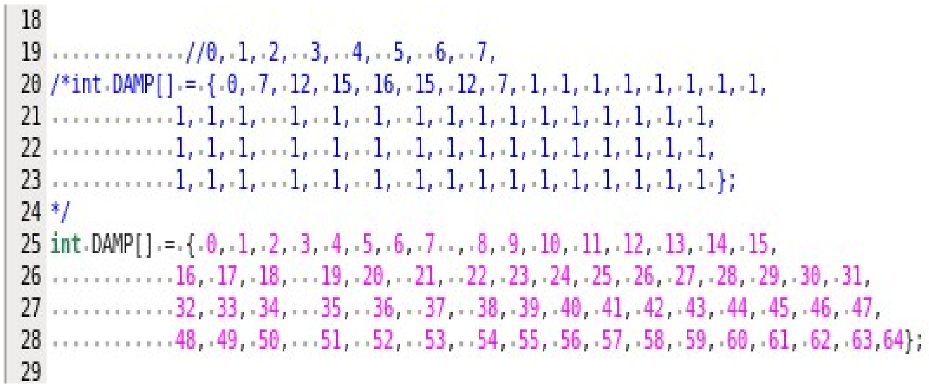
[26] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In *In Proceedings of Crypto*, pages 17–36. Springer-Verlag, 2005.

[27] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *In EUROCRYPT*. Springer-Verlag, 2005.

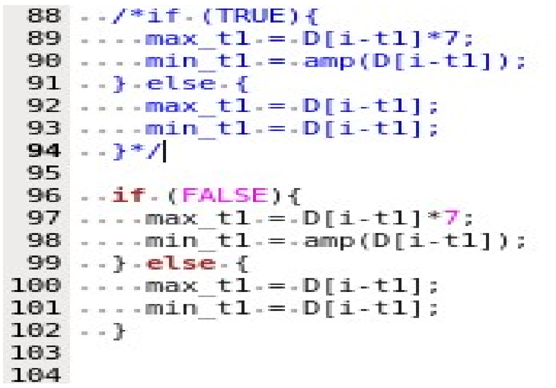
[28] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on sha-0. In *In Crypto*, pages 1–16. Springer-Verlag, 2005.

[29] Hongbo Yu and Xiaoyun Wang. Multi-collision attack on the compression functions of md4 and 3-pass haval. In *Proceedings of the 10th international conference on Information security and cryptology*, ICISC'07, pages 206–226, Berlin, Heidelberg, 2007. Springer-Verlag.

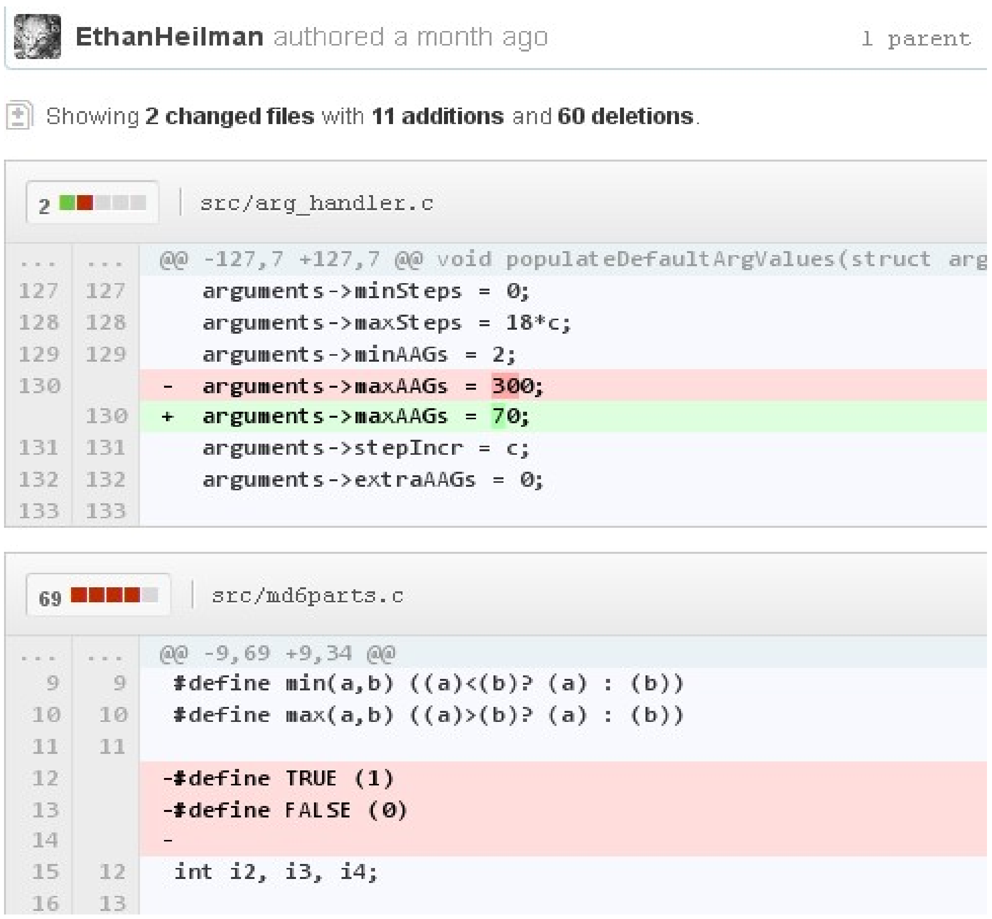
A Changes in Ethan Heilman's code



```

            //0, 1, 2,  3,  4,  5,  6,  7,
/*int DAMP[] = { 0, 7, 12, 15, 16, 15, 12, 7, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1,   1,  1,  1,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1,   1,  1,  1,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1,   1,  1,  1,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
*/
int DAMP[] = { 0, 1, 2, 3, 4, 5, 6, 7  , 8, 9, 10, 11, 12, 13, 14, 15,
            16, 17, 18,   19, 20,  21,  22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
            32, 33, 34,   35,  36,  37,  38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
            48, 49, 50,   51,  52,  53,  54, 55, 56, 57, 58, 59, 60, 61, 62, 63,64};

```



```
  /*if (TRUE){
    max_t1 = D[i-t1]*7;
    min_t1 = amp(D[i-t1]);
  } else {
    max_t1 = D[i-t1];
    min_t1 = D[i-t1];
  }*/

  if (FALSE){
    max_t1 = D[i-t1]*7;
    min_t1 = amp(D[i-t1]);
  } else {
    max_t1 = D[i-t1];
    min_t1 = D[i-t1];
  }


```



EthanHeilman authored a month ago   1 parent

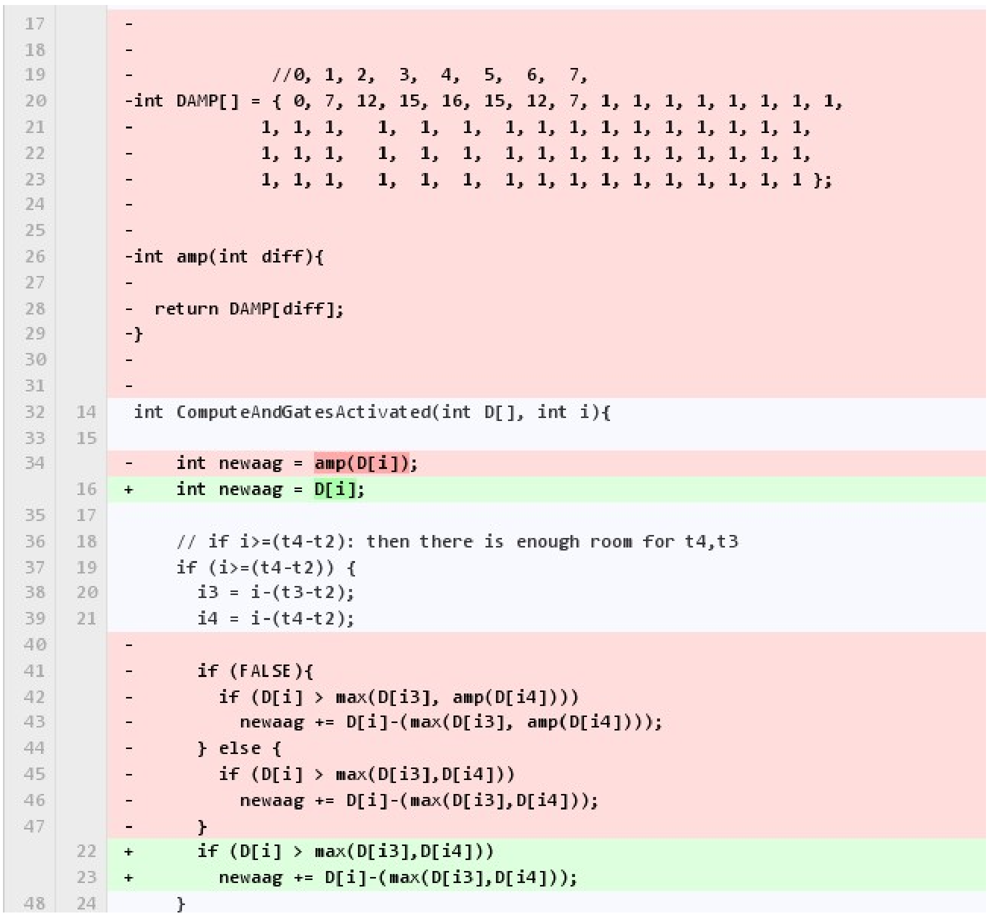
Showing **2 changed files** with **11 additions** and **60 deletions**.

2 | src/arg_handler.c

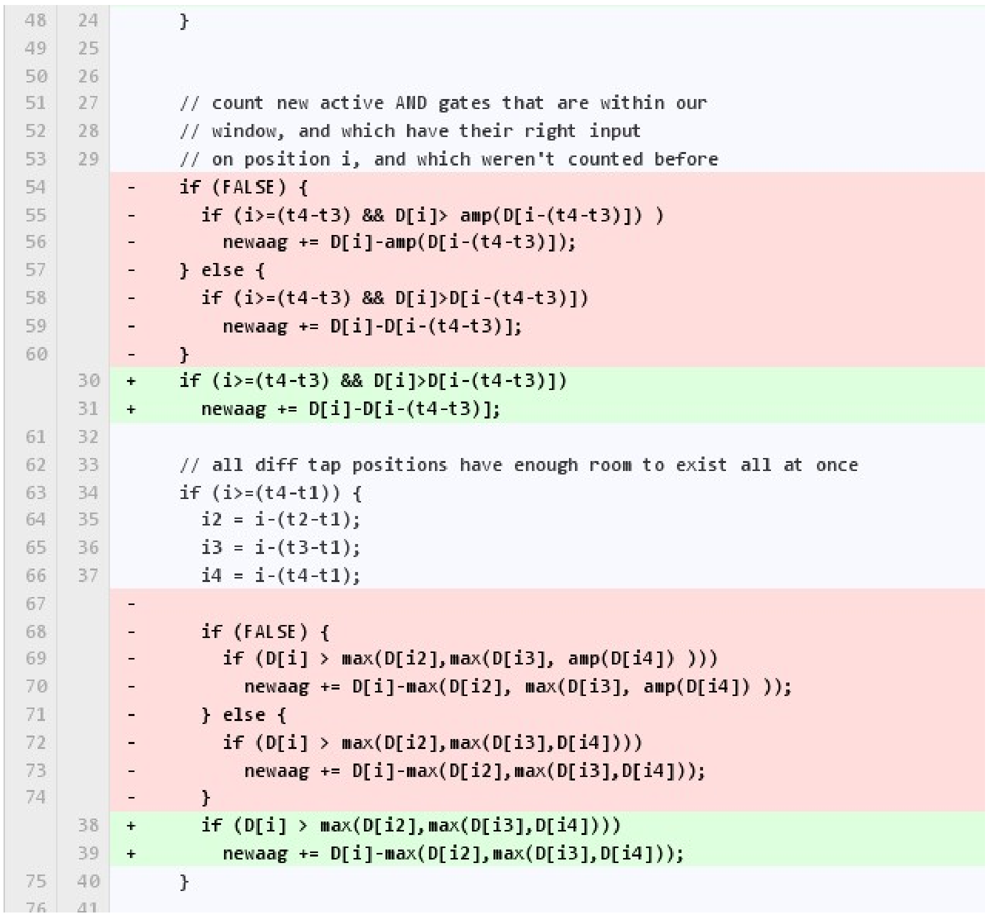
```
@@ -127,7 +127,7 @@ void populateDefaultArgValues(struct arg
     arguments->minSteps = 0;
     arguments->maxSteps = 18*c;
     arguments->minAAGs = 2;
-    arguments->maxAAGs = 300;
+    arguments->maxAAGs = 70;
     arguments->stepIncr = c;
     arguments->extraAAGs = 0;

```

69 | src/md6parts.c

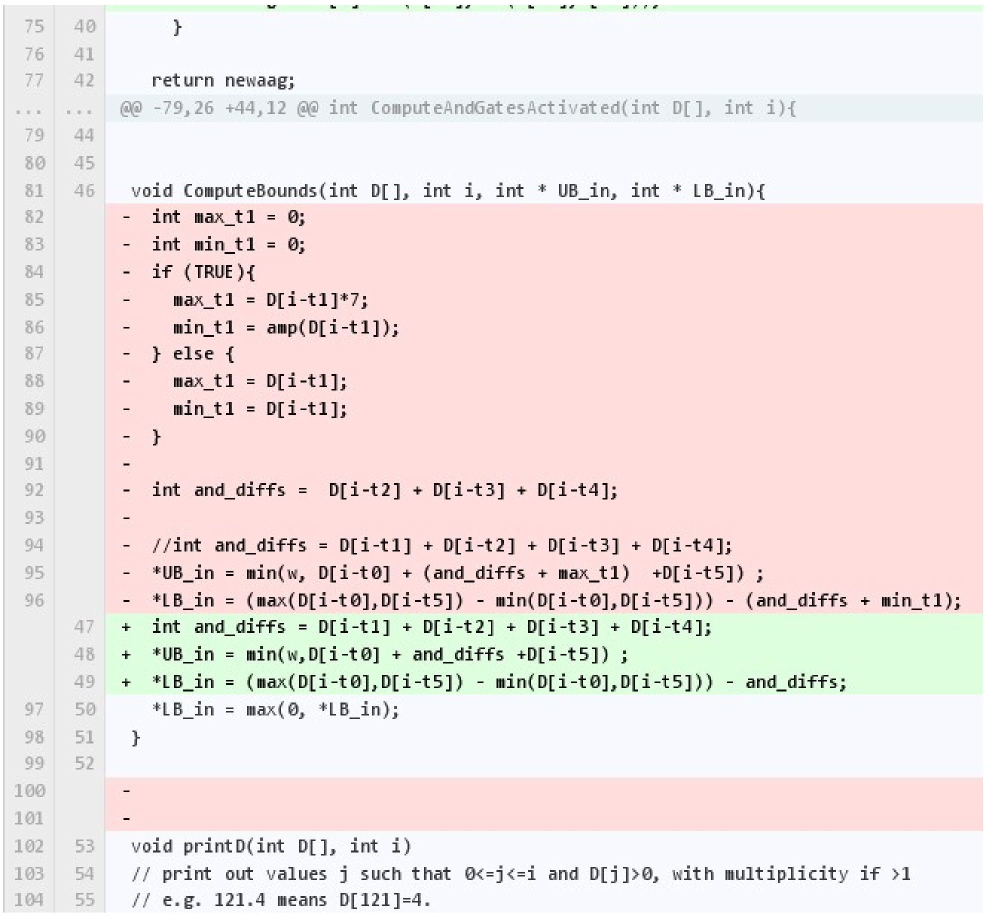
```
@@ -9,69 +9,34 @@
 #define min(a,b) ((a)<(b)? (a) : (b))
 #define max(a,b) ((a)>(b)? (a) : (b))

-#define TRUE (1)
-#define FALSE (0)
-
 int i2, i3, i4;

```



```
-
-
-             //0, 1, 2,  3,  4,  5,  6,  7,
-int DAMP[] = { 0, 7, 12, 15, 16, 15, 12, 7, 1, 1, 1, 1, 1, 1, 1, 1,
-            1, 1, 1,   1,  1,  1,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
-            1, 1, 1,   1,  1,  1,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
-            1, 1, 1,   1,  1,  1,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
-
-
-int amp(int diff){
-
-  return DAMP[diff];
-}
-
-
 int ComputeAndGatesActivated(int D[], int i){

-    int newaag = amp(D[i]);
+    int newaag = D[i];

     // if i>=(t4-t2): then there is enough room for t4,t3
     if (i>=(t4-t2)) {
       i3 = i-(t3-t2);
       i4 = i-(t4-t2);
-
-      if (FALSE){
-        if (D[i] > max(D[i3], amp(D[i4])))
-          newaag += D[i]-(max(D[i3], amp(D[i4])));
-      } else {
-        if (D[i] > max(D[i3],D[i4]))
-          newaag += D[i]-(max(D[i3],D[i4]));
-      }
+      if (D[i] > max(D[i3],D[i4]))
+        newaag += D[i]-(max(D[i3],D[i4]));
     }
```



```
      }


      // count new active AND gates that are within our
      // window, and which have their right input
      // on position i, and which weren't counted before
-     if (FALSE) {
-       if (i>=(t4-t3) && D[i]> amp(D[i-(t4-t3)]) )
-         newaag += D[i]-amp(D[i-(t4-t3)]);
-     } else {
-       if (i>=(t4-t3) && D[i]>D[i-(t4-t3)])
-         newaag += D[i]-D[i-(t4-t3)];
-     }
+     if (i>=(t4-t3) && D[i]>D[i-(t4-t3)])
+       newaag += D[i]-D[i-(t4-t3)];

      // all diff tap positions have enough room to exist all at once
      if (i>=(t4-t1)) {
        i2 = i-(t2-t1);
        i3 = i-(t3-t1);
        i4 = i-(t4-t1);
-
-       if (FALSE) {
-         if (D[i] > max(D[i2],max(D[i3], amp(D[i4]) )))
-           newaag += D[i]-max(D[i2], max(D[i3], amp(D[i4]) ));
-       } else {
-         if (D[i] > max(D[i2],max(D[i3],D[i4])))
-           newaag += D[i]-max(D[i2],max(D[i3],D[i4]));
-       }
+       if (D[i] > max(D[i2],max(D[i3],D[i4])))
+         newaag += D[i]-max(D[i2],max(D[i3],D[i4]));
      }
```



```
     }

   return newaag;
@@ -79,26 +44,12 @@ int ComputeAndGatesActivated(int D[], int i){


 void ComputeBounds(int D[], int i, int * UB_in, int * LB_in){
-  int max_t1 = 0;
-  int min_t1 = 0;
-  if (TRUE){
-    max_t1 = D[i-t1]*7;
-    min_t1 = amp(D[i-t1]);
-  } else {
-    max_t1 = D[i-t1];
-    min_t1 = D[i-t1];
-  }
-
-  int and_diffs =  D[i-t2] + D[i-t3] + D[i-t4];
-
-  //int and_diffs = D[i-t1] + D[i-t2] + D[i-t3] + D[i-t4];
-  *UB_in = min(w, D[i-t0] + (and_diffs + max_t1)  +D[i-t5]) ;
-  *LB_in = (max(D[i-t0],D[i-t5]) - min(D[i-t0],D[i-t5])) - (and_diffs + min_t1);
+  int and_diffs = D[i-t1] + D[i-t2] + D[i-t3] + D[i-t4];
+  *UB_in = min(w,D[i-t0] + and_diffs +D[i-t5]) ;
+  *LB_in = (max(D[i-t0],D[i-t5]) - min(D[i-t0],D[i-t5])) - and_diffs;
   *LB_in = max(0, *LB_in);
 }

-
-
 void printD(int D[], int i)
 // print out values j such that 0<=j<=i and D[j]>0, with multiplicity if >1
 // e.g. 121.4 means D[121]=4.
```