MD5 ALGORITHM
Understanding and Implementation

Presented and
Prepared by

Muneeb Ahmed
B.Tech CSE
13048100334

muneebpandith.github.io

At
University of Kashmir
Srinagar

My Declaration

I, Muneeb Ahmed Roll No. 13048100334,, hereby declare that the work, which is presented in this report entitled "**MD5 Algorithm: Understanding and Implementation**",is an authentic record of my own work and thereby I reserve the content-right of the report, presentation and the source code.

March 2017

Muneeb Ahmed

Roll No.
13048100334

muneebpandith.github.io

What is MD5 Algorithm ?

MD5 was designed by Ronald Rivest in 1991 to replace an earlier hash function Md4. The abbreviation "MD" stands for "Message Digest."

The MD5 algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities. It can still be used as a checksum to verify data integrity, but only against unintentional corruption.

Like most hash functions, MD5 is neither encryption nor encoding. It can be cracked by brute-force attack and suffers from extensive vulnerabilities as detailed in the security section below.

Application

1. MD5 digests have been widely used in the software world to provide some assurance that a transferred file has arrived intact. For example, file servers often provide a pre-computed MD5 (known as md5sum) checksum for the files, so that a user can compare the checksum of the downloaded file to it.

2.Most unix-based operating systems include MD5 sum utilities in their distribution packages; Windows users may use the included PowerShell function "Get-FileHash", install a Microsoft utility.

3. Android Applications/ROMs also use this type of checksum.

4.Historically, MD5 has been used to store a one-way hash of a password, often with key stretching. But due to security concerns this practice has been discarded now.

5. MD5 is also used in the field of electronic discovery, in order to provide a unique identifier for each document that is exchanged during the legal discovery process.

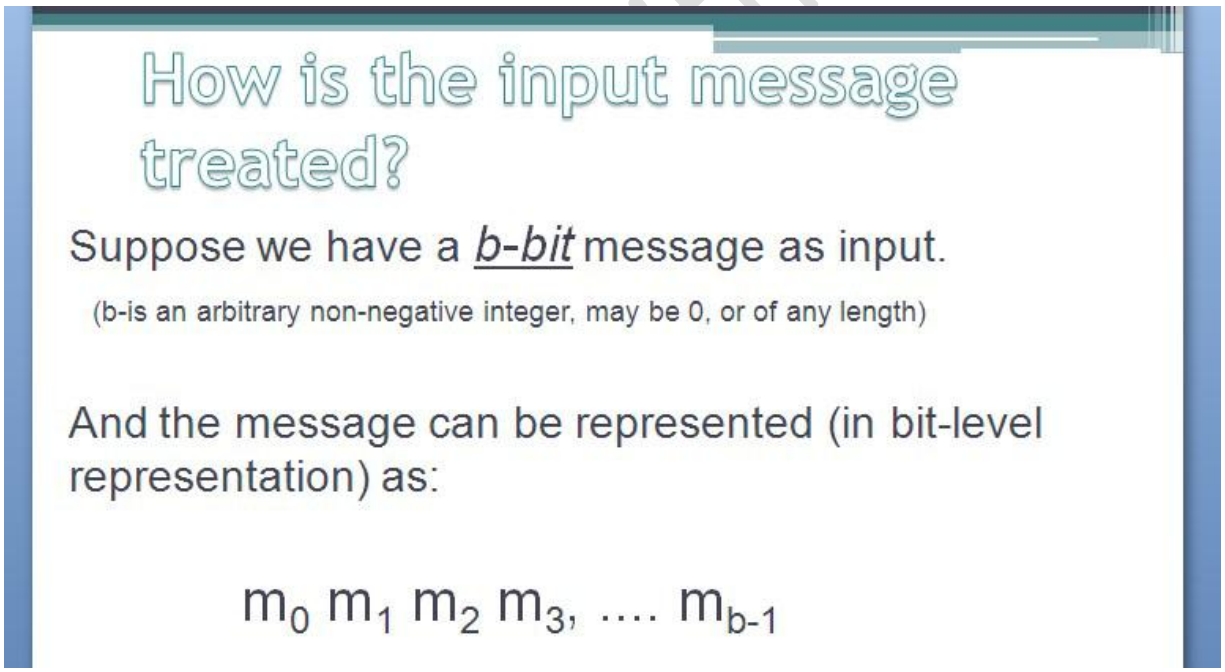
munnebpandith.github.io

Algorithm

1. TAKE THE INPUT
2. PADDING BITS
3. APPENDING LENGTH
4. INITIALIZING THE BUFFER
5. PROCESS MESSAGE AS 512 BITS

Steps 5 is repeated upto n-blocks

1. TAKING THE INPUT



MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit (64 byte) blocks

Then further grouped into (SIXTEEN-32-bit words in STEP 5.).

2. PADDING THE BITS

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512.

That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long.

Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Padding Procedure:

Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

```
void padding()
{
   int bytestobepadded=56-remainderbytes;        //we have to pad BYTESTOBEPADDED number of bytes (=448-remainderbits OR 56-remainder
   int offset=0;                                 //this contains the offset from TEXTLENGTH that is padded
    //Because we are dealing with bytes, we can't pad a binary bit by bit

    //Step 1: So, we can append a byte and not a bit! Therfore we append Byte as Hex(0x80) OR Bin(10000000)
    //Step 2: then pad 0s taking one byte at a time and not just a bit, because we dealing with bytes! as Hex(0x00) OR Bin(00000000)

    //So Step1: Pad 0x80
text[remainderbytes+offset]=0x80;     //this is the first byte we are padding as defined by Step 1

    //And then Step2: Pad 0x00 for rest of the bytes
 bytestobepadded=bytestobepadded-1;          //because 1st byte has been padded now bytestobepadded-1 more bytes to be padded
 offset++;                    //because 1st byte is padded now the next one

    while(bytestobepadded>0)     //loop until no bittobepadded is there
{

text[remainderbytes+offset]=0x00;        // next 8-bits are padded with 0x00
bytestobepadded--;                  // since 8-bits have been padded. Now we have BITSTOBEPADDED minus 8 no of bits to be padded mor
offset++;                           // as padding takes place 8-bits at a time, offset is incremented to next byte
}
text[remainderbytes+offset]='~';   //marks end of string


 }
```

Testing the output of PADDING function

```
Text: muneeb_farheen_usma~
Length in bytes:19
No Of Blocks (each of 512 bits in size) : 0
Remainder bytes : 19

AFTER PADDING
Text: muneeb_farheen_usmaÇ
Length in bytes:56
No Of Blocks (each of 512 bits in size) : 0
Remainder bytes : 56_
```

3. APENDING THE LENGTH

A 64-bit length wise representation of **b-bits** (*the length of the original message, before the padding bits were added*) is appended to the result of the previous step.

At this point the resulting message (after padding with bits and with the) has a length that is an exact multiple of 512 bits.

Now Mathematics,
We had a
Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

Code

```
112    void appendsize()
113    {
114   //int bytesleftforsize=8;          //the last 64-bits OR 8-bytes left after padding for appending size of original message
115   //we can have 64-bits representing the size of the original message
116
117   //sizecalculation of original message
118   //TEXTLENGTH has changed theoretically after padding but the variable has not been changed manually. so it will have the original v
119   //so we are actually converting TEXTLENGTH into hex/binary and then append it to the textmessage;
120   int n=textlength*8;    // l contain the no of bits of the original mesage for dec to bin calculation
121
122   int counterofbits=0;
123   unsigned int remainder;
124   int offset=0;
125
126   while(offset<8)
127       {
128    remainder=n % 2;
129    n=n/2;
130
131          text[56+offset]= (remainder << counterofbits ) | text[56+offset] ;
132    counterofbits++;            //must run from 0 to 7 in each iteration
133
134          if(counterofbits==8){ counterofbits=0;  offset++;}
135       }
136
137       //appending successful! :*
138     text[56+offset]='~';   //marks end of string
139
140
141    }
142
```

Output of the APPENDSIZEFUNCTION (after running the padding())

```
Text: muneeb_farheen_usma~
Length in bytes:19
No Of Blocks (each of 512 bits in size) : 0
Remainder bytes : 19

AFTER PADDING
Text: muneeb_farheen_usmaÇ
Length in bytes:56
No Of Blocks (each of 512 bits in size) : 0
Remainder bytes : 56

AFTER APPENDING THE Length
Text: muneeb_farheen_usmaÇ
Length in bytes:64
No Of Blocks (each of 512 bits in size) : 1
Remainder bytes : 0_
```

4. INITIALIZE BUFFERS

```
void Hash()
{
  //INITIALIZATION
  unsigned long int w[16];      //used for breaking a 5
  int offset,new_len,i;
  a = 0x67452301;
  b = 0xefcdab89;
  c = 0x98badcfe;
  d = 0x10325476;
```

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants.

In my implementation, HASH() Function does Steps 4 & 5

5. PROCESSING THE n-number of 512BIT BLOCKS

The main algorithm then uses each 512-bit message block in turn to modify the state.

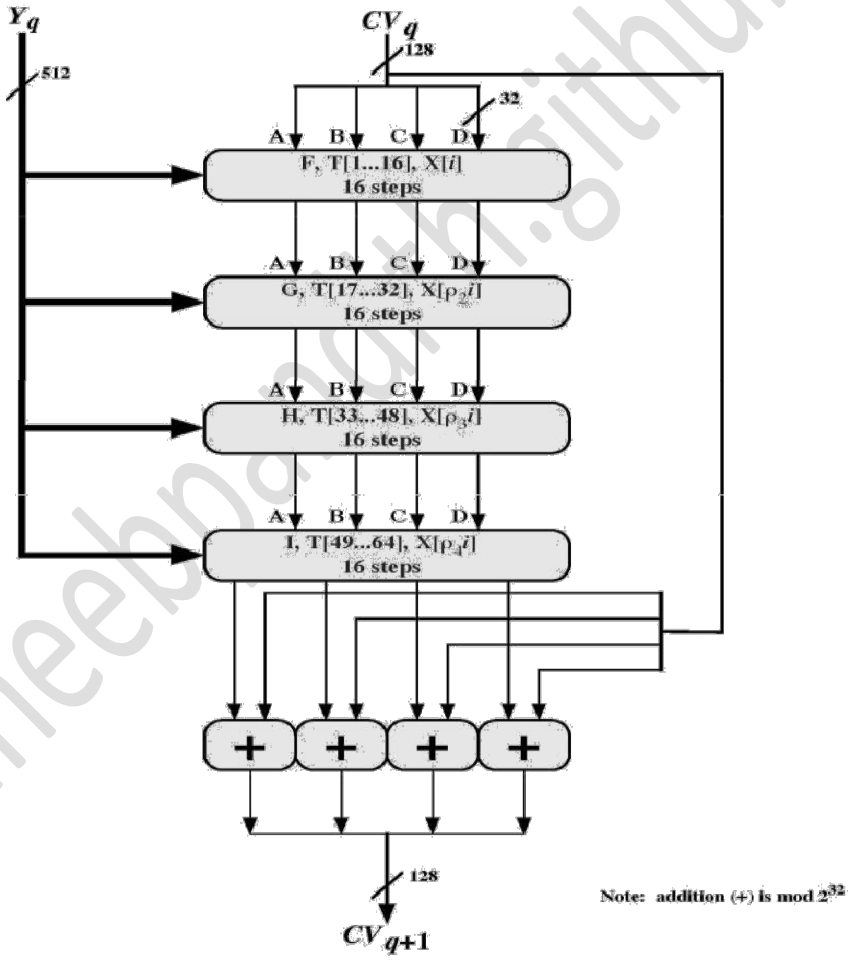
Using predefined functions namded as F,G,H & I. The processing of a message block consists of four similar stages, termed rounds, each round is composed of 16 similar operations based on a non-linear function F, modular addition, and left rotation.

muneebpandith.github.io

Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

There are 4 rounds for each 64-BYTE block. After dividing each block into 16- 4Byte words, Each round has 16 steps.



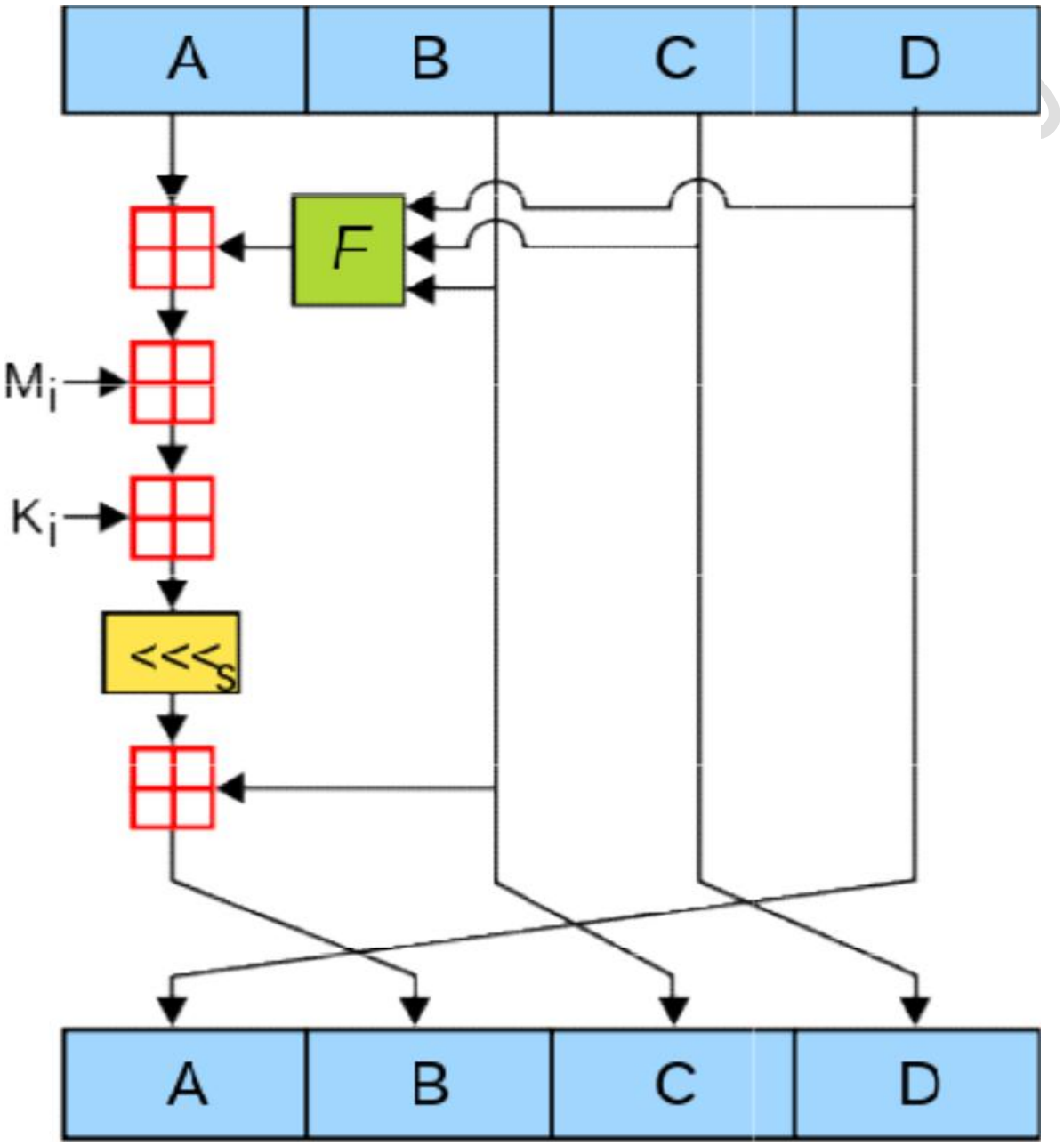
muneebalam.github.io

F(X,Y,Z) = XY v not(X) Z F OR FIRST round

G(X,Y,Z) = XZ v Y not(Z) F OR Second Round

H(X,Y,Z) = X xor Y xor Z For THIRD ROUND

I(X,Y,Z) = Y xor (X v not(Z)) FOR FOURTH ROUND



M[i] is the ith word input the Functions taken out of the divided 16 words from 64bytes block

This step uses a 64-element table K[1 ... 64] constructed from the sine function. Let K[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians.

MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. F is a nonlinear function; one function is used in each round. Mi denotes a 32-bit block of the message input, and Ki denotes a 32-bit constant, different for each operation. left shifts denotes a left bit rotation by s places; s varies for each operation. Addition denotes addition modulo 2_{32}.

muneebpandith.github.io

REFERENCES

1. RFC 1321
2. Some images have been taken from the RFC
3. Some images have been taken from public internet

muneebpandith.github.io