**CSEC-604**

**Cryptography and Authentication Assignment #4 (100 Points)**

# Introduction

Through this assignment, we will understand why and how MD5 is considered to be broken. It is widely acknowledged that the MD5 hash function is flawed. In 2005, researchers from Shandong University in China, Xiaoyun Wang, and Hongbo Yu, released a paper in which they described a method that could locate two distinct sequences of 128 bytes with the same MD5 hash.

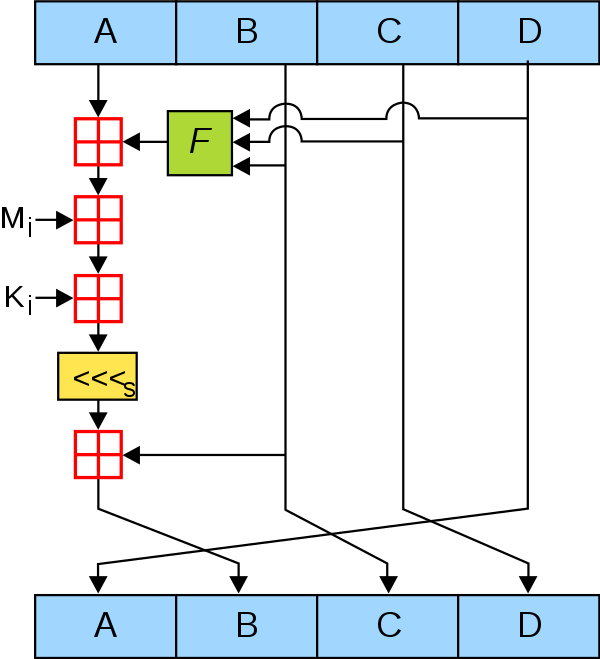
To understand why MD5 is considered to be broken, let us understand how the MD5 algorithm works.

**MD5 Algorithm** Input: Variable Length Output: 128 bits.

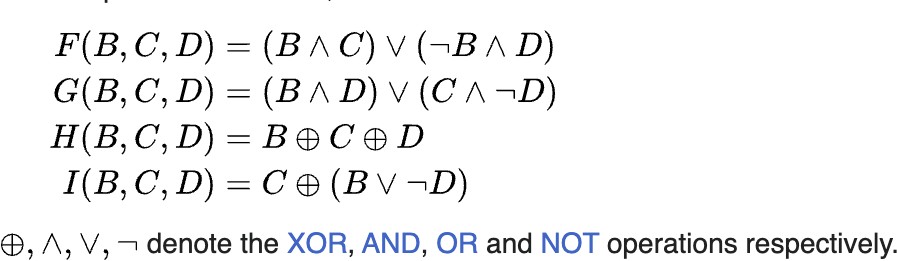
The input message is divided into 16 32-bit little-endian, 512-bit blocks, with padding added to make the message's length divisible by 512.

The padding works as follows: first, a single bit, 1, is appended to the end of the message. This is followed by as many zeroes as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with a 64-bit integer representing the length of the original message in bits.

The primary MD5 algorithm uses a 128-bit state that is split into 4 32-bit words called A, B, C, and D. They are set to a certain fixed constant at initialization. The main algorithm will then process each 512-bit message block in turn, changing the state with each block. Four comparable steps, known as rounds, make up the processing of a message block; each round is made up of 16 related operations based on the non-linear function F, modular addition, and left rotation.



# One MD5 Operation [Image Source: Wikipedia]

The following are the four possible functions that can be used in MD5 rounds.

# [Source: Wikipedia]

**Why is MD5 broken????**

It is considered to be broken because it is vulnerable to several types of attacks that can generate collisions. One of the most significant attacks on MD5 is the “collision attack”, which involves finding two different input messages that produce the same hash value. With this attack, an attacker could create a file with a malicious payload and a hash that matches the hash of a benign file. This would allow the attacker to substitute the malicious file for the benign file without detection, potentially leading to serious security breaches.

Finding collisions is difficult in real time and typically requires a significant amount of computational resources. However, it is important to note that the difficulty of finding a collision is not the only factor considered when evaluating the security of a cryptographic hash function. In the case of MD5, while it may be difficult to find a collision, there are known attacks that can exploit the weaknesses in the algorithm design.

Here’s an example of a pair that generates the same MD5 hash. (Source: Research Article published by Xiaoyun Wang and Hongbo Yu)

Each of these blocks has an MD5 hash **79054025255fb1a26e4bc422aef54eb4**



# 1. Find such other attacks on MD5 and explain how they work. (At least 2) (20 Points)

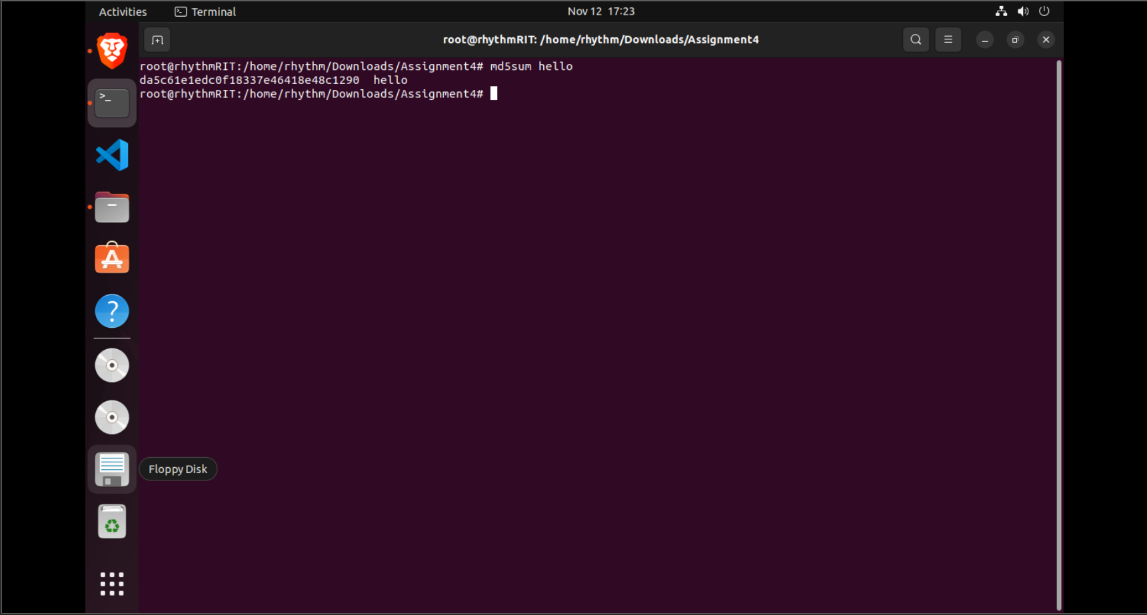
**Algorithm by Wang and Yu**

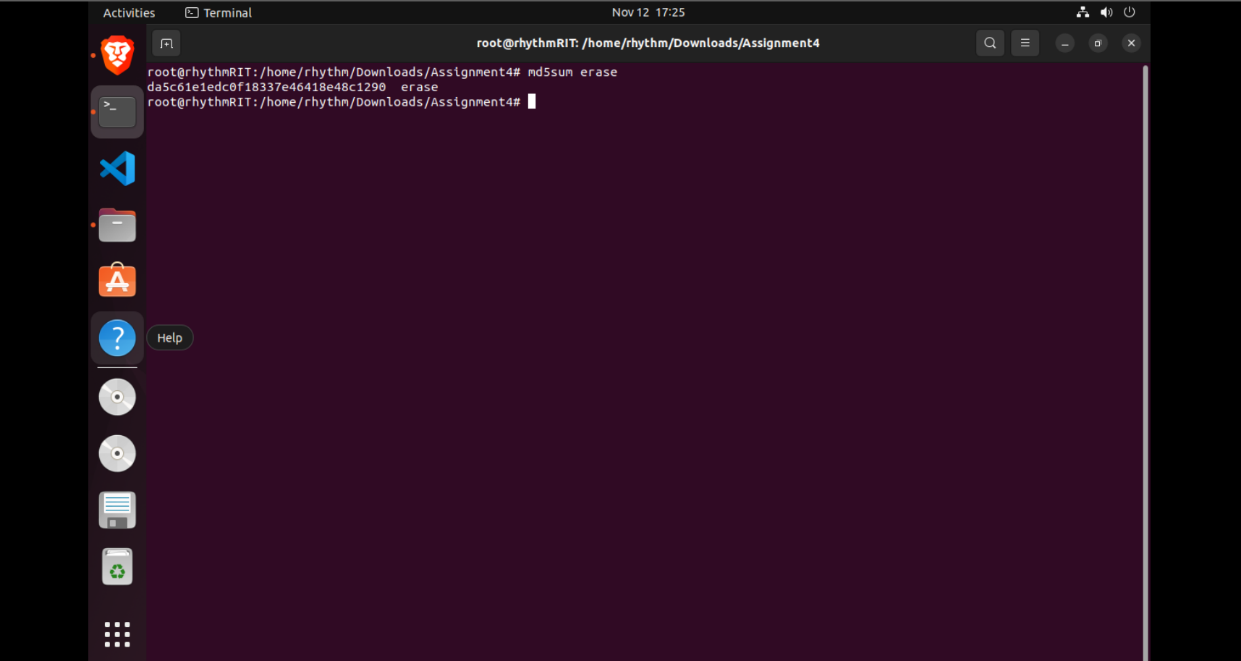
Before we understand the algorithm, let us do a fun exercise: (Adaptation of Peter Selinger’s Work)

**I have used Linux for this part of the assignment.**

For Linux, open the terminal and type the following commands take the screenshots

* 1. md5sum hello (for linux) **(5 Points)**
     1. What is the hash value of this file?

This is the hash value of the file “hello”: **da5c61e1edc0f18337e46418e48c1290**

* 1. Md5sum erase (for linux) **(5 Points)**
     1. What is the hash value of this file?

This is the hash value of the file “erase”: **da5c61e1edc0f18337e46418e48c1290**

1. If you want to see the contents of files you just found the hashes use the following commands **(10 Points)**

take the screenshots

* 1. For Linux:
* **chmod +x hello** followed by **./hello**
* **chmod +x erase** followed by **./erase**

1. What did you observe? **(10 Points)**

Now let us understand how the algorithm by Wang and Yu works. (Adaptation of Peter Selinger’s Work)

All of you might be wondering about the above results that you observed. The files that you downloaded were generated by exploiting two facts:

1. The block structure of MD5 functions
2. The technique proposed by Wang and Yu works for an arbitrary initialization vector.

We already saw how MD5 works. The process is done by an iteration method known as the Merkle Damgard method. The input file is first padded so that its length will be a multiple of 64 bits. It is then divided into individual 64-byte blocks m0, M1, ….. Mn-1. By computing a series of 16-byte states s0, s1,...... sn in accordance with the formula: si+1 - f(si, Mi), where f is a specific fixed function, one can calculate the MD5 hash.

Here:

S0 = Initialization Vector Sn = Computed MD5 Hash

The method that was proposed by Wang and Yu made it possible for S0 (Initialization Vector) to find two pairs of blocks, M, M’ and N, N’ such that f(f(s, M), M’) = f(f(s, N), N’). Because of this method now it is possible to find identical pairs except for the 128 bytes somewhere in the middle, which have identical MD5 hash.

# 1. Try and explain the method that was proposed by Wang and Yu in your own words

**(10 Points)**

# A fun Exercise (40 Points)

Now, let us try and create two files with identical MD5 hash.

In this exercise, we will be using the Evilize library on GitHub which is based on Patrick Stach’s implementation of Wang and Yu’s algorithm. (Adapted from the GitHub Repository) (Tool written by Patrick Stach)

1. Download the file from the following link
   1. [GitHub - mxrch/evilize: Use md5-collisions to make evil executables looking](https://github.com/mxrch/evilize) [like a good one.](https://github.com/mxrch/evilize)
2. For Windows:
   1. You will need to have GNU installed on this OS for this to work
      1. GNU Link: <https://sourceforge.net/projects/mingw/>
   2. Apart from this, you will also need to append .exe to all the executable files
3. Now the steps are same for all OS
   1. Unpack the archive
      1. tar zxf <evilize filename>
      2. cd <evilize filename>
      3. make
      4. These commands made a program under the name of evilize.
   2. Create a C program md5assignment.c with multiple behaviors. Instead of the usual top-level function main(), write two separate top-level functions main\_good() and main\_evil(). See the file hello-erase.c for a simple example. take the screenshot of the program you created
   3. Compiling the program and link against goodevil.o
      1. gcc md5assignment.c goodevil.o -o md5assignment
   4. Run the following command to create an initialization vector
      1. ./evilize md5assignment -i
   5. Create an MD5 collision by using the following command (replace the initialization vector with the one you formed in the previous command)
      1. ./md5coll <IV> > init.txt
      2. This step takes a lot of time. Take a coffee break.
   6. Create a pair of the good and evil programs
      1. ./evilize md5assignment -c init.txt -g file1 -e file2
   7. Check the MD5 hash of the files created.
   8. Run the files

Please attach screenshots of all the steps above to show your work. Try to include individual screenshots for each step and substep to show your work clearly.

# References

1. The above exercise is an adaptation of Peter Selinger’s Work. ([Peter Selinger: MD5](https://www.mathstat.dal.ca/~selinger/md5collision/) [Collision Demo](https://www.mathstat.dal.ca/~selinger/md5collision/))
2. It is based on the Algorithm written by Xiaoyun Wang, and Hongbo Yu ([How to Break](https://link.springer.com/chapter/10.1007/11426639_2) [MD5 and Other Hash Functions | SpringerLink](https://link.springer.com/chapter/10.1007/11426639_2))