

# MD5 Collision Attack Lab

A secure one-way hash function needs to satisfy two properties: the one-way property and the collision-resistance property. The one-way property ensures that given a hash value  $h$ , it is computationally infeasible to find an input  $M$ , such that  $\text{hash}(M) = h$ . The collision-resistance property ensures that it is computationally infeasible to find two different inputs  $M_1$  and  $M_2$ , such that  $\text{hash}(M_1) = \text{hash}(M_2)$ .

The learning objective of this lab is for students to really understand the impact of collision attacks, and see in first-hand what damages can be caused if a widely-used one-way hash function's collision-resistance property is broken.

To achieve this goal, students need to launch actual collision attacks against the MD5 hash function. Using the attacks, students should be able to create two different programs that share the same MD5 hash but have completely different behaviours. This lab covers a number of topics described in the following:

- One-way hash function, MD5
- The collision-resistance property
- Collision attacks

## Lab Tasks

### Task 1: Generating Two Different Files with the Same MD5 Hash

In this task, we will generate two different files with the same MD5 hash values. The beginning parts of these two files need to be the same, i.e., they share the same prefix. We can achieve this using the `md5collgen` program, which allows us to provide a prefix file with any arbitrary content. The way how the program works is illustrated in Figure 1. The following command generates two output files, `out1.bin` and `out2.bin`, for a given a prefix file `prefix.txt`:

```
$ python3 -c "print('\x' * 10, end='')" > prefix.txt
$ md5collgen -p prefix.txt -o out1.bin out2.bin
```



Figure 1: MD5 collision generation from a prefix

We can check whether the output files are distinct or not using the `diff` command. We can also use the `md5sum` command to check the MD5 hash of each output file. See the following commands.

```
$ diff out1.bin out2.bin
$ md5sum out1.bin
$ md5sum out2.bin
```

Since `out1.bin` and `out2.bin` are binary, we cannot view them using a text-viewer program, such as `cat` or `more`; we need to use a binary editor to view (and edit) them. A hex editor software called `bliss` is already installed in the VM. **Please use such an editor to view these two output files, and describe your observations.**

**In addition, you should answer the following questions. Attach screenshots to support your answer.**

Question 1. What is the length of your prefix file? If the length of your prefix file is not multiple of 64, what is going to happen?

Question 2. Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.

Question 3. Are the data (128 bytes) generated by `md5collgen` completely different for the two output files? Please identify all the bytes that are different.

## Task 2: Understanding MD5's Property

In this task, we will try to understand some of the properties of the MD5 algorithm. These properties are important for us to conduct further tasks in this lab. MD5 is a quite complicated algorithm, but from very high level, it is not so complicated. As Figure 2 shows, MD5 divides the input data into blocks of 64 bytes, and then computes the hash iteratively on these blocks. The core of the MD5 algorithm is a compression function, which takes two inputs, a 64-byte data block and the outcome of the previous iteration. The compression function produces a 128-bit IHV, which stands for "Intermediate Hash Value"; this output is then fed into the next iteration. If the current iteration is the last one, the IHV will be the final hash value. The IHV input for the first iteration (IHV<sub>0</sub>) is a fixed value.

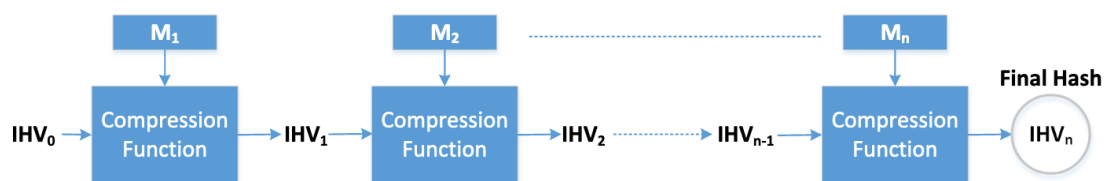


Figure 2: How the MD5 algorithm works

Your job in this task is to perform an experiment to demonstrate that this property holds for MD5.

### Step 1: Generate prefix and suffix that is multiple of 64

```
echo "$(python3 -c 'print("A"*63)') " > prefix
echo "$(python3 -c 'print("B"*63)') " > suffix
```

### Step 2: Create collision

```
md5collgen -p prefix -o out1.bin out2.bin
```

### Step 3: Append suffix

```
cat out1.bin suffix > file1
cat out2.bin suffix > file2
```

### Step 4: Check if MD5 hashes are the same?

```
md5sum file1 ; md5sum file2
```

**Give your observations.**

### Task 3: Generating Two Executable Files with the Same MD5 Hash

In this task, you are given the following C program. Your job is to create two different versions of this program, such that the contents of their xyz arrays are different, but the hash values of the executables are the same.

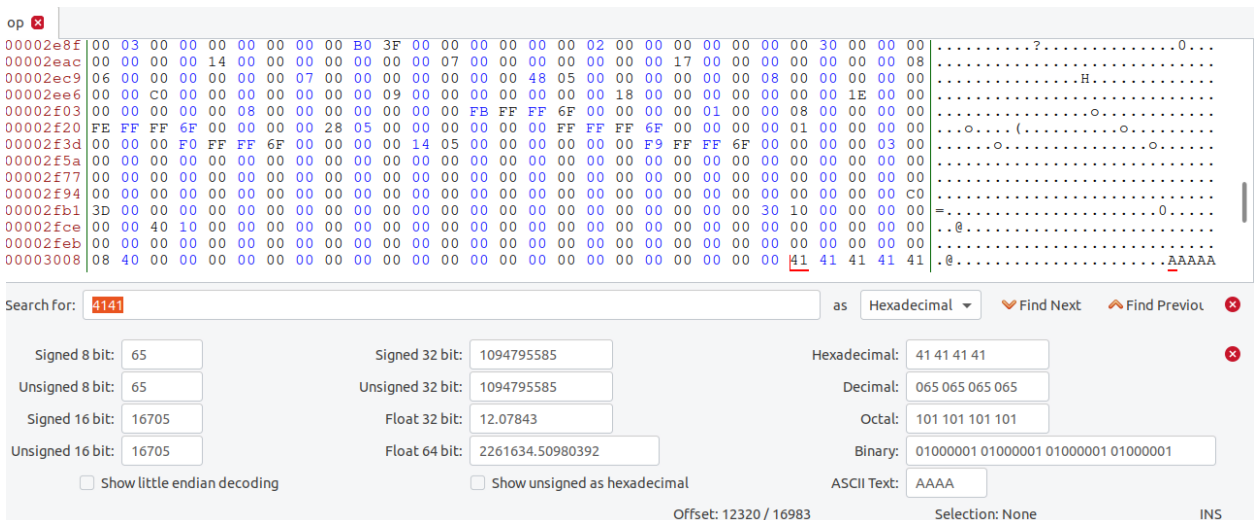
```
#include <stdio.h>
unsigned char xyz[200] = {
/* The actual contents of this array are up to you */
};
int main()
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

You may choose to work at the source code level, i.e., generating two versions of the above C program, such that after compilation, their corresponding executable files have the same MD5 hash value. However, it may be easier to directly work on the binary level. You can put some arbitrary values in the xyz array, compile the above code to binary. Then you can use a hex editor tool to modify the content of the xyz array directly in the binary file.

#### Steps:

1. Let us initialise the array with 200 A's so that it is easy to locate 200 A's in the binary.
2. Compile the program and using Bless find the location of continuous 0x41 in the executable output.
3. Now we have to divide the executable file into 3 parts – a prefix, 128 byte region and a suffix.

Note: Prefix has to be a multiple of 64 bytes

4. 

In the above example, you can see that the array starts from the offset 12320. But it is not a multiple of 64.

Hence I start my prefix from offset 12352. The next will be a 128 byte region. Hence my suffix will be the last 12480 ( 12352 + 128) bytes.

```
[11/07/24]seed@VM:~/Ankitha$ head -c 12352 op >prefix
[11/07/24]seed@VM:~/Ankitha$ tail -c +12480 op >suffix
[11/07/24]seed@VM:~/Ankitha$ md5collgen -p prefix -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix'
Using initial value: 516dbbbb1d06b5ab08e7504d1b0c8b5f

Generating first block: .
Generating second block: S01.....
Running time: 1.64533 s
[11/07/24]seed@VM:~/Ankitha$ cat out1.bin suffix > file1
[11/07/24]seed@VM:~/Ankitha$ cat out2.bin suffix > file2
```

5. Generate 2 files with prefix: md5collgen -p prefix -o P Q
6. Append suffix to make them like normal programs.
7. Compare the two programs using diff command
8. Make them executable: chmod x file1 file2
9. Now run the executables

Give your observations.

## Submission:

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising.