1. **[40 points + 10 Extra Credit] Short Lab Tasks**
   1. **Intro to MD5 Collisions**

MD5 was once the most widely used cryptographic hash function, but today it is considered dangerously insecure. This is because cryptanalysts have discovered efficient algorithms for finding collisions---pairs of messages with the same MD5 hash value.

The first known collisions were announced on August 17, 2004 by Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Here's one pair of colliding messages they published:

Message 1:

d131dd02c5e6eec4693d9a0698aff95c 2fcab58712467eab4004583eb8fb7f89

55ad340609f4b30283e488832571415a 085125e8f7cdc99fd91dbdf280373c5b

d8823e3156348f5bae6dacd436c919c6 dd53e2b487da03fd02396306d248cda0

e99f33420f577ee8ce54b67080a80d1e c69821bcb6a8839396f9652b6ff72a70

Message 2:

d131dd02c5e6eec4693d9a0698aff95c 2fcab50712467eab4004583eb8fb7f89

55ad340609f4b30283e4888325f1415a 085125e8f7cdc99fd91dbd7280373c5b

d8823e3156348f5bae6dacd436c919c6 dd53e23487da03fd02396306d248cda0

e99f33420f577ee8ce54b67080280d1e c69821bcb6a8839396f965ab6ff72a70

Copy the above hex strings into file1.hex and file2.hex. Convert each group of hex strings into a binary file. (On Linux, run xxd -r -p file.hex > file.)

1. What are the MD5 hashes of the two binary files? Verify that they're the same. (openssl dgst -md5 file1 file2)
2. What are their SHA-256 hashes? Verify that they're different. (openssl dgst -sha256 file1 file2)

You don't need to submit anything for the above questions.

* 1. **(15 points) MD5 Collisions using Fastcoll program**

In 2004, Wang's method took more than 5 hours to find a collision on a desktop PC. Since then, researchers have introduced vastly more efficient collision finding algorithms. You can compute your own MD5 collisions using a tool written by Marc Stevens that uses a more advanced technique.

You can download the fastcoll tool here:

<http://www.win.tue.nl/hashclash/fastcoll_v1.0.0.5.exe.zip> (Windows executable)

or

<http://www.win.tue.nl/hashclash/fastcoll_v1.0.0.5-1_source.zip> (source code)

If you use Linux or the VM we provided in class, you will also need to have installed the Boost libraries. On Ubuntu, you can install using apt-get install libboost-all-dev. On OS X, you can install Boost via the [Homebrew package manager](http://brew.sh) using brew install boost.

If you are compiling fastcoll from source, you can compile using this

g++ -O3 \*.cpp -lboost\_filesystem -lboost\_program\_options -lboost\_system -o fastcoll

**Generate Collision STEPS**

1. Create two files called file1 and file2 (you can use two of your own messages, for instance:

$echo “This is the first file” >file1

$echo “This is the second file” >file2

The two commands above would create files named file1 and file2 with the message “This is the first file” and “This is the second file” respectively.

1. Generate your own collision with this tool.   
   How long did it take? (time ./fastcoll -o file1 file2)
2. Show the Hex content of your files. To get a hex dump, run xxd -p file.
3. What are the MD5 hashes for file1 and file2?

You can use: openssl dgst -md5 file1 file2  
Verify that they're the same.

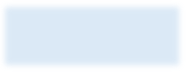
1. What are their SHA-256 hashes?

You can use: openssl dgst -sha256 file1 file2  
Verify that they're different.

* 1. **(20 points) MD5 collisions using the same file prefix**

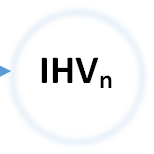
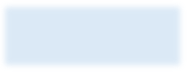
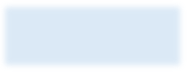
**IHV0 IHV1**

**IHVn**



Compression Function

**M1**



**Final Hash**

**IHV2**

**IHVn‐1**

Compression Function

Compression Function

**Mn**

**M2**

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 (IHV0) is a fixed value. The collision attack lets us generate two messages with the same MD5 hash and any chosen (identical) prefix. Due to MD5's length-extension behavior, we can append any suffix to both messages and know that the longer messages will also collide.

This lets us construct files that differ only in a binary "blob" in the middle and have the same MD5 hash, i.e. prefix || blobA || suffix and prefix || blobB || suffix.

We can leverage this to create two programs (shell scripts) that have identical MD5 hashes but wildly different behaviors. We're using shell scripts, but this could be done using a program in almost any language.

Put the following **three** lines (including an empty line at the end) into a file called prefix:

#!/bin/bash

cat << "EOF" | openssl dgst -sha256 > DIGEST

**<(intentionally left empty)**

and put these four lines (starting with a blank line) into a file called suffix:

EOF

digest=$(cat DIGEST | cut -c10- )

echo "The sha256 digest is $digest"

Now use fastcoll to generate two files with the same MD5 hash that both begin with prefix:.

fastcoll -p prefix -o col1 col2

Then append the suffix to both:

cat col1 suffix > file1.sh ; cat col2 suffix > file2.sh

The “;” operator is a command separator allowing us to run consecutive commands one after the other

Verify that file1.sh and file2.sh have the same MD5 hash but generate different output.

Extend this technique to produce another pair of programs, good and evil, that also share the same MD5 hash. One program should execute and print a benign payload: "I mean no harm." The second should execute and print a pretend malicious payload: echo or print "Your computer is hacked!"

**What to submit** Two scripts, good and evil, and your steps showing that have the same MD5 hash, have different SHA-256 hashes, and print the specified messages.

* 1. **[Extra Credit] (10 points) MD5 collisions in PDF**

The goal of this task is to understand that we can perform MD5 collisions on almost any file type we select (with some restrictions). The github site below has some interesting examples that show how you can manipulate images, documents, etc.:

<https://github.com/corkami/collisions>

<https://github.com/cr-marcstevens/hashclash>

Your task is to pick one class of files (e.g., PDF, JPG) and show how to create an MD5 collision. You can select any two files if they are not identical to ones presented in the site above. Note that it is important to demonstrate your work of generating those files through scripts or screenshots so downloading two files that have already been modified to produce an MD5 collision is not a valid answer.

**What to submit** Two files of your choice, file1 and file2, and your steps of you produce them to have identical MD5s but different SHA-256 hash values. Do not just submit the files but the also the detailed steps of how you created them.