# Lab03
# Hashes: Experimenting with hash collision attacks

(3.1 and 3.2 is intended to introduce you to MD5 collisions; you will not submit anything for it.)

## 3.1 MD5 Collisions (15 points)

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
```

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)
```

## 3.2 Generating Collisions Yourself

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(sourcecode)

If you are building fastcoll from source, you can compile using this makefile:
You will also need the Boost libraries. On Ubuntu, you can install these using
`apt-get install libboost-all-dev.`

1. Generate your own collision with this tool. How long did it take?
   ($ time ./fastcoll -o file1 file2)

2. What are your files? To get a hex dump, run $ xxd -p file.
3. What are their MD5 hashes? Verify that they're the same.
4. What are their SHA-256 hashes? Verify that they're different.

## 3.3 A Hash Collision Attack

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 that have identical MD5 hashes but wildly different behaviors. We'll use Python, but almost any language would do. Copy and paste the following three lines into a file called prefix: (Note: writing below lines yourself may lead to encoding mismatch and the error may occur while running the resulting python code)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
blob = """
```

and put these three lines into a file called suffix:

```
"""
from hashlib import sha256
print sha256(blob).hexdigest()
```

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.py; cat col2 suffix > file2.py).`

Verify that file1.py and file2.py 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 a benign payload: print "I come in peace."
The second should execute a pretend malicious payload: print "Prepare to be destroyed!".
Note that we may rename these program before grading them.

Submit Two Python 2.x scripts named good.py and evil.py that have the same MD5 hash, have different SHA-256 hashes, and print the specified messages.

## 3.4 Guess the number (20 points)

The goal of this task is to guess a number that we will announce in class on next week. You MUST complete this section and submit hash.hex before the announcement.

Here are the rules:
1. At the start of class next week I will announce the winning number, which will be an integer in the range [0;63].
2. Prior to the announcement, each student may email one guess. To keep everything fair, your guesses will be kept secret using the following procedure:
    (a) You'll create a PostScript file that, when printed, produces a single page that contains only the statement: "your name guesses n ".
    (b) You'll register your guess by emailing MD5 hash of your file to hash.hex.
    We must receive your hash value before the  announcement.
    (c) You and your project partner may collaborate and produce a file that includes both of your name, or you may choose to work independently.
3. If your guess is correct, you can claim the prize by emailing your PostScript file.ps. I will verify its MD5 hash, print it to a PostScript printer, and check that the statement is correct.