

# Crypto Lab – Exploring Collision-Resistance, Pre-Image Resistance and MACs

Copyright © 2006 - 2014 Wenliang Du, Syracuse University.

The development of this document is/was funded by three grants from the US National Science Foundation: Awards No. 0231122 and 0618680 from TUES/CCLI and Award No. 1017771 from Trustworthy Computing. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation. A copy of the license can be found at <http://www.gnu.org/licenses/fdl.html>.

## 1 Overview

The learning objective of this lab is for students to get familiar with pre-image resistant hash functions and Message Authentication Code (MAC). After finishing the lab, in addition to gaining a deeper understanding of the concepts, students should be able to use tools and write programs to generate hash values and a MAC for a given message.

## 2 Lab Environment

**Installing OpenSSL.** In this lab, we will use `openssl` commands and libraries. We have already installed `openssl` binaries in our VM. It should be noted that if you want to use `openssl` libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. We have already downloaded the necessary files under the directory `/home/seed/openssl-1.0.1`. To configure and install `openssl` libraries, go to the `openssl-1.0.1` folder and run the following commands.

```
You should read the INSTALL file first:
```

```
% sudo ./config
% sudo make
% sudo make test
% sudo make install
```

**Installing a hex editor.** In this lab, we need to be able to view and modify files of binary format. We have installed in our VM a hex editor called `GHex`. It allows the user to load data from any file, view and edit it in either hex or ascii. Note: many people told us that another hex editor, called `Bless`, is better; this tool might not be installed in the VM version that you are using, but you can install it yourself using the following command:

```
% sudo apt-get install bless
```

## 3 Lab Tasks

### 3.1 Task 1: Generating Message Digest and MAC

In this task, we will play with various hash algorithms. You can use the following `openssl dgst` command to generate the hash value for a file. To see the manuals, you can type `man openssl` and `man`

dgst.

```
% openssl dgst dgsttype filename
```

Replace the `dgsttype` with a specific hash algorithm, such as `-md5`, `-sha1`, `-sha256`, etc. In this task, you are encouraged to try at least 3 different algorithms. You can find the supported hash algorithms by typing `"man openssl"`.

### 3.2 Task 2: The Randomness of a Hash

To understand the properties of hash functions, we would like to do the following exercise for MD5 and SHA256:

1. Create a text file of any length.
2. Generate the hash value  $H_1$  for this file using a specific hash algorithm.
3. Flip one bit of the input file. You can achieve this modification using `ghex` or `Bless`.
4. Generate the hash value  $H_2$  for the modified file.
5. Observe whether  $H_1$  and  $H_2$  are similar or not. Write a short program to count how many bits are different between  $H_1$  and  $H_2$ .

Calculating the Hamming Distance is a mathematical way to determine the differences between two objects (or strings, boxes, files, hashes, etc.). For the first program you are going to submit on Sakai, you are going to write a program which calculates the Hamming distance between two different hashes for their *bits*. For example, take the hash "ab123" and ab122". These two hexadecimal values result in:

```
10101011000100100011
10101011000100100010
```

in binary respectively. The Hamming Distance between both of them is 1, because only one bit is different between the two hex values. For this program, input two hexadecimal values via command line arguments, and name your program `a2_YourOynen_hamming.py`. Important! Make sure that you can run your program in terminal like:

```
python3 a2_YourOynen_hamming.py ab123 ab122
```

The output for this case will be:

```
1
```

Another sample command and subsequent output:

```
python3 a2_YourOynen_hamming.py ff3a7 ff4b1
6
```

Next, create two files of at least 10 readable **ascii** characters (not hex values) that differ by **only 1 bit** (not just a character!). Hash both of these strings using a cryptographic hash and then observe the hamming distance between both hashes of your two strings.

For this task you will submit the following files to Sakai.

1. a2\_YourOnyen\_hamming.py
2. A file named **string1.txt** with a single string on a single line
3. A file named **string2.txt** with a single string on a single line with a one bit difference from string1.txt

### 3.3 Task 4a: Exploring Pre-image Resistance

In this task, we will investigate the difference between hash function's two properties: the pre-image resistance property versus collision-resistant property. We will use the brute-force method to see how long it takes to break each of these properties. Instead of using `openssl`'s command-line tools, you are required to write your own Python3 program to invoke the message hexdigest functions in Python's built in `hashlib`'s crypto library. A sample code can be found from <https://docs.python.org/3/library/hashlib.html>. Please get familiar with this sample code.

Since most of the hash functions are quite strong against the brute-force attack on those two properties, it will take us years to break them using the brute-force method. To make the task feasible, we reduce the length of the hash value to 24 bits. We can use any pre-image resistant hash function, but we only use the first 24 bits of the hash value in this task. Namely, we are using a modified pre-image resistant hash function.

Now, let us say that we have the SHA-256 sum:

```
d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592
```

(Whereas this SHA-256 sum happens to be of the string "The quick brown fox jumps over the lazy dog")

To test the pre-image resistance property, your goal is to find a string which collides with the first 24 bits of a given SHA-256 sum. For example, a 24 bit collision for the above hash would be `0xd7a8fb`. Write a Python3 program which will take as input a SHA-256 sum, and produce an output a string whose hash sum collides (for the first 24 bits) with the given hash sum. Use the `hashlib` library to generate the hashes. As soon as a colliding pre-image (the string) is found, your program should terminate. Name your file `a2_YourOnyen_preimage.py`. Your program should take the hash directly as a command line argument.

For example our input may look like:

```
d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592
```

and you would run your program like:

```
python3 a2_YourOnyen_preimage.py d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592
```

For the above input, my output is the following. Note that the SHA-256 sum of the below string collides in the first 24 bits.

```
19503334fhjasbfjhebw
```

For fun:

When you have completed the assignment, try to find a collision up to 28 bits or 32 bits. How much longer did it take?

### 3.4 Task 4b: The Collision Resistant Property

Next you will be testing the collision-resistant property. Now, write a python script which generates 1 line of output as before, except this time you choose both m1 and m2 (and keep searching until you find a collision between any of the pairs) Output a line with two strings separated by a single space. These two strings will have their first 24 bits collide with each other on a SHA-256 sum. For example, my output is:

```
12856fhjasbfjhebw 739fhjasbfjhebw
```

Your program must **not** print any hard coded values and should take less than 10 seconds to run. Note that the first 24 bits of each of these strings above collide. Your program should not take any input as a command line argument and will be run like:

```
python3 a2_onyen_collision.py
```

Answer the following questions:

1. What is the average number of trials it took you to break pre-image resistance based on 15 experiments?
2. What is the average number of trials it took you to break collision-resistance based on 15 experiments? What is the collision that you found, and what is their hash?
3. Which one is easier to break using brute force?

## 4 Submission

Submit your code on Sakai. You will submit the following files:

1. a2\_YourOnyen\_hamming.py
2. string1.txt
3. string2.txt
4. a2\_YourOnyen\_preimage.py
5. a2\_YourOnyen\_collision.py
6. YourOnyen\_report.txt (The answers to the above questions)