Alec Howard

November 8, 2022

CYB-3361-01

Lab 2: One-Way Hash Function & MAC Lab

1. Hash Functions for File Containing the String: “The bees are behind your eyes.”
    a. Md5: openssl dgst -md5 myfile.txt >> md5hash
        i. Output: MD5(myfile.txt)= 541e15c0a35a191f5dccc3f25b9d4a09
    b. Sha1: openssl dgst -sha1 myfile.txt >> sha1hash
        i. Output: SHA1(myfile.txt)= bd19c222260bcf3cd42d6624bc845c7a05c25721
    c. Sha256: openssl dgst -sha256 myfile.txt >> sha256hash
        i. Output: SHA256(myfile.txt)= 875dcde8acb951e08b0f9166b71662a6c99e56f14a48430a593fa4effd30e152

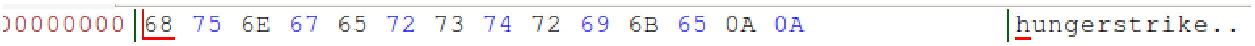
    I observed that the more complicated the hashing algorithm, the longer the hashed string became.

2. Keyed Hashes for File Containing the String: “We are in your walls.”
    a. HMAC-Md5:
        i. openssl dgst -md5 -hmac "abcdefg" myfile.txt
            1. Output: HMAC-MD5(myfile.txt)= 4bfb9bea00bdbd4ec06c3552cd106daf
        ii. openssl dgst -md5 -hmac "abcdefghijklmnop" myfile.txt
            1. Output: HMAC-MD5(myfile.txt)= eb1000066c8e15b8f60933ae62a0f9b9
    b. HMAC-Sha1:
        i. openssl dgst -sha1 -hmac "abcdefg" myfile.txt
            1. Output: HMAC-SHA1(myfile.txt)= 0d187f205235424daee49d2bd108d459873122ae
        ii. openssl dgst -sha1 -hmac "abcdefghijklmnop" myfile.txt
            1. Output: HMAC-SHA1(myfile.txt)= 6d22227c9b29273e863c3ee471bb24326490eb01
    c. HMAC-Sha256:
        i. openssl dgst -sha256 -hmac "abcdefg" myfile.txt
            1. Output: HMAC-SHA256(myfile.txt)= 89f53a4f402e2252bec5f1d24174888e987aac77639ec4c87c1f0eeea7293d3a
        ii. openssl dgst -sha256 -hmac "abcdefghijklmnop" myfile.txt
            1. Output: HMAC-SHA256(myfile.txt)= 5d415da24161c739947687cd203bc57bc071d3fcf964c484e4816c9fefb0b554

The key used for HMAC does not need to have a fixed size. It requires a key of at least 64 bits and any keys greater than that are first hashed and then used.

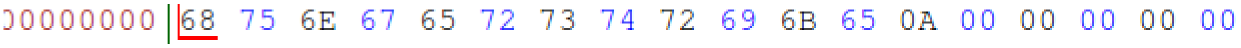
3. Randomness of One-Way Hash
    a. I created a text file containing the string: “hungerstrike”
    b. I hashed the file using the following command: openssl dgst -md5 myfile.txt >> md5hash
        i. Output: MD5(myfile.txt)= 3efdeb6ea539d76fd309ecbded71ce76
    c. I flipped a bit with the following command: printf '\x00' | dd of=myfile.txt bs=1 seek=55 count=1 conv=notrunc
    d. I hashed the flipped file: openssl dgst -md5 myfile.txt >> md5hash_flipped
        i. Output: MD5(myfile.txt)= bbe55c3363d3d46cef7420faf6d91b3d

Original Binary:



```
00000000 68 75 6E 67 65 72 73 74 72 69 6B 65 0A 0A          hungerstrike..
```

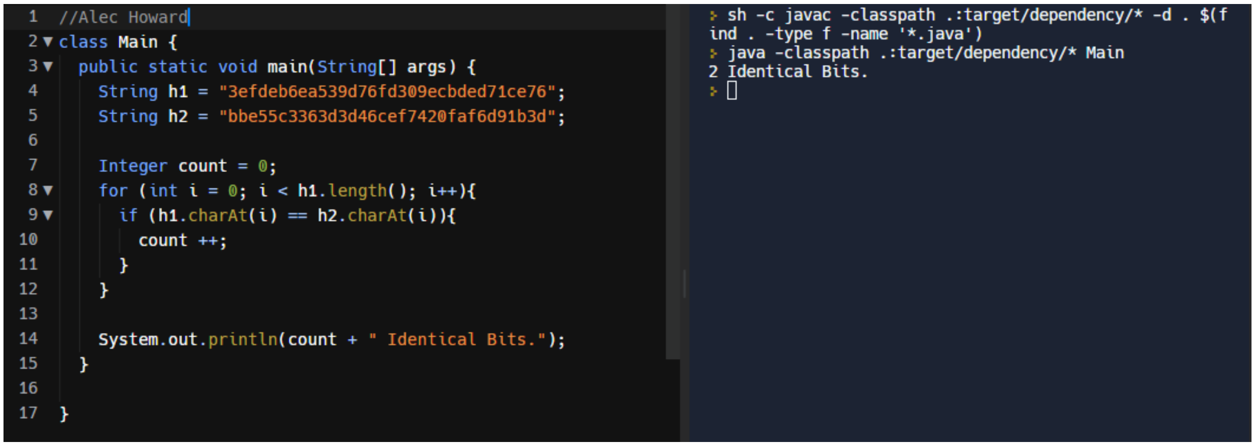
Flipped Binary:



```
00000000 68 75 6E 67 65 72 73 74 72 69 6B 65 0A 00 00 00 00 00
```

It appears that the 28th bit was affected.

H1 and H2 were very different, here is an algorithm I wrote to calculate the number of identical bits:



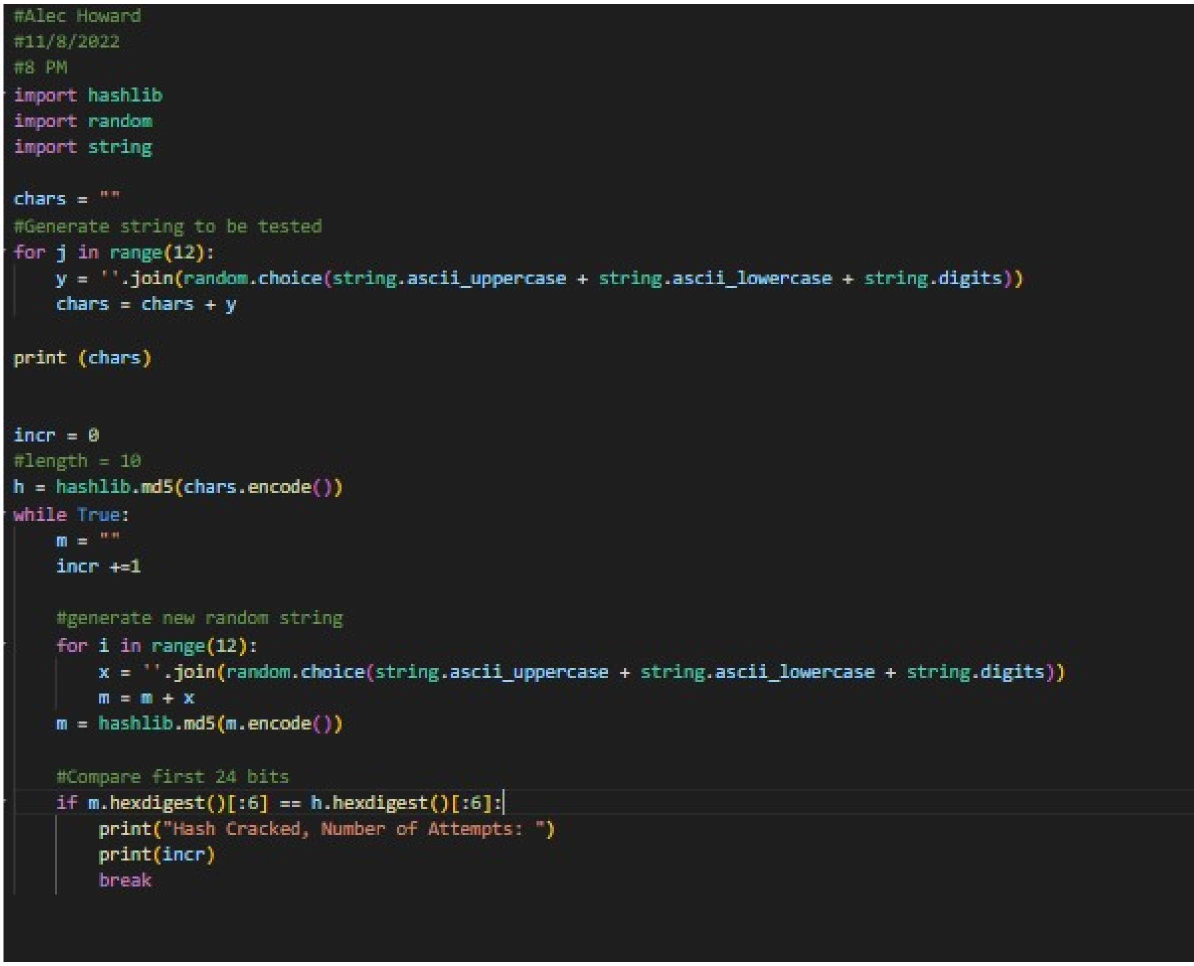
```java
//Alec Howard
class Main {
  public static void main(String[] args) {
    String h1 = "3efdeb6ea539d76fd309ecbded71ce76";
    String h2 = "bbe55c3363d3d46cef7420faf6d91b3d";

    Integer count = 0;
    for (int i = 0; i < h1.length(); i++){
      if (h1.charAt(i) == h2.charAt(i)){
        count ++;
      }
    }

    System.out.println(count + " Identical Bits.");
  }

}
```

```
› sh -c javac -classpath .:target/dependency/* -d . $(find . -type f -name '*.java')
› java -classpath .:target/dependency/* Main
2 Identical Bits.
› 
```

This shows that there are 2 identical bits the resulting hashes.

4. 4.1
    a. Theoretically it should take 2^23 or ~8388608 attempts
    b. I created a random string generator for the message and for each bruteforce attempt. Then they are both encrypted and a new string is tested until a match is found.

i.

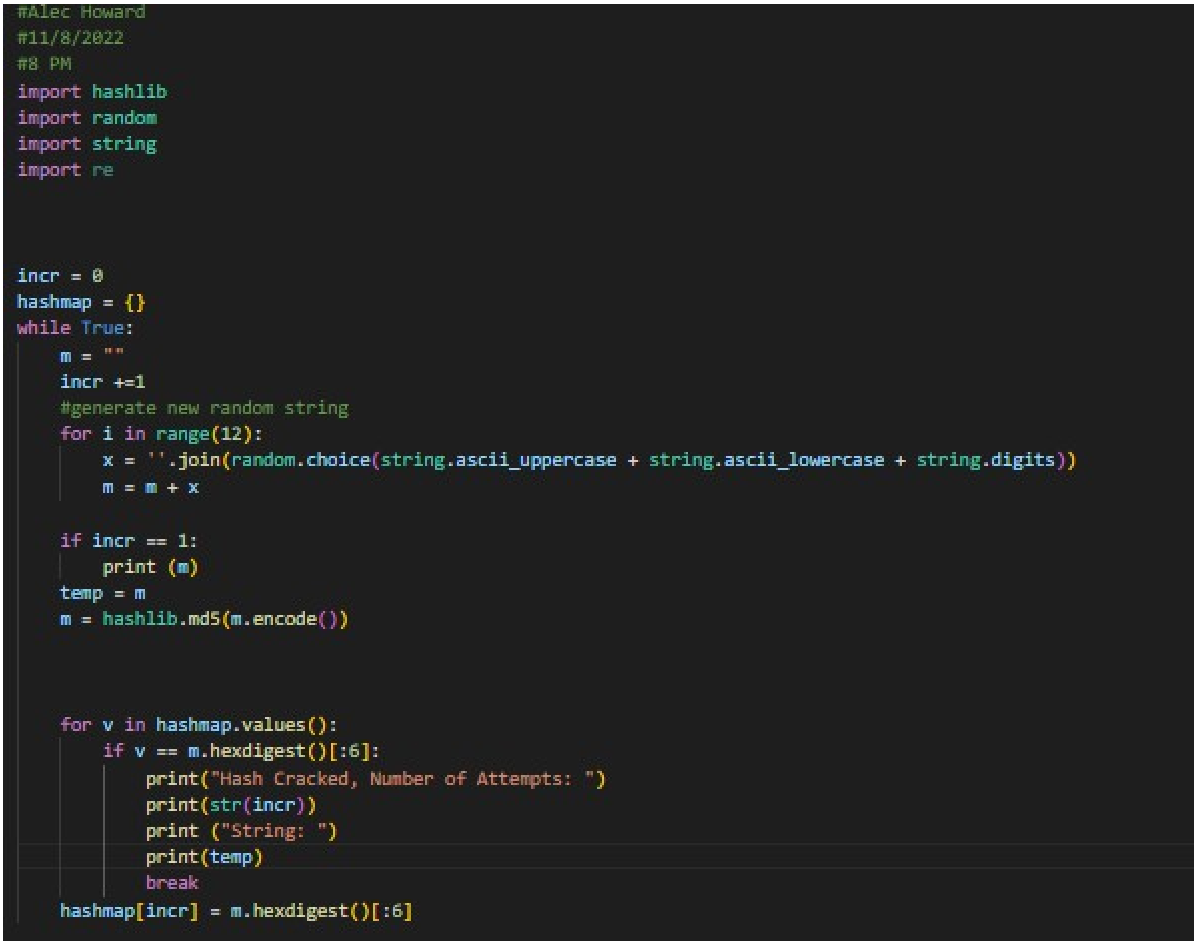


```
#Alec Howard
#11/8/2022
#8 PM
import hashlib
import random
import string

chars = ""
#Generate string to be tested
for j in range(12):
    y = ''.join(random.choice(string.ascii_uppercase + string.ascii_lowercase + string.digits))
    chars = chars + y

print (chars)


incr = 0
#length = 10
h = hashlib.md5(chars.encode())
while True:
    m = ""
    incr +=1

    #generate new random string
    for i in range(12):
        x = ''.join(random.choice(string.ascii_uppercase + string.ascii_lowercase + string.digits))
        m = m + x
    m = hashlib.md5(m.encode())

    #Compare first 24 bits
    if m.hexdigest()[:6] == h.hexdigest()[:6]:
        print("Hash Cracked, Number of Attempts: ")
        print(incr)
        break
```

c. Average Attempts: 5994245.67
    i. First Trial: 2700005 attempts
    ii. Second Trial: 8361147
    iii. Third Trial: 6921585

5. 4.2
    a. Theoretically, it should take 2^12 or, ~ 4096 attempts
    b. For my program, I added each randomly-generated strings to a hashmap if it wasn't already in it. If it appeared in the hashmap already, then I printed and broke the loop.

i.



```
#Alec Howard
#11/8/2022
#8 PM
import hashlib
import random
import string
import re



incr = 0
hashmap = {}
while True:
    m = ""
    incr +=1
    #generate new random string
    for i in range(12):
        x = ''.join(random.choice(string.ascii_uppercase + string.ascii_lowercase + string.digits))
        m = m + x

    if incr == 1:
        print (m)
    temp = m
    m = hashlib.md5(m.encode())



    for v in hashmap.values():
        if v == m.hexdigest()[:6]:
            print("Hash Cracked, Number of Attempts: ")
            print(str(incr))
            print ("String: ")
            print(temp)
            break
    hashmap[incr] = m.hexdigest()[:6]
```

ii. Two strings with the same first 24:
   1. LIElfTjcXf5B
   2. kmnusfqyde8K

c. Average Attempts: 7,957 attempts
   i. First Trial: 746 attempts
   ii. Second Trial: 10030 attempts
   iii. Third Trial: 13095 attempts

6. BONUS:
   a. Based on my observations, Collision-Free as far easier to break.
   b. The average for One-Way was 5994245.67 attempts, while the average for Collision-Free was only 7957 attempts