Patrik Kozak, Gabe Hendrickson
Professor Wang
CIS 458 - 01
12 September 2019

Lab 2

Question 01: Give at least three hash functions supported by the openssl dgst.

1. **md5**
2. **sha1**
3. **sha256**

Question 02: What are the hash functions you used to generate hash values. Did they work as expected?

1. **The hash functions I used to generate hash values include md5, sha1, and sha256. Yes, they worked as expected.**
2. **The commands I used to generate the hash values are as follows:**
   - **openssl dgst -md5 -hmac "MyPassword" test1.txt**
   - **openssl dgst -sha1 -hmac "GrandVstudent" test1.txt**
   - **openssl dgst -sha256 -hmac "CIS458" test1.txt**

Question 03: What are the hash functions you used to generate the keyed hash values? What is/are the key(s) you used?

1. **The first hash function I used was md5.**
   - **The key I used for this function is "MyPassword"**
2. **The second hash function I used was sha1.**
   - **The key I used for this function is "GrandVstudent"**
3. **The third hash function I used was sha256.**
   - **The key I used for this hash function is "CIS458"**

Question 04: Please describe your observations briefly. Please attach a screenshot to support your observations.

Any slight alteration in the file will result in a completely different hash value.

```
ashley@ashley-Aspire-E5-521:~/gvsu/cis458/lab/2$ openssl dgst -md5 lab02_hash.docx
MD5(lab02_hash.docx)= e2521adc1f7df564113075e97c2a422d
ashley@ashley-Aspire-E5-521:~/gvsu/cis458/lab/2$ openssl dgst -md5 lab02_hash_final.docx
MD5(lab02_hash_final.docx)= 78711dfc3c28928e545c02c21f8c99b5
```

Question 05: Attach your script of the sender's part.

The file is the original lab

```
1 #!/bin/bash
2 openssl  dgst -md5 -hmac $1 $2 > .authenticate
3 echo $1 >> .authenticate
4
```

Question 06: Attach your script of the receiver's part.

```
#!/bin/bash

# hashfunction  hashvalue the message and the password

key=$(sed -n '2p' .authenticate)
openssl dgst -md5 -hmac $key $1 >> .authenticate


thereHash=$( sed -n '1p' .authenticate | cut -d' ' -f2 )
yourHash=$(sed -n '3p' .authenticate  | cut -d' ' -f2)


echo comparing the following two hashes
echo $thereHash
echo $yourHash

sed -i '3d' .authenticate

if [ $thereHash == $yourHash ]; then
        echo files are the same
else
        echo files are different
fi
```

Question 07: Briefly describe how the script at the receiver's side works and how to test your script.
The sender must email the receiver with the original file and .authenticate file

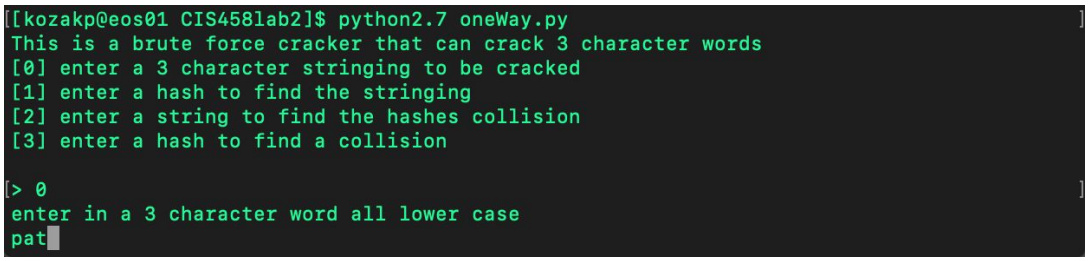
To run the sender Type
./sender keyword filename
To run the receiver type
./receiver filename

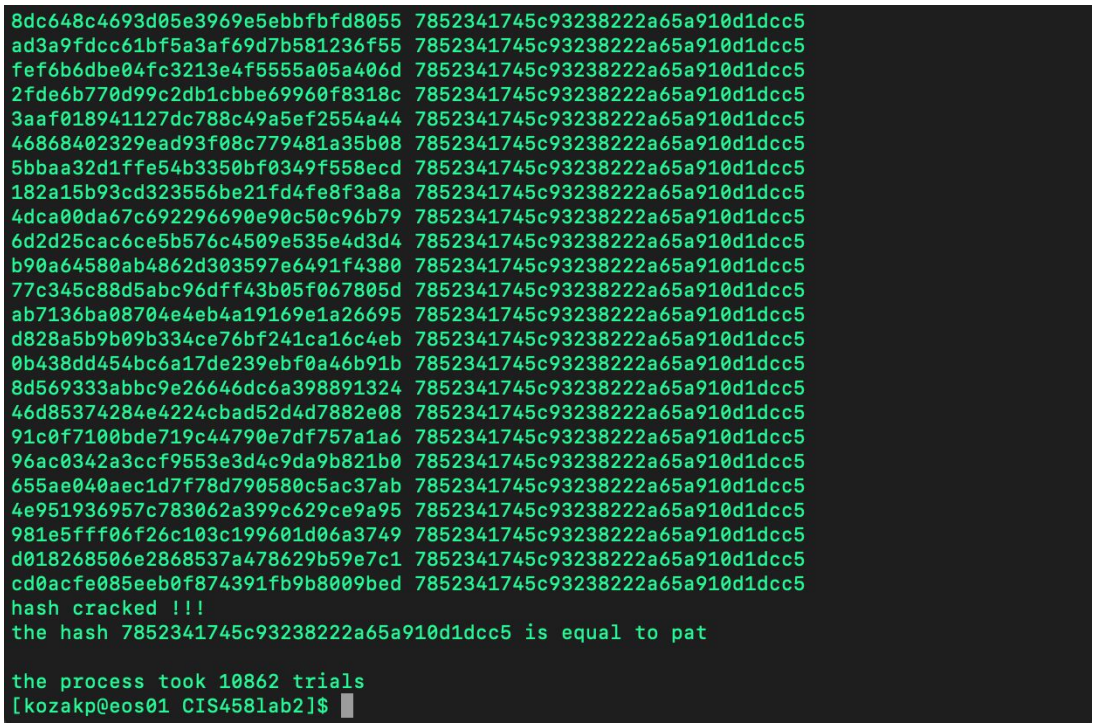
The sender gets a hash and saves to the .authenticate file then the user must email the .authenticate file. The receiver must put the file in the same directory as the test tile the receiver should then run the script with the file name if they are the same it will return files are the same if they are different it will return files are different

Question 08: How many trials it will take you to break the one-way property with the brute-force method? To make your answer reliable, you should repeat your experiment for multiple times

(e.g. 5-10 times) and report the average number of trials. Use screenshot to demonstrate your work if you have.

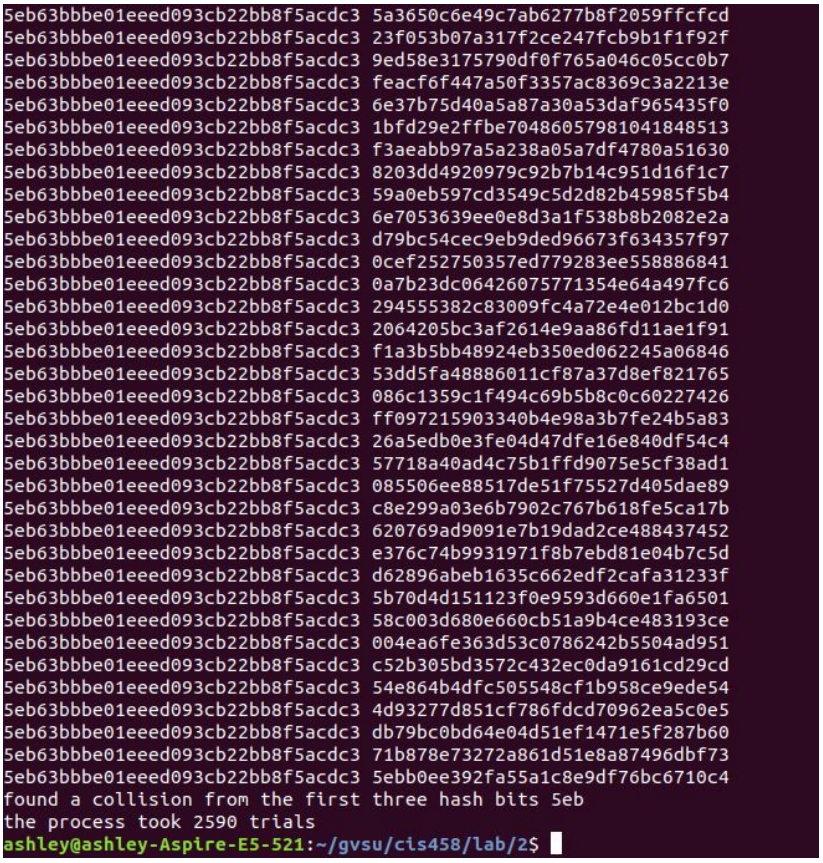
- **The way we implemented the one-way property involves reading through a 'combination' file that reads through every possible combination of strings and hashes the string until the correct hash is found that matches the string we input into the terminal. Therefore, there is no average but the program shows how many trials it took to break the one-way property.**
- **Here are screenshots showing an input of a string and then the amount of trials it takes to break:**





Question 09: How many trials it will take you to break the collision resistance property with the brute force method? Similarly, to make your answer reliable, you should repeat your experiment for multiple times (e.g. 5-10 times) and report the average number of trials. Use screenshot to demonstrate your work if you have.

- **The way we implemented the collection resistance there is no average because it takes the hash and then hashes that hash repeating this process until the first three characters of the hash match. In other words, each hash output is the previous hash, but hashed again. The number of trials needed to break the inputted string is shown in the screenshot below:**



Question 10: Based on your observation, which property is easier to break by using the brute-force method? Please justify your answer. Use screenshot to demonstrate your work if you have.

- **It's easier to break a hash with the collision property rather than a bruteforce match with the one-way property because you must go through every single possibility in a one way property. With the collision method there's a higher probability that you will find a collision in the hash. The number of trails are in the picture**

Question 11: (20 points) Please turn in your program code to Blackboard.