Timothy Jelinek

CIS313, Cryptography
Module 5 Lab – Hashing for Integrity and Password Storage

In this fifth module, we will explore hashing algorithms and their applicability to password storage and authentication as well as integrity checks. A hashing function is a one-way algorithm where it is very easy to calculate a hash for a given input but very difficult or impossible to find a given input for a specific hash. This means that if you have a password hash it can be difficult to find the original password if you only have the hash. As we will see, this often depends on how the hash is implemented. Hashing can also be used to establish integrity. A hash can be provided with a document or software to ensure that the document has not been modified from its original. One of the downsides of plain hashes for integrity is that hashes by themselves do not authenticate the creator or sender. This means that if an attacker hash access to change both the document or software and the hash, they can change the document or software and the hash. To be more secure, message authentication codes using symmetric encryption and digital signatures using asymmetric cryptography plus hashing can provide both authentication and integrity. In this lab we will only explore hashes for integrity and password storage/authentication.

You will be required to submit the following graded items as part of this lab:

- A screenshot and text of your unsalted SHA 256 password hash
- The results of a rainbow table lookup of your unsalted SHA 256 password hash
- The results of a rainbow table lookup of an unknown NTLM password hash
- A screenshot and text of your salted SHA 256 password hash
- The results of the crackstation.net rainbow table lookup of a salted password hash
- A screenshot and hash of a text file
- A screenshot and hash of a text file with one character changed, answered question

Lab 1 – Unsalted and Salted Password Hashing

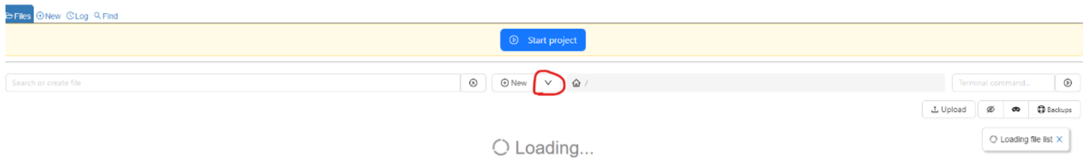
In this lab we will explore salted and unsalted password hashing, including illustrating the danger of unsalted password hashes.

If using CoCalc:

- Open your web browser and surf to https://cocalc.com/doc/terminal.html. (Note: If you have access to a Linux OS you can also use that Linux terminal/command line to do this lab.)
- Click on the Blue 'Try a Linux Terminal Now' button.

```
~$ sage
┌────────────────────────────────────────────────────────────────────┐
│ SageMath version 8.7, Release Date: 2019-03-23                     │
│ Create a "Sage Worksheet" file for the notebook interface.         │
│ Enhanced for CoCalc.                                               │
│ Using Python 2.7.15. Type "help()" for help.                       │
└────────────────────────────────────────────────────────────────────┘
sage: 3 + 9
12
sage: factor(12345)
3 * 5 * 823
sage:
Exiting Sage (CPU time 0m0.30s, Wall time 0m21.34s).
~$ lscpu | head
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  2
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
~$
```

- Click the blue "Use CoCalc Anonymously" button
- Select 'Projects' from the menu bar under the 'Signed in as' title
- Click the 'Create Project' button. You can leave the default project title as is.
- Click the blue 'Start Project' button.
- Click on the down arrow to the right of "New" and select Terminal



- Accept defaults for name for Terminal file, then click 'Create.'

If using Windows:

- If you are using the Cybersecurity Desktop, open the command line by typing cmd in the Type here to search box on the taskbar. The type the following command: set PATH=%PATH%;C:\Program Files\OpenSSL-Win64\bin

Animated Example

1. Next type the following command to generate an unsalted SHA 256 password hash. Notice how the password is 13 characters long and includes uppercase and lowercase letters, special characters, and numbers.

   CoCalc command:

   ```
   echo -n "Password123!!" | openssl dgst -sha256
   ```

   Windows command:

   ```
   echo | set /p="Password123!!" | openssl dgst -sha256
   ```

   This command does the following:

   - echo -n "Password123!!" - outputs the text Password123!! for the next command
   - openssl – This is the name of the command we are using. It has many cryptographic functions.

- dgst – This tells OpenSSL that we will be using the digest or hashing function
- -sha256 – This tells OpenSSL that we will be using SHA 256 as the hashing function

CoCalc Example:

[Animated Example]

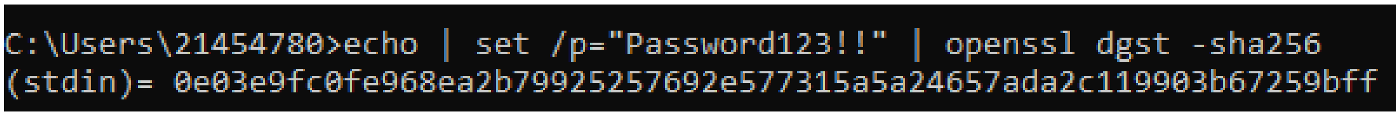
Windows Example:

[Animated Example]

Record your hash value of the password below:

Hash: **0e03e9fc0fe968ea2b79925257692e577315a5a24657ada2c119903b67259bff**

Paste a screenshot of the output of your command below.



```
C:\Users\21454780>echo | set /p="Password123!!" | openssl dgst -sha256
(stdin)= 0e03e9fc0fe968ea2b79925257692e577315a5a24657ada2c119903b67259bff
```

2. Copy the hash you obtained in step 1, paste it at [https://crackstation.net]. Have the website try to crack it. CrackStation uses rainbow tables of precomputed hashes to lookup your password.

How long did it take for the website to lookup this password? Keep in mind that this is 13-character password with uppercase, lowercase, numbers, and special characters.

Answer: **Immediately**

3. Windows uses a hashing algorithm called NTLM to store passwords. NTLM is based on MD4, a precursor to MD5 (MD5 is no longer used because it is considered insecure). Using [https://crackstation.net] or Google, find the password that corresponds to the following NTLM hash: 6616407B2AC7FBD4EAFD812541B261D2 and answer the following questions.

Is the hash base64 or hex encoded?

base64

What is the password?

!QAZ@WSX

4. Now we are going to add a random salt to the password. Run the following commands to salt the password. Prepend (add to the beginning) the random hex characters you get from the first command to the second.

CoCalc command:

```
openssl rand -hex 8
echo -n “<random hex> + Password123!!” | openssl dsgt -sha256
```

Windows command:

```
openssl rand -hex 8
echo | set /p=”<random hex> + Password123!!” | openssl dgst -sha256
```

CoCalc Example:

[Animated Example]

Windows Example:

[Animated Example]

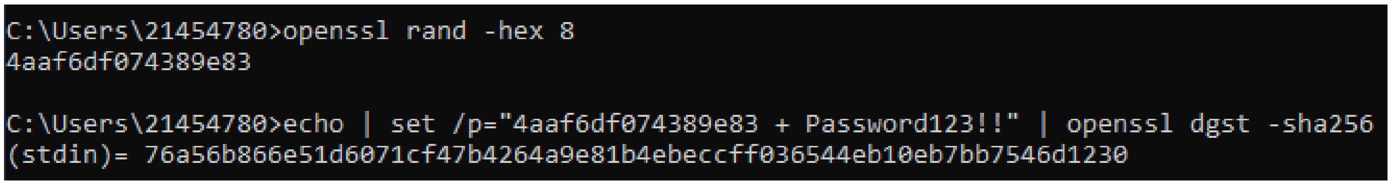
Record the following values:

Salt: 4aaf6df074389e83

Hash: 76a56b866e51d6071cf47b4264a9e81b4ebeccff036544eb10eb7bb7546d1230

Hash stored with the salt in the form <salt>:<hash>:

Paste a screenshot of the command being run.



```
C:\Users\21454780>openssl rand -hex 8
4aaf6df074389e83

C:\Users\21454780>echo | set /p="4aaf6df074389e83 + Password123!!" | openssl dgst -sha256
(stdin)= 76a56b866e51d6071cf47b4264a9e81b4ebeccff036544eb10eb7bb7546d1230
```

5. Go to https://crackstation.net again and paste in the new hash. Press the crack button.

What is the result now?

It says not found.

Why might this be?

By adding a random salt to the password hash, it made the hash more random and difficult to figure out.

Lab 2 - Hashing for file integrity.

In this lab we use hashing to verify the integrity of the file. We also change the content of the file and observe how the hash of that file changes. Often, hashing is used to verify the integrity of download files such as executables and disk images. They are often used to verify the integrity of forensic images taken for law enforcement purpose, to catalog malware files, or to verify the integrity of operating system executables, libraries, and configuration files.

6. Run the following command on Windows and Linux/CoCalc. It will create a file with correct text within the file.

CoCalc command

```
echo Now is the time for all good men to come to the aid of their
country. > integrity.txt
```

```
openssl dgst -sha256 integrity.txt
```

Windows command

```
echo | set /p=”Now is the time for all good men to come to the aid of
their country.“ > integrity.txt
```

```
openssl dgst -sha256 integrity.txt
```

CoCalc Example:

Animated Example

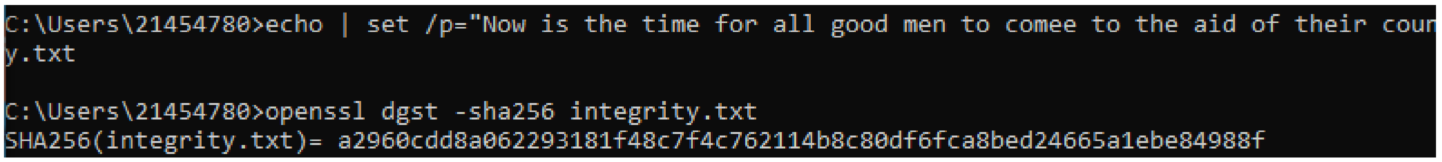
Windows Example:

Animated Example

Record the following values:

Hash: a2960cdd8a062293181f487f4c762114b8c80df6fca8bed24665a1ebe84988f

Paste a screenshot of the command being run.



```
C:\Users\21454780>echo | set /p="Now is the time for all good men to comee to the aid of their coun
y.txt

C:\Users\21454780>openssl dgst -sha256 integrity.txt
SHA256(integrity.txt)= a2960cdd8a062293181f48c7f4c762114b8c80df6fca8bed24665a1ebe84988f
```

7. Run the following command on Windows and Linux/CoCalc. It will create a file with correct text within the file. This command only removes a period from the file you create.

CoCalc command

```
echo Now is the time for all good men to come to the aid of their
country > integrity.txt
```

```
openssl dgst -sha256 integrity.txt
```

Windows command

echo | set /p=”Now is the time for all good men to come to the aid of their country “

```
openssl dgst -sha256 integrity.txt
CoCalc Example:
```

[Animated Example]

Windows Example:

[Animated Example]

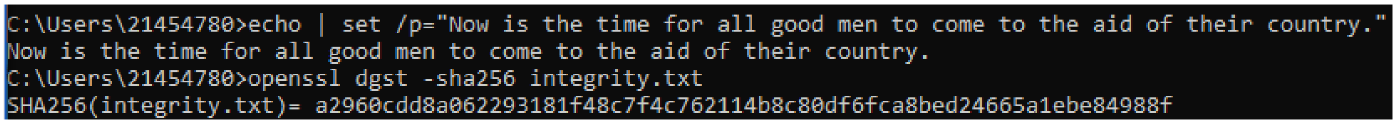
Record the following values:

Hash: a2960cdd8a062293181f48c7f4c762114b8c80df6fca8bed24665a1ebe84988f

How much did the hash change (estimate how many characters changed) just from changing on character in the original text?

Maybe two characters changed?

Paste a screenshot of the command being run.



```
C:\Users\21454780>echo | set /p="Now is the time for all good men to come to the aid of their country."
Now is the time for all good men to come to the aid of their country.
C:\Users\21454780>openssl dgst -sha256 integrity.txt
SHA256(integrity.txt)= a2960cdd8a062293181f48c7f4c762114b8c80df6fca8bed24665a1ebe84988f
```