

[Dashboard](#) / [My courses](#) / [COSC362](#) / [Lab Five: One Way Hash Functions](#)
/ [Lab Five: One Way Hash Functions](#)

Started on	Monday, 24 September 2018, 12:20 PM
State	Finished
Completed on	Tuesday, 25 September 2018, 4:27 PM
Time taken	1 day 4 hours
Marks	12.33/13.00
Grade	9.49 out of 10.00 (95%)

Information

Lab Five: One Way Hash Functions

In this lab we will learn about hash functions, and cover what they are, what uses they have, and testing their security. Hash functions are used pretty extensively in computing, and although most lack authentication, we rely on their security to validate our downloads and software updates.

The goals of this lab are to:

- Learn what a hash function is
- Hash some files with multiple hash functions with openssl
- Understand how to get authenticated hashes, or HMAC
- Learn just how random hashes are
- Test the security of the two properties behind hash functions:
 - The one way property, and the collision free property

All About Hash Functions

A **Hash Function** is any mathematical function that maps data of arbitrary to a fixed size output. They have uses in numerous places, as you would have probably heard of hash tables being an extremely fast and efficient data lookup method.

In this course, we are interested mostly in the security applications for hash functions, and how they are implemented.

A **Cryptographic hash function** takes a file, data, or message as input. The input is then iteratively compressed under a form of lossy compression, and a short fixed length output is produced. This is called a hash.

We can hash short messages:

```
SHA256("Hi 362! Hash functions are great!") =  
e3c1688b43a352bf291b0864731d205c1bc5eb369488b548c6b6a07a75ed5416
```

Or files, like `pic_original.bmp` from lab 4:

```
SHA1(pic_original.bmp) = dfa9c018aa3cd7aac60255b5dfb7bea5c51bece7
```

Or things like passwords:

```
MD5("password1234") = bdc87b9c894da5168059e00ebffb9077
```

We use cryptographic hash functions to sign messages between two people, or for software developers to release software and enable people to check to see if it has been tampered with. If the message changes, be it a change in paragraph, or even by one bit, the resulting hash will be different. This lets people see if letters have been modified, or malware placed inside software packages.

Cryptographic Hash Functions are a special class of hash functions with the following properties:

- They are deterministic, such that the same inputs always results in the same hash.
- They are fast, and hashes can be calculated quickly even for large files.
- It is infeasible to generate a message from its hash, apart from brute forcing all possible messages and hashing them.
- A small change (even 1 bit) to the message means a dramatic change to the hash.
- It is extremely difficult to craft two messages that map to the same hash.

Some common algorithms you may be familiar with are MD5, SHA1, SHA256, SHA512. Different hash functions have different internals, and different mechanisms of generating a specific hash.

Information

Task One: Using Hash Functions

We can hash files and words with the openssl command.

The syntax is like so:

```
openssl <dgst> <filename>
```

Where dgst is one of the supported digests. You can see what algorithms openssl implements with the "man openssl" command.

Try md5, sha1, sha256 or sha512.

Openssl can also read input from stdin:

We can hash individual words by piping them into stdin like so:

```
echo -n "Hello" | openssl sha256  
(stdin)= 185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969
```

Passing the "-n" flag into echo removes the newline, which changes the hash. Try it yourself!

We can hash files by simply passing them as an argument to openssl:

```
openssl sha1 pic_original.bmp  
(pic_original.bmp)= dfa9c018aa3cd7aac60255b5dfb7bea5c51bece7
```

Question 1

Correct

Mark 1.00 out of 1.00

What is the SHA256 hash of the resources file for Lab 5?

Answer:

```
f944c2f8d725297d813e7a50fe708dbbc1a00e7a2145e0b0d1b75d42f7
```



Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 0.67 out of 1.00

What is the SHA1 hash of "Hash functions are very useful things"

Make sure to remove the newline from the end of "echo"!

Answer:

cca3a7691deb1c91ec9096819c74db5db44b4ae6



Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.67/1.00**.

Question **3**

Correct

Mark 1.00 out of 1.00

If you downloaded a fresh copy of your favourite Linux distro, why should you check the hash of your download with the hash on the website?

Select one:

- ☒ a. If the download was corrupted, incomplete or tampered with, you can be alerted to this and download a new copy. ✓
- ☐ b. You can use the hash function to make the download smaller.
- ☐ c. You don't need to. The file downloaded fine, and you are happy to blindly trust your whole operating system install with a file from the internet.

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Information

Task Two: HMAC / Authenticated Hashing

Hash functions are excellent devices to check for modifications to the message, but in reality, they are only as secure as the channel where you got the hash from.

Let's say you wanted to download a web browser for your computer. You download web_browser.zip and web_browser.sha1.txt from the official website. You validate the hash against the one from the downloaded software and it's a perfect match.

But you install it, and now your computer has some nasty viruses. Why?

What happens if an attacker modifies the software, and re-generates new hashes for the infected versions? We get infected ourselves because we trusted the posted hash.

The way we can avoid this, is by adding authentication. HMAC is one way to do this.

HMAC, or keyed-hash message authenticated code, combines hash functions with a secret key. The secret key can then be exchanged out of channel, and you can detect when files have been modified maliciously.

The maths behind this is pretty simple, and you will learn this in lectures. We won't cover it here.

Openssl implements HMAC for all supported digest algorithms. You can try this yourself with the command:

```
openssl <dgst> -hmac "key" <filename>
```

Where dgst is one of md5, sha1, sha256, sha512, and key is your secret key / password, and filename is the file to hash.

Question 4

Correct

Mark 1.00 out of 1.00

What is the HMAC of "Super secret message" with the SHA256 algorithm, and the key "QWERTY1234"?

Remember to remove the newline from echo!

Answer:

```
6663331571dbc7afd0f9b34ff8cef46e99d67548b371948b351cd093acb
```



Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

What is the HMAC of the Lab 5 Resources zip file, with the SHA1 algorithm and key "362362362"?

Answer:

**Correct**

Marks for this submission: 1.00/1.00.

Question **6**

Correct

Mark 1.00 out of 1.00

If you look at the implementation of HMAC, you see that we need keys of a specific length, to match the block size of the hash function. What happens if we use a key smaller / larger / same size as the block size?

What happens if the key is larger?



What happens if the key is the same size?



What happens if the key is smaller?



Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Task Three: Randomness

So far, we have hashed some files, a few words and have authenticated them. But what happens when those files or words change? How much does the resulting hash change?

We will put this to the test. Let's do some basic examples:

```
echo -n "Hello" | openssl sha1
(stdin)= f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0

echo -n "Hfll0" | openssl sha1
(stdin)= a5f54e9436b7d69d2a3bed97e72659f291639efc
```

By swapping 'e' to 'f', or if we look down in the ASCII spec, we changed e / 0x65 -> f / 0x66, or a 1 bit change, which resulted in a dramatically different hash. The hash also changes drastically when you change larger amounts of the message too. Try this yourself.

To answer just how much the hash changes, we have included a program in the Lab 5 Resources zip file, called comparer.

You can compile comparer with:

```
gcc -o comparer Comparer.c
```

What comparer does, is iterate over two hashes, and counts the number of bits which are the same in each. It will then tell you how many are the same, which you can use to make a percentage.

Do the following:

1. Make a text file, and put some text in it.
2. Hash the file and save the hash to a file, called hash1
3. Open the text file, change one character, or more.
4. Hash the file and save the hash to a file, called hash2
5. Run comparer over the files with "comparer count hash1 hash2"

Compare the SHA1 and SHA256 algorithms. SHA1 is 20 bytes, and SHA256 is 32 bytes.

If you are looking for a fast way to do this, you can do it with:

```
# Make two different files
echo "test file please ignore" > file1
echo "test$file please ignore" > file2

# Hash both files, direct to binary hashes
openssl dgst -sha256 -binary -out hash1 file1
openssl dgst -sha256 -binary -out hash2 file2

# Run the comparer program
./comparer 32 hash1 hash2
```

Question **7**

Correct

Mark 1.00 out of 1.00

In the above example with the two messages:

"test file please ignore" and "test\$file please ignore"

When tested under sha256, how many bits are different?

Select one:

- ☒ a. 123 bits are different, which is < 50% ✓
- ☐ b. 203 bits are different, which is > 75%
- ☐ c. 83 bits are different, which is < 33%
- ☐ d. 237 bits are different, which is > 90%

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Question **8**

Correct

Mark 0.67 out of 1.00

Just how different are the hashes with SHA1 for the phrases:

"this will be very different" and "this wont be very different"

Select one:

- ☐ a. 149 bits are different, which is > 90%
- ☐ b. 47 bits are different, which is < 30%
- ☒ c. 86 bits are different, which is ~ 50% ✓

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.67/1.00**.

Task Four: One-Way Property versus Collision-Free Property

In this task, we will look into the security underpinning cryptographic hash functions. Namely the one-way property and the collision-free property.

The One-Way Property

The **one way property** is what make hash functions a "trap door function", which is that you cannot reconstruct the message from any given hash.

It is defined as:

For any given hash h , it is computationally infeasible to find a message x , where $H(x) = h$

Let's say I give you a hash:

```
ca21dcddb09a61249ee41445efcdd8fd9d568abbf955e582960d691ec72726b3
```

Can you reconstruct the message I hashed? Probably not, since I used SHA256, but if you do, let me know. I'll give you a chocolate fish for your efforts, while I go after the Bitcoin protocol.

Now, without doing any clever attacks on SHA256, it would take years, centuries or even the age of the universe to brute force every single message possible, to find one that matches that above hash.

But what if we did not use SHA256, and instead use a custom flawed hash function, which is limited to 24 bits? How many tries would it take?

We have prepared a program called OneWayBreaker.py which will do exactly this. Go ahead and run it, and run it several times to get an average number of tries it takes to brute force a randomly generated message.

The Collision-Free Property

The collision-free property comes in two variants. They are sometimes called different things, and they are:

Preimage resistance / weak collision resistance

and

Second preimage resistance / strong collision resistance

Now, weak collision resistance is defined as:

For any message x , it is computationally infeasible to find another message y , such that $y \neq x$, and the hashes match: $H(y) = H(x)$

Now, to demonstrate this, imagine you got a contract to sign for an internship. It says you will be paid \$18 an hour. It hashes to

```
2c624232cdd221771294dfbb310aca000a0df6ac8b66b696d90ef06fdefb64a3
```

If the hash algorithm had weak collision resistance, you would be able to tweak the contract to say you will be paid \$25 an hour, and then brute force adding some random bytes to the end of the file, such that the hash matches the original. Then you could simply submit that version of the contract and get paid more.

Strong collision resistance is slightly different. It is defined as:

It is computationally infeasible to find any pair of messages (x, y) such that the hashes match: $H(x) = H(y)$

This is where you can always find, in a reasonable amount of time, a collision for any two arbitrary messages. Any two messages at all. Very hard.

Again, we have prepared a program called CollisionFreeBreaker.py which implements a brute force algorithm for weak collision resistance. The program is incomplete, and you will need to finish it. Read along the pseudo code, and modify the program as necessary.

One-Way vs Collision-Free

Now that you are aware of both properties, what one is the easier one to break with brute force? Why?

Question 9

Correct

Mark 1.00 out of 1.00

How many trials will it take you to break the one-way property using the brute-force method?

You should repeat your experiment for multiple times, and report your average number of trials.

What are results from running OneWayBreaker.py a few times? Select the order of magnitude.

Select one:

- ☐ a. Hundreds of thousands
- ☐ b. Single millions
- ☐ c. Hundreds of millions
- ☒ d. Tens of millions ✓ It should be around 17 million. Aka $2^{24}=17M$.
- ☐ e. Tens of thousands

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Question **10**

Correct


Mark 3.00 out of 3.00

How many trials will it take you to break the collision-free property using the brute-force method?

You should repeat your experiment for multiple times, and report your average number of trials.

What are results from running the completed CollisionFreeBreaker.py a few times? Select the order of magnitude.

Select one:

- ☐ a. Hundreds of millions
- ☒ b. Thousands  Great.
- ☐ c. Millions
- ☐ d. Tens of thousands
- ☐ e. Tens of thousands
- ☐ f. Tens of millions

Your answer is correct.

Correct

Marks for this submission: 3.00/3.00.

Question **11**

Correct

Mark 1.00 out of 1.00

What property is easier to brute force? The one-way property or the collision-free property?

Look into the mathematics behind the two. How does the birthday problem make one simpler than the other?

Select one:

- ☒ a. The collision-free property ✓
- ☐ b. The one-way property

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

[◀ Lab Four: Resources](#)

Jump to...

[Lab Five: Resources ▶](#)