Lab 5

5.3 - In §[5.1](file:///Volumes/Hackintosh%20HDD%201/CSCE%20465/Network-Security-Textbook-CSCE-465/ch05lev1sec1.html#ch05lev1sec1) Introduction we discuss the devious secretary Bob having an automatic means of generating many messages that Alice would sign, and many messages that Bob would like to send. By the birthday problem, by the time Bob has tried a total of 232 messages, he will probably have found two with the same message digest. The problem is, both may be of the same type, which would not do him any good. How many messages must Bob try before it is probable that he'll have messages with matching digests, and that the messages will be of opposite types?

* Bob generates 2^32 messages. Therefore, there is a probability that type 1 matches type 2. This probability is . Because of this, a 2^32 type 2 will most likely have a match with a 2^32 type 1.

5.4 - In §[5.2.4.2](file:///Volumes/Hackintosh%20HDD%201/CSCE%20465/Network-Security-Textbook-CSCE-465/ch05lev1sec2.html#ch05lev3sec4) Hashing Large Messages, we described a hash algorithm in which a constant was successively encrypted with blocks of the message. We showed that you could find two messages with the same hash value in about 232 operations. So, we suggested doubling the hash size by using the message twice, first in forward order to make up the first half of the hash, and then in reverse order for the second half of the hash. Assuming a 64-bit encryption block, how could you find two messages with the same hash value in about 232 iterations? Hint: consider blockwise palindromic messages.

* The digest size still needs to be 64 bits. The iterations will also be the same. A collision will occur if a palindrome exists when the 2 messages are hashed.

5.14 - For purposes of this exercise, we will define random as having all elements equally likely to be chosen. So, a function that selects a 100-bit number will be random if every 100-bit number is equally likely to be chosen. Using this definition, if we look at the function "+" and we have two inputs, x and y, then the output will be random if at least one of x and y are random. For instance, y can always be 51, and yet the output will be random if x is random. For the following functions, find sufficient conditions for x, y, and z under which the output will be random:

| * ~x * If x is random, then y and z will be independent |
| --- |
| * x/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2295.GIFy * if y or x is random, then z will be independent |
| * x/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2228.GIFy * if x or y is random x != y, then z will be independent |
| * x/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2227.GIFy * is x or y is random and x != y, then z will be independent |
| * (x/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2227.GIFy)/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2228.GIF(~x/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2227.GIFz) [the selection function] * If x and y are different to get a non-zero value or if ~x and z differ |
|  |
| * (x/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2227.GIFy)/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2228.GIF(x/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2227.GIFz)/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2228.GIF(y/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2227.GIFz) [the majority function] * If x, y or z are different in 1 bit or more * (x /Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2295.GIF y /Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2295.GIF z) * Will be random if either 2 of the 3 are different by 1 bit or more * y/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2295.GIF(x/Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/U2228.GIF-z) * Will be random if either x or ~z are different |
|  |

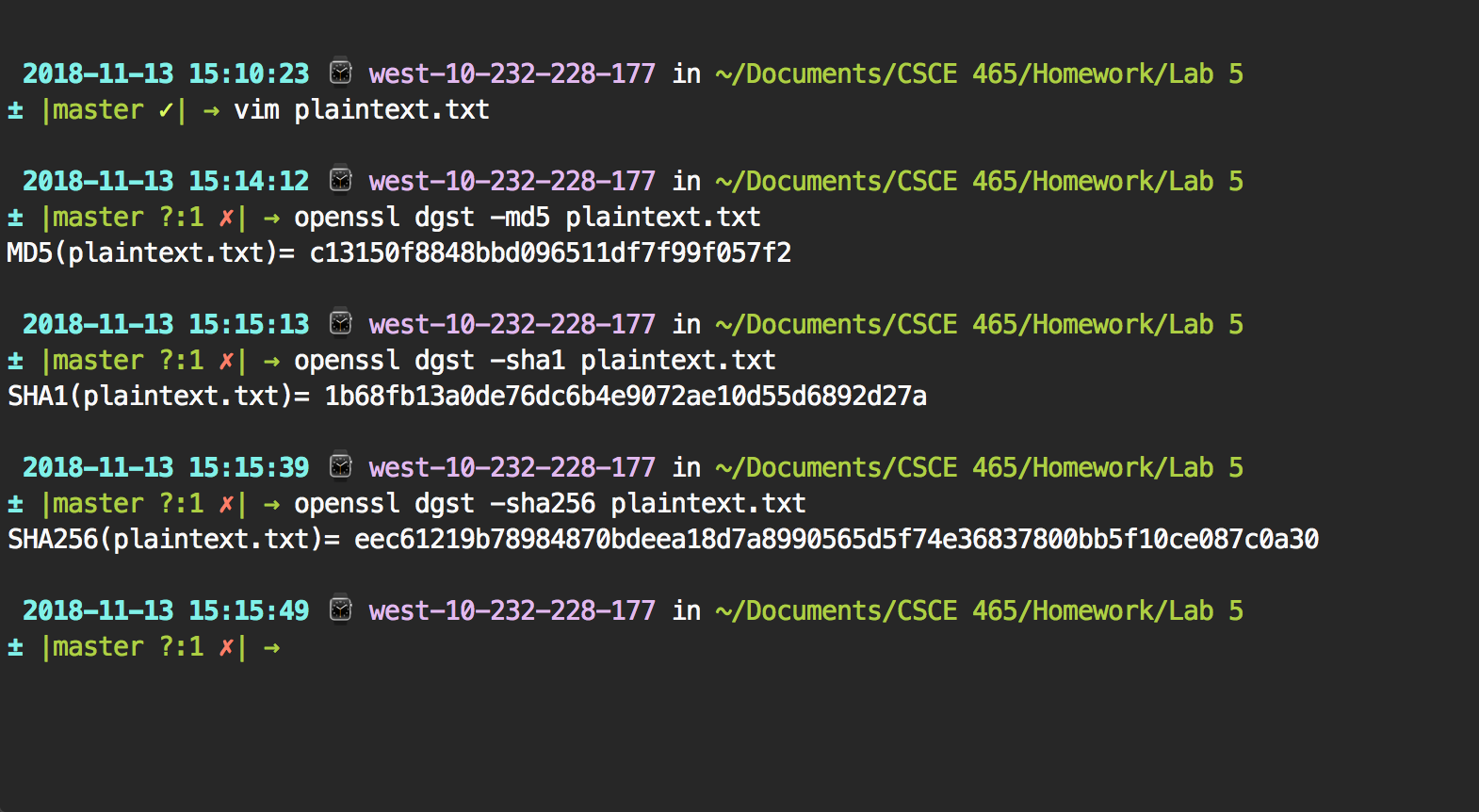
6.2 - In section §[6.4.2](file:///Volumes/Hackintosh%20HDD%201/CSCE%20465/Network-Security-Textbook-CSCE-465/ch06lev1sec4.html#ch06lev2sec11) Defenses Against Man-in-the-Middle Attack, it states that encrypting the Diffie-Hellman value with the other side's public key prevents the attack. Why is this the case, given that an attacker can encrypt whatever it wants with the other side's public key?

* The hacker will not be able to decrypt the Diffie Helman values. Because of this, he will not be able to compute the shared secrets.

6.8 - Suppose Fred sees your RSA signature on m1 and on m2 (i.e. he sees /Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/183equ01.jpgmod n and /Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/183equ02.jpgmod n). How does he compute the signature on each of /Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/183equ03.jpgmod n (for positive integer j), /Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/183equ04.jpgmod n, m1·m2 mod n, and in general /Volumes/Hackintosh HDD 1/CSCE 465/Network-Security-Textbook-CSCE-465/183equ05.jpgmod n (for arbitrary integers j and k)?

* If Freddie boi saw my RSA signature and tried to compute the signature, he would need to raise my signature on m1 to the jth power modulus n. From there, compute the inverse mod n of m1 ( ), then (m1 \* m2)mod(n). Then (m1j)dmod(n) = (m1d)jmod(n), then (m1-1)dmod(n) = (m1d)-1mod(n) and then (m1\* m2)dmod(n) = (m1d)m2dmod(n).
* So when he gets the general case m1jm2kmod(n), he can get my signature on m1sgnjmod(n) and raise it to the jth power and get signature m2sgnkmod(n) and raise to the kth power to get .

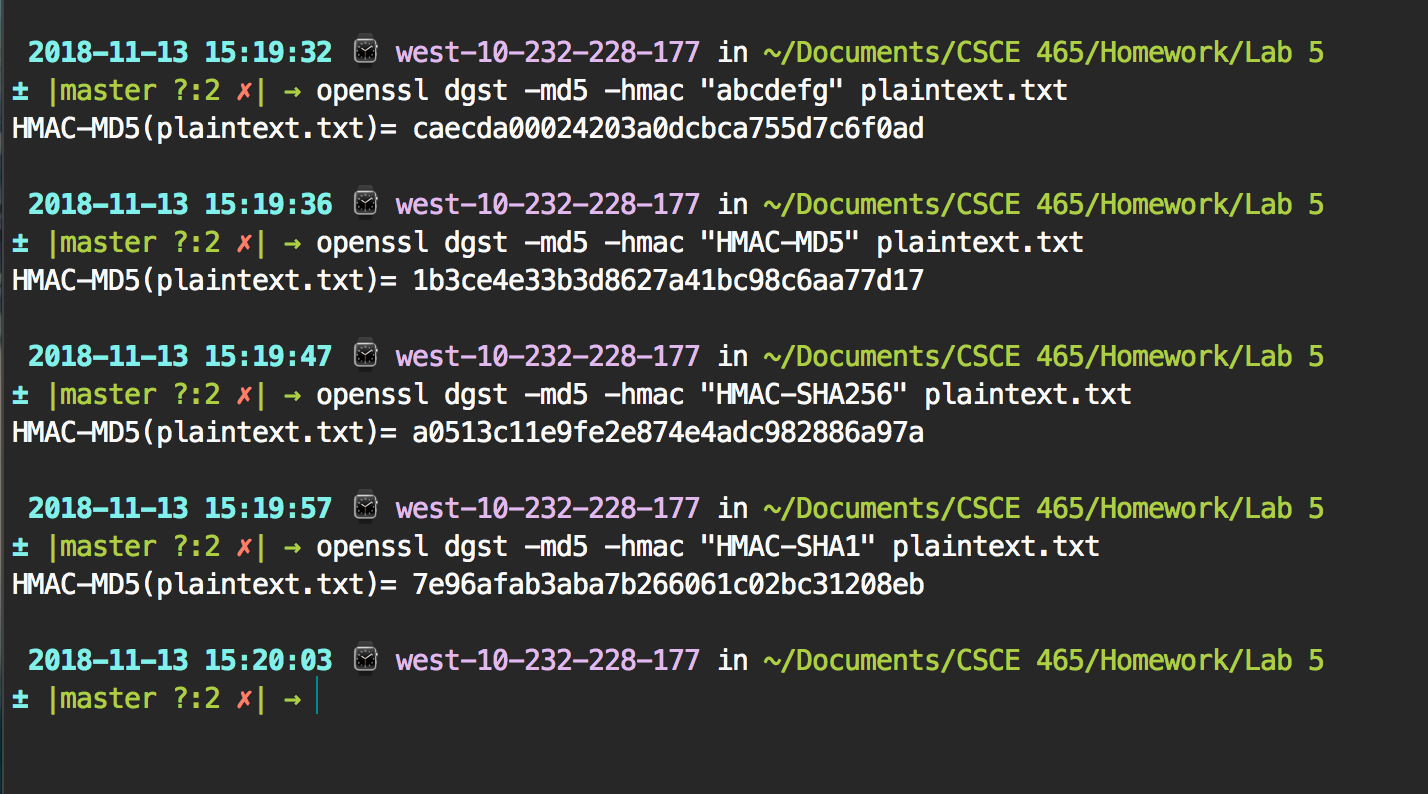
Task 1:



The only difference between them is the length of the hash that they generate. The order of them in length from shortest to longest is md5, sha1, and sha256.

Task 2:

With Different Key Sizes



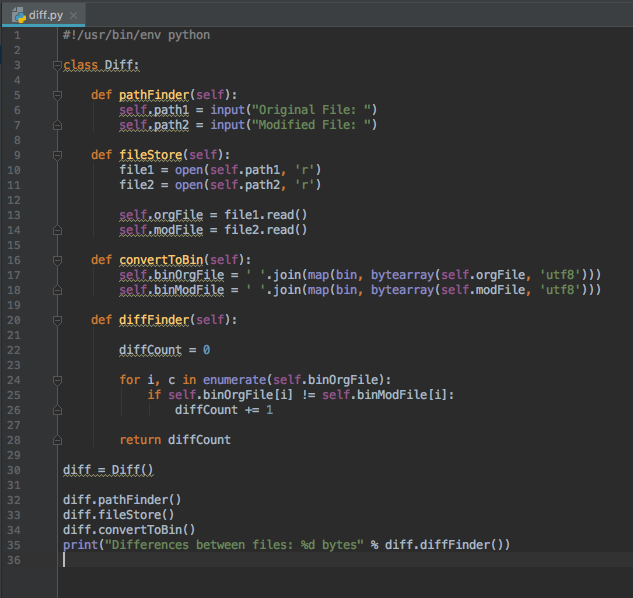
With Different HMAC Types



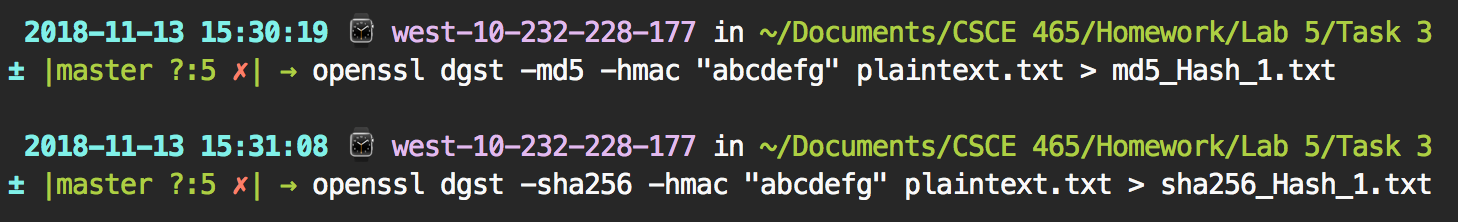
It is not necessary to used a fixed-size key in HMAC. It is not dependent on the length of the key as long as there is a key.

Task 3:

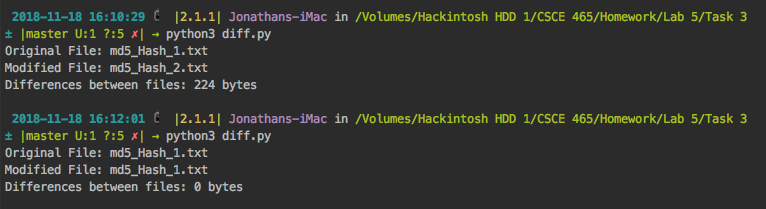
Difference Counter Code



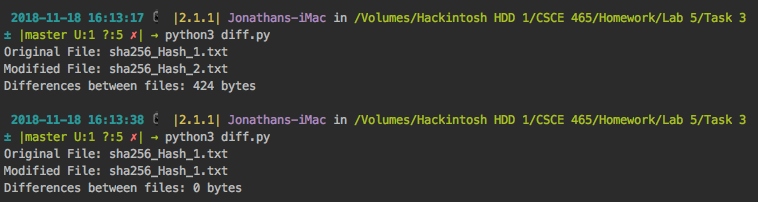
Generating Hash



Differences for MD5



Differences for SHA256



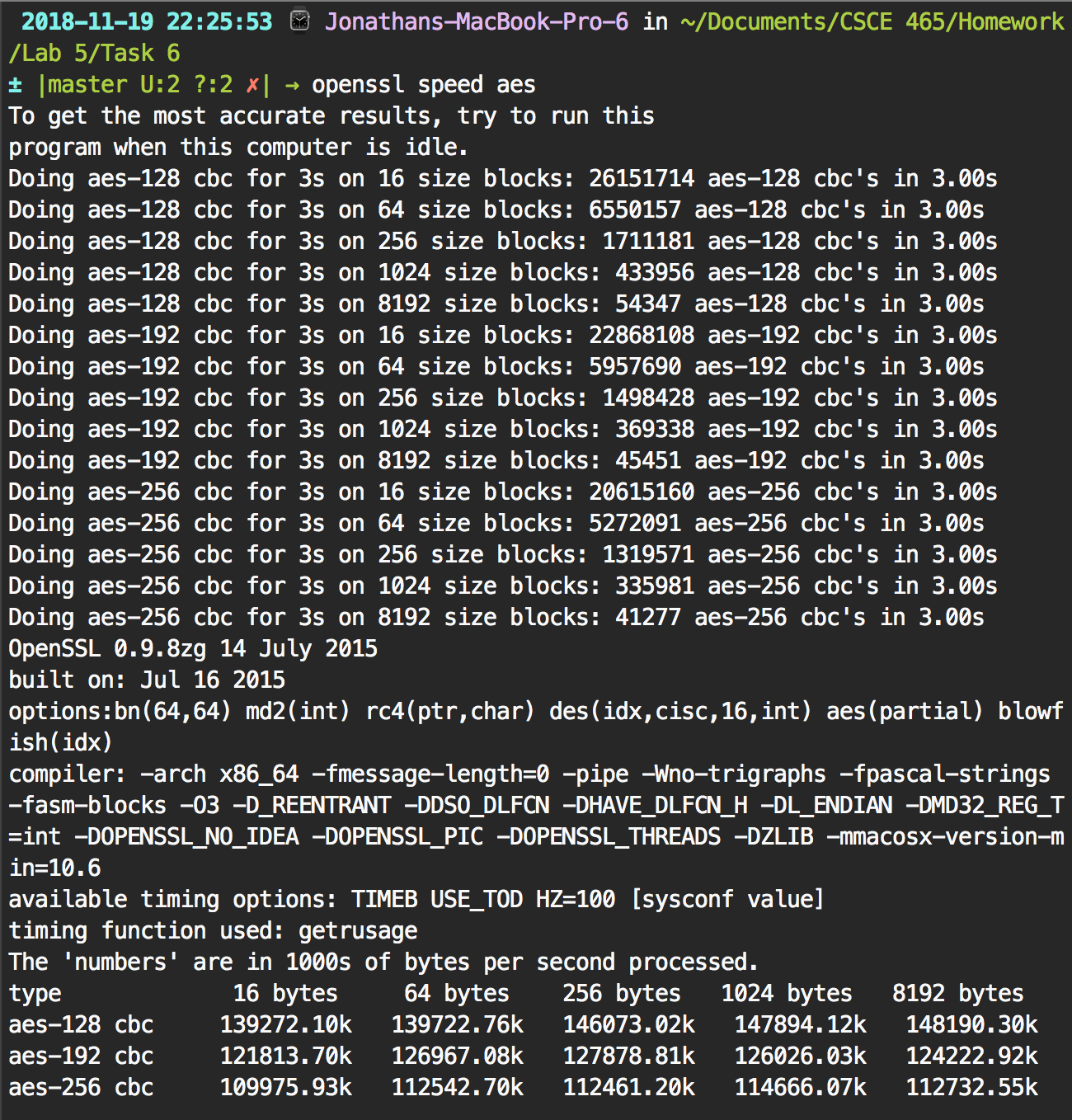
The H1 and H2 are completely different. This shows that even just a difference in 1 bit can completely change the outcome of the hash. This was showed by the python program I wrote that counts the number of different bits (after converting the text file to binary) that are different between H1 and H2.

Task 4:

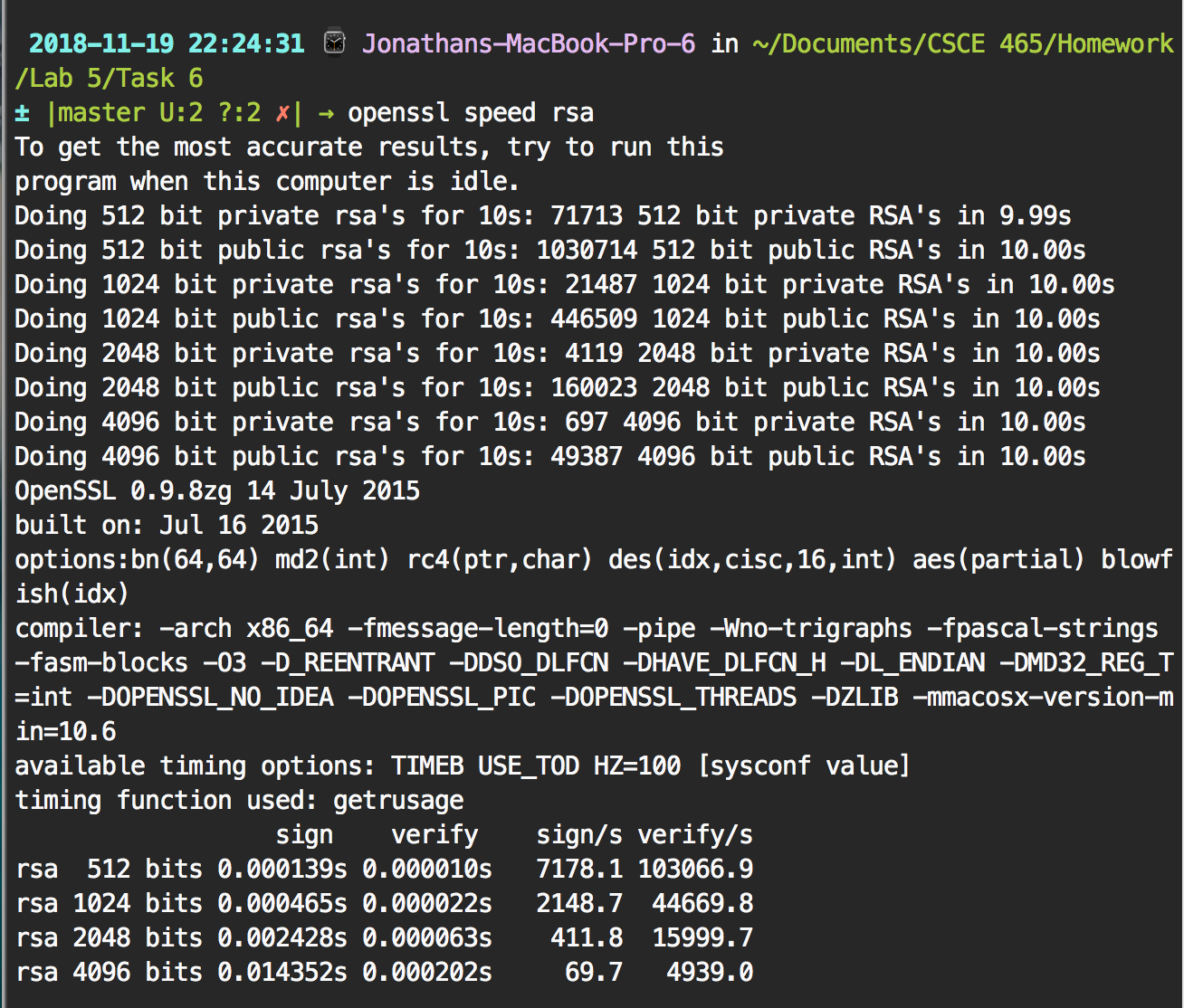
***PUT STUFF HERE!!!!!!!!!!!!!!!!!!!!!!!!***

Task 5: PUT OTHER SPEED TIMES HERE

AES OpenSSL Speed Times



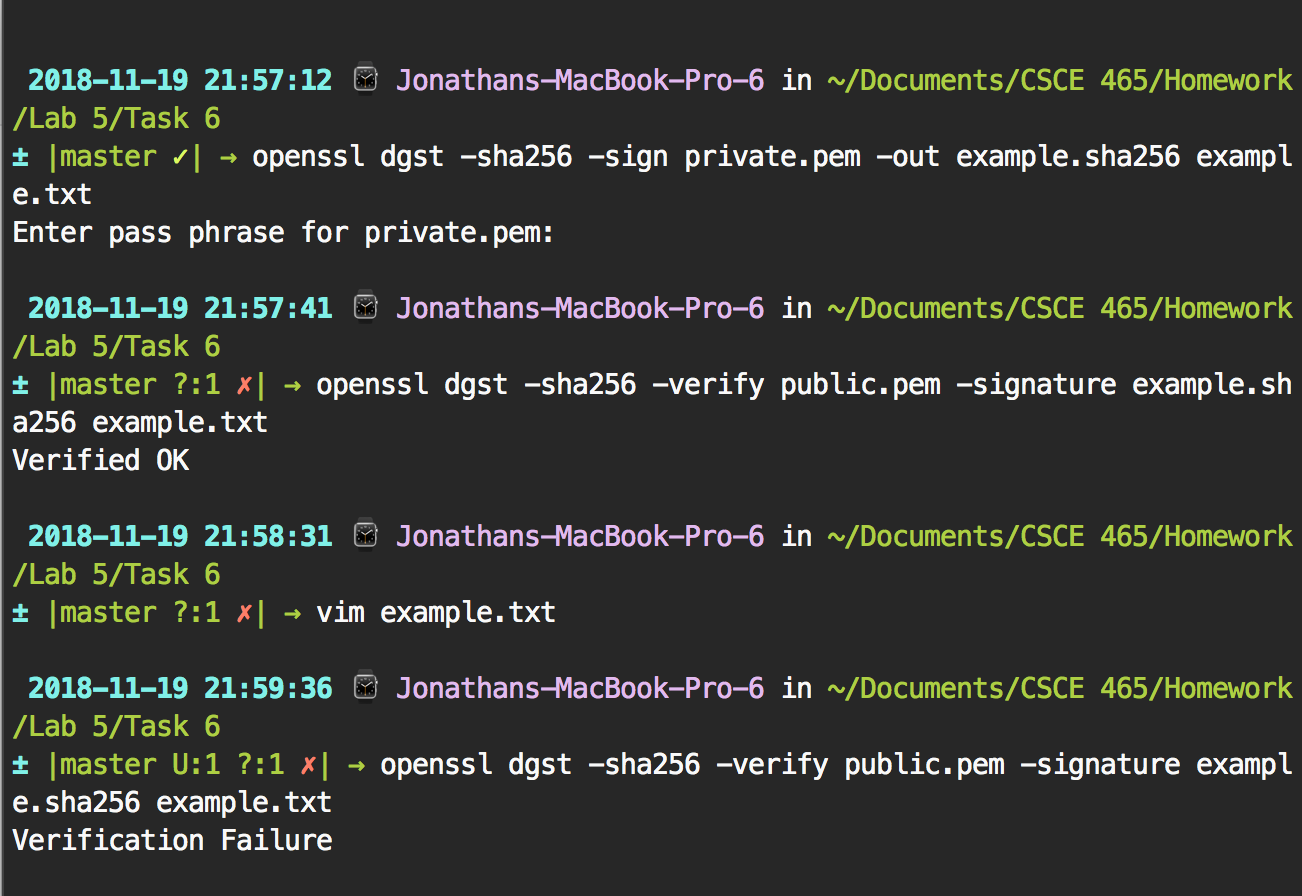
RSA3 OpenSSL Speed Times



The times between my trials of both RSA and AES are similar to the openssl speed tests (using the system time since real time includes typing in the sudo password).

Task 6:

Output Of Signature Verification



First I made a sha256 signature for example.txt and saved the output of it in example.sha256 using my private key. Then I used dgst with the verify flag to verify if the file has been tampered with. I then changed the file and verified it again. It failed after I modified it and that is why signatures are important. This helps you to verify if the file has been modified in anyway. This can potentially prevent man-in-the-middle attack because you can make sure that what you receive is not malicious.