EN4720: Security in Cyber-Physical Systems
Exercise — Encryption and Hashing
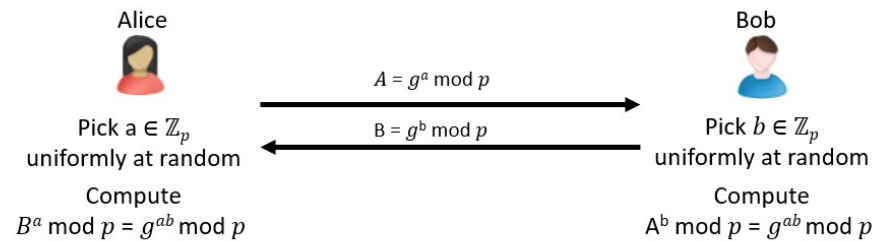
Name: Thalagala B. P.
Index No: 180631J

April 21, 2023

This is an individual exercise!
Due Date: 21 April 2023 by 11.59 PM

Problem 1 - Symmetric Encryption & Key Exchange

Alice and Bob are using the Diffie-Hellman protocol as shown in Figure 1. They want to extend their key exchange mechanism to include their friend Charlie.

- Setup:
  - Prime p, generator g of $\mathbb{Z}_p^*$ (public parameters)

Alice   Bob

$A = g^a \bmod p$

Pick $a \in \mathbb{Z}_p$ $B = g^b \bmod p$ Pick $b \in \mathbb{Z}_p$
uniformly at random   uniformly at random

Compute   Compute
$B^a \bmod p = g^{ab} \bmod p$ $A^b \bmod p = g^{ab} \bmod p$

- Alice and Bob can use as shared key:
  - $X = g^{ab} \bmod p$

Figure 1: Diffie Hellman Protocol

1. Design an extension of the Diffie-Hellman protocol to allow secure key exchange between the three parties.

The below method can be used to extend the DH protocol to share a common key between three parties. The method is graphically represented in the Figure 2 as well. The answer was adopted from $https://crypto.stackexchange.com/a/1027$
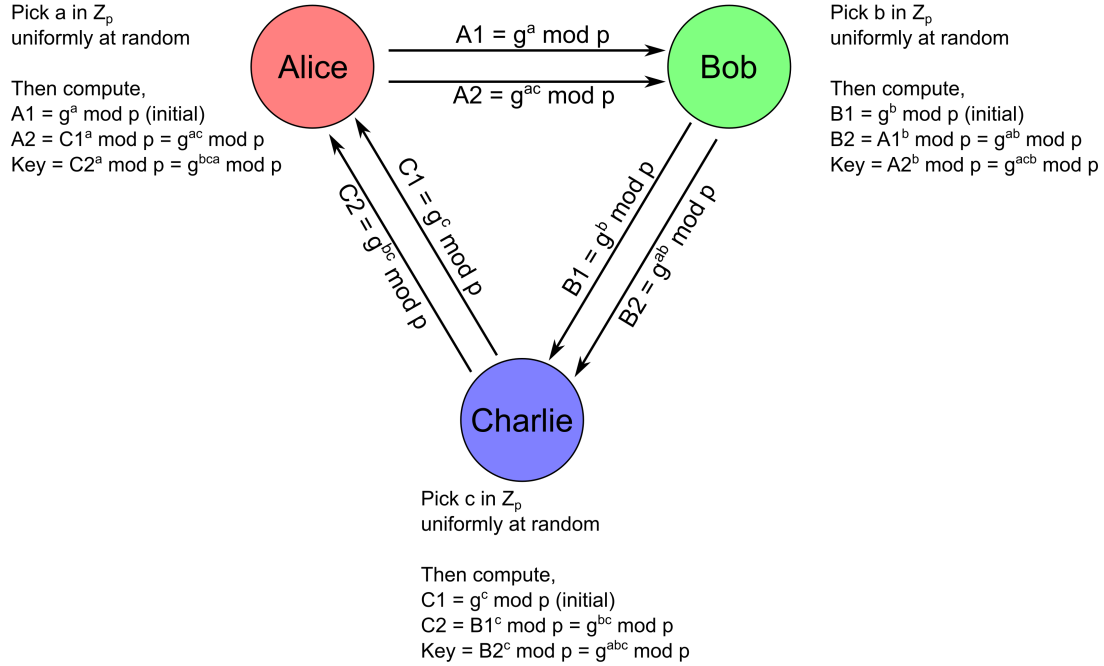
Pick a in Z_p
uniformly at random

Then compute,
A1 = g^a mod p (initial)
A2 = $C1^a$ mod p = g^{ac} mod p
Key = $C2^a$ mod p = g^{bca} mod p

A1 = g^a mod p
A2 = g^{ac} mod p

Pick b in Z_p
uniformly at random

Then compute,
B1 = g^b mod p (initial)
B2 = $A1^b$ mod p = g^{ab} mod p
Key = $A2^b$ mod p = g^{acb} mod p

C1 = g^c mod p
C2 = g^{bc} mod p

B1 = g^b mod p
B2 = g^{ab} mod p

Pick c in Z_p
uniformly at random

Then compute,
C1 = g^c mod p (initial)
C2 = $B1^c$ mod p = g^{bc} mod p
Key = $B2^c$ mod p = g^{abc} mod p

Figure 2: Extended Diffie-Hellman Protocol for three parties to share a common key

(a) *Alice, Bob and Charlie each picks a random number from the chosen multiplicative group ($\mathbb{Z}_p$). Let those numbers be a, b and c respectively, and Alice, Bob and Charlie are in a loop as shown in the Figure 2.*

(b) *Initially each party computes $Init_{val} = g^i$ mod p (**A1, B1, C1** in the Figure 2), for their selected i where $i \in \{a, b, c\}$ and passes to the next person in the loop.*

(c) *Upon receiving an $Init_{val}$ from the previous party of the loop, each party again computes $Inter_{val} = Init_{val}^i$ mod p, (**A2, B2, C2** in the Figure 2) and passes to the next person in the loop.*

(d) *Finally, upon receiving the $Inter_{val}$ from the previous party, each party calculates $Key_{val} = Inter_{val}^i$ mod p (**Key** in the Figure 2), which becomes the common key between the three parties.*

The key will take the form g^{abc} mod p after the above described process and will be a common key that can be used among the three parties for information encryption and decryption.

2

2. Alice, Bob, and Charlie are planning to use $(p, g) = (23, 9)$ to agree on a key for a Caesar cipher. Assume that the Caesar cipher is right shifting the number of letters equal to the key to produce the ciphertext (e.g., if the key is 5 and the plaintext letter is U, it will be replaced by Z in the ciphertext). Assuming 4, 7, and 13 to be secret exponents of Alice, Bob, and Charlie respectively, find the common key that they need to use in Caesar's cipher.

$$
\begin{aligned}
Key &= g^{abc} \bmod p \\
&= 9^{4 \times 7 \times 13} \bmod 23 \\
&= 9
\end{aligned}
$$

3. Derive the encrypted message for the message "Hello" using the above encryption key. Show all the major steps in deriving the answer.

   (a) *Since the key is 9, each letter in the original message is right shifted by 9 letters to produce the Caesar cipher.*

   (b) *Assume there is no requirement to preserve the case sensitivity of the message. Then the message is "HELLO"*

   (c) *Now for each letter in the word, replace it by the letter 9 slots to its right. Then 'H' is replaced by 'Q', 'E' is replaced by 'N', 'L' is replaced by 'U' and 'O' is replaced by 'X'.*

   (d) *Then the ciphertext will be "QNUUX"".*

Problem 2 - Symmetric Encryption - Security Analysis

For this part of the exercise you are required to install CrypTool. CrypTool is a software tool that comes with several encryption techniques.

Instructions

1. Create a text file named **"sampletext.txt"** in your computer. Add the text "EN4720: This is a sample text file for encryption." to the text file and save it.

2. Open the **"sampletext.txt"** file with CrypTool software.

3. Encrypt your message with RC2 encryption following **Encrypt/Decrypt >Symmetric (modern) >RC2**. Choose key size of 8 bits and add some random hexadecimal key to encrypt your text file. (Add screenshots)

4. Run brute force analysis of the encrypted message by following **Analysis >Symmetric Encryption (modern) >RC2**. (Add screenshots)

5. Repeat steps 3,4 for the key sizes 16, 24, 64, and 128 bits.

6. Answer the questions given below.

Questions

1. Add screenshots requested in Step #3.
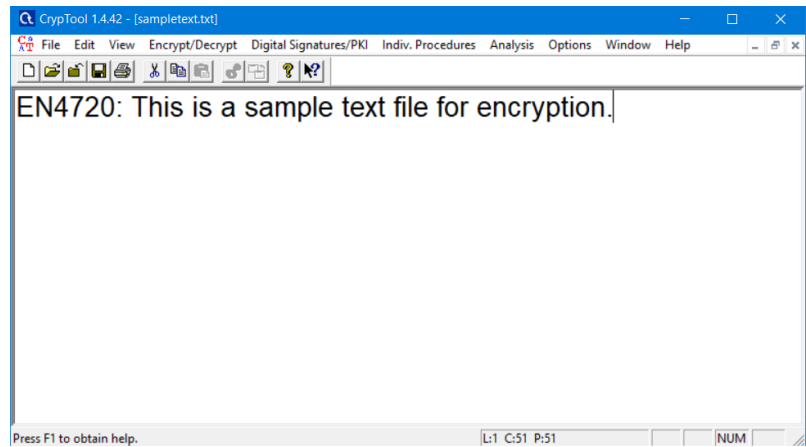
Figure 3: Open the *sampletext.txt* file using the CrypTool software
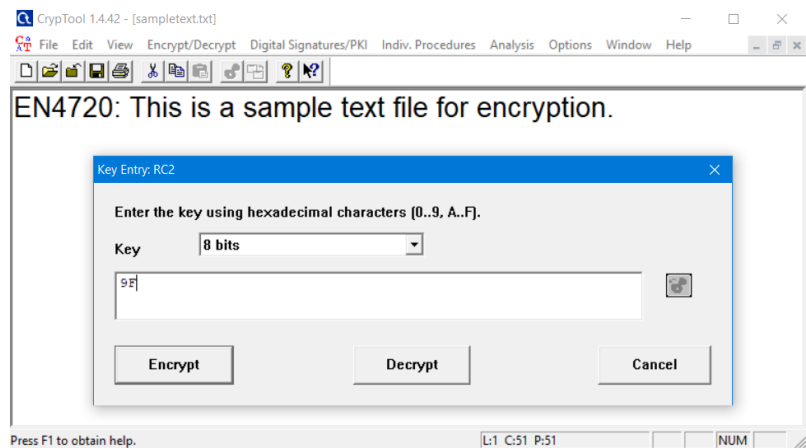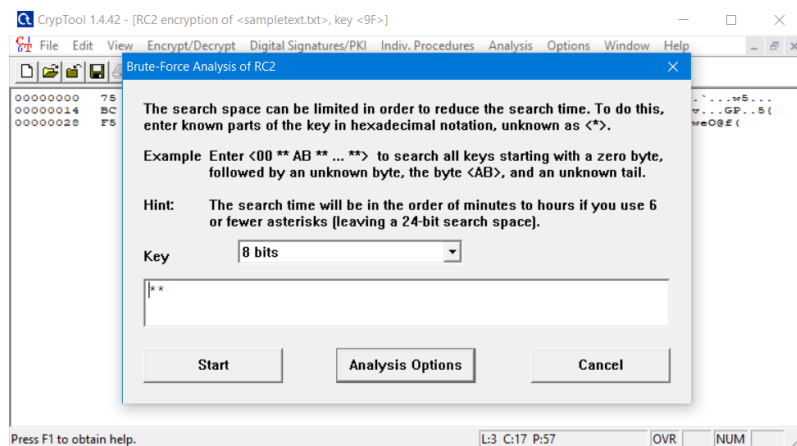

Figure 4: Enter the 8 bit Hexadecimal key for encryption
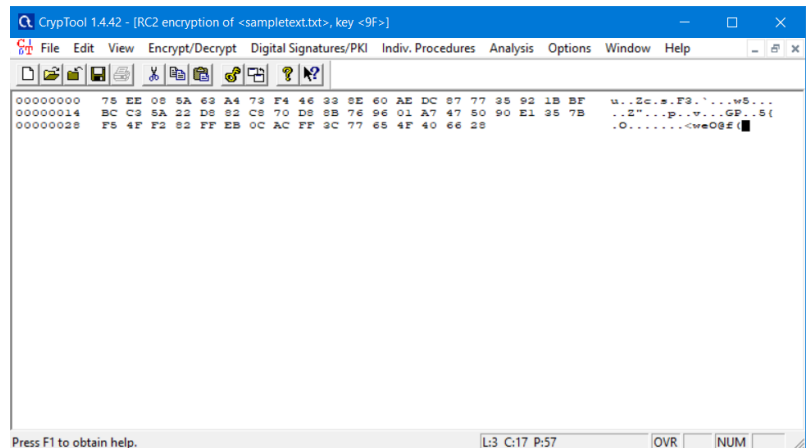
Figure 5: Encrypted text using the selected key

2. Add screenshots requested in Step #4.

Figure 6: Configure the Brute Force analysis of RC2

Figure 7: All possible keys obtained through the Brute Force attack

Figure 8: Decrypted message
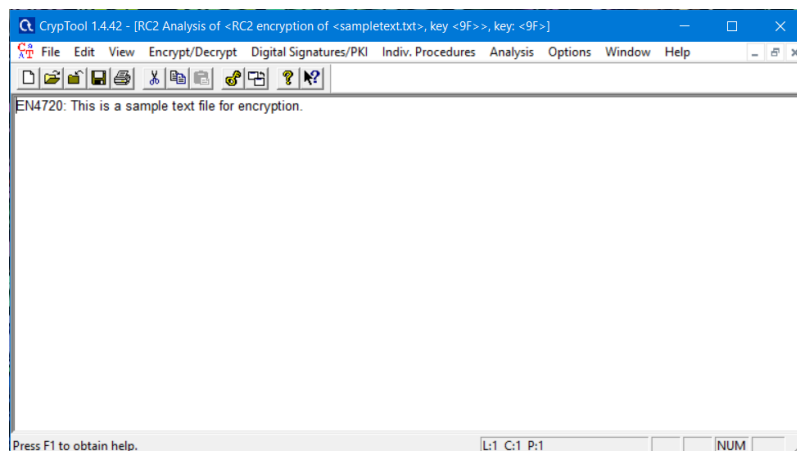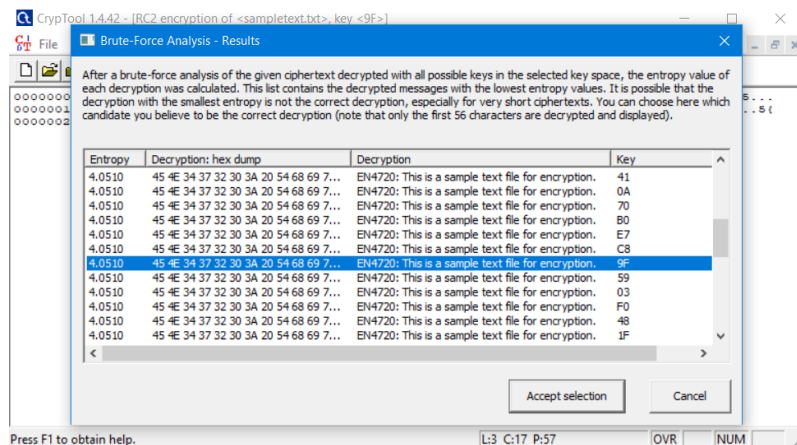
3. Complete Table 1.

Table 1: Time taken to brute force

Key length (bits)	Time to brute force	Maximum number of keys
8	< 1 s	33
16	< 1 s	1
24	1 min 34 s	1
64	3.8×10^6 years	None
128	7.5×10^{25} years	None

4. What is the impact of key length on time to decrypt a message using brute force?

   When the key length is increased, time to decrypt the message grows drastically

5. What could be done to diminish the amount of time required to perform the attack?

   - *Guess some of the characters or enter the known parts in the key and fix them, rather than letting computer to perform every possible combination of the characters. This limits the search space of keys.*
   - *Carry out the decryption in parallel using several computers.*

Problem 3 - Public Key Encryption

You are given that $n = 65$ and $\phi(n) = 48$ for a certain RSA scheme.

1. Calculate the private key d assuming the public key $e = 7$.

   d, takes the form $1 = e * d \mod \phi = 7 * d \mod 48$ and it can be calculated using the Extended Euclidean Algorithm. The steps are given below.

   (a) Euclidean Algorithm

   $$48 = 6 * (7) + 6$$
   $$7 = 1 * (6) + 1$$

   (b) Back Substitution

   $$1 = 7 - 1 * (6)$$
   $$1 = 7 - 1 * (48 - 6 * (7))$$
   $$1 = 7 * (7) - 1 * (48)$$

   (c) The coefficient in front of the $e = 7$, is the required d. Therefore $d = 7$.

2. Show all major steps of encryption and decryption using textbook RSA for the message given below. Specifically, derive the ciphertext and show that the plaintext can be recovered at the receiver after decrypting.
   $m =$ last digit of your index number (e.g., if the index number is 180234N, $m = 4$). If the last digital is 0 (zero) or 1 (one), use $m = 5$.

   (a) Encryption for $m = 5$ since last digit of index number is 1.

   $$ciphertext = m^e \mod n$$
   $$= 5^7 \mod 65$$
   $$= 60 \quad \because (5^7 = 1201 * 65 + 60)$$

   (b) Decryption

   $$plaintext = c^d \mod 65$$
   $$= 60^7 \mod 65$$
   $$= \left\{ (60^2 \mod 65) * (60^2 \mod 65) * (60^2 \mod 65) * (60^1 \mod 65) \right\} \mod 65$$
   $$= \{(25) * (25) * (25) * (60)\} \mod 65$$
   $$= 5 \quad \because (25^3 * 60 = 14423 * 65 + 5)$$

   $\therefore$ the plaintext can be recovered at the receiver after decrypting.

3. Find the candidates for the primes p, q that were used to generate this RSA scheme.

   (a) 48 can be written as, $48 = 12 * 4$
   (b) 65, can be written as, $65 = (12 + 1) * (4 + 1)$
   $\therefore$ the used prime factors to generate n, are 13 and 5.

4. Suppose that c is the ciphertext that you find above for the message m. Show how an attacker can recover the plaintext message m from the ciphertext c assuming that he can request a decryption of a single ciphertext $c' \neq c$.

    (a) *The number $n = 65$ and $e = 7$ is already a public knowledge. In order to decrypt any given message, attacker has to figure out only the private exponent d.*

    (b) *Once the attacker has access to a ciphertext c' which corresponds to some plaintext m', all he has to do is guess some private exponent, decrypt c' using it and check it matches with m'.*

    (c) *Once he found a match, it means he has recovered the private key d. Now he can decrypt any ciphertext which intends for the victim party.*

Problem 4 - Hash Functions
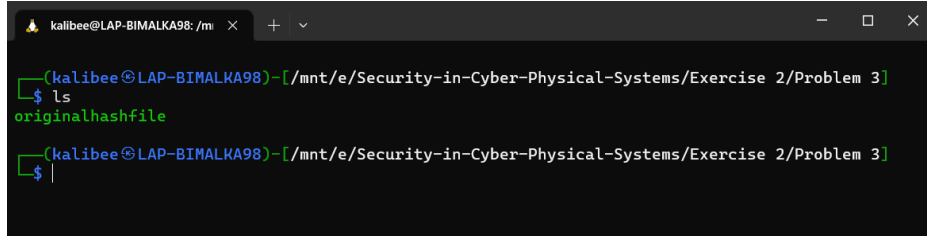
For this part of the exercise you are required to install Kali Linux. Kali Linux comes preinstalled with several hashing utilities. These include **md5sum**, **sha1sum**, **sha256sum**, and **sha512sum**. Follow the given instructions to complete the problem.

Instructions

1. Create a new directory and a new text file "**originalhashfile**" in the directory using **nano originalhashfile** command. Add the text "EN4720: Some text here" in the created file and save the file using the hotkey **CTRL+X** followed by **Y** and **Enter**.

2. Run the *ls* command in the directory to show the file (add screenshot) and run the *cat* command to show the contents of the created file (add screenshot).

3. calculate the hash digest of the **originalhashfile** using these hash algorithms: MD5, SHA1, SHA256, SHA512 (add screenshots). Add your observations in Table 2. You can use the following commands to get the results.

   - md5sum *<hashfile_name>*
   - sha1sum *<hashfile_name>*
   - sha256sum *<hashfile_name>*
   - sha512sum *<hashfile_name>*

4. Change the contents of the **originalhashfile** by adding your index at the end of the line ("EN4720: Some text here *<index_no>*") and save it as "**modifiedhashfile**" in the same directory.

5. Run the *ls* command in the directory to show the files (add screenshot) and run the *cat* command to show the contents of the modified file (add screenshot).

6. Calculate the hash digest of the **modifiedhashfile** using these hash algorithms: MD5, SHA1, SHA256, SHA512 (add screenshots). Add your observations in Table 3.

7. Answer the questions given below.
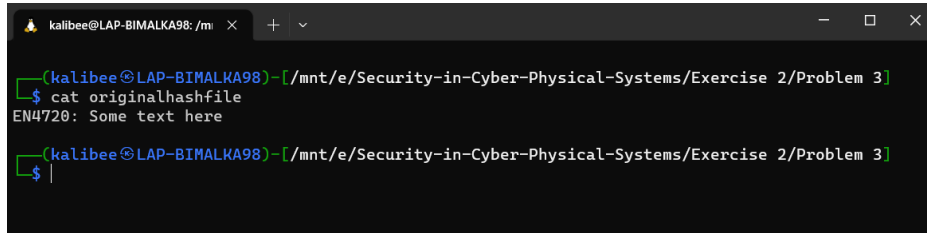
Questions

1. Add screenshots requested in Step #2.

Figure 9: Output of the `ls` command
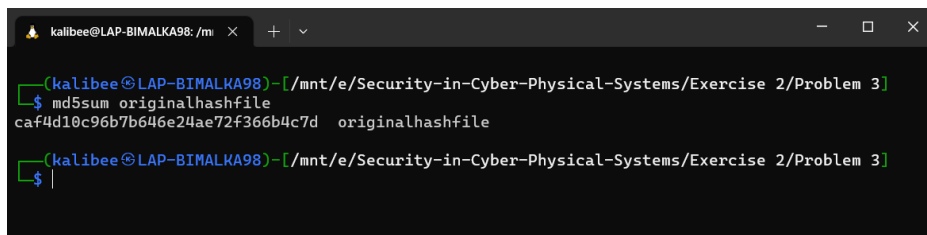
Figure 10: Output of the `cat originalhashfile` command

2. Complete Table 2. Add screenshots requested in Step #3 below the table.

Table 2: Hash digests of the original file

Hash Algorithm	Hash of the Original File
MD5	caf4d10c96b7b646e24ae72f366b4c7d
SHA1	e64870421f78773bf65437f515fd384170ce8839
SHA256	cc544b593a0faa91f34774289a66b7400d1ce3fa36cfaec64a958228c5ead764
SHA512	cc7aebdfdf19a2f9882deb0ff1fc01a5be9f25cb2b916bd179b4ec88436113c5dc 24618487bed98fcc3d11574bafabde3a5c785eed6bccef496a7817d132d73a

Figure 11: Output of the `md5sum originalhashfile` command

Figure 12: Output of the `sha1sum originalhashfile` command

Figure 13: Output of the `sha256sum originalhashfile` command

Figure 14: Output of the `sha512sum originalhashfile` command

3. Among the hash algorithms used, which one is the most cryptographically secure? Justify your answer.

> *sha512sum*. *Security of a hash function depends on three factors: 1. Preimage resistance/ One-waynes, 2. Collision Resistance, and 3. Second Preimage resistance. When it comes to* *sha512sum* *which is a SHA-2 hash, with the modern computing capabilities it is* **hard to attack and break** *the mentioned three properties of it.*

4. Why is MD5 not considered as a reliable hashing algorithm?

> *MD5 is an already broken hashing algorithm which is susceptible to collision attacks and birthday attacks. That is security experts have demonstrated methods, other than naive brute force attacks which can be used to break the hash even with limited computing capabilities.*

5. Which of the algorithms fall under the category of SHA-2?

> *sha256sum* *and* *sha512sum*

6. Add screenshots requested in Step #5.

Figure 15: Output of the `ls` command

Figure 16: Output of the `cat modifiedhashfile` command

7. Complete Table 3. Add screenshots requested in Step #6 below the table.

Table 3: Hash digests of the modified file

Hash Algorithm	Hash of the Modified File
MD5	e335818d74506e6ef35419fea121b102
SHA1	b002d4e2319a4bce10a19779bcd37b6829c4ee1a
SHA256	2621e717d282129cac13b08f3f2c3b45a44af25f56b2b2d762fa553a6e9f54f8
SHA512	0716eb7534e87ee353c2bf1ab16ca3a6747c19c1234426209e11d6c9ce1244e0cb f2a4bc5a8175e8f26b4d457ab2a8f8fdf90568e8726b2d557a3780514a861c

Figure 17: Output of the `md5sum modifiedhashfile` command

Figure 18: Output of the `sha1sum modifiedhashfile` command

13

Figure 19: Output of the `sha256sum modifiedhashfile` command

Figure 20: Output of the `sha512sum modifiedhashfile` command

8. Comparing the output from one algorithm in Table 2 and Table 3, is there a difference between the hash digests? If yes, why?

> *Let's compare the hash digest of* `SHA1` *algorithm for the two files.*
>
> *(a)* `originalhashfile` - `e64870421f78773bf65437f515fd384170ce8839`
> *(b)* `modifiedhashfile` - `b002d4e2319a4bce10a19779bcd37b6829c4ee1a`
>
> *As it can be observed,*
>
> - *In terms of the length there is no difference between the two hash digests digests (a property of the hashing algorithms), even though we have two different input files.*
> - *In terms of the similarity there is a drastic change between the two hash digests (the avalanche effect), even though we made only a simple change to the input file.*