Cryptography Engineering Quiz. 2

Problem 1

1. Import *hashlib* and *time* library.
2. Use *open* and *readlines* to read in ‘password.txt’ , and store them in a list called ‘password_list’ .

```python
import hashlib
import time

# read in password list
password_list = []
with open("password.txt",'r') as file:
    for line in file.readlines():
        password_list.append(line.strip())
```

3. The ‘crack_hash function’ crack a given hash value by comparing it with hashed versions of passwords from the ‘password_list’. The hash algorithm used here is SHA1. The function will return the password, the number of attempts, and the time taken if successfully crack.

```python
# crack hash
def crack_hash(hash, password_list):
    start_time = time.time()
    attempts = 0
    for password in password_list:
        attempts += 1
        # convert string to bit, binary to hex
        hashed_password = hashlib.sha1(password.encode()).hexdigest()
        if hashed_password == hash:
            end_time = time.time()
            time_taken = end_time - start_time
            return password, attempts ,time_taken
    return None, attempts, None
```

4. Finally, call the ‘print_ans’ function to print the hash, password, attempts and time_taken.

```python
def print_ans(hash, password, attempts, time_taken):

    if password :
        print("Hash: "+ hash )
        print("Password: " + password)
        #time_taken_formatted = str(datetime.timedelta(seconds=time_taken))
        time_taken_formatted = "{:02}:{:02}:{:09.6f}".format(int(time_taken // 3600),int((time_taken % 3600) // 60),time
        print(f"Took {attempts} attempts to crack input hash. Time Taken: {time_taken_formatted}\n")
    else:
        print("Failed to crack hash.\n")
```

5. For question (c), I first crack the hashed salt by calling the ‘crack_hash’ function with input salt and ‘password_list’. After knowing the actual string salt, I then call the ‘crack_salthash’ function. To compare each password concatenated with salt with the hash value until find the correct one. The final answer of attempts and taken time is the sum of crack salt and crack_hash.

```python
# for question c. hash + salt
def crack_salthash(hash, password_list, salt):
    start_time = time.time()
    attempts = 0
    for password in password_list:
        attempts += 1
        # convert string to bit, binary to hex
        concatenated_str = salt + password
        hashed_password = hashlib.sha1(concatenated_str.encode()).hexdigest()
        if hashed_password == hash:
            end_time = time.time()
            time_taken = end_time - start_time
            return password, attempts ,time_taken
    return None, attempts, None
```

```python
salt_str, attempts_salt ,time_taken_salt = crack_hash(salt , password_list)
password_3, attempts_3 ,time_taken_3 = crack_salthash(hash3 , password_list, salt_str)
```

```python
print_ans(hash3,password_3, attempts_salt+attempts_3 ,time_taken_3+time_taken_salt)
```

Answer:

a) Easy hash: ef0ebbb77298e1fbd81f756a4efc35b977c93dae

```
Hash: ef0ebbb77298e1fbd81f756a4efc35b977c93dae
Password: orange
Took 124 attempts to crack input hash. Time Taken: 00:00:00.001000
```

b) Medium hash: 0bc2f4f2e1f8944866c2e952a5b59acabd1cebf2

```
Hash: 0bc2f4f2e1f8944866c2e952a5b59acabd1cebf2
Password: starfish
Took 2681 attempts to crack input hash. Time Taken: 00:00:00.002017
```

c) Leet hacker hash: 9d6b628c1f81b4795c0266c0f12123c1e09a7ad3

Hint: The salt term here is: dfc3e4f0b9b5fb047e9be9fb89016f290d2abb06

The original plaint text of the salt is ‘redbull’. And the attempts of only cracking the hacker hash after finding salt is 2854.

```
Hash: 9d6b628c1f81b4795c0266c0f12123c1e09a7ad3
Password: puppy
Took 5639 attempts to crack input hash. Time Taken: 00:00:00.006182
```

```
Hash: 9d6b628c1f81b4795c0266c0f12123c1e09a7ad3
Password: redbullpuppy
Took 2854 attempts to crack input hash. Time Taken: 00:00:00.002019
```

d) Extra Credit: 44ac8049dd677cb5bc0ee2aac622a0f42838b34d

Hint: This hash constitutes two terms separated by one space

1. Using “hashcat”, a password cracking tool to crack this hash. Since it will require a significant amount of time and GPU resources.
2. Preparing two txt.file, one is to reverse the order of the original password.txt, the other is adding a space in front of each of the password, which we call

it reversed_passwords.txt and space_passwords.txt.

3. Then type in the command in cmd:

   hashcat -m 100 -a 1 44ac8049dd677cb5bc0ee2aac622a0f42838b34d reversed_passwords.txt space_passwords.txt

```
C:\hashcat>hashcat -m 100 -a 1 44ac8049dd677cb5bc0ee2aac622a0f42838b34d reversed_passwords.txt space_passwords.txt
hashcat (v6.2.6) starting
```

4. Then hashcat will crack the hashed password for you. The answer should be: z745100 and wujuchawiapra53

```
44ac8049dd677cb5bc0ee2aac622a0f42838b34d:z745100 wujuchawiapra53

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 100 (SHA1)
Hash.Target......: 44ac8049dd677cb5bc0ee2aac622a0f42838b34d
Time.Started.....: Wed Mar 13 21:07:39 2024 (49 secs)
Time.Estimated...: Wed Mar 13 21:08:28 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (reversed_passwords.txt), Left Side
Guess.Mod........: File (space_passwords.txt), Right Side
Speed.#1.........: 60813.1 kH/s (1.14ms) @ Accel:32 Loops:32 Thr:256 Vec:1
Speed.#2.........:   471.6 kH/s (0.55ms) @ Accel:16 Loops:4 Thr:32 Vec:1
Speed.#3.........:  1063.9 MH/s (0.50ms) @ Accel:1024 Loops:1024 Thr:512 Vec:1
Speed.#4.........: 68601.1 kH/s (1.93ms) @ Accel:2048 Loops:1024 Thr:64 Vec:1
Speed.#*.........:  1193.8 MH/s
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 55990101728/999996000004 (5.60%)
Rejected.........: 0/55990101728 (0.00%)
Restore.Point....: 0/999998 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:200320-200352 Iteration:0-32
Restore.Sub.#2...: Salt:0 Amplifier:2500-2504 Iteration:0-4
Restore.Sub.#3...: Salt:0 Amplifier:94208-95232 Iteration:0-1024
Restore.Sub.#4...: Salt:0 Amplifier:24576-25600 Iteration:0-1024
Candidate.Engine.: Device Generator
Candidates.#1....: 623500 zaqxswedf -> beauceron zaqzxc
Candidates.#2....: BEAUDOG Vl13ad03ik1995 -> collyer vl1500
Candidates.#3....: xx01109 wu3g3cp -> 623414 wurstbrot90
Candidates.#4....: vjht008 vxepuwosto -> XX0109 vyoyl
Hardware.Mon.#1..: Temp: 55c Util:  0% Core:1427MHz Mem: 685MHz Bus:8
Hardware.Mon.#2..: N/A
Hardware.Mon.#3..: N/A
Hardware.Mon.#4..: N/A
```

Problem 2

1. Import *hashlib* and *time* library.
2. Use *open* and *read* to read in ‘video.mp4’ in binary mode and store the contents as ‘data’.
3. The ‘calculate_speed’ function calculates the time taken to hash the data using the specified algorithm. Create a new hash object, update the hash with the input data and ‘hexdigest’ is called to compute the digest of the data.

```python
import hashlib
import time

def calculate_speed(data,algo):
    start_time = time.time()
    hash_object = hashlib.new(algo)
    hash_object.update(data)
    hash_object.hexdigest()
    end_time =time.time()
    return end_time - start_time

# open file
with open ("video.mp4",'rb') as f:
    data = f.read()
```

4. Define a list named ‘algorithms’ which contains names of different algorithms, and initialize an empty dictionary called speed which we will store each algorithm’s taken time to hash the video file.

```python
#calculate speed
algorithms = ["md5","sha1","sha224","sha256","sha512","sha3-224","sha3-256","sha3-512"]
speeds = {}

for algo in algorithms:
    speed = calculate_speed(data,algo)
    speeds[algo] = speed
```

5. Finally, use sorted function to sort the printed results by the algorithm’s speed.

```python
#sort
sorted_algorithms = sorted(algorithms, key=lambda x: speeds[x])
for algo in sorted_algorithms:
    speed = speeds[algo]
    print(f"{algo}'s taken time: {speed}")
```

Answer:

a) Write a Python 3 program to compare the speed of the hash algorithms.

```
sha1's taken time: 0.14218950271606445
sha256's taken time: 0.15635204315185547
sha224's taken time: 0.1590721607208252
sha512's taken time: 0.372298002243042
md5's taken time: 0.47606348991394043
sha3-224's taken time: 0.49292612075805664
sha3-256's taken time: 0.5770871639251709
sha3-512's taken time: 1.0448195934295654
```

b) Which one is the fastest?

sha1 is the fastest.

c) Rank the speed of each hash function.

sha1 > sha256 > sha224 > sha512 > md5 > sha3-224 > sha3-256 > sha3-512

Problem 3

Given the transposition cipher, decrypt this ciphertext.

“UONCS VAIHG EPAAH IGIRL BIECS TECSW PNITE TIENO IEEFD OWECX TRSRX STTAR TLODY FSOVN EOECO HENIO DAARQ NAELA FSGNO PTE”

1. First, I try to find the suitable dimension of the rectangle. Using the code below to find out every possible way of dimension and then calculate each average of differences between expected and actual number of vowels.

```
ciphertext = "UONCS VAIHG EPAAH IGIRL BIECS TECSW PNITE TIENO IEEFD OWECX TRSRX STTAR TLODY FSOVN EOECO HENIO DAARQ NAELA

ciphertext= ciphertext.replace(" ","")
total_length = len(ciphertext)

start=0
for i in range(1,total_length+1):
    # different possible way of dimension
    if (total_length % i ==0):
        exp_vowel = (total_length/i)*0.4
        #each row (total: i rows)
        actual_vowel=[0]*i
        #calculate actual num of vowel
        for j in range(total_length):
            if (ciphertext[j]=='A' or ciphertext[j]=='E' or ciphertext[j]=='I' or ciphertext[j]=='O' or ciphertext[j]=='U
                actual_vowel[j%i]+=1
        #sum of each row's diff
        sum_diff = 0
        for k in range(i):
            sum_diff += abs(exp_vowel-actual_vowel[k])
        print(f"For {i} x {total_length/i:.0f} rectangle, the avg of the differences is {sum_diff/i}.")
```

2. This is the printed result:

```
For 1 x 98 rectangle, the avg of the differences is 0.20000000000000284.
For 2 x 49 rectangle, the avg of the differences is 1.5.
For 7 x 14 rectangle, the avg of the differences is 0.6571428571428573.
For 14 x 7 rectangle, the avg of the differences is 0.557142857142857.
For 49 x 2 rectangle, the avg of the differences is 0.5510204081632651.
For 98 x 1 rectangle, the avg of the differences is 0.47959183673469374.
```

Although 1 x 98 and 98 x 1 rectangles have the smallest and second smallest avg. of diff. but it is not a possible way. Therefore, we try to transport the cipher text to a 49 x 2 rectangle, which is the third smallest. But the result seems weird. Finally, we decided to fill the ciphertext in a 14 x 7 rectangle：

5	2	7	6	5	3	4	Num of vowel	diff
U	H	S	E	T	E	Q	3	0.2
O	I	W	F	T	O	N	3	0.2
N	G	P	D	A	E	A	3	0.2
C	I	N	O	R	C	E	3	0.2
S	R	I	W	T	O	L	2	0.8
V	L	T	E	L	H	A	2	0.8
A	B	E	C	O	E	F	4	1.2
I	I	T	X	D	N	S	2	0.8
H	E	I	T	Y	I	G	3	0.2
G	C	E	R	F	O	N	2	0.8
E	S	N	S	S	D	O	2	0.8
P	T	O	R	O	A	P	3	0.2
U	E	I	X	V	A	T	4	1.2
O	C	E	S	N	R	E	3	0.2

3. After filling the cipher text, we try to change the order of each column with the aim of finding the most possible plaintext. According to the hint, the first two letters are ‘TH’, we can find the final result should be like this :

1	2	3	4	5	6	7
T	H	E	Q	U	E	S
T	I	O	N	O	F	W
A	G	E	A	N	D	P
R	I	C	E	C	O	N
T	R	O	L	S	W	I
L	L	H	A	V	E	T
O	B	E	F	A	C	E
D	I	N	S	I	X	T
Y	E	I	G	H	T	I
F	C	O	N	G	R	E
S	S	D	O	E	S	N
O	T	A	P	P	R	O
V	E	A	T	A	X	I
N	C	R	E	A	S	E

Answer:

The plaintext is :

“THE QUESTION OF WAGE AND PRICE CONTROLS WILL HAVE TO BE FACED IN SIXTY EIGHT IF CONGRESS DOES NOT APPROVE A TAX INCREASE.”