111550076 楊璟琛

Problem 1

You are expect to write a Python 3 program that breaks SHA1 hashes in a **brute force** manner. Please use the password list below, and copy them locally for ease of use.
https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/Common-Credentials/10-million-password-list-top-1000000.txt

For each hash value, your program should output the actual clear text **password**, count **the number of tries** before reaching a solution, and time **how long it takes** to break the hash, if found. For example:

```
$ python problem1.py
Hash: db3ae03df555104cd021c6308d5d11cfa40aac41
Password: hotmom
Took 30568 attempts to crack input hash. Time Taken: 0:00:00.073000
... and so on
```

Here are the provided SHA1 hashes you need to break:

a) **Easy hash:** ef0ebbb77298e1fbd81f756a4efc35b977c93dae

b) **Medium hash:** 0bc2f4f2e1f8944866c2e952a5b59acabd1cebf2

c) **Leet hacker hash:** 9d6b628c1f81b4795c0266c0f12123c1e09a7ad3
Hint: The salt term here is: dfc3e4f0b9b5fb047e9be9fb89016f290d2abb06
This is concatenated before hashing with another word to produce the salted hash.

```python
import hashlib
import time

def find_matching_word(file_path, target_hash):
    attempt_count = 0
    start_time = time.time()

    with open(file_path, 'r') as file:
        for line in file:
            words = line.split()
            for word in words:
                attempt_count += 1
                encoded_word = hashlib.sha1(word.encode()).hexdigest()
                if encoded_word == target_hash:
                    return word, attempt_count, time.time() - start_time
    return None, attempt_count, time.time() - start_time

def find_matching_word_c(file_path, target_hash):
    attempt_count = 0
    start_time = time.time()

    with open(file_path, 'r') as file:
        for line in file:
            words = line.split()
            for word in words:
                word = 'redbull' + word
                attempt_count += 1
                encoded_word = hashlib.sha1(word.encode()).hexdigest()
                if encoded_word == target_hash:
                    return word, attempt_count, time.time() - start_time
    return None, attempt_count, time.time() - start_time

file_path = 'data.txt'
target_hash = '9d6b628c1f81b4795c0266c0f12123c1e09a7ad3'

matching_word, attempts, duration = find_matching_word_c(file_path, target_hash)

print(f"Hash: {target_hash}")
print(f"Password: {matching_word}")
print(f"Took {attempts} attempts to crack input hash. Time Taken: {duration:.10f} seconds")
```

直接一條條用 hashlib 的 sha1 執行比對

(a)

```
Hash: ef0ebbb77298e1fbd81f756a4efc35b977c93dae
Password: orange
Took 124 attempts to crack input hash. Time Taken: 0.0000000000 seconds
```

(b) 同上

```
Hash: 0bc2f4f2e1f8944866c2e952a5b59acabd1cebf2
Password: starfish
Took 2681 attempts to crack input hash. Time Taken: 0.0000000000 seconds
```

(c)

find the salt is redbull,

```
Hash: dfc3e4f0b9b5fb047e9be9fb89016f290d2abb06
Password: redbull
Took 2785 attempts to crack input hash. Time Taken: 0.0091781616 seconds
```

在所有 test 前面加 redbull，找到

```
Hash: 9d6b628c1f81b4795c0266c0f12123c1e09a7ad3
Password: redbullpuppy
Took 2854 attempts to crack input hash. Time Taken: 0.0000000000 seconds
```

Problem 2

Checksums are crucial for ensuring data integrity in digital communications and storage. By generating a small, fixed-size data snippet or "hash" from a block of digital data using specific algorithms, checksums allow the verification of the integrity without requiring the original data.

You need to download this video file: https://commondatastorage.googleapis.com/gtv-videos-bucket/sample/BigBuckBunny.mp4

Please calculate the checksums of the downloaded video file by using various hash functions, including MD5, SHA1, SHA-2(sha224, sha256 and sha512), and SHA-3(sha3-224, sha3-256 and sha3-512), and answer the following questions.

a) Write a Python 3 program to compare the speed of the hash algorithms.
Hint: You can use hashlib or time library

b) Which one is the fastest?

c) Rank the speed of each hash function.

```python
import hashlib
import time
import requests

def download_file(url):
    local_filename = url.split('/')[-1]
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(local_filename, 'wb') as f:
            for chunk in r.iter_content(chunk_size=8192):
                f.write(chunk)
    return local_filename

def compute_checksum(filename, algorithm):
    hash_func = hashlib.new(algorithm)
    with open(filename, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""):
            hash_func.update(chunk)
    return hash_func.hexdigest()

def measure_speed(filename, algorithms):
    times = {}
    for algorithm in algorithms:
        start_time = time.time()
        checksum = compute_checksum(filename, algorithm)
        times[algorithm] = (time.time() - start_time, checksum)
    return times

url = "https://commondatastorage.googleapis.com/gtv-videos-bucket/sample/BigBuckBunny.mp4"
filename = download_file(url)

algorithms = ['md5', 'sha1', 'sha224', 'sha256', 'sha512', 'sha3_224', 'sha3_256', 'sha3_512']

times = measure_speed(filename, algorithms)

for algorithm in sorted(times, key=times.get):
    time_taken, checksum = times[algorithm]
    print(f"{algorithm} took {time_taken:.6f} seconds - Checksum: {checksum}")

fastest = min(times, key=lambda k: times[k][0])
print(f"The fastest hash algorithm is {fastest}.")

print("Ranking the hash algorithms by speed:")
for rank, algorithm in enumerate(sorted(times, key=times.get), start=1):
    print(f"{rank}. {algorithm} ({times[algorithm][0]:.6f} seconds)")
```

要下載 requests

指令：

pip install requests

version: 2.31.0

用 hashlib 套用所有 function，比較計算時間

(a)

```
sha1 took 0.180321 seconds - Checksum: b29ae9b33d33304b3b966f2921cc5bfb3cb3
c3ce
sha224 took 0.187838 seconds - Checksum: 2dd11ca85546f0bf1029299f5d38383ab0
f0942b61ae1b92b5a384be
sha256 took 0.189437 seconds - Checksum: 1cadc5e09cbb81044e256f9fc67090fcf8
6d7a596145eb615844fe15341451e6
sha512 took 0.314085 seconds - Checksum: e6eaef73af4b739daf7e8874e1f3b87b4d
320f954347e912c6cbb33f686c428b94832c46f7928e9cf685e14452f5a0e3209edae501ac2
22fa6eaae7dbbb7488a
md5 took 0.332803 seconds - Checksum: cab08b36195edb1a1231d2d09fa450e0
sha3_224 took 0.416202 seconds - Checksum: 26c55e271dc576d3db2653dc952ab530
3cc521ff788acd63a9f16716
sha3_256 took 0.425209 seconds - Checksum: 02db744889e01a17accabbb69a0eca49
a39058ed560d673170c631f096bef1be
sha3_512 took 0.721913 seconds - Checksum: 58d0bc115ddaa7a8a03245b054be6e9b
59d338508d00313b486b81430f51514c1ca5b3d569093ea795e0d97c2c17861925af55250ff
f5a4a2250b5897d381dba
The fastest hash algorithm is sha1.
Ranking the hash algorithms by speed:
1. sha1 (0.180321 seconds)
2. sha224 (0.187838 seconds)
3. sha256 (0.189437 seconds)
4. sha512 (0.314085 seconds)
5. md5 (0.332803 seconds)
6. sha3_224 (0.416202 seconds)
7. sha3_256 (0.425209 seconds)
8. sha3_512 (0.721913 seconds)
```

(c)

#(b)

Problem 3

Given the transposition cipher:

UONCS VAIHG EPAAH IGIRL BIECS

TECSW PNITE TIENO IEEFD OWECX

TRSRX STTAR TLODY FSOVN EOECO

HENIO DAARQ NAELA FSGNO PTE

Please decrypt this ciphertext.

```
UHSETEQ
OIWFTON
NGPDAEA
CINORCE
SRIWTOL
VLTELHA
ABECOEF
IITXDNS
HEITYIG
GCERFON
ESNSSDO
PTOROAP
AEIXVAT
ACESNRE
```

arrange it to 7 x 14, and use python code to find average vowel 比例.

有做，但內容相似，不再贅述

```
3, Difference: 0.20
3, Difference: 0.20
3, Difference: 0.20
3, Difference: 0.20
2, Difference: 0.80
2, Difference: 0.80
4, Difference: 1.20
2, Difference: 0.80
3, Difference: 0.20
2, Difference: 0.80
2, Difference: 0.80
3, Difference: 0.20
4, Difference: 1.20
3, Difference: 0.20
0.3979591836734694
0.557142857142857
```

difference 的平均和14×7的結果比較小，合理

和理想值(0.4)差不多，7×14合理，接著用人工智慧思考，助教題示TH開頭，猜THE，裡面有Q跟U，猜會連

在一起，最终找出顺序

```
THEQUES
TIONOFW
AGEANDP
RICECON
TROLSWI
LLHAVET
OBEFACE
DINSIXT
YEIGHTI
FCONGRE
SSDOESN
OTAPPRO
VEATAXI
NCREASE
```

"The question of wage and price control will have to be faced in sixty-eight if Congress does not approve a tax increase."

```
text_combined = "UONCSVAIHGEPAAHIGIRLBIECSTECSWPNITETIENOIEEFDOWECXTRSRXSTTARTLODYFSOVNEOECOHENIODAARC
columns_new = 14

transposed_text_new = [''.join(text_combined[i::columns_new]) for i in range(columns_new)]
formatted_text_new = '\n'.join(transposed_text_new)
print(formatted_text_new)

vowels = "AEIOU"
ideal_proportion = 0.28
proportions_diff = []
total_proportion = 0
for col in transposed_text_new:
    vowel_count = sum(1 for char in col if char.upper() in vowels)
    proportion = vowel_count / len(col)
    diff = abs(proportion - ideal_proportion)
    proportions_diff.append((proportion, diff))
    total_proportion += proportion

for item in proportions_diff:
    print(f"Proportion: {item[0]}, Difference: {item[1]}")

average_proportion = total_proportion / columns_new
print(average_proportion)

new_order_corrected = [5, 2, 6, 7, 1, 4, 3]
reordered_corrected = [''.join(line[i - 1] for i in new_order_corrected) for line in transposed_text_r
reordered_corrected_formatted = '\n'.join(reordered_corrected)
print(reordered_corrected_formatted)
```