Answers

#1

ThaiLanD

#2

Time taken: 4.04618501663208 seconds

Table size: 2650956

#3

Time taken for a single SHA-1 hash: 8.106231689453125e-06 seconds

#4

Let x be the length of the password and the password contains only letters and digits (a-z, A-Z, 0-9)

totalCombinations = (26 + 26 + 10)^x = 62^x

timeToGenerate = totalCombinations * timeTaken

timeToGenerate = 62^x * 8.106231689453125e-06 seconds

#5

Time to generate all possible combinations for a 8-character password: 20485.13290475492 days / 56.12365179384909 years

The proper length of the password is 8 (need more than 1 year to break)

#6 Salt explanation: Salt is a random string added to the password before hashing. It protects against precomputed hash attacks (rainbow tables).

Code

```python
import hashlib
import time

# Generate all possible combinations of a password
def generate_password_combinations(password):
   passwords = []

   # Helper function to generate all possible combinations of a password
   def helper(passwords, password, index=0, current_combination=''):
       if index == len(password):
           passwords.append(current_combination)
           return

       char = password[index]
       helper(passwords, password, index + 1, current_combination + char.lower())
       helper(passwords, password, index + 1, current_combination + char.upper())

       char_to_number = {'o': '0', 'i': '1', 'l': '1'}
       if char.lower() in char_to_number:
           helper(passwords, password, index + 1, current_combination +
char_to_number[char.lower()])

   helper(passwords, password)
   return passwords

# Decode the password using the 10k most common passwords
def decode_password(passwordHash, passwords):
   for p in passwords:
       combinations = generate_password_combinations(p.strip())
       for c in combinations:
           if hashlib.sha1(c.encode()).hexdigest() == passwordHash:
               return c
   return ''

# Generate a rainbow table using the 10k most common passwords
def generateRainbowTable(passwords):
   rainbowTable = {}
   for p in passwords:
       combinations = generate_password_combinations(p.strip())
       for c in combinations:
           rainbowTable[c] = hashlib.sha1(c.encode()).hexdigest()
   return rainbowTable
```

```python
# Read the 10k most common passwords
try:
   with open('/Users/peeralis/Documents/Computer
Security/Activity01/10k-most-common.txt', 'r') as passwordFile:
       passwordList = passwordFile.readlines()
except FileNotFoundError:
   print("Password file not found.")
   exit(1)

#1
print('#1\n', decode_password('d54cc1fe76f5186380a0939d2fc1723c44e8a5f7',
passwordList), end='\n\n')

#2
start = time.time()
rainbowTable = generateRainbowTable(passwordList)
end = time.time()
print('#2\nTime taken:', end - start, 'seconds')
print('Table size:', len(rainbowTable), end='\n\n')
# print(tabulate(rainbowTable.items(), headers=['Password', 'Hash'],
tablefmt='grid'))

#3
password = "password"
startTime = time.time()
hashValue = hashlib.sha1(password.encode()).hexdigest()
endTime = time.time()

timeTaken = endTime - startTime
print("#3\nTime taken for a single SHA-1 hash:", timeTaken, end=' seconds\n\n')

#4 assume that the password is x characters long and contains only letters and
digits
print('#4\nLet x be the length of the password and the password contains only
letters and digits (a-z, A-Z, 0-9)')
print('totalCombinations = 62^x')
print('timeToGenerate = totalCombinations * timeTaken')
print('timeToGenerate = 62^x *', timeTaken, 'seconds\n')

#5
passwordLength = 1
while True:
   totalCombinations = 62**passwordLength
   timeToGenerate = totalCombinations * timeTaken
   # if time to generate is more than 1 year, break
```

```
    if timeToGenerate > 365*24*3600:
        print(f'#5\nTime to generate all possible combinations for a
{passwordLength}-character password: {timeToGenerate/3600/24} days /
({timeToGenerate/3600/24/365} years)')
        print(f'The proper length of the password is {passwordLength} (need more
than 1 year to break)\n')
        break
    passwordLength += 1

# 6. Explanation of salt
print("#6 Salt explanation: Salt is a random string added to the password before
hashing. It protects against precomputed hash attacks (rainbow tables).")
```