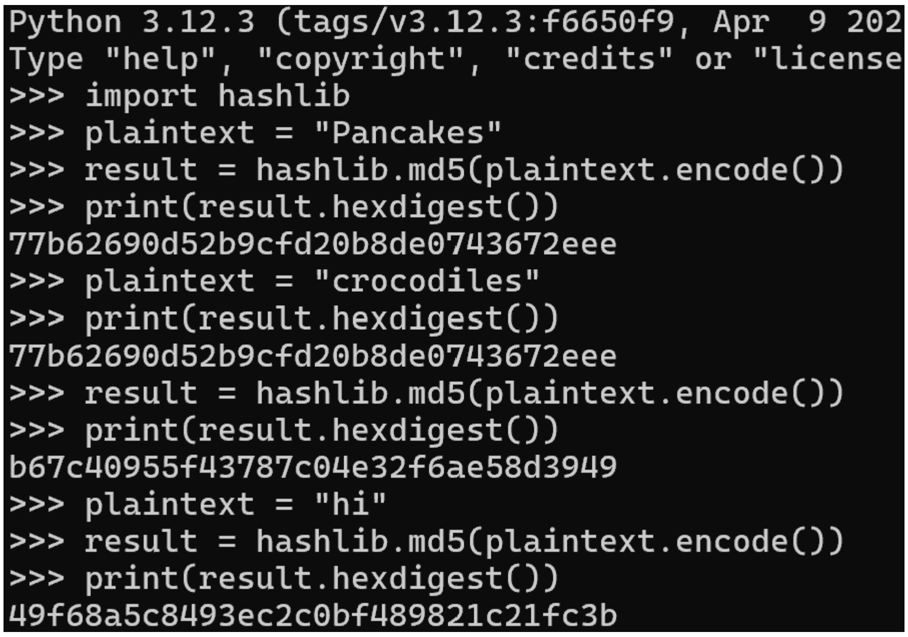
LAB 3 REPORT

Contents

Part I:



```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 202
Type "help", "copyright", "credits" or "license
>>> import hashlib
>>> plaintext = "Pancakes"
>>> result = hashlib.md5(plaintext.encode())
>>> print(result.hexdigest())
77b62690d52b9cfd20b8de0743672eee
>>> plaintext = "crocodiles"
>>> print(result.hexdigest())
77b62690d52b9cfd20b8de0743672eee
>>> result = hashlib.md5(plaintext.encode())
>>> print(result.hexdigest())
b67c40955f43787c04e32f6ae58d3949
>>> plaintext = "hi"
>>> result = hashlib.md5(plaintext.encode())
>>> print(result.hexdigest())
49f68a5c8493ec2c0bf489821c21fc3b
```

All the lengths of the hash are 32 characters longs, or 128 bits longs, regardless of the input string

There is no visible correlation between the hash and the input string. A small change in the plain text results in a completely different hash.

Cryptographic weakness of MD5 :

- Since MD5 hashing is deterministic (A plaintext produces the same hash every single time), it is vulnerable to collision attacks, where hackers may find 2 different plaintext that hash to the same value
- It is also vulnerable to preimage attacks, where a hacker can find a plaintext that hashes to a target

Part II

```
Hash ddaafa5d551a582bc924d09cc8d33ee5 matched password: aseas
Hash 96f6065d8f2dd1376eff88fba65d1d83 matched password: cance
Hash 836626589007d7dd5304c8d22815fffc matched password: di5gv
Hash a74edf83748e3c4fa5f31ec10bad79db matched password: dsmto
Hash 1b31905c59f481958d2eb72158c27ac7 matched password: egunb
Hash a8218c67a5b4e652e30a59372e07df59 matched password: hed4e
Hash 81466b6bb4be5a48e2230be1338bcde6 matched password: lou0g
Hash 6e313b70d12de950443527a33d802b76 matched password: mlhdi
Hash 78c1b8edd1bc3ffc438432479289a9e1 matched password: nized
Hash de952f5454fb0ee79bca249f80e9fe8f matched password: ofror
Hash a92b66a9802704ca8616c4b092378272 matched password: opmen
Hash 644674d142ba2174a80889f833b32563 matched password: owso9
Hash 1b4baba3ae3be69857b323cf6b7fcd80 matched password: sso55
Hash 0d5b558d5f6744deaaf5b016c6c77a57 matched password: tpoin
Hash d4efdba5e9725e77c9b9051fa8136f0a matched password: tthel
total time: 50.343761920928955
```

Total time taken = 50.3s

On average it took 3.4 seconds to crack each string

It is possible to amortize the brute forcing attempts by first sorting the passwords alphanumerically

Part III:

Modifications made between ex2.py to ex3.py:

- Changed length to brute force from 5 to 6

It took 8939 seconds to crack the salted plaintexts, compared to 50.3 seconds for the unsalted plaintext.

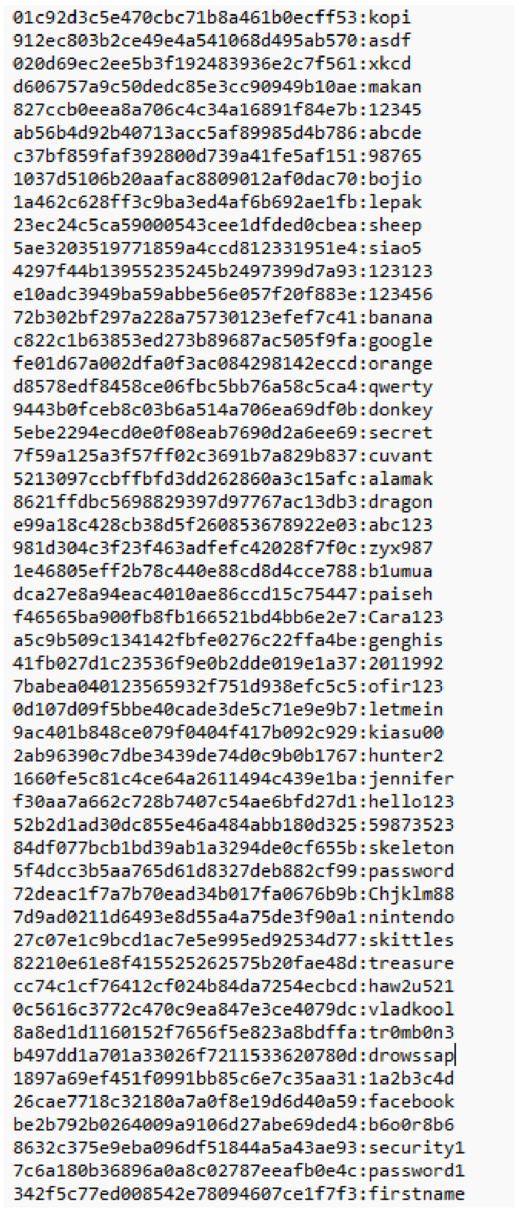
Part IV:

I managed to crack 106 hashes out of 148

-Approaches used to crack the hashes:

Brute force using tools:

To maximise the efficiency of brute forcing, I used a password cracker called hashcat to brute force plaintext that had less than 10 characters. Hashcat utilises GPU processing power, which made brute-forcing plaintexts up to 9 characters a quick task, it took around 7 minutes for hashcat to crack these 52 hashes:



```
01c92d3c5e470cbc71b8a461b0ecff53:kopi
912ec803b2ce49e4a541068d495ab570:asdf
020d69ec2ee5b3f192483936e2c7f561:xkcd
d606757a9c50dedc85e3cc90949b10ae:makan
827ccb0eea8a706c4c34a16891f84e7b:12345
ab56b4d92b40713acc5af89985d4b786:abcde
c37bf859faf392800d739a41fe5af151:98765
1037d5106b20aafac8809012af0dac70:bojio
1a462c628ff3c9ba3ed4af6b692ae1fb:lepak
23ec24c5ca59000543cee1dfded0cbea:sheep
5ae3203519771859a4ccd812331951e4:siao5
4297f44b13955235245b2497399d7a93:123123
e10adc3949ba59abbe56e057f20f883e:123456
72b302bf297a228a75730123efef7c41:banana
c822c1b63853ed273b89687ac505f9fa:google
fe01d67a002dfa0f3ac084298142eccd:orange
d8578edf8458ce06fbc5bb76a58c5ca4:qwerty
9443b0fceb8c03b6a514a706ea69df0b:donkey
5ebe2294ecd0e0f08eab7690d2a6ee69:secret
7f59a125a3f57ff02c3691b7a829b837:cuvant
5213097ccbffbfd3dd262860a3c15afc:alamak
8621ffdbc5698829397d97767ac13db3:dragon
e99a18c428cb38d5f260853678922e03:abc123
981d304c3f23f463adfefc42028f7f0c:zyx987
1e46805eff2b78c440e88cd8d4cce788:b1umua
dca27e8a94eac4010ae86ccd15c75447:paiseh
f46565ba900fb8fb166521bd4bb6e2e7:Cara123
a5c9b509c134142fbfe0276c22ffa4be:genghis
41fb027d1c23536f9e0b2dde019e1a37:2011992
7babea040123565932f751d938efc5c5:ofir123
0d107d09f5bbe40cade3de5c71e9e9b7:letmein
9ac401b848ce079f0404f417b092c929:kiasu00
2ab96390c7dbe3439de74d0c9b0b1767:hunter2
1660fe5c81c4ce64a2611494c439e1ba:jennifer
f30aa7a662c728b7407c54ae6bfd27d1:hello123
52b2d1ad30dc855e46a484abb180d325:59873523
84df077bcb1bd39ab1a3294de0cf655b:skeleton
5f4dcc3b5aa765d61d8327deb882cf99:password
72deac1f7a7b70ead34b017fa0676b9b:Chjklm88
7d9ad0211d6493e8d55a4a75de3f90a1:nintendo
27c07e1c9bcd1ac7e5e995ed92534d77:skittles
82210e61e8f415525262575b20fae48d:treasure
cc74c1cf76412cf024b84da7254ecbcd:haw2u521
0c5616c3772c470c9ea847e3ce4079dc:vladkool
8a8ed1d1160152f7656f5e823a8bdffa:tr0mb0n3
b497dd1a701a33026f7211533620780d:drowssap
1897a69ef451f0991bb85c6e7c35aa31:1a2b3c4d
26cae7718c32180a7a0f8e19d6d40a59:facebook
be2b792b0264009a9106d27abe69ded4:b6o0r8b6
8632c375e9eba096df51844a5a43ae93:security1
7c6a180b36896a0a8c02787eeafb0e4c:password1
342f5c77ed008542e78094607ce1f7f3:firstname
```

Using https://crackstation.net/:

Crackstation is an online service that can be used to crack password hashes, it works by comparing a given hash to a large database of precomputed hash values derived from common passwords.

Using this method, I managed to crack 54 more hashes

Using a dictionary attack:

Finally, using hashcat, I attempted a dictionary attack. This works similarly to Crackstation, where the given hash is compared to a dictionary of common passwords.

For the attack, I used publicly available password list rockyou.txt, which contains 14,341,564 unique passwords. This method did not perform well though, as 0 hashes were cracked through this method

Main challenges and limitation of my approach

<u>Password Complexity and Length:</u>

Challenge: Complex passwords with a mix of uppercase letters, lowercase letters, numbers, and special characters can be difficult to crack.

Limitation: Brute forcing longer passwords exponentially increases the time required. Hashcat can efficiently handle passwords up to a certain length, but beyond that, the time required becomes impractical.

<u>Limited Dictionary Coverage:</u>

Challenge: Dictionaries, including rockyou.txt, are based on common passwords and may not cover less common or highly unique passwords.

Limitation: Even though rockyou.txt contains over 14 million passwords, it may not include the specific password being used, especially if the user has chosen a unique or complex password.

<u>Precomputed Hash Limitations:</u>

Challenge: Precomputed hash databases like CrackStation are only as good as the data they contain.

Limitation: If the password is not in the precomputed database, it will not be cracked. Additionally, precomputed databases typically do not cover all possible passwords, especially if salts are used.