**COMPUTER ARCHITECTURE AND ORGANIZATION**

**DIGITAL ASSIGNMENT-2**

Saahil Ahmad 22BRS1023

Contribution: Architecture Diagram + Code + Output

Shane Boby 22BPS1040

Contribution: Summary + Code + Output

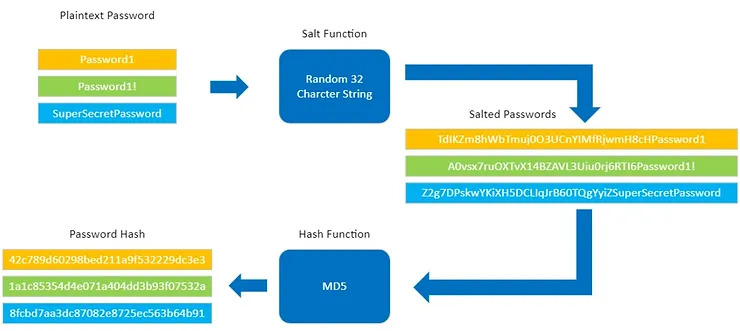
**SUMMARY**

Traditional password cracking methods often consume a lot of time and have low efficiency when dealing with long and complex passwords. This project uses the power of parallel processing to greatly enhance the speed and efficiency of password recovery, making it a very useful tool.

The system used in our project uses Python’s multiprocessing capabilities to distribute the password cracking load among multiple cores. This offers a valuable contribution to the field of cybersecurity by showcasing the potential of parallel processing for password recovery. Its ethical application can assist individuals and organizations in identifying and mitigating password vulnerabilities, ultimately strengthening digital security in an increasingly connected world.

The code is a Python script for performing password cracking. Two functions are defined: check\_password and parallel\_password\_cracker. The check\_password takes a password and a target hash as input, hashes the password using the MD5 algorithm, and returns the password if the hash matches the target hash. The parallel\_password\_cracker function takes a target hash and a list of passwords as input, creates a multiprocessing pool with the number of processes equal to the number of CPU cores, and applies the check\_password function to each password in the list in parallel. If a password is found that matches the target hash, it is returned. Otherwise, none is returned. The script also includes an example usage of the parallel\_password\_cracker function, which attempts to crack an example MD5 hash for the password “password” using a list of example passwords.

**ARCHITECTURE DIAGRAM**



**WITH MULTIPROCESSING**

import hashlib

import multiprocessing

import time

# Function to check if a given password matches the target hash

def check\_password(password, target\_hash):

# Change the hashing algorithm as needed

hashed\_password = hashlib.md5(password.encode()).hexdigest()

if hashed\_password == target\_hash:

return password

return None

# Function to perform parallel password cracking

def parallel\_password\_cracker(target\_hash, password\_list):

# Use all available CPU cores

pool = multiprocessing.Pool(processes=multiprocessing.cpu\_count())

start\_time = time.time()

result = pool.starmap(

check\_password, [(password, target\_hash) for password in password\_list])

pool.close()

pool.join()

end\_time = time.time()

# Filter out None values (passwords that didn't match)

valid\_passwords = list(filter(None, result))

if valid\_passwords:

return valid\_passwords[0], end\_time - start\_time # Return the first valid password found and the execution time

else:

return None, end\_time - start\_time

if \_name\_ == "\_main\_":

# Example MD5 hash for "password"

target\_hash = "68b2789181096f58c5b6eb2303870742"

# Example password list (you should use a real password dictionary)

password\_list = ["shaneboby233", "2022", "shane2022", "shaneboby", "asdasda", "afegfag", "tg3u2hrui23",

"2yedfhqbw fj", "ru3h2iuehfiuw", "dfgavudw", "fwgugfye", "fhqwyrh923wdb",'mumble', 'plump', 'helium', 'esoteric', 'glisten', 'tarantula', 'jester', 'chisel',

'flourish', 'scorch', 'hectic', 'gloomy', 'brittle', 'scuttle', 'juggle', 'barrel',

'wrangle', 'fossil', 'thunder', 'twirl', 'bumble', 'finicky', 'cautious', 'atlas',

'scowl', 'awkward', 'paddle', 'benign', 'virtue', 'halcyon', 'spruce', 'orbit',

'jaunt', 'grimace', 'opulent', 'amble', 'guffaw', 'squawk', 'muddle', 'lavish',

'meandering', 'eulogy', 'forage', 'fickle', 'dreary', 'flimsy', 'bellow', 'satire',

'acrobat', 'persimmon', 'hazy', 'blunder', 'flounder', 'snooze', 'wobble', 'infamous',

'tangent', 'gamut', 'jaded', 'staunch', 'deplete', 'exodus', 'enthuse', 'waltz',

'epicure', 'ponder', 'cringe', 'aquamarine', 'doozy', 'agape', 'piety', 'vicarious',

'hobnob', 'lunge', 'deluge', 'gawky', 'parlay', 'phlegmatic', 'parsimony', 'impeach',

'convalesce', 'maelstrom', 'syllogism', 'malevolent', 'blase', 'luminary', 'histrionic',

'famished', 'exegesis', 'quelled', 'prodigious', 'obsequious', 'overture', 'verdant',

'halting', 'abhorrent', 'edifying', 'profane', 'verbiage', 'lackadaisical', "shaneboby2022"]

result, execution\_time = parallel\_password\_cracker(target\_hash, password\_list)

if result:

print(f"Password found: {result}. \nExecution time: {execution\_time} seconds.\n")

else:

print(f"Password not found in the given list.\n Execution time: {execution\_time} seconds.\n")

**WITHOUT MULTIPROCESSING**

import hashlib

import time

# Function to check if a given password matches the target hash

def check\_password(password, target\_hash):

# Change the hashing algorithm as needed

hashed\_password = hashlib.md5(password.encode()).hexdigest()

if hashed\_password == target\_hash:

return password

return None

# Function to perform sequential password cracking

def sequential\_password\_cracker(target\_hash, password\_list):

start\_time = time.time()

for password in password\_list:

result = check\_password(password, target\_hash)

if result:

end\_time = time.time()

return result, end\_time - start\_time

end\_time = time.time()

return None, end\_time - start\_time

if \_name\_ == "\_main\_":

# Example MD5 hash for "password"

target\_hash = "68b2789181096f58c5b6eb2303870742"

# Example password list (you should use a real password dictionary)

password\_list = ["shaneboby233", "2022", "shane2022", "shaneboby", "asdasda", "afegfag", "tg3u2hrui23",

"2yedfhqbw fj", "ru3h2iuehfiuw", "dfgavudw", "fwgugfye", "fhqwyrh923wdb",'mumble', 'plump', 'helium', 'esoteric', 'glisten', 'tarantula', 'jester', 'chisel',

'flourish', 'scorch', 'hectic', 'gloomy', 'brittle', 'scuttle', 'juggle', 'barrel',

'wrangle', 'fossil', 'thunder', 'twirl', 'bumble', 'finicky', 'cautious', 'atlas',

'scowl', 'awkward', 'paddle', 'benign', 'virtue', 'halcyon', 'spruce', 'orbit',

'jaunt', 'grimace', 'opulent', 'amble', 'guffaw', 'squawk', 'muddle', 'lavish',

'meandering', 'eulogy', 'forage', 'fickle', 'dreary', 'flimsy', 'bellow', 'satire',

'acrobat', 'persimmon', 'hazy', 'blunder', 'flounder', 'snooze', 'wobble', 'infamous',

'tangent', 'gamut', 'jaded', 'staunch', 'deplete', 'exodus', 'enthuse', 'waltz',

'epicure', 'ponder', 'cringe', 'aquamarine', 'doozy', 'agape', 'piety', 'vicarious',

'hobnob', 'lunge', 'deluge', 'gawky', 'parlay', 'phlegmatic', 'parsimony', 'impeach',

'convalesce', 'maelstrom', 'syllogism', 'malevolent', 'blase', 'luminary', 'histrionic',

'famished', 'exegesis', 'quelled', 'prodigious', 'obsequious', 'overture', 'verdant',

'halting', 'abhorrent', 'edifying', 'profane', 'verbiage', 'lackadaisical',"shaneboby2022"]

result, execution\_time = sequential\_password\_cracker(target\_hash, password\_list)

if result:

print(f"Password found: {result}. \nExecution time: {execution\_time} seconds.\n")

else:

print(f"Password not found in the given list.\n Execution time: {execution\_time} seconds.\n")

\

****