CSE3502 - Information Security Management

School of Computer Science & Engineering

J COMPONENT PROJECT REPORT

Submitted to: Professor Sendhil Kumar K.S

P. Mridula Menon 18BCB0125

Pranav Kapoor 18BCE2184

Manan Bhand 18BCE0915

Vartika Trivedi 18BCE0962

DECLARATION

We hereby declare that the project entitled “Password cracking using Dictionary Attacks" submitted by Pranav Kapoor, Mridula Menon, Manan Bhand and Vartika Trivedi for the award of the degree of Bachelor of Technology in Computer Science to VIT is a record of bonafide work carried out by us under the supervision of Prof.Sendhil Kumar K.S.

We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

- **Pranav Kapoor**
- **Mridula Menon**
- **Manan Bhand**
- **Vartika Trivedi**

ABSTRACT:

In today’s day and age, databases are of prime importance, so their security and protection is also the foremost agenda for the companies managing and maintaining them.

Sadly, a variety of attacks exist which can hamper the normal functionality of the relational databases. In this project we take into consideration the
‘DICTIONARY ATTACKS’.

Dictionary Attacks are a method of using a program to try a list of words on the interface or program that is protecting the area that you want to gain access to.

A weakness of dictionary attacks is that it obviously relies on words supplied by a user, typically real words, to function. Examples of programs that use dictionary attacks: John the Ripper, L0phtCrack, and Cain and Abel.

Specifically considering ‘JOHN THE RIPPER’ attack it can be prevented by using a method called ‘SALTING’.

A salt is random data that is used as an additional input to a one-way function that "hashes" a password or passphrase. Salts are closely related to the concept of the nonce.

The primary function of salts is to defend against dictionary attacks or against its hashed equivalent, a pre-computed rainbow table attack.
For detecting this attack, we can see the average time is taken by the attack to crack a password, depending on the result we can find whether the ‘John The Ripper’ uses ‘Dictionary Attacks’ or the general brute force method.

KEYWORDS – password cracking; password security; brute force attack; dictionary attack; hashed passwords

1. INTRODUCTION

1.1 Theoretical Background

To understand how to protect yourself from a password attack, you should become familiar with the most commonly used types of attacks.

With that information, you can use password cracking tools and techniques to regularly audit your own organization's passwords and determine whether your defences need bolstering.

The most common type of attack is password guessing. Attackers can guess passwords locally or remotely using either a manual or automated approach.

Password guessing isn't always as difficult as you'd expect. Most networks aren't configured to require long and complex passwords, and an attacker needs to find only one weak password to gain access to a network.

1.2 Motivation

A dictionary attack is a technique or method used to breach the computer security of a password-protected machine or server.

A dictionary attack attempts to defeat an authentication mechanism by systematically entering each word in a dictionary as a password or trying to determine the decryption key of an encrypted message or document.

Dictionary attacks are often successful because many users and businesses use ordinary words as passwords.

This attack method can also be employed as a means to find the key needed to decrypt encrypted files.

1.3 Aim

The aim of this study is to crack as many passwords as possible, in order to demonstrate just how predictable and weak they really are. We were interested in the differences between the used cracking techniques, their success and time consumption. Finally, we performed the analysis and compared cracked and uncracked password characteristics using different hashing algorithms.

1.4 Objective(s) of the proposed work

The objective of this project is to attack various passwords both salted and non-salted , and check all the passwords using 4 different algorithms, to check the strength of password and to be able to use new methods to make passwords so that it is protected from dictionary attacks.

2. Literature review

2.1. Survey of the Existing Models/Work

Traditional hashing techniques employ one powerful and unbreakable hashing algorithm which can protect the hashing algorithm's properties pre-image resistance and collision resistance.

However, some existing solutions do a hashing a file twice by adding an appropriate salt and pepper to it will enhance its security and make it more collision resistant and more pre-image resistant. SHA-256 eliminates the collision attack weakness that MD5 exhibits while the advantage of obtaining a smaller number of bytes in the output is also retained by using MD5.

Hence, the purpose of hashing it with SHA-256 is to improve security, while the purpose of MD5 is to reduce the number of bytes in the message digest.

This is the reason behind using the two Multiple Hashing Using SHA-256 and MD5 653 algorithms. It is also worth mentioning that repeated hashing might destroy the security that hashing algorithms provide.
Every hashing algorithm diffuses the message as much as possible.

2.2 Summary/Gaps identified in the Survey

The existing solutions uses at max only 2 algorithms which is not sufficient for today’s time.

As the passwords are getting more and more complex today with even using different algorithms for hashing a password , thus we have decided to implement 4 algorithms in our project which are:

1. SHA1
2. SHA2
3. MD5
4. SHA3

Not only that , we also introduced the concept of salting in our project which saltes the password to make it more safe.

Further , each password in the dictionary is matched with the hash key on the basis of 4 approaches:

The 6 possible hashes computed for each word from the dictionary are:

- SHA1(word)
- SHA1(drow) (reversed word)
- SHA1(wrd) (word without vowels)
- SHA1(salt||word) (salted word)
- SHA1(salt||drow) (salted reversed word)
- SHA1(salt||wrd) (salted word without vowels)

Thus , by using all these approaches , it ensures that we are able to crack the given password.

3. OVERVIEW OF THE PROPOSED SYSTEM

3.1 Introduction and Related Concepts

<u>How to prevent a password dictionary attack?</u>

The length of the password is an effective defence against brute-force attacks. The best strategy for creating a long password, that is also memorable, is to make it a passphrase.

A passphrase is a sentence or phrase, with or without spaces, typically more than 20 characters longer. The words making up a passphrase should be meaningless together to make them less susceptible to social engineering.

But a passphrase is only a good choice when it doesn't appear on a list of

leaked passwords.

Another critical measure to prevent a dictionary attack is to stop password reuse between different password-protected systems.

User training can help educate on the importance of not reusing passwords. However, the only way to remove this possibility is to blacklist leaked passwords at password creation.

3.2 Framework, Architecture or Module for the Proposed System(with explanation)

System Architecture of Password cracking using Dictionary Attacks

The design goal of the hybrid password cracking system is to crack encrypted target passwords by incrementally increasing the size of the password cracking space. Therefore, the system consists of three attack stages: the Dictionary attack followed by the TDT-model attack, and the Brute-force attack; this system is called the DTB password cracking system.

The cracking strategy is to crack encrypted passwords using the Dictionary attack and the TDT-model attack first in finite time, and to then run the Brute- force attack when some encrypted passwords have not been cracked.

The system can be used in DTB mode where all stages are enabled or in DB mode where only the TDT-model attack stage is disabled.

System architecture of DTB password cracking system

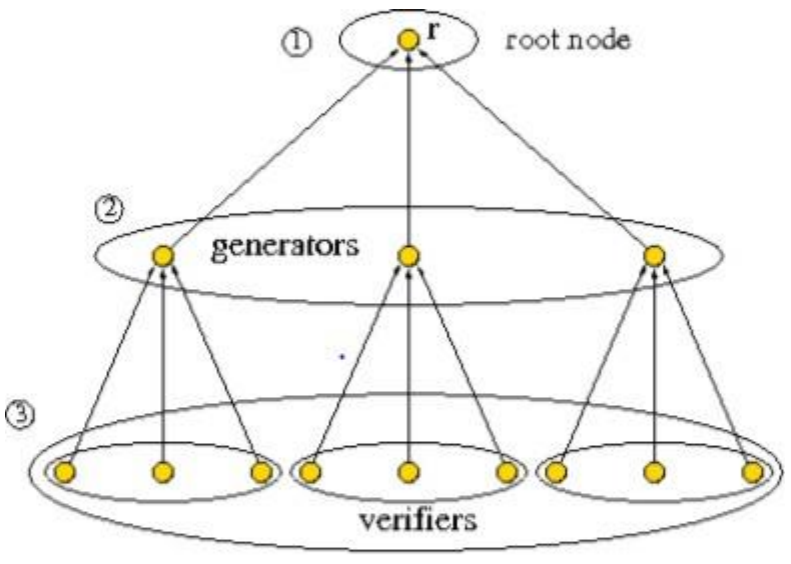


Figure 7. Nodes organization.

Since this network has been conceived to work with heterogeneous systems in a geographic context, the proposed architecture guarantees the following requirements:
scalability: the number of network nodes can be easily increased, augmenting the available computational power. load balancing: the computational load

must be distributed among nodes according to their capabilities, so that to prevent local starvation.

flexibility: since the availability of each node in the network is unpredictable, the architecture must be able to adapt itself to variations of available resources by changing the load distribution.

fault tolerance: possible failures of a node must not subvert the overall computation, thus the system must able to re-assign any workload and to recover local computation

The first level is constituted by a single "root" node, denoted as r, that is responsible for the compilation of the dictionary.

Second network is devoted to the generation of passphrases starting from the dictionary compiled in the first phase by the "root" node.

The third level consists of a variable number of nodes, named "verifiers" and denoted as v, that form the "verification network". Such a network is in charge of verifying whether any of the generated passphrases decrypt the private key

<u>ALGORITHMS SUPPORTED</u>

SHA1 Algorithm

There are two possible line formats: the first one contains an unsalted password while the second contains a salted password along with the salt.
The passwords are hashed using SHA-1 (see attack source code for implementation in the Java Cryptography Extension).

When a salt is used, it is simply concatenated together with the passwords as follows: salt || password. The attack simply reads the dictionary line by line and computes 6 different possible hashed passwords for the word contained in each line.

These 6 possible hashes are compared to each of the passwords contained in the password.txt file for a match. If there is a match, we recovered a password. If not, we simply keep reading the dictionary line by line.

MD5 Algorithm

The MD5 hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used for authenticating the original message.
The MD5 message digest hashing algorithm processes data in 512-bit blocks, broken down into 16 words composed of 32 bits each. The output from MD5 is a 128-bit message digest value.

Computation of the MD5 digest value is performed in separate stages that process each 512-bit block of data along with the value computed in the preceding stage. The first stage begins with the message digest values initialized using consecutive hexadecimal numerical values. Each stage includes four message digest passes which manipulate values in the current data block and values processed from the previous block.
The final value computed from the last block becomes the MD5 digest for that block.

SHA-2 Algorithm:
SHA-2 is a family of hashing algorithms to replace the SHA-1 algorithm. SHA-2 features a higher level of security than its predecessor.

The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.

As SHA-224 is simply truncated SHA-256 with different initialization values, and they share the same internal structure, so these are implemented together.

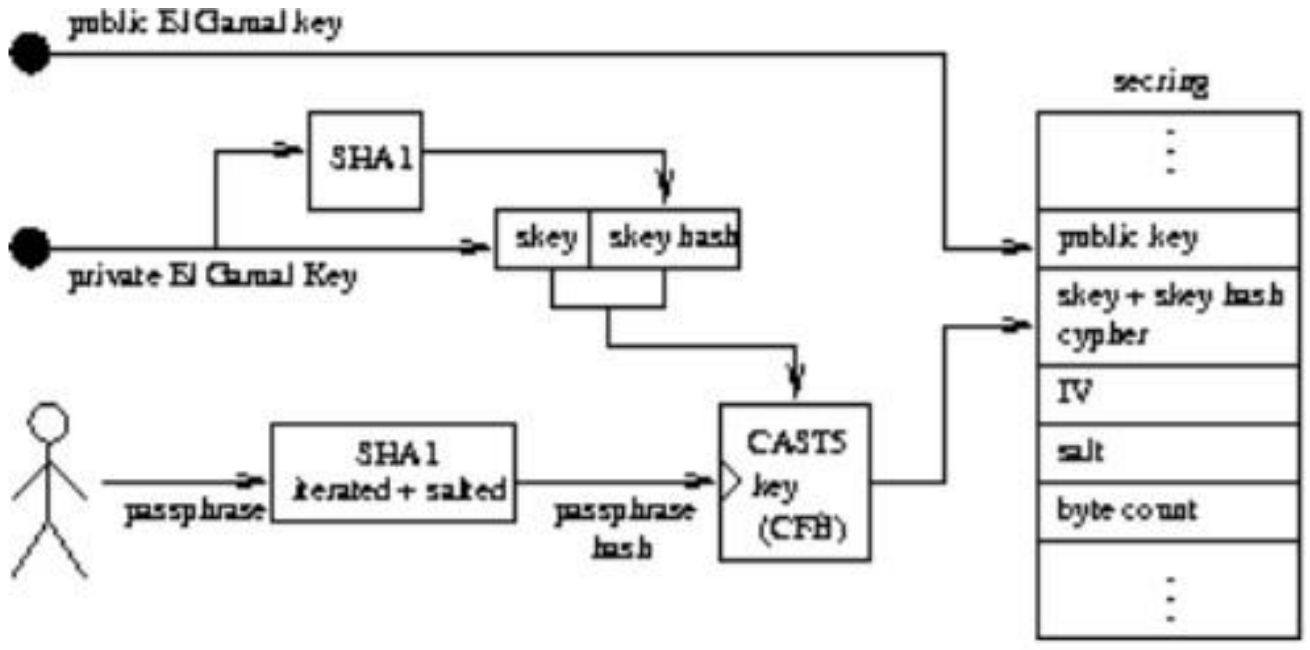
SHA3 Algorithm:

SHA-3 (Secure Hash Algorithm 3) is a set of cryptographic hash functions defined in FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.

The SHA-3 family consists of six hash functions with digests (hash values) that are 128, 224, 256, 384 or 512 bits: SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256.

3.3 Proposed System Model

Three Phases of Attacks



4. PROPOSED SYSTEM ANALYSIS AND DESIGN

4.1 Introduction

A straightforward dictionary attack is limited to exact matches, but is still surprisingly successful, as users tend to choose simple and predictable passwords. As suggested in [3] and [4], used words included most common names and surnames from several languages, pet names, band names, car manufacturers, sports teams, famous people, and colours.

These specific phrases were added to a large wordlist containing most common passwords in several languages, among others, obtained in several well-known leaks, forming a 14,457,264-word (134MB) dictionary for a straightforward attack.

How is it done?

Basically, it is trying every single word that is already prepared. It is done using automated tools that try all the possible words in the dictionary.

Some Password Cracking Software:

- John the Ripper
- L0phtCrack
- Aircrack-ng

Description of Modules/Programs – Hash suite:

Storing user passwords in the plain text naturally results in an instant compromise of all passwords if the password file is compromised.

To reduce this danger, Windows applies a cryptographic hash function, which transforms each password into a hash, and stores this hash.

This hash function is one-way in the sense that it is infeasible to infer a password back from its hash, except via the trial-and-error approach described below.

To authenticate a user, the password presented by the user is hashed and compared with the stored hash.

Hash Suite, like all other password hash crackers, does not try to "invert" the hash to obtain the password (which might be impossible).

It follows the same procedure used by authentication: it generates different candidate passwords (keys), hashes them and compares the computed hashes with the stored hashes.

This approach works because users generally select passwords that are easy to remember, and as a side-effect, these passwords are typically easy to crack.

Another reason why this approach is so very effective is that Windows uses password hash functions that are very fast to compute, especially in an attack (for each given candidate password). A Hash-Based Password Cracker tool. Works on any word list.

Supports most of the commonly used hashing algorithms to store passwords . No dependencies needed, just python.

4.2 REQUIREMENT ANALYSIS

4.2.1 FUNCTIONAL REQUIREMENTS

4.2.1.1 FEATURES

- Efficient
- Easy-to-Use
- Lightweight
- No Dependencies
-Wordlist Independent

4.2.1.2 User requirements

Python , command prompt

4.2.2 NON FUNCTIONAL REQUIREMENTS

The user is experiencing almost the same interaction as in the original login protocol (that uses no RTTs). The only difference is that he is required to pass an RTT in two instances (1) when he tries to login from a new computer for the first time, and (2) when he enters a wrong password (with probability p).

We assume that most users are likely to use a small set of computers to access their accounts, and use other computers very infrequently (say, while traveling without a laptop). We also have evidence, from the use of RTTs in Yahoo!, Alta Vista and Paypal, that users are willing to a nswer RTTs if they are not asked to do so very frequently. Based on these observations we argue that the suggested protocol could be implemented without substantially downgrading the usability of the login system.

4.2.3 ORGANIZATIONAL REQUIREMENTS

Safety of users

By using the project we ensure that the users password are thoroughly examined using all the approaches and algorithms available in the project so that we can check the strength of the password and thus suggest some alternatives ways to make a password more strong and robust.

4 METHODOLOGY ADOPTED

The list of passwords that we recover using the attack is a text file in which each line contains a user account name followed by a password.

There are two possible line formats: the first one contains an unsalted password while the second contains a salted password along with the salt.

The passwords are hashed using SHA-1 (see attack source code for implementation in the Java Cryptography Extension).

When a salt is used, it is simply concatenated together with the passwords as follows: salt || password.

The attack simply reads the dictionary line by line and computes 6 different possible hashed passwords for the word contained in each line.

These 6 possible hashes are compared to each of the passwords contained in the password.txt file for a match.

If there is a match, we recovered a password.
If not, we simply keep reading the dictionary line by line.

The 6 possible hashes computed for each word from the dictionary are:

- SHA1(word)
- SHA1(drow) (reversed word)
- SHA1(wrd) (word without vowels)
- SHA1(salt||word) (salted word)
- SHA1(salt||drow) (salted reversed word)
- SHA1(salt||wrd) (salted word without vowels)

CODE IMPLEMENTATION

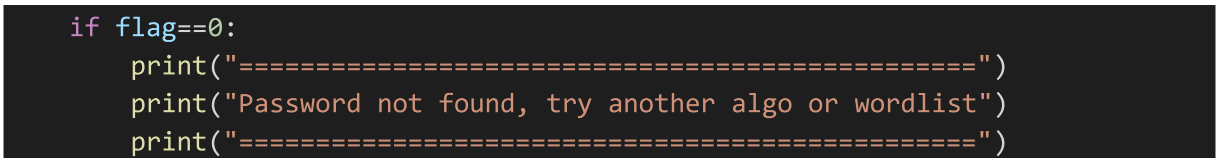
Dictionary attack code :--

```python
import hashlib
import os
import uuid
# Defining the Cracking Functions:
# remove vowels
def rem_vowel(string):
    vowels = ('a', 'e', 'i', 'o', 'u')
    for x in string.lower():
        if x in vowels:
            string = string.replace(x, "")
# MD5
def fun_md5():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running MD5")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.md5(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        if flag == 0:
            print("Please wait....Searching...")
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")


# SHA1
def fun_sha1():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA-1")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:

        salt = uuid.uuid4().hex
        enc_word=word.encode('utf-8')
        enc_word_rev=word[::-1].encode('utf-8')
        enc_word_salted=enc_word+salt.encode('utf-8')
        enc_word_rev_salted=enc_word_rev+salt.encode('utf-8')

        vowelstring=word
        rem_vowel(vowelstring)
        enc_word_vowel=vowelstring.encode('utf-8')

        salt = uuid.uuid4().hex

        # SHA1 STRING SALTED

        salted=hashlib.sha1(enc_word_salted.strip()).hexdigest()

        # SHA1 STRING

        digest=hashlib.sha1(enc_word.strip()).hexdigest()

        # SHA1 STRING REVERSED

        reverse=hashlib.sha1(enc_word_rev.strip()).hexdigest()

        # SHA1 STRING REVERSED SALTED

        reverse_salted=hashlib.sha1(enc_word_rev_salted.strip()).hexdigest()

        # SHA1 STRING WITHOUT VOWELS
        without_vowel=hashlib.sha1(enc_word_vowel.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        elif reverse == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        elif without_vowel == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        elif salted == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        elif reverse_salted == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break
        if flag == 0:
            print("Please wait....Searching...")

    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")

# SHA 2 Algorithms

# SHA2-224
def fun_sha224():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA2-224")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.sha224(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        if flag == 0:
            print("Please wait....Searching...")
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")

# SHA2-256
def fun_sha256():
    try:
        pass_file = open(word_list,"r")
```

```python
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA2-256")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.sha256(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        if flag == 0:
            print("Please wait....Searching...")
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")

# SHA2-384
def fun_sha384():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA2-384")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.sha384(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        if flag == 0:
            print("Please wait....Searching...")
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")

# SHA2-512
def fun_sha512():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA2-512")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.sha512(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        if flag == 0:
            print("Please wait....Searching...")
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
```

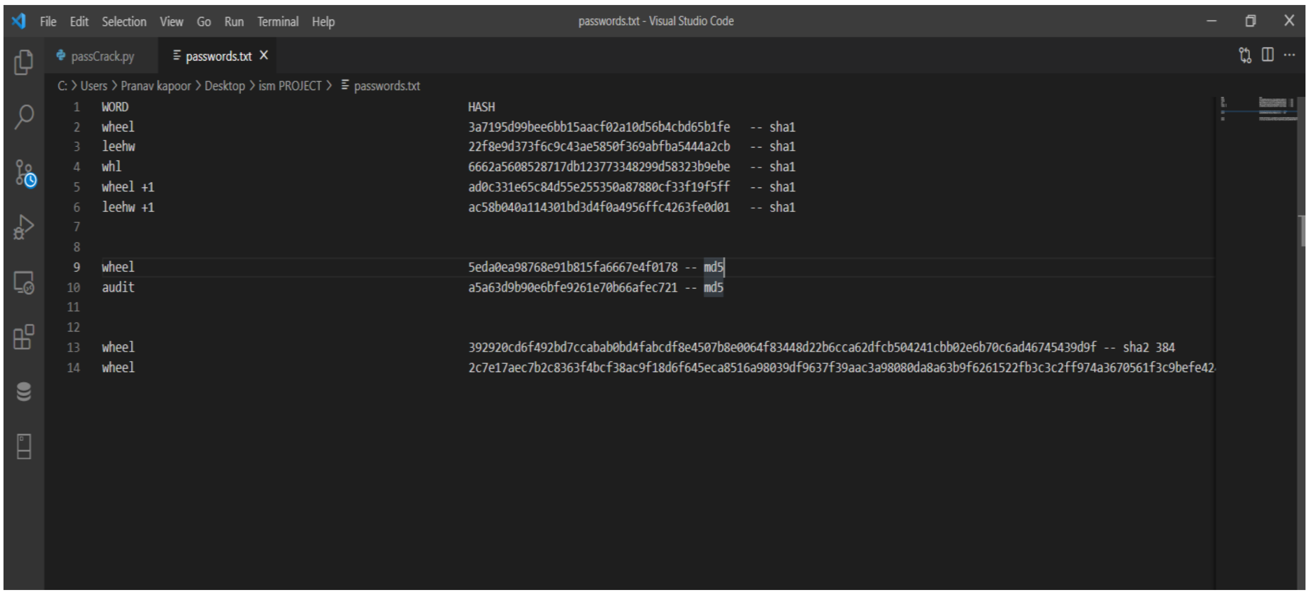
```python
        print("================================================")

#  SHA 3 Algorithms

# SHA3-224
def fun_sha3_224():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA3-224")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.sha3_224(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        if flag == 0:
            print("Please wait....Searching...")
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")

# SHA3-256
def fun_sha3_256():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA3-256")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.sha3_256(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        if flag == 0:
            print("Please wait....Searching...")
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")

# SHA3-384
def fun_sha3_384():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA3-384")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.sha3_384(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
```

```
                flag=1
                break

            if flag == 0:
                print("Please wait....Searching...")
        if flag==0:
            print("================================================")
            print("Password not found, try another algo or wordlist")
            print("================================================")

# SHA3-512
def fun_sha3_512():
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()

    flag=0
    counter=0
    print("Running SHA3-512")
    print("=================")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        digest=hashlib.sha3_512(enc_word.strip()).hexdigest()

        counter+=1
        if digest == pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break

        if flag == 0:
            print("Please wait....Searching...")
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")

# Driver Code
print("=============================\n")

print("A Hash Based password Cracker")
print("=============================")

pass_hash = input("Enter The hash Key: ")

know=input("Do you know the Hashing Algorithm used ??\n\tY\tN\n: ")
print("=============================")
word_list = input("Enter the name of the wordlist: ")

print("=============================")
if know =='Y'or know =='y':
    print("Select the Algotithm used : ")
    choice=int(input("[0]: MD5\n[1]: SHA-1\n[2]: SHA-2\n[3]: SHA-
3\nEnter Your Choice: "))
    print("=============================")
    if choice==0:
        fun_md5()
    elif choice==1:
        fun_sha1()
    elif choice==2:
        ch_sha2=int(input("[1]: SHA2-224\n[2]: SHA2-256\n[3]: SHA2-
384\n[4]: SHA2-512\nEnter Your Choice: "))
        if ch_sha2==1:
            fun_sha224()
        elif ch_sha2==2:
            fun_sha256()
        elif ch_sha2==3:
            fun_sha384()
        elif ch_sha2==4:
            fun_sha512()
        else:
            print("=============================")
            print("Enter a Valid Choice!!")
            print("=============================")
    elif choice==3:
        ch_sha2=int(input("[1]: SHA3-224\n[2]: SHA3-256\n[3]: SHA3-
384\n[4]: SHA3-512\nEnter Your Choice: "))
        if ch_sha2==1:
            fun_sha3_224()
        elif ch_sha2==2:
            fun_sha3_256()
        elif ch_sha2==3:
            fun_sha3_384()
        elif ch_sha2==4:
            fun_sha3_512()
        else:
            print("=============================")
            print("Enter a Valid Choice!!")
            print("=============================")
    else:
        print("=============================")
        print("Enter a Valid Choice!!")
        print("=============================")

else:
    print("=============================")
    print("Trying all avaliable algorithms: Please wait..")
    try:
        pass_file = open(word_list,"r")
    except:
        print("File Not Found")
        quit()
    flag=0
    counter=0
    print("=================")
    print("Please wait.. This will take some time")
    Lines=pass_file.readlines()
    for word in Lines:
        enc_word=word.encode('utf-8')
        # MD5
        digest_md5=hashlib.md5(enc_word.strip()).hexdigest()
        # SHA1
        digest_SHA1=hashlib.sha1(enc_word.strip()).hexdigest()
        # SHA2
        digest_SHA2_224=hashlib.sha224(enc_word.strip()).hexdigest()
        digest_SHA2_256=hashlib.sha256(enc_word.strip()).hexdigest()
        digest_SHA2_384=hashlib.sha384(enc_word.strip()).hexdigest()
        digest_SHA2_512=hashlib.sha512(enc_word.strip()).hexdigest()
        # SHA3
        digest_SHA3_224=hashlib.sha3_224(enc_word.strip()).hexdigest()
        digest_SHA3_256=hashlib.sha3_256(enc_word.strip()).hexdigest()
        digest_SHA3_384=hashlib.sha3_384(enc_word.strip()).hexdigest()
        digest_SHA3_512=hashlib.sha3_512(enc_word.strip()).hexdigest()

        counter+=1
        if digest_md5 == pass_hash or digest_SHA1==pass_hash or digest_SHA2_22
4==pass_hash or digest_SHA2_256==pass_hash or digest_SHA2_384==pass_hash or di
gest_SHA2_512==pass_hash or digest_SHA3_224== pass_hash or digest_SHA3_256== p
ass_hash or digest_SHA3_384==pass_hash or digest_SHA3_512==pass_hash:
            print("=================")
            print("Password Found")
            print(word)
            print("=================")
            print("Passwords Checked: "+str(counter))
            flag=1
            break
```



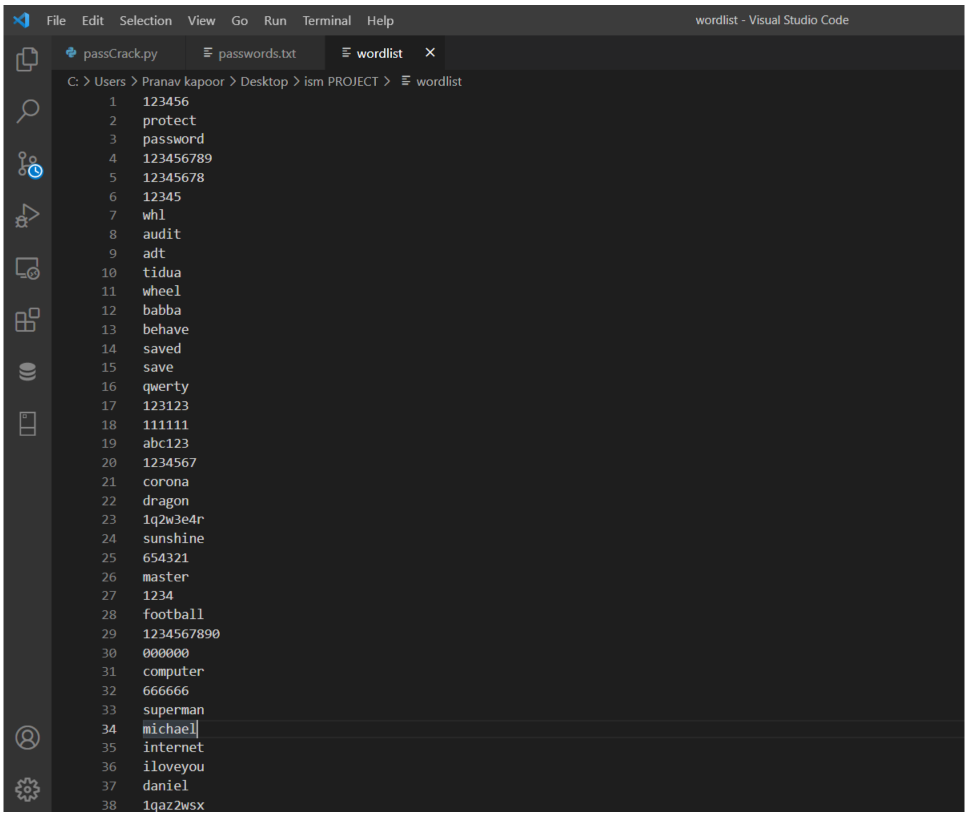
```
    if flag==0:
        print("================================================")
        print("Password not found, try another algo or wordlist")
        print("================================================")
```

Passwords.txt


File Edit Selection View Go Run Terminal Help
passwords.txt - Visual Studio Code
passCrack.py
passwords.txt
C: > Users > Pranav kapoor > Desktop > ism PROJECT > passwords.txt

```
WORD                                              HASH
wheel                                             3a7195d99bee6bb15aacf02a10d56b4cbd65b1fe   -- sha1
leehw                                             22f8e9d373f6c9c43ae5850f369abfba5444a2cb   -- sha1
whl                                               6662a5608528717db123773348299d58323b9ebe   -- sha1
wheel +1                                          ad0c331e65c84d55e255350a87880cf33f19f5ff   -- sha1
leehw +1                                          ac58b040a114301bd3d4f0a4956ffc4263fe0d01   -- sha1


wheel                                             5eda0ea98768e91b815fa6667e4f0178 -- md5
audit                                             a5a63d9b90e6bfe9261e70b66afec721 -- md5


wheel                                             392920cd6f492bd7ccabab0bd4fabcdf8e4507b8e0064f83448d22b6cca62dfcb504241cbb02e6b70c6ad46745439d9f -- sha2 384
wheel                                             2c7e17aec7b2c8363f4bcf38ac9f18d6f645eca8516a98039df9637f39aac3a98080da8a63b9f6261522fb3c3c2ff974a3670561f3c9befe42
```

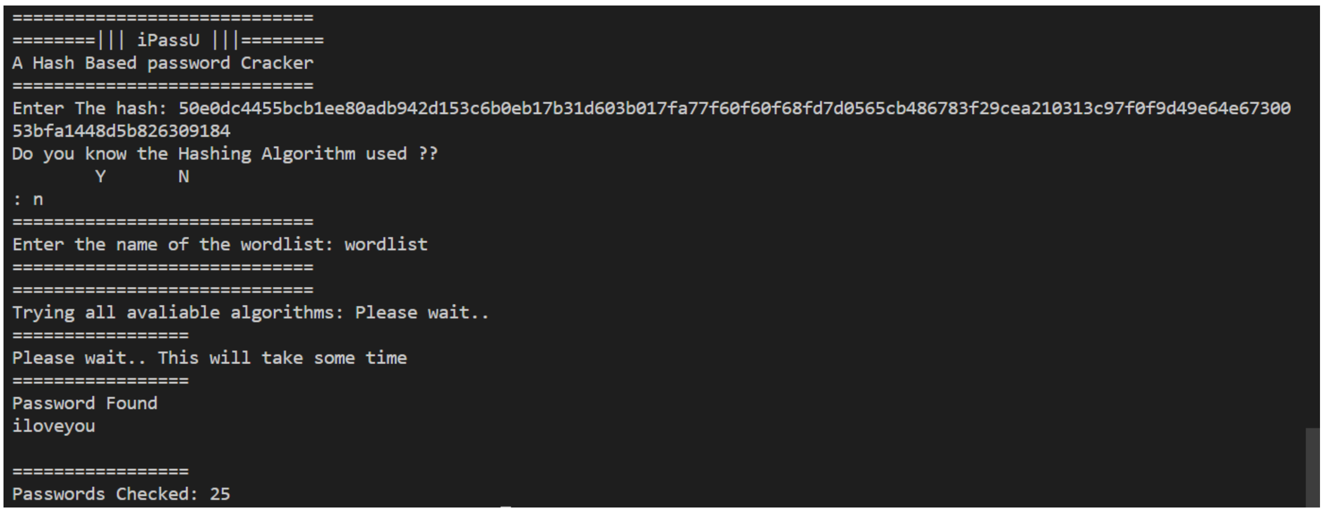
Dictionary


File Edit Selection View Go Run Terminal Help
wordlist - Visual Studio Code
passCrack.py
passwords.txt
wordlist
C: > Users > Pranav kapoor > Desktop > ism PROJECT > wordlist

```
123456
protect
password
123456789
12345678
12345
whl
audit
adt
tidua
wheel
babba
behave
saved
save
qwerty
123123
111111
abc123
1234567
corona
dragon
1q2w3e4r
sunshine
654321
master
1234
football
1234567890
000000
computer
666666
superman
michael
internet
iloveyou
daniel
1qaz2wsx
```

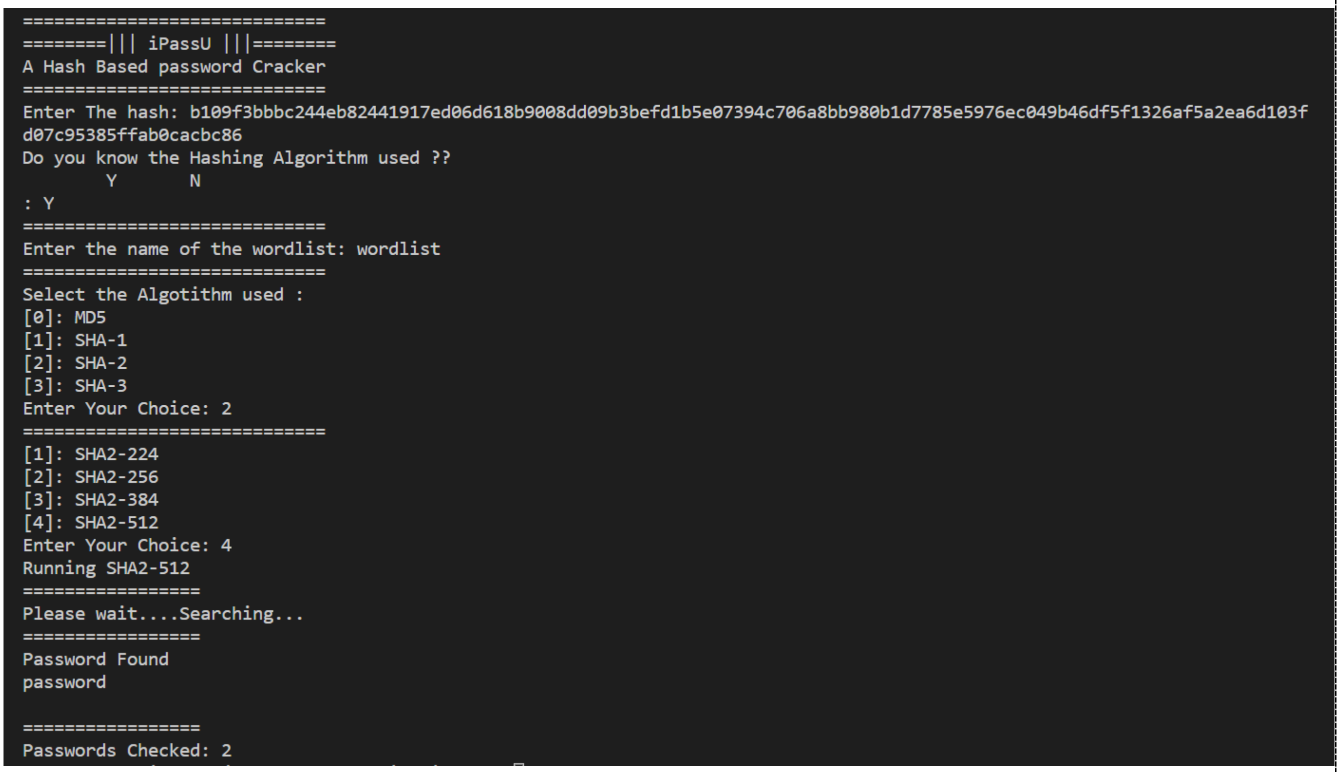
6. RESULTS AND DISCUSSION

After inputting the hash , we choose the hash algorithm to evaluate.



```
============================
========||| iPassU |||========
A Hash Based password Cracker
============================
Enter The hash: 50e0dc4455bcb1ee80adb942d153c6b0eb17b31d603b017fa77f60f60f68fd7d0565cb486783f29cea210313c97f0f9d49e64e67300
53bfa1448d5b826309184
Do you know the Hashing Algorithm used ??
        Y       N
: n
============================
Enter the name of the wordlist: wordlist
============================
============================
Trying all avaliable algorithms: Please wait..
================
Please wait.. This will take some time
================
Password Found
iloveyou

================
Passwords Checked: 25
```

After that , we input the name of the wordlist to be searched in , and then get the result of a decrypted password.



```
============================
========||| iPassU |||========
A Hash Based password Cracker
============================
Enter The hash: b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df5f1326af5a2ea6d103f
d07c95385ffab0cacbc86
Do you know the Hashing Algorithm used ??
        Y       N
: Y
============================
Enter the name of the wordlist: wordlist
============================
Select the Algotithm used :
[0]: MD5
[1]: SHA-1
[2]: SHA-2
[3]: SHA-3
Enter Your Choice: 2
============================
[1]: SHA2-224
[2]: SHA2-256
[3]: SHA2-384
[4]: SHA2-512
Enter Your Choice: 4
Running SHA2-512
================
Please wait....Searching...
================
Password Found
password

================
Passwords Checked: 2
```

7. REFERENCES

[1] R. Morris and K. Thompson, “ Password Security: A Case History,”Commun. ACM, vol. 22, no. 11, pp. 5 94–5 97, Nov. 1979.

[2] D. C. Feldmeier and P. R. Karn, “UNIX password security - ten Advances in Cryptology, pp. 44-63, 1989.

[3] M. Zviran and W. J. Haga, “Passwo rd security : an empirical study,” Journal of Management Information Systems, vol. 15, pp. 161–185, 1999.

[4] D. Klein, “Foiling the cracker: A survey of, and improvements to, Workshop, pp. 5-14, 1990.

[5] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, an d SIGCHI Conference on Human Factors in Computing Syst em s, 2013, pp. 2379–2388.

[6] V. Tan eski, M. Heričko and B. Brumen, “ Imp act of Security Technology, Electronics and Microelectr onics (MI PRO) 38th International Convention, 2015, pp. 1351–1355

[7] Froilan E. De Guzman, Bobby D. Gerardo, Ruji P. Humanoid Nanotechnology Information TechnologyCommunication and Control Environment and Management (HNICEM) 2018 IEEE 10th International Conference on, pp. 1-6, 2018.

[8] .Vishal Vishal, Rahul Johari, "SOAiCE: Simulation of Attacks in Cloud Computing Environment", Cloud Computing Data Science & Engineering (Confluence) 2018 8th International Conference on, pp. 14-15, 2018.

[9] .Michael Angelo D. Brogada, Ariel M. Sison, Ruji P. Medina, "Cryptanalysis on the Head and Tail Technique for Hashing Passwords", Systems Process and Control (ICSPC) 2019 IEEE 7th Conference on, pp. 137-142, 2019.

[10] .Shriya S Shetty, Rithika R Shetty, Tanisha G Shetty, Divya Jennifer D'Souza, "Survey of hacking techniques and it's prevention", Power Control Signals and Instrumentation Engineering (ICPCSI) 2017 IEEE International Conference on, pp. 1940-1945, 2017.