

**CSE3502 - Information Security Management**

**School of Computer Science & Engineering**

**J COMPONENT PROJECT REPORT**

**PASSWORD CRACKING USING DICTIONARY ATTACK**

**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, thus their security and protection is also the foremost agenda for the businesses managing and maintaining them.

It is a matter of disgrace that there square measure a range of attacks exist which may hamper the traditional practicality of the relative databases. during this project we have a tendency to take into thought the ‘DICTIONARY ATTACKS’.

Dictionary Attacks is an approach of using a program to run each 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.

A salt is random information that's used as an extra input to a unidirectional perform that "hashes"

a word or passphrase. Salts square measure closely associated with the thought of the nowadays.

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. **Theoretical Background**

In order to guard yourself from a word attack or a breach, we must always become aware of the foremost unremarkably used sorts 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 defense need bolstering.

The most common type of attack is password guessing. Hackers can make out passwords locally or remotely using either a manual or an automated approach.

Password shot isn't as tough as you'd expect. Most networks are not organized to need long and complicated passwords, and an aggressor must notice only 1 weak word to realize access to a network.

**1.2 Motivation**

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

A dictionary attack tries to make an authentication mechanism fail by sequentially entering each word in a dictionary as a password or trying to find the decryption key of an encrypted message or document.

Dictionary attacks are often possible because as many people and businesses use simple words as their 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 varied passwords each preserved and non-salted , and check all the passwords mistreatment four totally different algorithms, to ascertain the strength of word and to be able to use new methods to form passwords so it's shielded from lexicon attacks.

1. **Literature review**
   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 aim of hashing it with SHA-256 is to enhance security, whereas the aim of MD5 is to scale back the quantity of bytes within 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.

* 1. **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 words have gotten a lot of and a lot of complicated these days with even mistreatment totally different algorithms for hashing a password , so we've got determined to implement four algorithms in our project that 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:

## We will be undertaking 6 use cases 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.

used by the students to access the university’s online

1. **OVERVIEW OF THE PROPOSED SYSTEM**

**3.1 Introduction and Related Concepts**

# How to prevent a password dictionary attack?

The length of the word is AN economical defense against brute-force attacks. the foremost effective strategy for creating AN extended word, that is together persistent, is to create it a passphrase.

A passphrase is also a sentence or phrase, with or whereas not areas, typically quite twenty characters longer. The words making up a passphrase have to be compelled to be nonsensical on to create them less at risk of social engineering.

But a passphrase is simply an honest alternative once it doesn’t seem on a listing of

leaked passwords.

Another important live to forestall a lexicon attack is to prevent watchword use between totally different password-protected systems.

User employment can facilitate educate on the importance of not reusing passwords. However, the only real because of subtract this risk is to blacklist leaked words at watchword creation.

**3.2 Framework, design or Module for the projected System(with explanation)**

**System design of watchword cracking mistreatment lexicon Attacks**

The design goal of the hybrid watchword cracking system is to crack encrypted target watchwords by incrementally increasing the scale of the password cracking house. Therefore, the system consists of 3 attack stages: the lexicon attack followed by the TDT-model attack, and also the Brute-force attack; this technique is named the DTB watchword cracking system.

The cracking strategy is to crack encrypted passwords mistreatment the lexicon attack and also the TDT-model attack 1st in finite time, and to then run the Brute- force attack once some encrypted passwords haven't been cracked.

The system is utilized in DTB mode where all stages unit enabled or in unit mode where only the TDT-model attack stage is disabled.

**System design of DTB watchword cracking system**

****

Since this network has been formed to figure with heterogeneous systems during a geographic context, the projected design guarantees the subsequent requirements:

**Scalability:**

The amount of network nodes will be simply increased , augmenting the offered machine power. load balancing: the machine load must be distributed among nodes in line with their capabilities, in order that to forestall native starvation.

**Flexibility:**

Since the provision of every node within the network is unpredictable, the design should be ready to adapt itself to variations of obtainable resources by dynamical the load distribution.

**Fault tolerance**

Doable failures of a node should not subvert the general computation, therefore the system should ready to re-assign any employment and to recover native computation

The first level is recognized by one “root” node, denoted as r, that is responsible for the compilation of the lexicon.

Second network is dedicated to the generation of passphrases ranging from the lexicon compiled within the 1st part by the “root” node.

The third level consists of a variable range of nodes, named “verifiers” and denoted as v, that kind the “verification network”. Such a network is responsible of confirming whether or not any of the generated passphrases rewrite the non-public key

**ALGORITHMS SUPPORTED**

**SHA1 algorithmic rule**

There are 2 achievable line formats: the primary one contains AN tasteless watchword whereas the second contains a salt-cured watchword at the side of the salt.

The passwords are hashed mistreatment SHA-1 (see attack ASCII text file for implementation within the Java Cryptography Extension).

When a salt is utilized, it's just concatenated at the facet of the words as follows: salt || watchword.

The attack just reads the lexicon line by line and computes vi utterly totally different come-at-able hashed passwords for the word contained in each line.

These six doable hashes square measure compared to every of the watchwords contained within the password.txt file for a match. If there's a match, we tend to recovered a watchword. If not, we tend to merely keep reading the lexicon line by line.

**MD5 algorithmic rule**

The MD5 hashing algorithmic rule could be a unidirectional science perform that accepts a message of any length as input and returns as output a fixed-length digest price to be used for authenticating the initial message.

The MD5 message digest hashing algorithmic rule processes knowledge in 512-bit blocks, weakened into sixteen words composed of thirty two bits every. The output from MD5 could be a 128-bit message digest price.

Computation of the MD5 digest price is performed in separate stages that method every 512-bit block of knowledge at the side of the worth computed within the preceding stage. the primary stage begins with the message digest values initialized mistreatment consecutive hex numerical values. every stage includes four message digest passes that manipulate values within the current knowledge block and values processed from the previous block.

The final price computed from the last block becomes the MD5 digest for that block.

**SHA-2 Algorithm:**

SHA-2 is also a family of hashing rules to exchange the SHA-1 algorithmic rule. SHA-2 choices future level of security than its precursor.

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

As SHA-224 is only truncated SHA-256 with utterly totally different info values, which they share constant internal structure, so these unit implemented on.

**SHA3 Algorithm:**

SHA-3 (Secure Hash algorithmic rule 3) could be a set of science hash functions outlined 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 square measure 128, 224, 256, 384 or 512 bits: SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256.

* 1. **Proposed System Model**

**Three Phases of Attacks**

Diagram

Description automatically generated

****

1. **PROPOSED SYSTEM ANALYSIS AND DESIGN**

**4.1 Introduction**

## A simple lexicon attack is restricted to precise matches, however remains amazingly booming, as users tend to decide on straightforward and foreseeable passwords. As steered in [3] and [4], used words enclosed commonest names and surnames from many languages, pet names, band names, automobile makers, sports groups, celebrated folks, and hues.

These specific phrases were additional to associate degree outsized wordlist containing commonest passwords in several languages, among others, obtained in several well-known leaks, forming a fourteen,457,264-word (134MB) lexicon for a straightforward attack.

**How is it done?**

Basically, it's attempting each single word that's already ready. it's done exploitation machine-driven tools that strive all the doable words within the lexicon.

Some secret Cracking Software:

• John the liquidator

• L0phtCrack

• Aircrack-ng

Description of Modules/Programs – Hash suite:

Storing user secrets within the plain text naturally leads to an immediate compromise of all passwords if the password file is compromised.

To reduce this danger, Windows applies a cryptographical hash operate, that transforms every secret into a hash, and stores this hash.

This hash operate is unidirectional within the sense that it's unfeasible to infer a secret back from its hash, except via the trial-and-error approach delineated below.

To certify a user, the secret bestowed by the user is hashed and compared with the hold on hash. Hash Suite, like all different secret hash barmy, doesn't attempt to "invert" the hash to get the secret (which can be impossible).

It follows constant procedure employed by authentication: it generates totally different candidate passwords (keys), hashes them and compares the computed hashes with the hold on hashes. This approach works as a result of users usually choose passwords that ar straightforward to recollect, and as a side-effect, these passwords ar usually straightforward to crack.

Another reason why this approach is therefore terribly effective is that Windows uses secret hash functions that ar in no time to figure, particularly in associate degree attack (for every given candidate password). A Hash-Based secret Cracker tool. Works on any thesaurus.

Supports most of the usually used hashing algorithms to store passwords . No dependencies required, simply python.

* 1. **REQUIREMENT ANALYSIS** 
     1. **FUNCTIONAL REQUIREMENTS**
        1. **PRODUCT PERSPECTIVE:**

The design target is login pages wherever the user enters login and password to login into their account.

**4.2.1.2 FEATURES**

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

**4.2.1.3. User requirements**

Python , command prompt

**4.2.2 NON FUNCTIONAL REQUIREMENTS**

3.2.1 Usability

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 answer 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.

* + 1. **ORGANIZATIONAL REQUIREMENTS**
       1. **Product Requirements:**
          1. **Reliability**-

It is a reliable project in terms of consistency of output and its accuracy every time we run it.

* + - * 1. **Portability:**

Since there are no heavyweight hardware requirements the project is easily portable.

* + - * 1. **Usability:**

Usability level is intermediate

* + - 1. **Organizational requirements:**
         1. **Implementation Requirements:**

Python command prompt, Hash converter for text to SHA conversion

* + - * 1. **Engineering Standard requirements;**

The project follows all good practices of engineering standards.

* + - 1. **Operational requirements:**
         1. **Health and safety**

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.

* + - * 1. **Legality:**

The project helps institutions in assessing the strength of their password protected systems to figure out ways to help strengthen it further

* + 1. **SYSTEM REQUIREMENTS**
       1. **S/W Requirements:**

Python command prompt, Hash converter

1. **METHODOLOGY ADOPTED**

The list of Passwords that we have a tendency to recover mistreatment the attack may be a document within which every line contains a user account name followed by a password.

There square measure 2 potential line formats: the primary one contains AN unseasoned secret whereas the second contains a salt-cured secret beside the salt.

The passwords square measure hashed mistreatment SHA-1 (see attack ASCII text file for implementation within the Java Cryptography Extension).

When a salt is employed, it's merely concatenated at the side of the Passwords as follows: salt || password.

The attack merely reads the lexicon line by line and computes six completely different potential hashed passwords for the word contained in every line.

These six potential hashes square measure compared to every of the Passwords contained within the password.txt file for a match.

If there's a match, we have a tendency to recovered a secret.

If not, we have a tendency to merely keep reading the lexicon line by line.

## We will be undertaking 6 use cases 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 :--

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")

    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")

        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\_224==pass\_hash or digest\_SHA2\_256==pass\_hash or digest\_SHA2\_384==pass\_hash or digest\_SHA2\_512==pass\_hash or digest\_SHA3\_224== pass\_hash or digest\_SHA3\_256== pass\_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**

Text

Description automatically generated

**Dictionary**

Text

Description automatically generated

1. **RESULTS AND DISCUSSION**

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

**ALL ALGORITHM TESTING**

**Text

Description automatically generated**

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

**SHA 2 ALGO**

**Text

Description automatically generated**

**SHA 3 ALGO**

Text

Description automatically generated

**MD5 ALGO**

Text

Description automatically generated

1. **REFERENCES**
2. R. Morris and K. Thompson, “ Password Security: A Case History,”Commun. ACM, vol. 22, no. 11, pp. 5 94–5 97, N ov.

1979.

1. D. C. Feldmeier and P. R. Karn, “UNIX password security - ten Advances in Cryptology, pp. 44-63, 1989.
2. M. Zviran and W. J. Haga, “Passwo rd security : an empirical study,” Journal of Management Information Systems, vol. 15, pp.

161–185, 1999.

1. D. Klein, “Foiling the cracker: A survey of, and improvements to,

Workshop, pp. 5-14, 1990.

1. S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, an d SIGCHI Conference on Human Factors in Computing Syst em s,

2013, pp. 2379–2388.

1. 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
2. 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.
3. .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.
4. .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.
5. .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.