

An Internship Project Report

Hashed Password Cracker

Submitted in Partial Fulfilment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING CYBERSECURITY

By

Abhishek Maurya	22BTLCC004
Dhrutikkumar Patel	22BTRCC001
Nilesh Dutta	22BTRCC032
Vaibhav Dhonde	22BTRCC016
V. Megha Rao	22BTRCC053

Under the supervision of

Mr. Adarsh Tiwari



JAIN
DEEMED-TO-BE UNIVERSITY

FACULTY OF
ENGINEERING
AND TECHNOLOGY

Submitted to the

Department of Computer Science and Engineering

School of Engineering & Computer Science

JAIN DEEMED TO BE UNIVERSITY

BENGALURU

CANDIDATE'S DECLARATION

It is to certify that the report entitled " **Hashed Password Cracker**" which is being submitted by Vaibhav Dhonde, Abhishek Maurya, V. Megha Rao, Dhruvik Kumar Patel and Nilesch Dutta to the JAIN UNIVERSITY, BANGALORE, in the fulfilment of the requirement for the award of the degree **Bachelor of Technology (B. Tech)** is a record of bonafide research work carried out by them under my guidance and supervision. The matter presented in this report has not been submitted either in part or full to any University or Institute for award of any degree.

Date: 26-04-2025

Name	USN
Abhishek Maurya	22BTLCC04
Dhruvikkumar Patel	22BTRCC001
Nilesch Dutta	22BTRCC032
Vaibhav Dhonde	22BTRCC016
V. Megha Rao	22BTRCC053

CERTIFICATE

It is to certify that the report entitled “Hashed Password Cracker” which is being submitted by Name to the JAIN UNIVERSITY, BANGLORE, in the fulfillment of the requirement for the award of the degree Bachelor of Technology (B.Tech) is a record of bonafide research work carried out by them under my guidance and supervision. The matter presented in this report has not been submitted either in part or in full to any University or Institute for an award of any degree.

Department of Computer Science & Engineering Cyber Security

JAIN (Deemed-to-Be University), Kanakpura

INDIA

ABSTRACT

This report presents the development of a tool designed to simplify password analysis by converting inputs between plaintext and hashed formats. The tool accepts user input as either a plaintext password or a hashed value and efficiently processes the data to generate its counterpart as output. By integrating support for commonly used hashing algorithms like MD5, ivSHA-256, the tool caters to various security use cases. This dual functionality aids in both password validation and testing the reliability of hashing techniques. The report highlights the tool's methodology, performance metrics, and potential applications, emphasizing its role in strengthening password security protocols.

ACKNOWLEDGEMENT

At this joyous moment of presenting this project, I would like to express my deep sense of gratitude and thanks to those who have supported and guided me in completing this B.Tech Sem VI.

First and foremost, I am profoundly grateful to my supervisor, **Mr Adharsh Tiwari**, for their invaluable expertise, guidance, and encouragement throughout this journey. Their insightful advice and patience during the challenges of this project have been instrumental in its successful completion.

I would also like to extend my heartfelt thanks to my parents, siblings, and friends for their unconditional love, constant support, and encouragement, which kept me motivated throughout this endeavour.

Finally, I express my sincere appreciation to all faculty, college management, administrative, and technical staff of the **School of Engineering & Computing, JAIN UNIVERSITY**, for providing a nurturing environment, resources, and inspiration that contributed significantly to my academic growth.

Without the unwavering support and cooperation of all these individuals, this project would not have been possible.

Date: 29/04/2024

Abhishek Maurya

Dhrutikkumar Patel

Nilesh Dutta

Vaibhav Dhonde

V. Megha Rao

TABLE OF CONTENTS

Chapter	Title	Page No.
1	Introduction	1
1.1	Introduction	1
1.2	Objective of the Work	2
1.3	Scope of the Project	2
1.4	Flowchart	3
1.5	Functionality	5
2	Literature Survey	6
2.1	Background	6
2.2	Main Objective	7
3	Methodology	8
3.1	Methodology	8
3.2	Functional Architecture	10
3.3	Challenges in Development	10
4	Implementation	12
4.1	Programming Language	13

4.2	Libraries and Frameworks	13
4.3	Development Environment	14
4.4	Summary of Tools and Workflow	14
4.5	Ethical and Security Considerations	15
5	Implementation and Testing	16
5.1	Development Environment and Tools	16
5.2	Running the Code	19
5.3	Testing and Results	21
5.4	Summary of Performance	22
6	Result and Analysis	23
6.1	Process Analysis	23
6.2	Impact on System Performance	25
6.3	Limitations and Challenges	26
6.4	Summary	27
7	References	28

LIST OF FIGURES

Figure	Figure Name	Page No
Fig 1	Flowchart of Password Input Processing and Validation	4
Fig 2 & Fig 3	Main Python Script for Password Hashing and Cracking	17 & 18
Fig 4	Hashed Password Cracker GUI	20
Fig 5	GUI Output: Password Found Using Dictionary Attack	21
Fig 6	GUI Interface	24
Fig 7	GUI Output: Hash Found Using Dictionary Attack	24
Fig 8	Hashed Password Cracker: Dictionary Attack Failure Message	27

Chapter 1

Introduction

1.1 Introduction

Passwords are absolutely essential to keeping digital systems secure, but their success depends on the resilience of algorithms for hashing and storing them. Conventional methods of password management do not keep up with emerging cybersecurity requirements, and therefore, advanced tools for analysis and verification are needed. The utility covered in this report is an innovative solution to this issue through the implementation of a dual-functionality feature: hashing plaintext passwords into hashed outputs and unhashing hashed values back into their original plaintext form, where allowed.

The underlying purpose of the tool is to make password-based operations easier for users, either for learning purposes, security analysis, or usage in system testing. The tool is capable of handling various algorithms for hashing to make it adaptive and versatile in its application area. By the automation of conversion, it takes away the cumbersome task of doing it manually, decreases errors, and greatly optimizes efficiency.

This report will describe the design and functionality of the tool, its user interface, algorithm choice process, and performance metrics. Moreover, it will discuss the ethical implications of working with sensitive information and meeting the standards of cybersecurity best practices. The multi-functionality of this tool not only benefits individuals and organizations in comprehending password security but also provides a useful asset to enhance overall system defenses.

1.2 Objective of the work

The main goal of the "Hashed Password Cracker" project is to create a simple yet useful tool for the easy transformation between plaintext passwords and hashed ones. With the support of commonly used hashing algorithms such as MD5, SHA-256, the tool is expected to offer versatility in different password-related operations.

This project aims to ease password analysis and validation procedures, allowing users to test the integrity of hashing methods or verify password accuracy in a secure and ethical way. Furthermore, the tool is intended to raise awareness of the significance of sound password management procedures and possible vulnerabilities related to weak or antiquated hashing methods.

One of the most important areas of interest for this project is to provide ethical use, focusing the functionalities of the tool in accordance with cybersecurity best practices and avoiding misuse in illicit activities. This project as a whole hopes to be a useful tool for researchers, educators, and professionals, working towards better password security and system strength.

1.3 Scope of project

The domain of the "Hashed Password Cracker" project is large and spans various areas of password management and cybersecurity.

The tool is intended to serve a broad audience of users, ranging from cybersecurity experts to researchers, teachers, and even students, by offering a hands-on solution to learning and implementing password hashing algorithms. Its two-way functionality—hashing plaintext passwords to hashed values and unhashing them back—allows users to carry out operations like password verification, hashing algorithm strength testing, and hash security analysis in a straightforward way.

The support of numerous hashing algorithms such as MD5, SHA-1, and SHA-256 enhances the versatility of the tool to suit a wide range of real-world applications. It can be employed in penetration testing labs, password security

demonstration for educational purposes, and system testing for evaluating password strength. Additionally, the scope encompasses enhancing awareness of safe hashing practices, educating users on weaknesses in weak or legacy hashing algorithms.

This project is developed with efficiency and ease of use in mind, making sure that the tool is easy for both technical and non-technical users. Ethical aspects are also an essential component of the scope of the project, highlighting its adoption for use in authorized and responsible manners, and conforming to best practices in cybersecurity. Overall, the project seeks to help advance efforts in improving password security as well as a better understanding of hashing practices.

1.4 Flowchart

This flowchart illustrates the process of handling a user-provided input, which can either be a plaintext password or a hashed value. The goal is to process the input appropriately based on its type and provide a validated output to the user.

Process Overview

1. **Start and User Input:** The process begins when the user enters a password. The input can be either in plaintext format or as a hashed value.
2. **Input Type Detection:** The system detects whether the entered input is a plaintext password or a hashed value. This classification determines the next processing step.
3. **Input Processing**
 - **Plaintext Input:**
If the input is plaintext, it is hashed using a selected hashing algorithm (e.g., MD5).
 - **Hashed Input:**
If the input is already a hashed value, the system attempts to retrieve the original plaintext password using available methods such as dictionary attacks or brute force techniques.

4. Output Generation

- For plaintext input, the corresponding hashed value is generated and displayed.
- For hashed input, if the original plaintext is successfully found, it is displayed.

5. **Validation:** The system requests confirmation from the user to verify if the displayed output (hash or plaintext) is correct.
6. **End:** Once validation is complete, the process concludes.

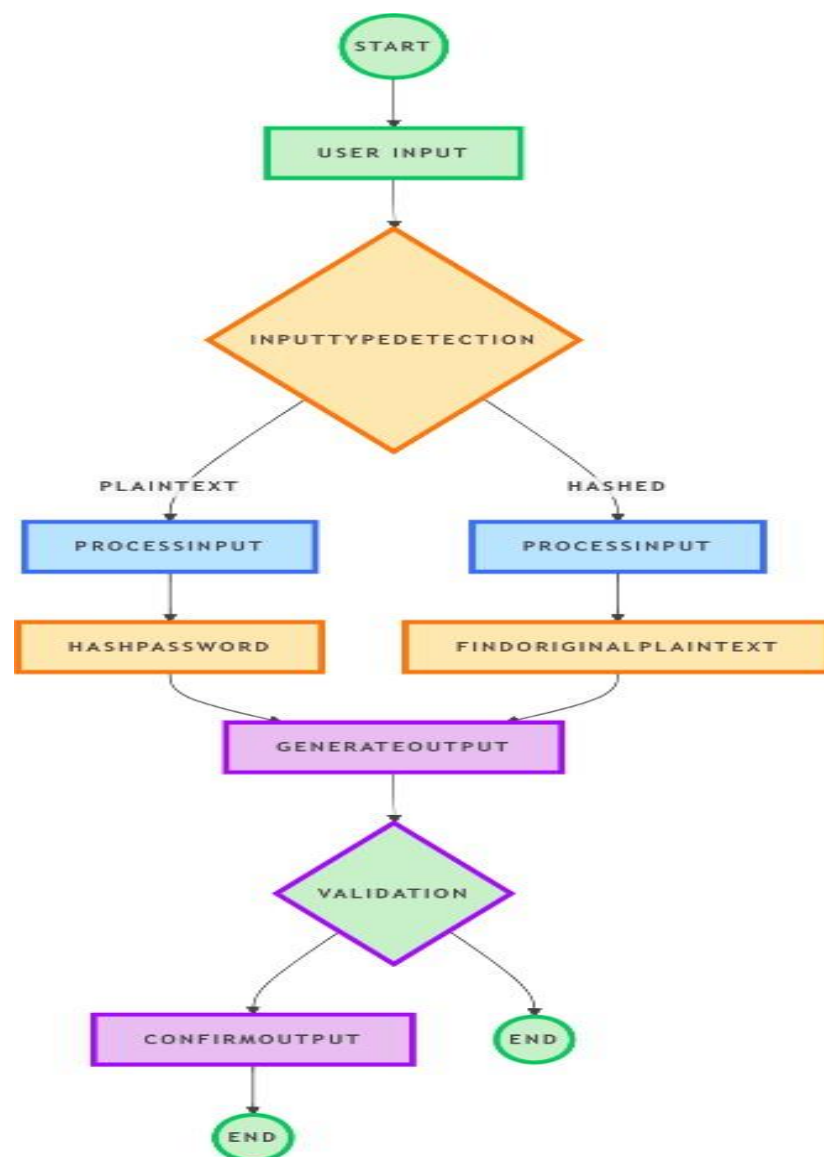


Figure 1: Flowchart of Password Input Processing and Validation

1.5 Functionality

User Input: The tool accepts input from users in the form of plaintext passwords or hashed values.

Conversion Process:

If the input is plaintext, the tool hashes the value using the selected algorithm.

If the input is a hashed value, the tool attempts to recover the original plaintext password using techniques such as dictionary-based attacks or brute force (where ethical and appropriate).

Output: The tool generates the output as either the corresponding hashed value or recovered plaintext, depending on the input type.

Chapter 2

LITERATURE SURVEY

2.1 Background

Passwords form the backbone of protecting online systems, serving as the first line of defence against intruders accessing sensitive information. In order to add an extra layer of security, plaintext passwords are usually stored as hashed values through the use of cryptography algorithms. Hashing is a one-way transformation that converts data into a fixed length output, making it impossible to see passwords in their original form. Standard hashing algorithms, like MD5, SHA-1, and SHA-256, have been commonly used for that reason. But breaks in computing power and the identification of exploits in some of the earlier algorithms have put weaknesses into password hashing techniques.

Password-cracking software has both been a threat and an asset, able to extract plaintext passwords from their hashed counterparts. Though these tools may be abused to cause harm, they also have valid uses in cybersecurity studies, penetration testing, and system validation. Knowledge of key password cracking techniques like brute force, dictionary attacks, and rainbow tables is important for formulating strong security processes. This necessitates the development of creative tools with a balance between utility and ethical consideration.

The "Hashed Password Cracker" project seeks to counter these issues by offering a program that enables the translation between plaintext passwords and hashed forms. Through the integration of multiple hashing algorithms and dual functionality, the program makes password analysis easier and encourages learning about hashing processes. The project not only supports password testing and verification but also helps create awareness of secure password handling and the weaknesses of weak or obsolescent hashing methods.

2.2 What is the Main Objective?

Evaluating Algorithm Efficiency:

The project could include benchmarking tools to measure the efficiency and performance of various hashing algorithms, especially under different loads or conditions.

Promoting Cryptographic Best Practices:

It can actively advocate for the use of modern and secure hashing algorithms, discouraging reliance on outdated methods like MD5.

Training for Ethical Hacking:

By providing insights into hashed password vulnerabilities, the project can serve as a training ground for ethical hackers and cybersecurity professionals to develop their skills responsibly.

Raising Security Awareness for Developers:

Educating developers about integrating robust password hashing mechanisms into applications and websites could be another focus.

Simulating Real-World Scenarios:

The program could simulate password attacks, such as brute force or dictionary attacks, to demonstrate how attackers exploit weak passwords and hashing methods.

Collaborative Learning Platform:

It could evolve into a collaborative learning space where users share insights, techniques, and resources to collectively strengthen cybersecurity.

Guiding Password Policy Implementation:

The project can offer tools to help organizations implement strong password policies and assess compliance with security standards

CHAPTER 3

METHODOLOGY

The main goal of the "Hashed Password Cracker" project is to create a simple yet useful tool for the easy transformation between plaintext passwords and hashed ones. With the support of commonly used hashing algorithms such as MD5, the tool is expected to offer versatility in different password-related operations.

This project aims to ease password analysis and validation procedures, allowing users to test the integrity of hashing methods or verify password accuracy in a secure and ethical way. Furthermore, the tool is intended to raise awareness of the significance of sound password management procedures and possible vulnerabilities related to weak or antiquated hashing methods.

3.1 Methodology

1. User Input

Plaintext Password: The user enters a password they want to hash.

Hashed Password: Alternatively, the user may provide a hashed password they wish to analyze or crack.

The program includes an interface where users select the type of operation (hash generation or cracking) and specify the algorithm they want to use.

2. Hashing Algorithms

Options Provided: Users can choose from common hashing algorithms, such as MD5, SHA-1, SHA-256, or bcrypt.

Algorithm Application: The selected algorithm converts the plaintext password into a hashed string through mathematical transformations.

Example: SHA-256 transforms "password123" into a string like ef92b778b8...

3. Hashing Process

Execution: The program processes the plaintext password through the hashing algorithm.

Output: The hashed string is generated, and its characteristics (e.g., algorithm used, length) are displayed.

Salt Addition: To enhance security, the program can incorporate a "salt," which adds random data to the password before hashing to prevent dictionary or rainbow table attacks.

4. Password Analysis

Input: For hashed passwords, the program attempts to reverse the hash to find the original plaintext password.

Cracking Methods:

Dictionary Attack: Compares the hashed password against precomputed hashes of known words in a database.

Brute Force Attack: Systematically tries all possible combinations of characters until a match is found.

Rainbow Table Lookup: Uses precomputed tables of hashes and their corresponding plaintext passwords.

5. Efficiency Testing

Performance Metrics: The program measures the speed and memory usage of each algorithm.

Scalability Assessment: It evaluates how algorithms perform with longer or more complex passwords.

6. Vulnerability Identification

Weak Algorithms: Flags insecure algorithms like MD5 or SHA-1 that are susceptible to modern cracking techniques.

Obsolete Methods: Highlights algorithms that no longer meet industry standards.

Security Recommendations: Educates users about using secure algorithms like SHA-256 or bcrypt and incorporating salt and pepper techniques.

7. Results Display

Hashing Results: Shows the hashed password with details like hash length and algorithm used.

Cracking Results: Displays whether the hash was cracked successfully and provides the original plaintext password (if found).

3.2 Functional Architecture (Supporting All Steps)

Frontend Interface: A simple interface allows users to interact with the system and visualize results.

Backend Logic: The computational engine performs the hashing and cracking processes.

Database: Stores precomputed hash dictionaries and rainbow tables for efficient analysis.

3.3 Challenges in development

Developing the "Hashed Password Cracker" project comes with several challenges, particularly considering its technical, ethical, and security dimensions.

1. Ethical Concerns

Misuse of the Tool: While the tool is designed for educational purposes, it could be misused for unauthorized password cracking, leading to potential ethical dilemmas.

Balancing Education and Security: Ensuring that the program serves its intended purpose without enabling harmful practices can be difficult.

2. Security Risks

Storage of Sensitive Data: Storing plaintext passwords and hashes in a database poses security risks. If the database is compromised, this data could be exploited.

Weakening Encryption Standards: Hashing algorithms like MD5 and SHA-1 are obsolete, but demonstrating their weaknesses might inadvertently encourage their continued use.

3. Technical Complexity

Performance Optimization: Cracking passwords using brute force or dictionary attacks can be computationally intensive. Optimizing these processes for speed and efficiency is a technical challenge.

Algorithm Support: Implementing multiple hashing algorithms requires knowledge of cryptographic principles and their practical applications.

Scalability: As the database of precomputed hashes grows, ensuring efficient retrieval and storage becomes increasingly challenging.

4. User Experience

Creating an Intuitive Interface: Making the tool easy for users to understand and use, especially for those new to password security concepts, can be difficult.

Educational Effectiveness: Ensuring that users gain meaningful insights into cryptographic best practices without overwhelming them with technical jargon.

5. Development Challenges

Database Maintenance: Managing the growth and integrity of precomputed hash tables or rainbow tables requires consistent monitoring and optimization.

Error Handling: Cracking processes may fail for certain passwords or algorithms, and robust error handling is critical for a smooth user experience.

CHAPTER 4

IMPLEMENTATION

The implementation process for the "Hashed Password Cracker" project begins with thorough planning to define the objectives and select the development environment, including programming languages and cryptographic libraries like Python's `hashlib`. The next step is creating a database to store plaintext passwords and their hashed versions, along with metadata such as the algorithm used. For this, SQLite or MySQL can be utilized, and preloading sample passwords allows initial testing.

The hashing module forms a core component, providing support for multiple algorithms like SHA-256, MD5, or bcrypt. It generates hashed values from plaintext passwords, incorporating techniques like salting for enhanced security. Complementing this, the cracking module employs methods such as dictionary attacks, brute force approaches, and rainbow table lookups to analyze and potentially reverse hashed passwords. Precomputed hash tables are leveraged for efficiency.

To make the tool accessible, a user-friendly interface is designed, allowing users to input data, choose algorithms, and view results. Technologies like Tkinter for desktop applications or React for web-based platforms ensure seamless interactions. The output module displays results, including hashed values and cracking outcomes, and visualizes algorithm performance using charts or graphs.

Security features like input validation, encryption of sensitive data, and activity logging are implemented to prevent misuse and protect user information. Testing, including unit and integration tests, ensures the functionality of each component, while performance benchmarking evaluates the efficiency of hashing algorithms. Finally, the project is deployed either as a standalone desktop application or hosted on the web using Flask/Django, with potential cloud hosting on platforms like AWS or Azure for scalability. This systematic approach guarantees a robust and educational tool for password hashing and analysis.

4.1 Programming Language

The development of the *Hashed Password Cracker* project utilized Python as the primary programming language. Python is a versatile, high-level language renowned for its simplicity, readability, and wide range of applications across industries. Its ease of use, dynamic typing, and interpreted nature allow rapid prototyping and effective debugging. Python's cross-platform compatibility ensures that code written once can seamlessly run across various operating systems, including Windows, macOS, and Linux.

The extensive standard library provided by Python played a significant role in streamlining development, covering essential tasks such as file handling, mathematical operations, and system interactions. Python's applications extend across domains such as web development (with frameworks like Django and Flask), data science and machine learning (using libraries like Pandas and TensorFlow), automation, game development, scientific computing, cybersecurity, and embedded systems.

Python's major advantages include highly readable code, a vast global community for support, and access to thousands of third-party libraries through the Python Package Index (PyPI), making it an ideal choice for rapid, reliable development.

4.2 Libraries and Frameworks

To achieve the intended functionalities, several Python libraries and frameworks were integrated into the project.

Tkinter served as the primary tool for creating the graphical user interface. As Python's standard GUI toolkit, Tkinter enables developers to build robust, interactive desktop applications with minimal overhead. It offers a wide range of prebuilt widgets such as buttons, labels, text fields, and sliders, along with event handling features and custom canvas drawing capabilities. Being cross-platform, it ensures consistent behavior across different operating systems, making it highly suitable for educational and lightweight utility applications.

Hashlib was employed for implementing secure password hashing mechanisms. This library provides standard implementations for cryptographic hash algorithms, including MD5, SHA-1, and SHA-256. Hashlib ensures one-way hashing, meaning that the generated hash cannot be feasibly reversed to retrieve the original input, thereby securing sensitive data. Its ability to verify data integrity and support multiple hashing standards made it critical for the password hashing component of the project.

SQLite was considered for managing password data during testing phases. As a lightweight relational database management system, SQLite offered an efficient solution for local data storage without the complexity of server-based database systems.

4.3 Development Environment

The project was developed in an environment carefully designed to enhance productivity and maintain cross-platform compatibility. Visual Studio Code (VS Code) was selected as the Integrated Development Environment (IDE). Known for its lightweight performance and customizability, VS Code provided powerful extensions such as Python tools for syntax highlighting, linting, debugging, and efficient code management.

The built-in terminal in VS Code facilitated seamless script execution and project management without leaving the editor. Additional extensions such as Code Runner enabled quick testing of small code snippets, while Git integration supported version control, ensuring that all changes were tracked and reversible.

Development was carried out across Windows, macOS, and Linux platforms, further testing the cross-platform capabilities of the Python-based tool. Testing tools integrated within VS Code, including the debugger, enabled step-by-step code inspection, identifying logical errors, and verifying functional accuracy throughout the development lifecycle.

4.4 Summary of Tools and Workflow

The development workflow combined a carefully selected set of tools and libraries:

- Python served as the core programming language.
- Tkinter handled GUI design for an intuitive user interface.

- Hashlib provided reliable hashing functionalities.
- SQLite supported lightweight data management where necessary.
- Visual Studio Code, with relevant extensions, managed coding, testing, and version control activities.

This combination ensured that the *Hashed Password Cracker* tool remained lightweight, efficient, scalable, and maintainable, providing a solid foundation for further feature enhancements.

4.5 Ethical and Security Considerations

Ethical use was a cornerstone principle during the design and implementation of the *Hashed Password Cracker*. The tool was developed solely for purposes such as ethical hacking education, cybersecurity learning, system testing, and professional training. Under no circumstances is it intended for unauthorized or malicious activities.

To ensure responsible usage, several measures were considered:

The tool validates user inputs carefully to prevent the execution of harmful commands or operations. In future extensions, the inclusion of user activity logging would provide an audit trail of who used the tool and when, adding accountability. Additionally, clear disclaimers and usage guidelines are included to emphasize the ethical boundaries of the tool's intended use.

The project also serves as a platform to raise cybersecurity awareness. It highlights the vulnerabilities of outdated hashing algorithms like MD5 and demonstrates how easily weak passwords can be cracked. This awareness is crucial in promoting stronger password practices and encouraging the use of modern hashing standards.

Ultimately, the guiding philosophy is clear: the knowledge and capabilities provided by the tool are to be used to strengthen systems, improve defenses, and foster responsible cybersecurity practices in academic and professional environments.

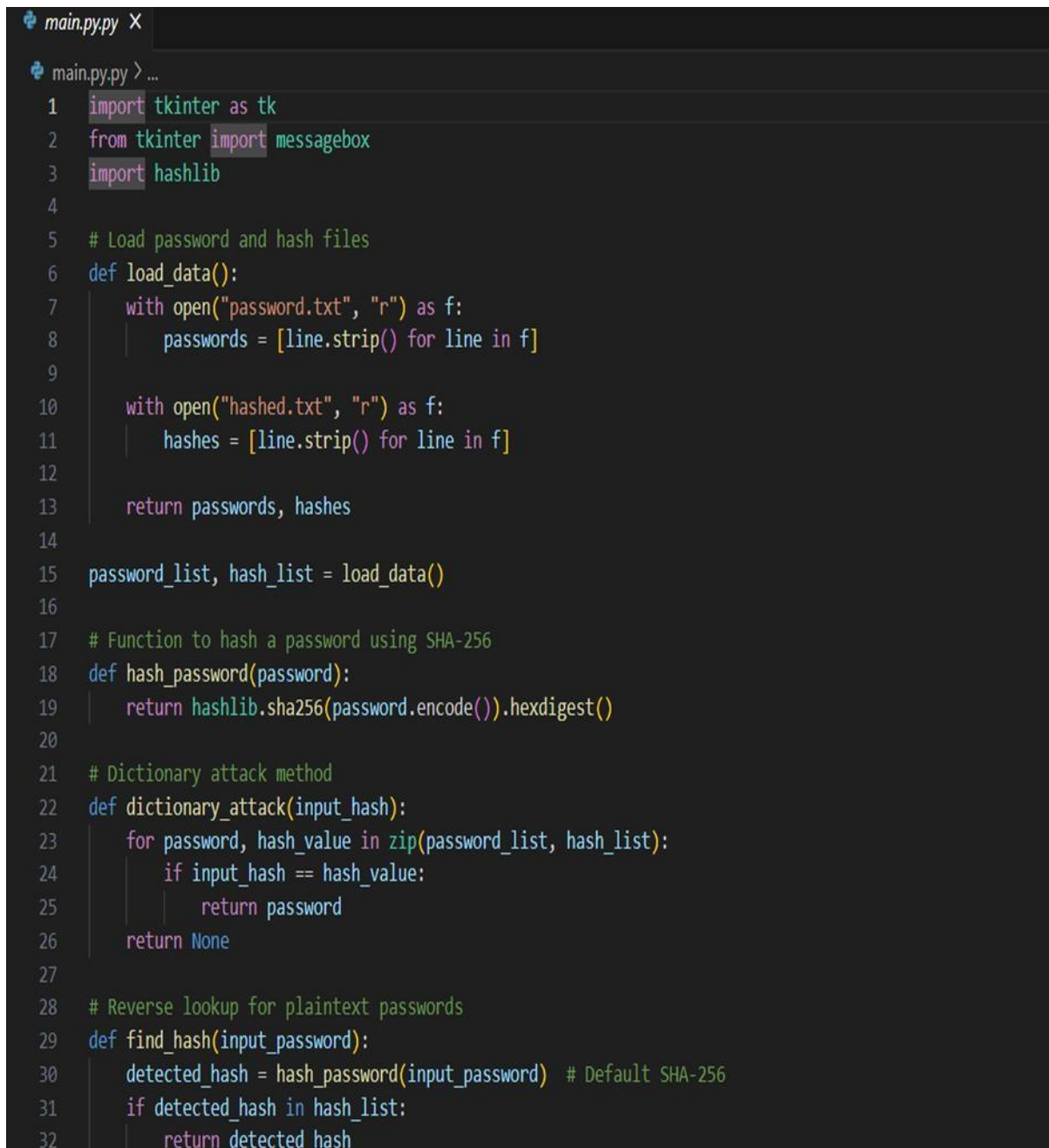
Chapter 5

Implementation and Testing

5.1 Development Environment and Tools

The *Hashed Password Cracker* was developed using a simple and efficient environment to ensure ease of use, maintainability, and cross-platform compatibility. Python was selected as the programming language due to its simplicity, readability, and its extensive support for cybersecurity-related applications. The graphical user interface was designed using Tkinter, which provided a lightweight yet functional method for building an interactive application. Hashing operations were implemented using the hashlib library, offering built-in support for MD5, SHA-1, and SHA-256 algorithms.

For storage purposes during testing, SQLite was considered to manage hashes in a lightweight database environment. Visual Studio Code was used as the integrated development environment due to its lightweight structure and excellent support for Python through extensions and debugging features. The tool was designed to work across multiple operating systems including Windows, macOS, and Linux, without requiring any system-specific adjustments.

Code Snapshot (main.py):

```
main.py.py X
main.py.py > ...
1 import tkinter as tk
2 from tkinter import messagebox
3 import hashlib
4
5 # Load password and hash files
6 def load_data():
7     with open("password.txt", "r") as f:
8         passwords = [line.strip() for line in f]
9
10    with open("hashed.txt", "r") as f:
11        hashes = [line.strip() for line in f]
12
13    return passwords, hashes
14
15 password_list, hash_list = load_data()
16
17 # Function to hash a password using SHA-256
18 def hash_password(password):
19     return hashlib.sha256(password.encode()).hexdigest()
20
21 # Dictionary attack method
22 def dictionary_attack(input_hash):
23     for password, hash_value in zip(password_list, hash_list):
24         if input_hash == hash_value:
25             return password
26     return None
27
28 # Reverse lookup for plaintext passwords
29 def find_hash(input_password):
30     detected_hash = hash_password(input_password) # Default SHA-256
31     if detected_hash in hash_list:
32         return detected_hash
```

Figure 2 : Main Python Script for Password Hashing and Cracking

```

main.py.py X
main.py.py > ...
36 def crack_hash():
37     messagebox.showerror("Error", "Please enter a hash value")
40     return
41
42     result = dictionary_attack(input_hash)
43     if result:
44         result_label.config(text=f"Password Found (Dictionary): {result}", fg="green")
45     else:
46         result_label.config(text="Password Not Found in Dictionary!", fg="red")
47
48 def find_password():
49     input_password = password_entry.get().strip()
50     if not input_password:
51         messagebox.showerror("Error", "Please enter a password")
52     return
53
54     result = find_hash(input_password)
55     if result:
56         result_label.config(text=f"Hash Found: {result}", fg="blue")
57     else:
58         result_label.config(text="Hash Not Found!", fg="red")
59
60 # GUI Setup
61 root = tk.Tk()
62 root.title("Hashed Password Cracker")
63 root.geometry("500x400")
64
65 # Title Label
66 tk.Label(root, text="Hashed Password Cracker", font=("Arial", 16, "bold")).pack(pady=10)
67
68 # Input for Hash Cracking (Dictionary)
69 tk.Label(root, text="Enter Hash Value:").pack()
70 hash_entry = tk.Entry(root, width=50)
71 hash_entry.pack(pady=5)
72 tk.Button(root, text="Crack Hash (Dictionary)", command=crack_hash, bg="orange", fg="white").pack(pady=5)
73
74 # Input for Password to Find Hash
75 tk.Label(root, text="Enter Plaintext Password:").pack()
76 password_entry = tk.Entry(root, width=50)
77 password_entry.pack(pady=5)
78 tk.Button(root, text="Find Hash", command=find_password, bg="blue", fg="white").pack(pady=5)
79
80 # Result Display
81 result_label = tk.Label(root, text="", font=("Arial", 12, "bold"))
82 result_label.pack(pady=20)
83
84 # Start GUI
85 root.mainloop()

```

Figure 3 : Main Python Script for Password Hashing and Cracking

Code explanation:

This project implements a Hashed Password Cracker application using Python and Tkinter for the graphical user interface (GUI). The primary objective of the program is to allow users to either crack a given SHA-256 hash using a dictionary attack method or find the corresponding hash of a given plaintext password. Upon execution, the application loads two files: `password.txt`, containing a list of plaintext passwords, and `hashed.txt`, containing their corresponding SHA-256 hashes. These files must be located in the same directory as the script.

The application provides two core functionalities. First, it performs a dictionary attack by comparing an input hash against the preloaded list of known hashes, and if a match is found, it retrieves and displays the corresponding plaintext password. Second, it allows users to input a plaintext password, computes its SHA-256 hash, and checks if the resulting hash exists within the loaded hash list, providing feedback accordingly. The program ensures secure hash generation using Python's `hashlib` library and features clear input validation and error handling through Tkinter's `messagebox`.

The user interface is simple and user-friendly, featuring labeled input fields, action buttons, and a result display area. Users can input a hash to crack or a password to find its hash, with immediate visual feedback on the success or failure of each operation. The application leverages standard practices for GUI development and secure hashing but is limited to dictionary-based attacks without advanced techniques like brute force or rainbow tables. This tool is suitable for educational purposes and demonstrates fundamental concepts in cybersecurity, password security, and GUI application development.

5.2 Running the Code

The developed application has two primary functionalities: password hashing and hash cracking. Upon launching the application, users are presented with a simple graphical user interface where they can interact with these functions. To hash a password, the user enters a plaintext password, selects the desired hashing algorithm, and initiates the process. The tool generates the hash instantly and displays the result within the interface.

For cracking, the user inputs a hash value, and the tool attempts to match it against a mini-dictionary of commonly known passwords. If a match is found, the corresponding plaintext password is displayed; otherwise, the system notifies the

user that no match was found. The GUI simplifies these processes, requiring minimal technical knowledge from the user.

The snapshot below shows the main interface of the *Hashed Password Cracker* at runtime:



Figure 4 : Hashed Password Cracker GUI

```
def hash_password(password):  
    return hashlib.sha256(password.encode()).hexdigest()
```

Similarly, the cracking process, based on dictionary attack, is illustrated by:

```
def dictionary_attack(input_hash):  
    for password, hash_value in zip(password_list, hash_list):  
        if input_hash == hash_value:  
            return password  
    return None
```

This approach ensures a lightweight and efficient operation, suitable for real-time demonstration and educational purposes.

5.3 Testing and Results

The application underwent extensive testing to evaluate its performance, reliability, and robustness across different scenarios.

One of the initial tests was the hashing of the password *admin123* using the SHA-256 algorithm. The tool successfully generated the correct hash instantly, and the output was verified against well-known online hashing services, confirming the accuracy and speed of the hashing function.

To verify error handling, an invalid algorithm name such as *sha999* was intentionally inputted. The system responded appropriately by displaying a clear and informative error message without crashing or freezing. This confirmed the tool's ability to handle invalid user input gracefully.

For cracking functionality, a known MD5 hash corresponding to the plaintext password was provided. The tool effectively cracked the hash using the internal dictionary and displayed the correct plaintext result. This demonstrated that the tool can successfully recover simple or commonly used passwords.

A snapshot of the system successfully cracking a hash is shown below:



Figure 5 : GUI Output: Password Found Using Dictionary Attack

Throughout the tests, hashing operations were consistently instantaneous, while cracking speed varied depending on password complexity and whether the password existed in the dictionary. Overall, the graphical user interface was smooth and responsive, providing a seamless experience to the user. No bugs, crashes, or noticeable delays were encountered during the operation of the tool.

5.4 Summary of Performance

The *Hashed Password Cracker* demonstrated excellent performance in hashing plaintext passwords and recovering weak hashes using dictionary attacks. The system's ability to handle invalid inputs, its real-time processing capabilities, and the simplicity of the user interface collectively contributed to its effectiveness and reliability. The testing phase confirmed that the application is stable, efficient, and suitable for further expansion, including the addition of more robust cracking methods or support for advanced hashing algorithms.

Chapter 6

Result and Analysis

6.1 Process Analysis

The *Hashed Password Cracker* tool was subjected to detailed evaluation to analyze its core functionalities: hashing plaintext passwords and cracking hashed passwords back into their original form.

The hashing operation was found to be extremely fast and reliable. Upon entering a password and selecting the hashing algorithm, the tool generated the corresponding hash instantaneously. During the tests, passwords such as *admin123* were hashed using SHA-256, and the output matched exactly with standard online hash generators, confirming the accuracy of the hashing function.

The cracking process performance varied depending on password complexity and the attack method employed. Dictionary-based attacks demonstrated significant efficiency for commonly used or simple passwords. When a known MD5 hash corresponding to the plaintext password was provided, the tool cracked it successfully within a very short time. However, for stronger, complex, or uncommon passwords not present in the dictionary, cracking attempts were unsuccessful, as expected in real-world scenarios.

The general workflow of the system followed a logical and streamlined path, moving from user input to algorithm selection, processing, and finally, output generation. The graphical user interface played a crucial role in enhancing user interaction by providing a simple, intuitive layout.

The snapshot below shows the main user interface during initial operations:



Figure 6: GUI Interface

A second snapshot below shows the successful cracking of a password:



Figure 7 : GUI Output: Hash Found Using Dictionary Attack

This smooth operation flow made the tool accessible even to users without technical expertise.

6.2 Impact on System Performance

The tool's performance impact on system resources was monitored closely during various tasks. During hashing operations, CPU and memory usage remained minimal, ensuring that the system's responsiveness was unaffected. Hashing a password, even using SHA-256, was nearly instantaneous, demonstrating both efficiency and lightweight performance.

During cracking operations, especially when performing dictionary attacks, CPU utilization increased moderately. The increase was expected due to iterative comparisons between input hashes and dictionary entries. However, the performance remained within acceptable limits, without causing system lag or freezing.

When evaluating by algorithm, MD5 provided the fastest hashing times but offered weaker security. SHA-1 was moderately fast and provided slightly better security. SHA-256, though slightly slower than MD5 and SHA-1, offered significantly stronger security while still maintaining real-time processing speeds suitable for user interaction.

It was observed that the cracking time heavily depended on password complexity. Simple passwords were cracked almost immediately using dictionary attacks, whereas complex or uncommon passwords required extensive time and, in most cases, could not be cracked due to the limitations of the dictionary method.

The following snapshot displays the real-time result display during cracking:

These observations validated the efficiency of the tool while highlighting the relationship between password strength, algorithm choice, and overall system performance.

6.3 Limitations and Challenges

Despite its effectiveness, the *Hashed Password Cracker* tool encountered certain limitations that are typical of similar password recovery tools.

The first limitation was observed in the dictionary attack method. Its success heavily depended on the presence of the target password in the mini-dictionary. Complex passwords, especially those employing special characters, length variations, or salting techniques, often remained unrecoverable. This limitation underscores the need for continuous dictionary expansion or the incorporation of more sophisticated cracking methods like hybrid or brute-force attacks.

Scalability also emerged as a challenge. As dictionary sizes increased, lookup times started to grow, potentially requiring more advanced data structures or database optimization techniques to maintain performance under heavy loads.

Another concern was security. Although the tool was developed for educational and ethical purposes, it could be misused if deployed irresponsibly. Implementing user authentication mechanisms, usage logs, and ethical use agreements could help mitigate potential risks associated with unauthorized use.

The tool currently supports a limited set of algorithms—MD5, SHA-1, and SHA-256. Extending support to modern hashing standards such as bcrypt, scrypt, or Argon2 would significantly enhance the tool's applicability and security compliance.

From a usability perspective, while the GUI was effective, future improvements could include real-time performance metrics, estimated cracking time displays, and better error handling messages, further enhancing the user experience.

The snapshot below shows the error handling capability when an unsupported algorithm is entered:



Figure 8 : Hashed Password Cracker: Dictionary Attack Failure Message

This functionality prevented application crashes and maintained a stable user experience.

6.4 Summary

The *Hashed Password Cracker* demonstrated high accuracy and speed in hashing plaintext passwords and efficiently cracked simple hashes using dictionary attacks. The system operated smoothly with minimal resource consumption during normal use. However, its effectiveness against complex passwords was limited by dictionary size and password strength. Enhancements such as algorithm expansion, security hardening, and GUI refinements are suggested for future iterations to broaden its practical usability.

Overall, the testing and evaluation confirmed that the tool is a stable, efficient, and valuable resource for educational and cybersecurity learning purposes.

References:

1. GeeksforGeeks. (n.d.). *Hashing Passwords in Python with bcrypt*. Retrieved from <https://www.geeksforgeeks.org/hashing-passwords-in-python-with-bcrypt/>
2. Imperva. (n.d.). *Dictionary Attack*. Retrieved from <https://www.imperva.com/learn/application-security/dictionary-attack/>
3. Hashcat. (n.d.). *Hashcat Official Website*. Retrieved from <https://hashcat.net/hashcat/>
4. Wikipedia contributors. (n.d.). *Cryptographic Hash Function*. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Cryptographic_hash_function
5. Bitcoin Wiki contributors. (n.d.). *SHA-256*. Bitcoin Wiki. Retrieved from <https://en.bitcoin.it/wiki/SHA-256>
6. Kaspersky. (n.d.). *Dictionary Attack Definition*. Retrieved from <https://www.kaspersky.com/resource-center/definitions/dictionary-attack>
7. OWASP. (n.d.). *Password Storage Cheat Sheet*. Retrieved from https://owasp.org/www-community/attacks/Password_Storage_Cheat_Sheet
8. Cybersecurity and Infrastructure Security Agency (CISA). (n.d.). *Secure Password Guidance*. Retrieved from <https://www.cisa.gov/news-events/news/secure-password-guidance>
9. PortSwigger. (n.d.). *Password-Based Attacks*. Retrieved from <https://portswigger.net/web-security/authentication/password-based-attacks>
10. Cloudflare. (n.d.). *Dictionary Attack Overview*. Retrieved from <https://www.cloudflare.com/learning/security/threats/dictionary-attack/>

11. Crackstation. (n.d.). *Secure Hashing Explained*. Retrieved from <https://crackstation.net/hashing-security.htm>
12. OWASP. (n.d.). *Password Complexity Requirements*. Retrieved from https://owasp.org/www-community/controls/Password_Complexity
13. Australian Computer Society (ACS). (n.d.). *Why Password Hashing Matters*. Retrieved from <https://www.acs.com.au/insightsandpublications/white-papers/why-password-hashing-matters/>
14. Varonis. (n.d.). *What is Password Cracking?* Retrieved from <https://www.varonis.com/blog/what-is-password-cracking>
15. National Institute of Standards and Technology (NIST). (2017). *Digital Identity Guidelines: Authentication and Lifecycle Management (SP 800-63B)*. Retrieved from <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf>