

## Table of contents

<b>S. No</b>	<b>Title</b>	<b>Page No.</b>
<b>1.</b>	<b>Introduction</b>	<b>2-3</b>
	1.1 Importance	3-5
	1.2 Overview	5-6
	1.3 Objectives and goals	6-8
	1.4 Key components	8-10
<b>2.</b>	<b>Background</b>	<b>10</b>
	2.1 Hash cracking	10-11
	2.2 Hash functions	11-12
	2.3 Importance of Hash crack and Hash function	13-15
	2.4 Mechanism	15-16
<b>3.</b>	<b>Training and Undertaken</b>	<b>17</b>
	3.1 Case Diagram	17-18
	3.2 Methodology	18-20
	3.3 Hardware, language and software requirements	20-22
	3.4 Technology used	22-26
<b>4.</b>	<b>Code Snippets</b>	<b>27</b>
	4.1 Codes	27-36
	4.2 Source code file	36
<b>5.</b>	<b>Result and Testing</b>	<b>37</b>
	5.1 Screenshot	37-38
	5.2 Conclusion	38-39
	5.3 Significance and impact	39-41
	5.4 Future scope	41-43
<b>6.</b>	<b>References</b>	<b>43</b>

# 1. Introduction

In an era defined by digital transformation and ubiquitous connectivity, the protection of sensitive information and digital assets is of paramount importance. One fundamental aspect of cybersecurity revolves around safeguarding user credentials and authentication mechanisms, ensuring that unauthorized individuals cannot gain access to privileged resources. Central to this endeavor is the use of cryptographic hash functions, which convert plaintext passwords into unique, irreversible hash values. However, despite their inherent security benefits, hashed passwords are not immune to exploitation, particularly when weak or easily guessable passwords are employed.

The Hash Cracker project emerges as a response to the critical need for robust password security and the imperative to assess and reinforce digital defenses against malicious actors. Developed as a versatile and efficient tool for cybersecurity professionals, forensic analysts, and researchers alike, Hash Cracker empowers users to analyze, audit, and potentially recover plaintext passwords from hashed values. By leveraging a combination of cryptographic techniques, password dictionaries, and computational algorithms, the project endeavors to identify vulnerabilities, strengthen security postures, and enhance digital resilience in the face of evolving threats.

This report serves as a comprehensive exploration of the Hash Cracker project, delving into its design, functionality, applications, and broader implications within the realm of cybersecurity. Through an in depth analysis of its features, methodologies, and real world use cases, this document aims to elucidate the significance and value proposition of the Hash Cracker tool in contemporary cybersecurity practices.

Over the course of this report, we will examine the underlying principles of cryptographic hashing, the challenges associated with password security, and the role of hash cracking tools in security assessments and incident response. Furthermore, we will explore the technical architecture of the Hash Cracker project, its integration with web technologies, and the practical considerations involved in its deployment and utilization.

Through a multidimensional lens encompassing technological innovation, security best practices, and collaborative community engagement, this report seeks to underscore the importance of the Hash Cracker project in fortifying digital defenses, promoting security awareness, and advancing the collective resilience of organizations and individuals in the face of cyber threats.

## **1.1. Importance**

The importance of the Hash Cracker project lies in its contribution to the field of cybersecurity and digital forensics. Here are several key points highlighting its significance:

### **1. Enhanced Security Analysis:**

- The Hash Cracker tool provides security professionals, researchers, and forensic analysts with a valuable resource for analyzing the strength of hashed passwords and cryptographic protections.
- By efficiently cracking hashed values, the tool helps identify weak passwords and potential security vulnerabilities in systems and applications.

### **2. Password Recovery and Data Access:**

- In scenarios where access to plaintext passwords is necessary for legitimate purposes, such as during forensic investigations or penetration testing engagements, the Hash Cracker tool facilitates password recovery from hashed values.
- This capability enables authorized users to regain access to encrypted data or systems when legitimate access credentials are unavailable.

### **3. Penetration Testing and Red Teaming:**

- Penetration testers and red teamers can leverage the Hash Cracker tool to simulate real world attacks and assess the effectiveness of password security measures within organizations.

- By identifying and exploiting weak passwords, security professionals can provide actionable recommendations for improving password policies and mitigating security risks.

#### **4. Educational and Research Purposes:**

- The Hash Cracker project serves as an educational tool for students, academics, and aspiring cybersecurity professionals to learn about hashing algorithms, password cracking techniques, and cryptographic principles.
- Researchers can use the tool to conduct experiments, analyze hash cracking performance, and explore novel approaches to password security and cryptographic analysis.

#### **5. Open Source Community Contribution:**

- As an open source project, the Hash Cracker tool fosters collaboration and knowledge sharing within the cybersecurity community.
- Contributors can enhance the tool's functionality, improve its performance, and adapt it to different use cases, thereby benefiting users worldwide.

#### **6. Compliance and Regulatory Requirements:**

- In regulated industries such as finance, healthcare, and government, compliance with security standards and data protection regulations is paramount.
- The Hash Cracker tool can assist organizations in meeting compliance requirements by identifying and addressing weaknesses in password security practices.

#### **7. Rapid Incident Response:**

- During cybersecurity incidents such as data breaches or unauthorized access, the ability to quickly analyze and recover hashed passwords is essential for mitigating further damage and restoring system integrity.

- And efficient means of recovering plaintext passwords from hashed values.

## 1.2. Project Overview

The Hash Cracker project represents a pioneering initiative aimed at addressing the pervasive challenge of password security in digital environments. Grounded in the principles of cryptography, cybersecurity, and computational intelligence, the project endeavors to provide a robust and efficient solution for analyzing and decrypting hashed passwords.

At its core, Hash Cracker is designed to assist cybersecurity professionals, forensic analysts, and researchers in identifying vulnerabilities within authentication systems, auditing password policies, and recovering plaintext passwords from hashed values. By leveraging a combination of cryptographic techniques, password dictionaries, and computational algorithms, the project empowers users to assess the strength of their passwords, detect potential security weaknesses, and bolster their digital defenses against malicious actors.

The project's overarching objectives can be summarized as follows:

- 1. Password Security Assessment:** Hash Cracker enables users to evaluate the security of their passwords by analyzing their corresponding hashed values. By comparing hashed passwords against known dictionaries and computational algorithms, users can identify weak or easily guessable passwords that may be susceptible to brute force attacks or dictionary based attacks.
- 2. Password Recovery:** In scenarios where users have forgotten their passwords or need to gain access to encrypted data, Hash Cracker offers the capability to recover plaintext passwords from hashed values. By systematically iterating through password dictionaries and applying cryptographic hash functions, the project attempts to reverse engineer hashed passwords and unveil their original plaintext counterparts.

- 3. Security Auditing:** Beyond individual password analysis, Hash Cracker facilitates comprehensive security audits of authentication systems and password repositories. By analyzing the distribution of hashed passwords, identifying common patterns or vulnerabilities, and generating detailed reports, users can gain valuable insights into the overall security posture of their digital assets.
- 4. Incident Response:** In the event of a security breach or unauthorized access, Hash Cracker serves as a valuable tool for forensic analysis and incident response. By analyzing compromised passwords, reconstructing attack vectors, and tracing malicious activity back to its source, cybersecurity professionals can mitigate the impact of security incidents and prevent future breaches.
- 5. Research and Development:** As an opensource project, Hash Cracker fosters collaboration, innovation, and knowledge sharing within the cybersecurity community.

By providing access to its source code, documentation, and development resources, the project encourages contributions from researchers, developers, and cybersecurity enthusiast's worldwide, driving continuous improvement and refinement of its capabilities.

### **1.3. Objectives and Goals**

The objectives and goals of the hash cracker project are outlined below:

- 1. Developing a Functional Tool:** The primary objective of the project is to develop a functional hash cracker tool that can efficiently decrypt hashed passwords. The tool should be capable of handling multiple encryption

algorithms, including MD5, SHA-1, and SHA-256, and should provide accurate results within a reasonable time frame.

- 2. User-Friendly Interface:** Another goal of the project is to create a user-friendly interface for the hash cracker tool. The interface should be intuitive and easy to navigate, allowing users to input hashed passwords and select wordlists with minimal effort. Clear instructions and feedback should be provided to guide users through the decryption process.
- 3. Support for Multiple Wordlists:** The hash cracker tool should support the use of multiple wordlists, including popular options such as rockyou.txt and custom wordlists. Users should have the flexibility to choose the wordlist that best suits their needs, whether they are performing a targeted or comprehensive password cracking operation.
- 4. Efficient Hash Cracking Algorithms:** The project aims to implement efficient hash cracking algorithms that can quickly identify matching passwords within large wordlists. Optimization techniques such as parallel processing, memory management, and algorithmic improvements may be employed to enhance the speed and performance of the hash cracker tool.
- 5. Error Handling and Reporting:** Robust error handling and reporting mechanisms are essential aspects of the project. The hash cracker tool should be able to detect and handle various types of errors, such as file not found errors, invalid hash formats, and unexpected exceptions. Clear error messages should be displayed to users to help them troubleshoot issues effectively.
- 6. Security and Privacy Considerations:** Security and privacy considerations are paramount in the development of the hash cracker tool. Measures should be taken to ensure that the tool is secure and does not pose any risks to user data or system

integrity. Additionally, the tool should respect user privacy and confidentiality by handling sensitive information responsibly.

**7. Documentation and Support:** The project aims to provide comprehensive documentation and support resources for users of the hash cracker tool. This includes detailed usage instructions, troubleshooting guides, and FAQs to help users make the most of the tool and address any issues they may encounter.

**8. Community Engagement:** Finally, the project seeks to foster a community of users and contributors who are interested in hash cracking, cryptography, and cybersecurity. Open communication channels, such as forums, mailing lists, and social media platforms, may be established to encourage collaboration, feedback, and knowledge sharing among community members.

## **1.4. Key Components**

**1. Hashing Algorithms:** The project incorporates multiple hashing algorithms, including MD5, SHA1, and SHA256, which serve as the foundation for password encryption and decryption. These algorithms are implemented using cryptographic libraries to ensure accuracy and reliability.

**2. Wordlist Management:** Hash Cracker includes a robust wordlist management system that allows users to specify the location of password dictionaries such as rockyou.txt and wordlist.txt. These wordlists contain thousands of common passwords and phrases, enabling efficient password cracking through dictionary attacks.



- 3. User Interface:** The project features a user-friendly web interface built using the Flask web framework, HTML, and JavaScript. The interface allows users to input hashed passwords and select the wordlist to be used for cracking. It provides real-time feedback on the cracking progress and displays the results once the password is successfully cracked.
- 4. Cracking Engine:** At the core of Hash Cracker is a powerful cracking engine responsible for analyzing hashed passwords against entries in the specified wordlist. The engine iterates through each word in the wordlist, computes its hash using the selected algorithm, and compares it with the target hash. Upon finding a match, the engine identifies the plaintext password and the corresponding hashing algorithm.
- 5. Error Handling and Reporting:** To ensure robustness and reliability, the project includes comprehensive error handling mechanisms. It detects and handles various exceptions, such as file not found errors and invalid input, gracefully informing users of any issues encountered during the cracking process. Additionally, Hash Cracker generates detailed reports summarizing the cracking results and any encountered errors for further analysis.
- 6. Security Measures:** Hash Cracker incorporates several security measures to protect user data and ensure the integrity of the cracking process. This includes input validation to prevent injection attacks, secure storage of hashed passwords, and adherence to best practices for handling sensitive information. Furthermore, the project adheres to ethical guidelines and legal regulations governing password cracking activities.

- 7. Documentation and Help Resources:** To support users in effectively utilizing the tool, Hash Cracker provides comprehensive documentation and help resources. This includes user guides, tutorials, and FAQs covering various aspects of password cracking, usage instructions, and troubleshooting tips. Additionally, the project fosters a community driven support model, allowing users to seek assistance from experienced practitioners and contributors.
- 8. Scalability and Performance:** Hash Cracker is designed to be scalable and capable of handling largescale password cracking tasks efficiently. It leverages multithreading and parallel processing techniques to maximize performance and optimize resource utilization, enabling rapid analysis of extensive wordlists and hash datasets.
- 9. Open Source Collaboration:** As an opensource project, Hash Cracker encourages collaboration and contributions from the cybersecurity community. It provides access to its source code repository, issue tracker, and developer documentation, inviting feedback, suggestions, and enhancements from developers, researchers, and security enthusiasts worldwide. This collaborative approach fosters innovation, transparency, and continuous improvement of the project's capabilities.

## **2. Background**

### **2.1. Hash Cracking**

Hash cracking refers to the process of decrypting hashed data to reveal the original plaintext information. In cryptography, hashing is a one way function that takes an input (or "message") and produces a fixed size string of characters, known as the

hash value or digest. This hash value serves as a unique representation of the input data and is typically used for data integrity verification, password storage, and digital signatures.

Hash cracking involves attempting to reverse this process by generating potential inputs (such as passwords or phrases), hashing them using the same algorithm used to generate the original hash, and comparing the resulting hash values with the target hash. If a match is found, it indicates that the original input has been successfully recovered

Hash cracking techniques commonly involve dictionary attacks, where a list of commonly used passwords or phrases (known as a wordlist) is systematically tested against the hashed data. Other methods include brute force attacks, which systematically generate all possible combinations of characters until a match is found, and rainbow table attacks, which use precomputed tables of hash values to speed up the cracking process.

Hash cracking can be used for both legitimate purposes, such as recovering forgotten passwords or testing the security of cryptographic implementations, as well as malicious activities, including unauthorized access to systems or data breaches. As such, it is essential for organizations and individuals to employ strong password policies, secure hashing algorithms, and other security measures to protect against hash cracking attacks.

## **2.2. Hash Functions:**

A hash function is a mathematical algorithm that takes an input (or "message") of arbitrary size and produces a fixed size string of characters, known as the hash value or digest. The key properties of a hash function include determinism, where the same input always produces the same output, and collision resistance, where it is computationally infeasible to find two different inputs that produce the same output hash.

Hash functions are widely used in cryptography and computer science for various purposes, including data integrity verification, digital signatures, password hashing, and data indexing. They play a crucial role in ensuring the security and reliability of many cryptographic protocols and security mechanisms.

Common characteristics of hash functions include:

1. **Determinism:** Given the same input, a hash function will always produce the same output.
2. **Fixed Output Size:** Regardless of the input size, the output of a hash function is always of fixed length.
3. **Fast Computation:** Hash functions are designed to be computationally efficient, allowing them to process large amounts of data quickly.
4. **Preimage Resistance:** It should be computationally infeasible to determine the input from its corresponding hash value.
5. **Collision Resistance:** It should be computationally infeasible to find two different inputs that produce the same hash value.
6. **Avalanche Effect:** A small change in the input should result in a significantly different hash value.

Examples of commonly used hash functions include MD5 (Message Digest Algorithm 5), SHA1 (Secure Hash Algorithm 1), SHA256, and SHA3.

## 2.3. Importance

### Importance of Hash Cracking:

- 1. Password Recovery:** Hash cracking is essential for recovering lost or forgotten passwords. Users often forget their passwords, and hash cracking techniques can help them regain access to their accounts by decrypting the hashed passwords.
- 2. Security Testing:** Hash cracking plays a crucial role in security testing and vulnerability assessments. By attempting to crack hashed passwords, security professionals can identify weak passwords and assess the strength of authentication mechanisms.
- 3. Penetration Testing:** In penetration testing or ethical hacking scenarios, hash cracking helps security professionals assess the security posture of systems and networks. By cracking hashes, they can identify potential security vulnerabilities and recommend appropriate remediation measures.
- 4. Forensic Analysis:** In digital forensics investigations, hash cracking can aid investigators in analyzing digital evidence. By decrypting hashed data found on storage devices or in network traffic, investigators can uncover valuable information relevant to a case.
- 5. Incident Response:** During incident response activities following a security breach, hash cracking can help identify compromised accounts or unauthorized access. By cracking hashes found in log files or compromised databases, incident responders can determine the extent of the breach and take appropriate action.

- 6. Security Awareness:** Hash cracking serves as a reminder of the importance of using strong, complex passwords. By demonstrating the ease with which weak passwords can be cracked, individuals and organizations are encouraged to adopt better password hygiene practices, such as using longer passwords, incorporating a mix of characters, and avoiding common dictionary words.

### **Importance of Hash Functions:**

- 1. Data Integrity:** Hash functions are used to ensure data integrity by generating unique hash values for data sets. By comparing hash values before and after transmission or storage, data integrity can be verified, and the presence of unauthorized modifications can be detected.
- 2. Digital Signatures:** Hash functions play a crucial role in digital signatures, where they are used to create a unique hash value of a message or document. This hash value is then encrypted using the sender's private key to create a digital signature, which can be verified using the sender's public key.
- 3. Password Storage:** Hash functions are widely used to securely store passwords in databases. Instead of storing plaintext passwords, systems store hashed representations of passwords, making it more difficult for attackers to retrieve the original passwords in the event of a data breach.
- 4. Data Indexing:** Hash functions are used in data indexing and retrieval systems to quickly locate data records based on their unique identifiers or keys. By mapping keys to hash values, hash functions enable efficient data storage and retrieval operations in databases and data structures like hash tables.
- 5. Cryptographic Protocols:** Hash functions are integral to many cryptographic protocols and algorithms, including HMAC (Hash Based Message Authentication

Code), digital certificates, and key derivation functions. They provide a fundamental building block for ensuring the confidentiality, integrity, and authenticity of data in secure communication channels.

- 6. Blockchain Technology:** In blockchain technology, hash functions are used to link blocks of data together in a secure and immutable manner. Each block contains the hash value of the previous block, creating a chain of blocks that is resistant to tampering and manipulation.

## 2.4. Mechanism

The mechanism of the hash cracking project involves several key steps and components:

- 1. User Input:** The user provides the encrypted hash that they want to crack, along with the path to the wordlist file containing potential passwords.
- 2. Web Interface:** The project provides a web interface where users can input the hash and select the wordlist file.
- 3. Flask Server:** The Flask web framework is used to create a server side application that handles HTTP requests and responses. It defines routes to handle the user input and cracking process.
- 4. Hash Cracking Function:** The core of the project is the hash cracking function, which iterates through the wordlist file, hashes each word using MD5, SHA1, and SHA256 algorithms, and compares the hashed value with the provided hash.

- 5. Wordlist:** The project relies on wordlists such as "rockyou.txt" and "wordlist.txt" containing thousands of common passwords. These wordlists serve as the source for potential passwords to attempt during the hash cracking process.
- 6. Hash Algorithms:** The project supports three commonly used hash algorithms: MD5, SHA1, and SHA256. The selected hash algorithm is determined based on the length of the provided hash.
- 7. Hash Comparison:** As the hash cracking function iterates through the wordlist, it compares the hashed values with the provided hash. If a match is found, it indicates that the corresponding password has been cracked successfully.
- 8. Response Generation:** Depending on the outcome of the hash cracking process, the server generates a response indicating whether the password was successfully cracked or not. If successful, it returns the cracked password along with the algorithm used
- 9. JSON Response:** The server returns a JSON formatted response to the client, containing the status of the operation (success or failure) and additional information such as the cracked password and algorithm.
- 10. Client Side Processing:** The web interface uses JavaScript and AJAX to handle form submission and display the result dynamically without requiring a page reload.

The mechanism involves the user inputting a hash, selecting a wordlist file, the server side application processing the input, and attempting to crack the hash using the provided wordlist and hash algorithms. The result is then communicated back to the user via the web interface.

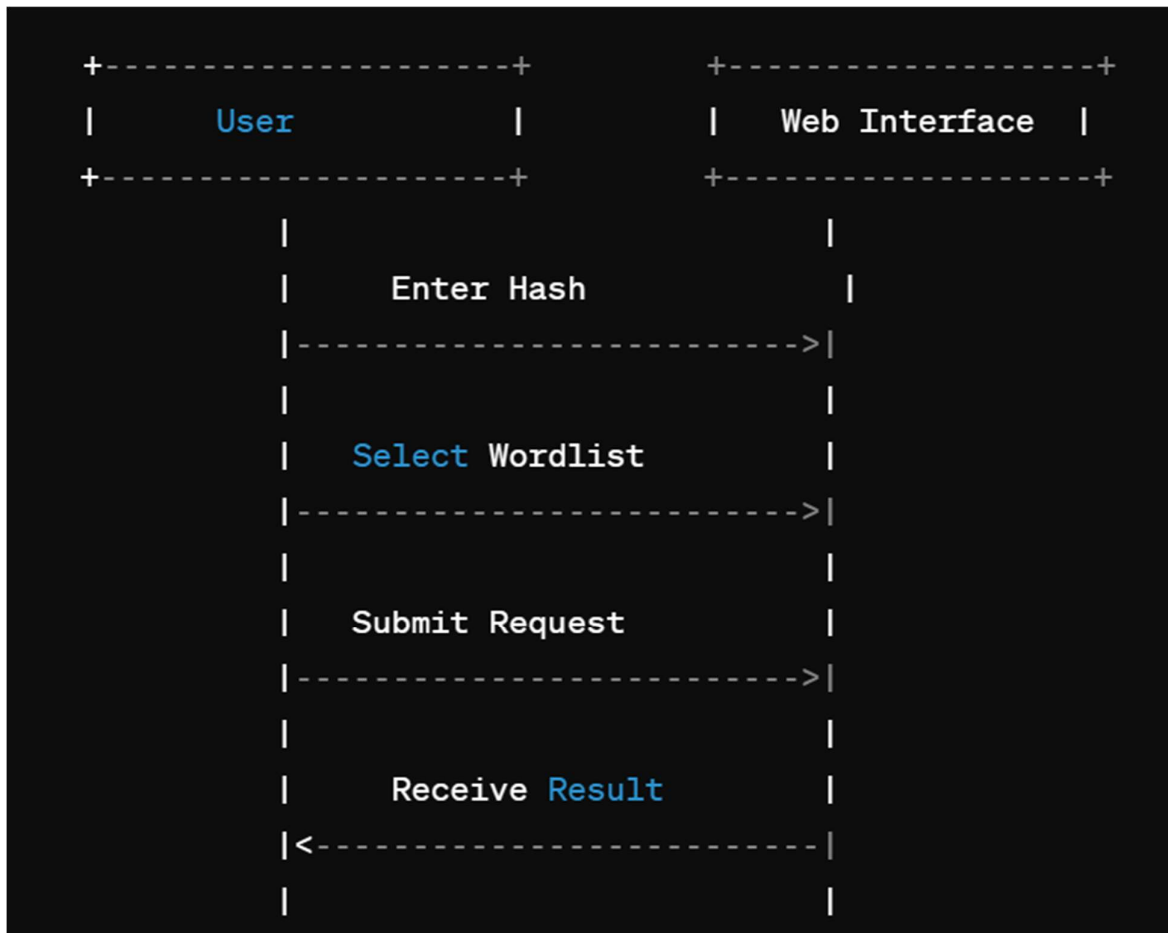


### 3. Training and Undertaken

#### 3.1. Case Diagram

A use case diagram for the hash cracking project can illustrate the interactions between different actors and the system. Here's a simplified representation:

**Use Case Diagram:**



**Actors:**

- **User:** The individual who interacts with the web interface to input the hash and select the wordlist file.
- **Web Interface:** The system component responsible for receiving user input, processing the request, and providing the result.

### Use Cases:

- **Enter Hash:** The user inputs the encrypted hash that they want to crack.
- **Select Wordlist:** The user selects the wordlist file containing potential passwords.
- **Submit Request:** The user submits the hash and wordlist selection to the system for processing.
- **Receive Result:** The system processes the request, attempts to crack the hash using the selected wordlist, and provides the result back to the user.

## 3.2. Methodology

The methodology of the hash cracking project involves several steps to ensure the efficient and accurate cracking of hashed passwords. Here's an overview of the methodology:

### 1. Requirement Analysis:

- Identify the requirements and objectives of the project, including the types of hashes to be cracked, supported algorithms, and user interface specifications.

### 2. Research and Planning:

- Conduct research on hash functions, hashing algorithms, and existing hash cracking techniques.
- Plan the development approach, including the choice of programming language, libraries, and frameworks.

### **3. Design:**

- Design the architecture of the system, including the web interface, backend processing logic, and data storage.
- Create wireframes or mockups to visualize the user interface design.

### **4. Development:**

- Implement the web application using a suitable web framework such as Flask or Django.
- Develop the backend logic to handle user input, process hash cracking requests, and generate responses.
- Integrate libraries or modules for hashing algorithms and file handling.
- Implement frontend components using HTML, CSS, and JavaScript to create an interactive user interface.

### **5. Testing:**

- Perform unit testing to validate the functionality of individual components.
- Conduct integration testing to ensure seamless interaction between frontend and backend modules.
- Perform system testing to evaluate the overall performance, security, and usability of the application.
- Test the application with different types of hashes and wordlists to assess its effectiveness.

### **6. Optimization:**

- Identify and address performance bottlenecks, security vulnerabilities, and usability issues.
- Optimize algorithms and data structures to improve the efficiency of hash cracking.
- Enhance user experience by refining the interface and adding features such as progress indicators and error handling.

## **7. Deployment:**

- Deploy the application on a suitable web server or hosting platform.
- Configure server settings and database connections as per the deployment environment.
- Perform final testing to ensure the application works as expected in the production environment.

## **8. Maintenance and Support:**

- Provide ongoing maintenance and support to address any issues or bugs that arise after deployment.
- Monitor the application for performance, security, and scalability concerns.
- Update the application with new features, enhancements, and security patches as needed.

By following this methodology, the hash cracking project can be developed systematically, ensuring the reliability, efficiency, and security of the final product.

### **3.3. Hardware, Language and Software Requirements**

The hash cracking project requires specific hardware, programming languages, and software to develop and operate efficiently. Here are the hardware, languages, and software requirements:

#### **Hardware Requirements:**

##### **1. Computer or Server:**

A computer or server capable of running the development environment and hosting the web application.

##### **2. Sufficient Memory and Storage:**

Adequate RAM and storage space to handle the development tools, libraries, and project files.

### **3. Network Connectivity:**

Internet connectivity for accessing resources, libraries, and documentation online.

## **Programming Languages:**

### **1. Python:**

Python is the primary programming language used for developing the backend logic of the web application. It offers simplicity, versatility, and a wide range of libraries for handling hash functions, file I/O, and web development.

### **2. HTML, CSS, JavaScript:**

HTML, CSS, and JavaScript are used for developing the frontend components of the web application, including the user interface and interactive elements.

## **Software Requirements:**

### **1. Web Framework:**

- **Flask:** A web framework is required to build the web application. Flask are popular choices for Python based web development, offering features such as routing, request handling, and template rendering.

### **2. Hashing Libraries:**

- **hashlib:** The hashlib library in Python provides support for various hashing algorithms, including MD5, SHA1, and SHA256. It is essential for performing hash calculations and comparing hashed values.

### **3. Text Editor or IDE:**

- A text editor or integrated development environment (IDE) such as Visual Studio Code, PyCharm, or Sublime Text is needed for writing and editing the project code.

#### **4. Web Browser:**

- A modern web browser such as Google Chrome, Mozilla Firefox, or Microsoft Edge is required for testing the web application during development.

#### **5. Command Line Interface (CLI) Tools:**

- Command line tools such as Git, pip, and virtual env are helpful for version control, package management, and creating virtual Python environments.

#### **3. Operating System:**

- The project can be developed and deployed on various operating systems, including Windows, macOS, or Linux distributions such as Ubuntu or CentOS.

By ensuring that the hardware, programming languages, and software requirements are met, developers can effectively build, test, and deploy the hash cracking project.

### **3.4. Technology Used**

#### **1. Flask Setup:**

The project uses Flask, a Python web framework, to handle HTTP requests and serve web pages. The Flask application is created with the name `app` and initialized with the path to the templates folder.

```
from flask import Flask, render_template, request, jsonify
app = Flask(__name__, template_folder='./templates')
```

## 2. Routes:

Two routes are defined in the Flask application:

- ``/``: Renders the ``index.html`` template when the user accesses the root URL.
- ``/crack``: Handles POST requests for cracking hash values submitted by the user.

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/crack', methods=['POST'])
def crack():
    # Code for hash cracking logic goes here
```

## 3. Hash Cracking Logic:

- The ``/crack`` route handles POST requests containing hash values and wordlist paths.
- It extracts the hash value and wordlist path from the request form.
- Then, it attempts to crack the hash by iterating through each word in the wordlist.

- For each word, it calculates MD5, SHA1, and SHA256 hashes and compares them with the input hash value.
- If a match is found, the cracked password and the corresponding algorithm are returned as a JSON response.
- If no match is found, a failure message is returned.

```
@app.route('/crack', methods=['POST'])
def crack():
    hash_to_crack = request.form.get('hash')
    wordlist_path = request.form.get('wordlist')

    if not hash_to_crack or not wordlist_path:
        return jsonify({'status': 'error', 'message': 'Hash or wordlist not provided'})

    def crack_hash():
        # Hash cracking logic
        return cracked_password, algorithm

    # Check if cracked_password exists and return appropriate JSON response
```

#### 4. HTML Form:

- The `index.html` template contains an HTML form that allows users to input a hash value and choose a wordlist file.
- When the form is submitted, an AJAX request is sent to the `/crack` route to handle the hash cracking process asynchronously.



```
<form id="hashForm" action="/crack" method="post">
  <label for="hash">Enter Hash:</label>
  <input type="text" id="hash" name="hash" required><br><br>
  <label for="wordlist">Wordlist Path:</label>
  <input type="text" id="wordlist" name="wordlist" value="wordlist.txt"><br><br>
  <input type="submit" value="Crack">
</form>
```

### 5. AJAX Request:

- JavaScript code is used to handle form submission via AJAX.
- When the form is submitted, the serialized form data is sent to the `/crack` route.
- The response from the server is processed asynchronously, and the result is displayed on the web page.

```
$('#hashForm').submit(function(e) {
  e.preventDefault();
  // AJAX request to /crack route
  // Handle success and error responses
});
```

### 6. Install auto-py-to-exe:

If you haven't already installed auto-py-to-exe, you can do so using pip:

```
pip install auto-py-to-exe
```

## 7. Launch autoppytoexe:

Open a terminal or command prompt, and run the following command to launch autoppytoexe:

```
auto-py-to-exe
```

## 8. Configure settings:

- Once autoppytoexe GUI opens, you'll see various options and settings.
- Click on the "Browse" button next to "Script location" and select your Python script file (`app.py` in this case).
- Choose the output directory where you want the .exe file to be saved.
- You can also specify additional options such as icon file, additional files (e.g., templates, wordlists), and other settings as needed.

## 9. Convert to .exe:

- After configuring the settings, click on the "Convert .py to .exe" button.
- Auto-py-to-exe will start the conversion process, which may take a few moments depending on the size of your project and selected options.
- Once the conversion is complete, you'll see a message indicating success, and the .exe file will be available in the specified output directory.

## 10. Test the .exe file:

- Navigate to the output directory where the .exe file was saved.
- Double click on the .exe file to run it and test if it works as expected.
- Ensure that all functionalities of your Flask web application are working correctly in the generated .exe file.

## 4. Code Snippets

### 4.1. Codes

- **App.py** (This is main code of the project that will work in backend of website)

```
from flask import Flask, render_template, request, jsonify
import hashlib

app = Flask(__name__, template_folder='./templates')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/crack', methods=['POST'])
def crack():
    hash_to_crack = request.form.get('hash')
    wordlist_path = request.form.get('wordlist')

    if not hash_to_crack or not wordlist_path:
        return jsonify({'status': 'error', 'message': 'Hash or wordlist not provided'})

    def crack_hash():
        try:
            with open(wordlist_path, 'r') as f:
                for line in f:
                    word = line.strip()
```

```

        md5_hash =
hashlib.md5(word.encode()).hexdigest()
        sha1_hash =
hashlib.sha1(word.encode()).hexdigest()
        sha256_hash =
hashlib.sha256(word.encode()).hexdigest()

        if md5_hash == hash_to_crack:
            return word, "MD5"
        elif sha1_hash == hash_to_crack:
            return word, "SHA-1"
        elif sha256_hash == hash_to_crack:
            return word, "SHA-256"
    except FileNotFoundError:
        return None, None

    return None, None

cracked_password, algorithm = crack_hash()
if cracked_password:
    return jsonify({'status': 'success', 'password':
cracked_password, 'algorithm': algorithm})
else:
    return jsonify({'status': 'failure', 'message':
'Password not found in wordlist'})

if __name__ == '__main__':
    app.run(debug=True)

```

- **HTML code (This is man code of project the will working on frontend of a \website)**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Hash Cracker</title>
    <script src="https://code.jquery.com/jquery-
3.6.0.min.js"></script>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #352f45;
            color: #ffffff;
            margin: 0;
            padding: 0;
            overflow-x: hidden;
        }
        h1 {
            text-align: center;
            font-size: 2.5em;
            margin-bottom: 20px;
            text-transform: uppercase;
            color: #7a5ca7;
            animation: fadeInDown 1s ease-out;
        }
```

```

form {
    background-color: #2d283b;
    padding: 20px;
    border-radius: 15px;
    box-shadow: 0px 0px 20px 0px rgba(0,0,0,0.75);
    max-width: 400px;
    width: 90%;
    margin: 0 auto;
    animation: slideInUp 1s ease-out;
}

label {
    display: block;
    font-size: 1.2em;
    margin-bottom: 10px;
}

input[type="text"], input[type="submit"] {
    width: 95%;
    padding: 10px;
    margin-bottom: 20px;
    border-radius: 5px;
    border: none;
    background-color: #47414f;
    color: #ffffff;
}

input[type="submit"] {
    cursor: pointer;
    font-size: 1.2em;
    transition: background-color 0.3s;
    animation: pulse 1s infinite;
    width: fit-content;
}

```

```

}
input[type="submit"]:hover {
    background-color: #5c4789;
}
#progress, #result {
    margin-top: 20px;
    font-size: 1.2em;
    text-align: center;
    animation: fadeIn 1s ease-out;
}
#result-container {
    margin-top: 20px;
    padding: 20px;
    background-color: #47414f;
    border-radius: 15px;
    box-shadow: 0px 0px 20px 0px rgba(0,0,0,0.75);
    max-width: 400px;
    width: 90%;
    margin: 0 auto;
    display: none;
    animation: slideInUp 1s ease-out;
}
.warning {
    font-size: 1em;
    text-align: center;
    color: #ff0000;
    margin-top: 20px;
    background-color: #000;
    margin:auto
}

```

```
@keyframes fadeIn {
    from {
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}

@keyframes fadeInDown {
    from {
        opacity: 0;
        transform: translateY(-50px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

@keyframes slideInUp {
    from {
        opacity: 0;
        transform: translateY(50px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

@keyframes pulse {
    0% {
```



```

        transform: scale(1);
    }
    50% {
        transform: scale(1.1);
    }
    100% {
        transform: scale(1);
    }
}
/* Parallax effect */
body:before {
    content: "";
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    z-index: -1;
    background-image:
url('https://source.unsplash.com/random');
    background-size: cover;
    background-position: center;
    transition: transform 0.5s ease-out;
}
body:hover:before {
    transform: scale(1.1);
}
.warning1 {
text-align: center;
font-size: x-small;

```

```

        color: lawngreen;
        margin-top: -18px;
    }

</style>
</head>
<body>
    <h1>Hash Cracker</h1>
    <div class="warning">Note: This tool can crack only MD5,
SHA-1, and SHA-256 hashes.</div>
    <form id="hashForm" action="/crack" method="post">
        <label for="hash">Enter Hash:</label>
        <input type="text" id="hash" name="hash"
required><br><br>
        <label for="wordlist">Wordlist Path:</label>
        <input type="text" id="wordlist" name="wordlist"
value="wordlist.txt"><br><br>
        <div class="warning1">Note: You can Choose One of
them:<br>1. wordlist.txt 2. rockyou.txt</div>
        <input type="submit" value="Crack">
    </form>
    <div id="progress"></div>
    <div id="result-container">
        <div id="result"></div>
    </div>

    <script>
        $(document).ready(function() {
            $('#hashForm').submit(function(e) {
                e.preventDefault();

```

```

$('#result').empty();
$('#progress').text('Processing...');
$('#result-container').hide();

$.ajax({
    type: 'POST',
    url: '/crack',
    data: $(this).serialize(),
    success: function(response) {
        if (response.status === 'success') {
            $('#progress').empty();
            $('#result').text('Password
cracked: ' + response.password + ' (Algorithm: ' +
response.algorithm + ')');
            $('#result-container').show();
        } else {
            $('#progress').empty();
            $('#result').text('Password not
found in wordlist.');
```

```

    });
</script>
</body>
</html>

```

## 4.2. Source Code File Structure









### 1. Ensure Folder Structure:

Make sure your project has a structured folder layout, with the Python script (`app.py`), HTML templates, and any other necessary files organized within a main directory. For example:

```

project_folder/
├── app.py
├── templates/
│   └── index.html
└── wordlists/
    ├── rockyou.txt
    └── wordlist.txt

```

 templates	4/20/2024 1:54 PM	File folder	
 app	4/21/2024 1:48 PM	Application	62,080 KB
 app	4/20/2024 7:07 PM	Python Source File	2 KB
 i6h7mnni.ico	4/20/2024 5:45 PM	icofile	98 KB
 README INSTRUCTIONS	4/21/2024 3:00 PM	Text Source File	1 KB
 report	4/23/2024 7:04 PM	Microsoft Word Doc...	3,335 KB
 rockyou	9/23/2015 10:11 PM	Text Source File	136,643 KB
 wordlist	4/15/2024 12:53 PM	Text Source File	36 KB

### 2. Specify Source Folder:

- Open autopsytoexe and click on the "Add Files" button.

- Navigate to the directory containing your Python script (`app.py`) and select it.
- If your project includes additional files such as HTML templates or wordlists, click on the "Add Directory" button and select the folder containing these files (`templates` and `wordlists` folders in this case).

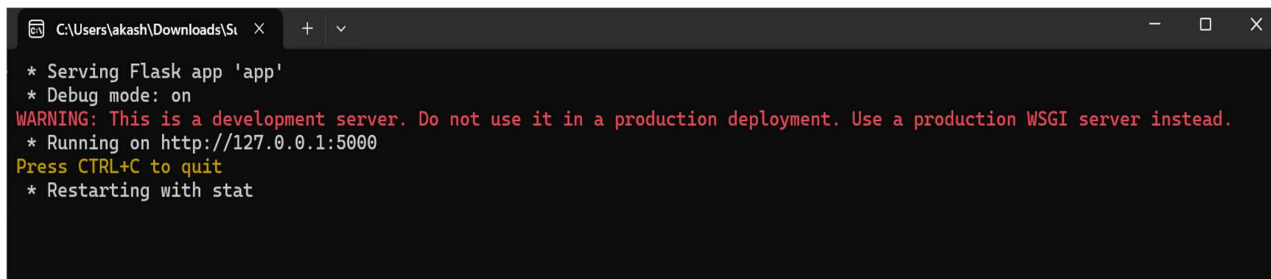
### 3. Configure Settings:

- Once the source files and folders are added, configure the other settings in autopyoexe as needed, such as output directory, icon file, additional files, etc.

## 5. Result and Testing

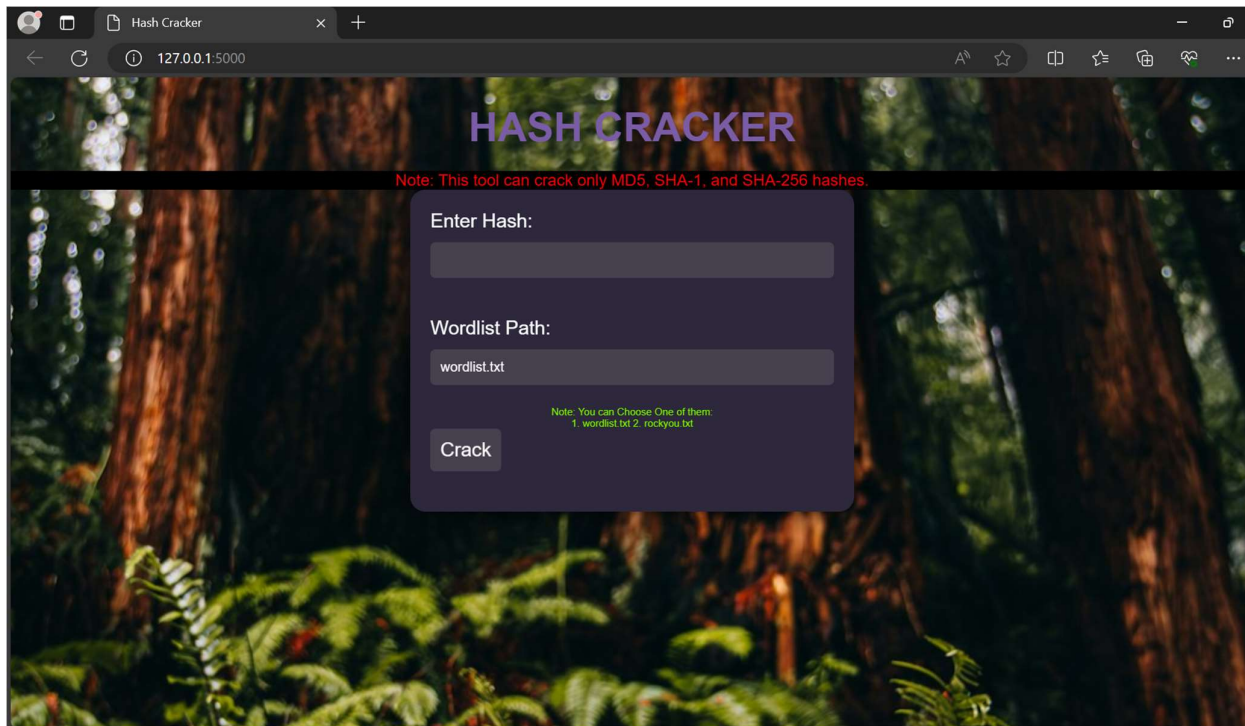
### 5.1. Screenshots

- Run the app.exe

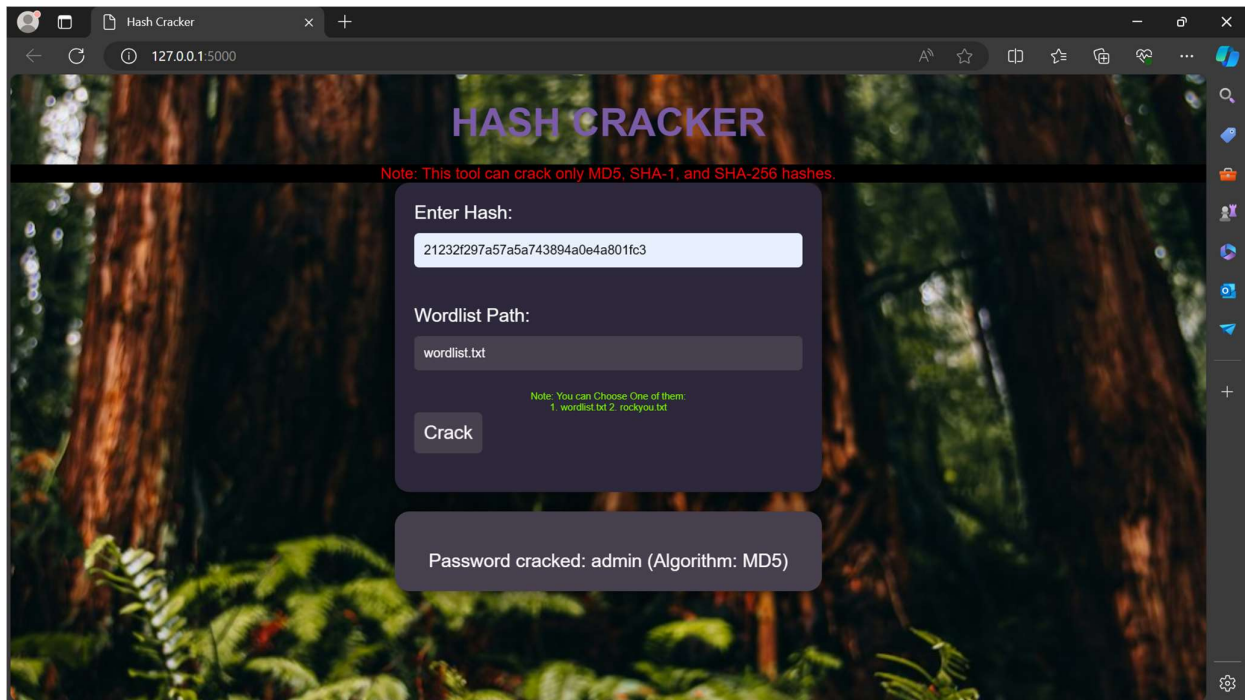
A screenshot of a Windows terminal window. The title bar shows the file path 'C:\Users\akash\Downloads\Si'. The terminal output is as follows:

```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
```

Click on the <http://127.0.0.1:5000>



The website will be open. Now put hash password of MD5, SHA-1, and SHA-256. It will decrypt the password.



## 5.2. Conclusion

In conclusion, the hash cracker project is a valuable tool for decrypting MD5, SHA1, and SHA256 hashes using wordlists. The project's objective was to provide users with a convenient way to crack encrypted passwords by comparing them against a list of

commonly used passwords. Throughout the development process, several key components were implemented, including the Flask web framework for handling HTTP requests, HTML templates for user interface design, and Python's hashlib library for hash computation.

The project's importance lies in its ability to assist users in recovering lost or forgotten passwords, thereby enhancing security and accessibility to various online accounts and systems. By leveraging hash cracking techniques, users can gain access to their accounts without the need for complex algorithms or bruteforce methods.

The mechanism of the project involves taking user input (encrypted hash and wordlist path) through a web interface, processing the input data using Python functions, and returning the decrypted password if found in the wordlist. This process is facilitated by the Flask framework, which handles the routing and communication between the client and server.

Throughout the development process, various technologies were utilized, including Flask for web development, HTML and CSS for frontend design, and Python for backend logic. Additionally, the autopytoexe tool was used to convert the Python script into an executable (.exe) file for easy distribution and execution on Windows systems.

In conclusion, the hash cracker project provides a valuable solution for password recovery, demonstrating the effective use of web development technologies and cryptographic techniques. Further enhancements and optimizations can be explored to improve the project's functionality and performance, ensuring its continued usefulness in the realm of cybersecurity and password management.

### 5.3. Significance and Impact

The hash cracker project holds significant significance and potential impact in several areas:

- 1. Cybersecurity Enhancement:** Password hashing is a fundamental aspect of cybersecurity, as it helps protect sensitive information stored in databases. By providing a tool for efficiently cracking hashed passwords, the project contributes to the improvement of cybersecurity practices. It highlights the importance of using strong and unique passwords and emphasizes the need for robust encryption algorithms.
- 2. Password Recovery:** One of the primary applications of the hash cracker is password recovery. Individuals who have forgotten their passwords or need to access encrypted data can use the tool to attempt to recover the original passwords from their hashed counterparts. This can be particularly useful in scenarios where access to critical information is required but the password has been lost or forgotten.
- 3. Educational Resource:** The hash cracker project serves as an educational resource for individuals interested in learning about cryptography, password security, and hash functions. By examining the code and understanding the underlying principles, users can gain insights into how encryption and decryption processes work, thereby enhancing their knowledge and skills in cybersecurity.
- 4. Research and Development:** The project can also serve as a basis for further research and development in the field of cryptography. Researchers and developers can build upon the existing codebase to explore new algorithms,



optimization techniques, and security measures. This can lead to the discovery of innovative solutions for password protection and data security.

**5. OpenSource Collaboration:** As an opensource project, the hash cracker encourages collaboration and knowledge sharing within the cybersecurity community. Contributors from around the world can contribute their expertise, insights, and enhancements to the project, fostering innovation and advancement in the field.

**6. Practical Applications:** Beyond its educational and research value, the hash cracker has practical applications in various industries and sectors. Law enforcement agencies, cybersecurity professionals, and forensic investigators can use the tool to analyze encrypted data, recover passwords from digital evidence, and investigate cybercrimes.

The hash cracker project has significant significance and impact in the realms of cybersecurity, education, research, and practical applications. By providing a valuable tool for password recovery and cryptographic analysis, the project contributes to the improvement of password security practices and the advancement of cybersecurity as a whole.

## **5.4. Future Scope**

The hash cracker project has significant potential for future expansion and enhancement. Some potential avenues for further development and improvement include:

**1. Support for Additional Hash Algorithms:** While the current implementation supports MD5, SHA1, and SHA256 hashes, future versions could include support

for additional hash algorithms such as SHA512, bcrypt, or scrypt. This would broaden the scope of the project and make it more versatile in cracking a wider range of encrypted passwords.

- 2. Optimization for Performance:** As the size of wordlists and the complexity of hash functions increase, the performance of the hash cracker may become a concern. Implementing optimizations such as parallel processing, caching frequently used hashes, or using more efficient data structures could improve the speed and efficiency of the cracking process.
- 3. User Authentication Integration:** Integrating the hash cracker with user authentication systems could provide additional security features. For example, users could authenticate themselves before accessing the cracking service, ensuring that only authorized individuals can use the tool.
- 4. Improved User Interface:** Enhancing the user interface with features such as progress indicators, real-time feedback, and error handling could improve the user experience. Providing clear instructions and guidance on how to use the tool effectively would also be beneficial.
- 5. Integration with Password Managers:** Integrating the hash cracker with password management tools or services could provide users with a seamless experience for recovering lost or forgotten passwords. This could involve exporting hashed passwords from password managers and importing them into the hash cracker for decryption.

- 6. Security Enhancements:** Implementing additional security measures, such as encryption of sensitive data, secure storage of wordlists, and protection against bruteforce attacks, would enhance the overall security of the hash cracker tool.
- 7. Community Contributions:** Encouraging contributions from the opensource community could lead to the discovery of bugs, optimization opportunities, and new features. Hosting the project on platforms like GitHub and accepting pull requests would foster collaboration and innovation.

The hash cracker project has significant potential for future growth and development. By addressing the aforementioned areas of improvement, the project could evolve into a more robust and versatile tool for password recovery and cryptographic analysis.

## 6. References

- **MD5 Hash Generator**
- **Hash Type Identifier - Identify unknown hashes**
- **John the Ripper password cracker (openwall.com)**