

# PROJECT REPORT

*Information Security*



**BS(CS)-3B**

**AIR UNIVERSITY ISLAMABAD**

**Group Members**

| Name          | Registration-ID |
|---------------|-----------------|
| Afia Aziz     | 231561          |
| Zumer Dhillun | 231597          |
| Zoya Azad     | 231579          |
| Duaa Dara     | 232481          |

**Submitted to: Miss Waseeqa Ghazanfer**

# ***DATA HASHING TOOL***

**Cryptographic Hashing** is crucial in information security, ensuring confidentiality, integrity, and authentication by converting input data into fixed-length, irreversible hash values. The **Data Hashing Tool** provides an intuitive platform to generate, compare, and analyze hashes using popular algorithms like MD5, SHA-1, SHA-256, and SHA-512, offering a hands-on approach to understanding hashing's role in digital security.

---

## **Core Principles:**

1. **Hash Function:** Converts variable-length input into fixed-length hash values (e.g., MD5, SHA-1, SHA-256).
  2. **Irreversibility:** Hashing is a one-way function, making it infeasible to reverse the hash.
  3. **Fixed-Length Output:** Regardless of input size, the output hash remains constant.
- 

## **Use Cases:**

1. **Password Storage:** Protects plaintext passwords in databases.
  2. **Data Integrity Verification:** Ensures data consistency during transmission or storage.
  3. **Digital Signatures:** Provides message authentication and integrity.
- 

## **Design and Architecture:**

- **Frontend:** HTML, CSS, and JavaScript for an interactive user interface.
- **Backend:** Flask with Python's hashlib for hashing.
- **Workflow:** Users input data, select an algorithm, and the backend processes and returns the hash.
- **Scalability:** Modular design supports easy integration of new features.

# LOADING SCREEN



## 1. Functionality:

The **Loading Screen** is the introductory page of the Data Hashing Tool, providing a polished, visually appealing transition before redirecting to the Welcome Page. It uses JavaScript's `setTimeout` function for automated redirection after a brief delay (e.g., 3.5 seconds), ensuring a smooth user experience.

---

## 2. Implementation:

The loading screen is implemented using JavaScript.

```
<script>
  setTimeout(() => {
    window.location.href = "/welcome";
  }, 3500);
</script>
```

---

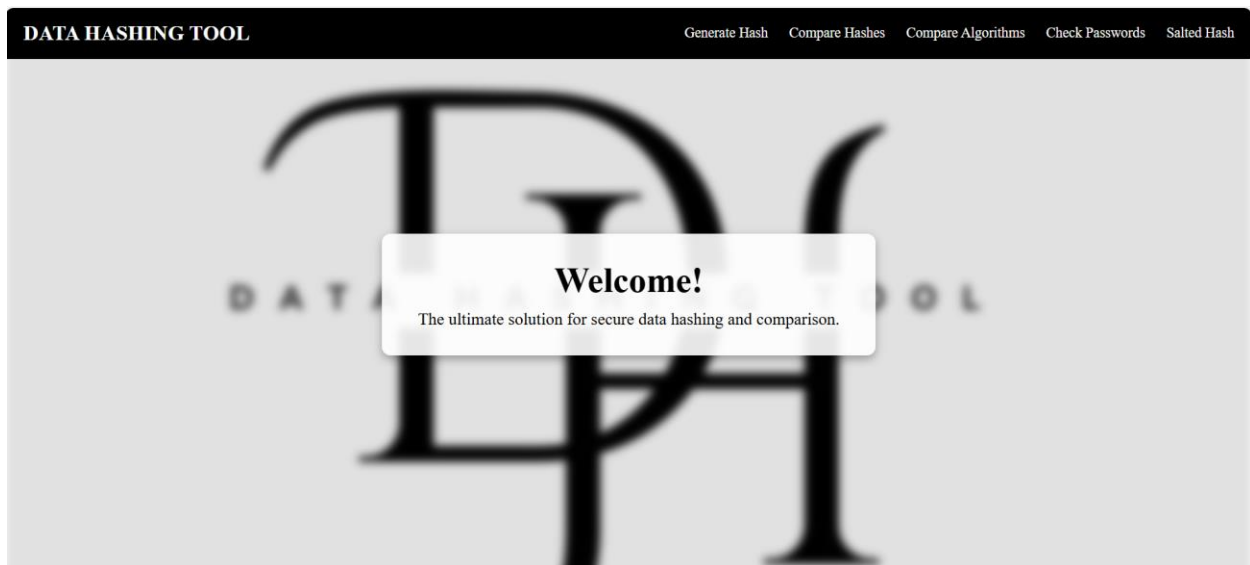
### 3. Purpose:

- Prepares users for the main interface with a professional touch.
  - Enhances user experience by creating a seamless transition.
- 

### 4. Benefits:

- Smooth navigation and dynamic entry.
- Creates a strong first impression through professional branding.

## WELCOME PAGE



### 1. Functionality:

The **Welcome Page** is the main entry point for users, offering intuitive navigation to key features of the Data Hashing Tool. It includes a navigation bar with links to **Generate Hash**, **Compare Hashes**, **Compare Algorithms**, **Check Passwords**, and **Salted Hash**. The page has a dynamic design with a full-screen background, a prominent welcome message, and a modern transparent content box.

---

## 2. Implementation:

```
<body>
  <div class="background"></div>
  <div class="header">
    <div class="title">DATA HASHING TOOL</div>
    <div class="nav-links">
      <a href="/generate-hash">Generate Hash</a>
      <a href="/compare-hashes">Compare Hashes</a>
      <a href="/compare-algorithms">Compare Algorithms</a>
      <a href="/password-checker">Check Passwords</a>
      <a href="/salted-hash">Salted Hash</a>
    </div>
  </div>
  <div class="welcome-box">
    <h1>Welcome!</h1>
    <p>The ultimate solution for secure data hashing and comparison.</p>
  </div>
</body>
```

---

## 3. Purpose:

- Introduces the tool with clarity and professionalism.
- Provides easy access to features through an intuitive navigation bar.

---

## 4. Benefits:

- **Engaging First Impression:** The visually appealing design invites users to explore the tool further.
- **Simplified Navigation:** Ensures users can quickly locate and access key features.

GENERATE HASH PAGE

DATA HASHING TOOL

[Generate Hash](#)[Compare Hashes](#)[Compare Algorithms](#)[Check Passwords](#)[Salted Hash](#)

Generate Hash

Input Type:

Text

Input Text:

Enter text here

Algorithm:

MD5

MD5

SHA1

SHA256

SHA512

DATA HASHING TOOL

[Generate Hash](#)[Compare Hashes](#)[Compare Algorithms](#)[Check Passwords](#)[Salted Hash](#)

Generate Hash

Input Type:

Text

Input Text:

Information Security

Algorithm:

MD5

Generate

DATA HASHING TOOL

[Generate Hash](#)[Compare Hashes](#)[Compare Algorithms](#)[Check Passwords](#)[Salted Hash](#)

Generate Hash

Input Type:

Text

Input Text:

Enter text here

Algorithm:

MD5

Generate

Generated Hash: 0889f4cee1d942ae2c1daf686377de0

## 1. Functionality:

The **Generate Hash** page lets users create hash values from text or file inputs, with options to select a hashing algorithm (MD5, SHA-1, SHA-256, SHA-512). The generated hash is displayed after form submission.

---

## 2. Implementation:

```
<h1>Generate Hash</h1>
    <form method="POST" enctype="multipart/form-data">
        <label for="input_type">Input Type:</label>
        <select name="input_type" id="input_type"
onchange="toggleInputFields()">
            <option value="text">Text</option>
            <option value="file">File</option>
        ``html
    </select>
    <div id="text-input">
        <label for="text">Input Text:</label>
        <textarea name="text" id="text" placeholder="Enter text
here"></textarea>
    </div>
    <div id="file-input" class="file-input">
        <label for="file">Upload File:</label>
        <input type="file" name="file" id="file">
    </div>
    <label for="algorithm">Algorithm:</label>
    <select name="algorithm" id="algorithm">
        <option value="md5">MD5</option>
        <option value="sha1">SHA1</option>
        <option value="sha256">SHA256</option>
        <option value="sha512">SHA512</option>
    </select>
    <button type="submit">Generate</button>
</form>
{% if hash_result %}
<p>Generated Hash: <strong>{{ hash_result }}</strong></p>
{% endif %}
```

## 3. Explanation:

Users select 'Text' or 'File' inputs via the toggleInputFields function, which updates the interface. They choose a hashing algorithm (MD5, SHA-1, SHA-256, SHA-512) from a dropdown, and upon submission, the backend generates and displays the hash.

---

#### 4. Purpose:

- **Educational:** Teaches users about cryptographic hashing and its role in data security.
  - **Practical:** Simplifies hash generation for text and files, aiding verification tasks.
- 

#### 5. Benefits:

- **Flexible Input:** Supports both text and file hashing.
  - **Algorithm Options:** Provides multiple popular algorithms.
  - **Real-Time Results:** Generates hashes instantly upon submission.
  - **User-Friendly:** Accessible design suitable for all users.
- 

#### 6. Backend Code for Generate Hash Functionality:

The backend implementation processes the selected input type (text or file) and generates the hash using the chosen algorithm. Below is the code snippet for the functionality:

```
@app.route('/generate-hash', methods=['GET', 'POST'])
def generate_hash():
    hash_result = None
    if request.method == 'POST':
        algorithm = request.form.get('algorithm')
        text = request.form.get('text', '')
        file = request.files.get('file')
        if text:
            if algorithm == 'md5':
                hash_result = hashlib.md5(text.encode()).hexdigest()
            elif algorithm == 'sha1':
                hash_result = hashlib.sha1(text.encode()).hexdigest()
```



```

elif algorithm == 'sha256':
    hash_result = hashlib.sha256(text.encode()).hexdigest()
elif algorithm == 'sha512':
    hash_result = hashlib.sha512(text.encode()).hexdigest()
elif file:
    file_content = file.read()
    if algorithm == 'md5':
        hash_result = hashlib.md5(file_content).hexdigest()
    elif algorithm == 'sha1':
        hash_result = hashlib.sha1(file_content).hexdigest()
    elif algorithm == 'sha256':
        hash_result = hashlib.sha256(file_content).hexdigest()
    elif algorithm == 'sha512':
        hash_result = hashlib.sha512(file_content).hexdigest()
return render_template('generate-hash.html', hash_result=hash_result)

```

## COMPARE HASH PAGE

DATA HASHING TOOL

Generate Hash Compare Hashes Compare Algorithms Check Passwords Salted Hash

**Compare Hashes**

Hash 1:  
000964cee1d942ae2c1da086377de0

Hash 2:  
000964cee1d942ae2c1da086377de0

Compare

The screenshot shows a web application titled "Compare Hashes" with a logo featuring the letters "DPH" inside a circle. Below the title, there are two input fields labeled "Hash 1:" and "Hash 2:". The first input field contains the text "Enter first hash" and the second contains "Enter second hash". Below these fields is a "Compare" button. At the bottom, it displays "Comparison Result: Match".

## 1. Functionality:

The **Compare Hash** page allows users to compare two hash values by inputting text or uploading files. After selecting inputs and a hashing algorithm (MD5, SHA-1, SHA-256, SHA-512), the page generates the hashes and informs the user whether they match.

---

## 2. Implementation:

```
</div>
    <h1>Compare Hashes</h1>
    <form method="POST">
        <label for="hash1">Hash 1:</label>
        <input type="text" name="hash1" id="hash1" placeholder="Enter
first hash" required>
        <label for="hash2">Hash 2:</label>
        <input type="text" name="hash2" id="hash2" placeholder="Enter
second hash" required>
        <button type="submit">Compare</button>
    </form>
    {% if result %}
    <p>Comparison Result: <strong>{{ result }}</strong></p>
    {% endif %}
</div>
```

---

## 3. Explanation:

- **Input Type Selection:** Users choose between 'Text' or 'File' for each input (Hash 1 and Hash 2) via dropdown menus, dynamically displaying either a `textarea` for text input or an `input[type="file"]` for file uploads.
  - **Algorithm Selection:** Users select a hashing algorithm (MD5, SHA-1, SHA-256, or SHA-512) from a dropdown menu.
  - **Hash Comparison:** The system generates hashes for both inputs using the selected algorithm and compares them, displaying whether they match or not.
- 

#### 4. Purpose:

- **Educational:** Helps users understand how hashing works and the importance of identical data inputs for matching hash values.
  - **Practical:** Provides a straightforward way to verify data integrity by comparing hashes for text or file inputs.
- 

#### 5. Benefits:

- **Flexible Inputs:** Supports both text and file comparisons.
  - **Algorithm Variety:** Enables users to select from multiple secure hashing algorithms.
  - **Efficient Verification:** Quickly determines if the two inputs produce matching hash values.
  - **User-Friendly:** Clear and intuitive interface simplifies the hash comparison process.
- 

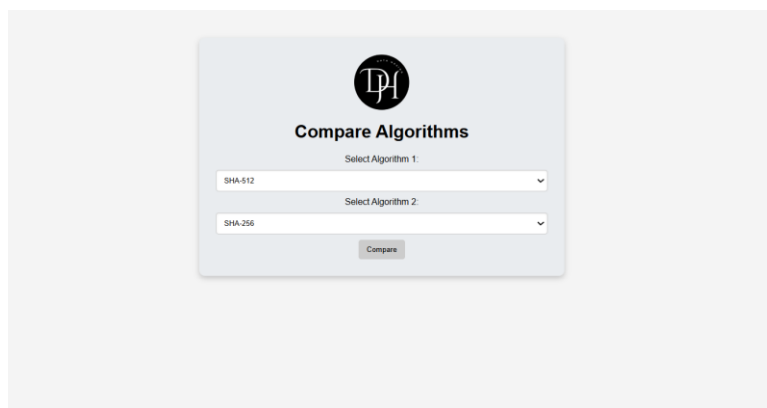
#### 6. Backend (Flask) Code for Handling Hash Comparison:

The backend logic processes the inputs, generates hashes, and compares them to provide the result.

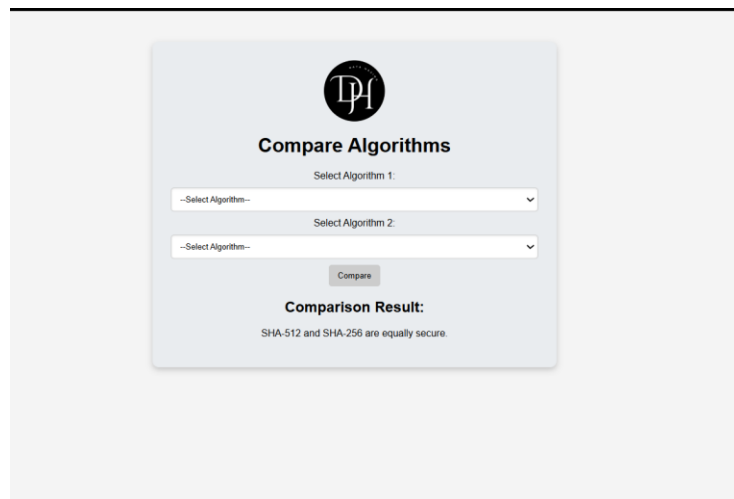
```
@app.route('/compare-hashes', methods=['GET', 'POST'])
def compare_hashes():
    result = None
```

```
if request.method == 'POST':
    hash1 = request.form.get('hash1')
    hash2 = request.form.get('hash2')
    result = "Match" if hash1 == hash2 else "Do not match"
    return render_template('compare-hashes.html', result=result)
```

## COMPARE ALGORITHMS



The screenshot shows a web form titled "Compare Algorithms" with a logo at the top. It contains two dropdown menus labeled "Select Algorithm 1:" and "Select Algorithm 2:". The first dropdown is set to "SHA-512" and the second is set to "SHA-256". Below the dropdowns is a "Compare" button.



This screenshot shows the same "Compare Algorithms" form after the comparison. The dropdown menus now display "--Select Algorithm--". Below the form, a section titled "Comparison Result:" displays the text "SHA-512 and SHA-256 are equally secure."

### 1. Functionality:

The **Compare Algorithms** page allows users to select two different hashing algorithms (such as MD5, SHA-1, SHA-256, SHA-512) and compare them. This page generates a result that shows the differences between the selected algorithms.

The comparison result may detail things such as speed, length of the hash, or other characteristics that differentiate the selected algorithms. The user selects two algorithms from dropdown menus and submits the form for comparison.

---

## 2. Implementation:

```
<div class="elevated-box">
  <div class="logo">
    
  </div>
  <h1>Compare Algorithms</h1>
  <form method="POST" action="/compare-algorithms">
    <label for="algorithm1">Select Algorithm 1:</label>
    <select name="algorithm1" id="algorithm1" required>
      <option value="">--Select Algorithm--</option>
      <option value="MD5">MD5</option>
      <option value="SHA-1">SHA-1</option>
      <option value="SHA-256">SHA-256</option>
      <option value="SHA-512">SHA-512</option>
    </select>
    <label for="algorithm2">Select Algorithm 2:</label>
    <select name="algorithm2" id="algorithm2" required>
      <option value="">--Select Algorithm--</option>
      <option value="MD5">MD5</option>
      <option value="SHA-1">SHA-1</option>
      <option value="SHA-256">SHA-256</option>
      <option value="SHA-512">SHA-512</option>
    </select>
    <button type="submit">Compare</button>
  </form>
  {% if comparison_result %}
    <h2>Comparison Result:</h2>
    <p>{{ comparison_result }}</p>
  {% endif %}
</div>
```

---

## 3. Explanation:

- **Algorithm Selection:** The page allows users to select two different hashing algorithms from dropdown menus. The supported algorithms are **MD5**, **SHA-1**, **SHA-256**, and **SHA-512**.

- **Form Submission:** Once users select two algorithms and submit the form, the page processes the selection and compares the chosen algorithms.
  - **Comparison Result:** If the user submits valid selections, the comparison result is displayed below the form. The result could be a detailed explanation comparing the differences in speed, hash length, security features, or other factors related to the selected algorithms.
- 

#### 4. Purpose:

- **Educational:** Provides users with insights into the strengths and weaknesses of different hashing algorithms by comparing their security levels and computational performance.
  - **Practical:** Helps users select the most appropriate hashing algorithm for their specific security needs by offering a detailed comparison.
- 

#### 5. Benefits:

- **Algorithm Analysis:** Compares hashing algorithms (MD5, SHA-1, SHA-256, SHA-512) for security and efficiency.
  - **User Awareness:** Increases understanding of algorithm vulnerabilities and use cases.
  - **Informed Decision-Making:** Helps users choose secure algorithms for password hashing or data verification.
  - **Comprehensive Overview:** Displays key metrics like hash length, computational cost, and attack resistance.
- 

#### 6. Backend (Flask) Code for Handling Algorithm Comparison:

The backend for the Compare Algorithms page retrieves user-selected algorithms, evaluates their hash length, computational efficiency, and security features, and generates a comparison report. It helps users understand the strengths and weaknesses of each algorithm for informed decision-making.

```
@app.route('/compare-algorithms', methods=['GET', 'POST'])
def compare_algorithms():
```

```

comparison_result = ""
if request.method == 'POST':
    algorithm1 = request.form['algorithm1']
    algorithm2 = request.form['algorithm2']
    sec1 = algorithm_security.get(algorithm1, 0)
    sec2 = algorithm_security.get(algorithm2, 0)
    if algorithm1 == algorithm2:
        comparison_result = f"{algorithm1} and {algorithm2} are equally
secure."
    elif sec1 > sec2:
        comparison_result = f"{algorithm1} is more secure than
{algorithm2}."
    elif sec1 < sec2:
        comparison_result = f"{algorithm2} is more secure than
{algorithm1}."
    else:
        comparison_result = f"{algorithm1} and {algorithm2} are equally
secure."
    return render_template('compare-algorithms.html',
comparison_result=comparison_result)

```

## CHECK PASSWORD

Generate Hash   Compare Hashes   Compare Algorithms   G

**Check Password Strength**

Enter Password:

\*\*\*\*\*

Check

Strength: Weak: Password must contain at least one uppercase letter.

### 1. Functionality:

The **Check Passwords** page lets users evaluate password strength based on criteria like length, character variety, and special characters. Weak passwords receive suggestions for improvement, helping users strengthen their security.

---

## 2. Implementation:

```
<div class="content-container">
  <div class="elevated-box">
    <div class="logo">
      
    </div>
    <h1>Check Password Strength</h1>
    <form method="POST">
      <label for="password">Enter Password:</label>
      <input type="password" name="password" id="password"
placeholder="Enter your password" required>
      <button type="submit">Check</button>
    </form>
    {% if password_strength %}
    <div class="result">
      <strong>Strength: {{ password_strength }}</strong>
    </div>
    {% if suggestions %}
    <div class="suggestions">
      <strong>Suggestions:</strong>
      <ul>
        {% for suggestion in suggestions %}
        <li>{{ suggestion }}</li>
        {% endfor %}
      </ul>
    </div>
    {% endif %}
    {% endif %}
  </div>
</div>
```

## 3. Explanation:

Users enter their password in a secure field and submit it via POST. The backend evaluates the password's strength based on length, character variety, and returns a rating (e.g., "Weak", "Moderate", "Strong"). If weak, improvement tips are



provided.

---

## 4. Purpose

- **Educational:** Helps users understand the importance of secure passwords and how to create them.
  - **Practical:** Provides instant feedback and actionable advice for improving password security.
- 

## 5. Benefits

- **Security Awareness:** Educates users on strong password practices.
  - **User Guidance:** Provides clear password improvement tips.
  - **Immediate Feedback:** Instantly evaluates password strength.
  - **Convenience:** Simple interface for easy password evaluation.
- 

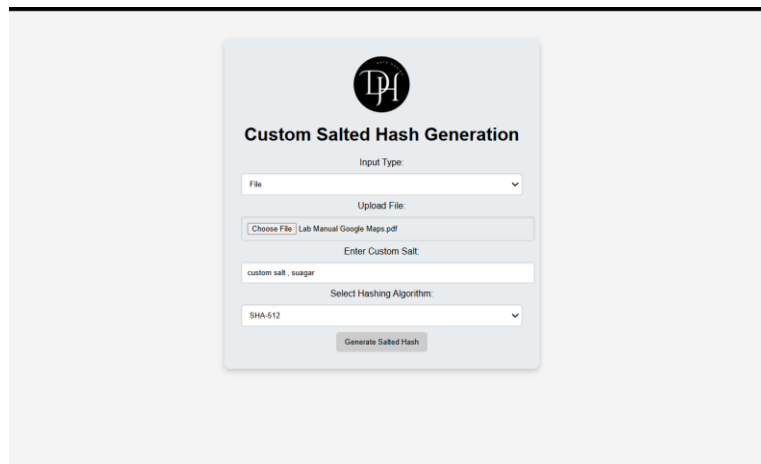
## 6. Backend (Flask) Code for Handling Password Strength Check:

The Flask backend evaluates the password using regex patterns to check compliance with security criteria. It assigns a strength rating and provides suggestions for improvement, if necessary.

```
@app.route('/password-checker', methods=['GET', 'POST'])
def password_checker():
    password_strength = None
    if request.method == 'POST':
        password = request.form.get('password')
        if len(password) < 8:
            password_strength = "Weak: Password must be at least 8 characters."
        elif not re.search(r"[A-Z]", password):
            password_strength = "Weak: Password must contain at least one
uppercase letter."
        elif not re.search(r"[a-z]", password):
            password_strength = "Weak: Password must contain at least one
lowercase letter."
        elif not re.search(r"\d", password):
            password_strength = "Weak: Password must contain at least one digit."
        elif not re.search(r"[!@#$$%^&*(),.?\"':{}|<>]", password):
```

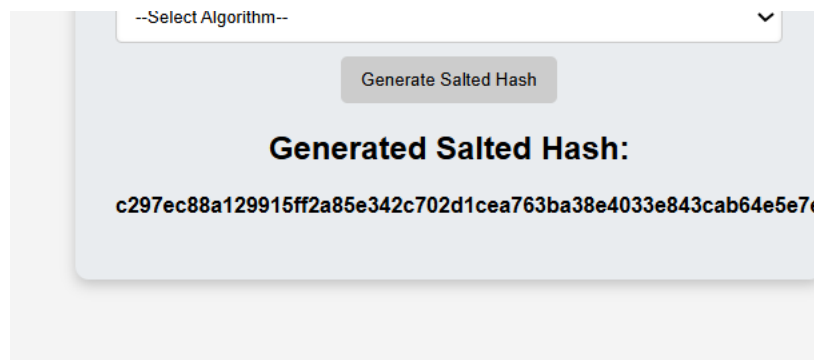
```
password_strength = "Moderate: Adding special characters can make it  
stronger."  
else:  
    password_strength = "Strong Password!"  
return render_template('password-checker.html',  
password_strength=password_strength)
```

## SALTED HASH



The screenshot shows a web application titled "Custom Salted Hash Generation" with a logo "TPH". It features a form with the following elements: an "Input Type" dropdown menu set to "File"; an "Upload File" section with a "Choose File" button and a file name "Lab Manual Google Maps.pdf"; an "Enter Custom Salt" text input field containing "custom salt : suagar"; and a "Select Hashing Algorithm" dropdown menu set to "SHA-512". A "Generate Salted Hash" button is located at the bottom of the form.

**Generating hash of the file using the salted hash**



This screenshot shows the result of the hash generation. At the top, there is a dropdown menu labeled "--Select Algorithm--". Below it is a "Generate Salted Hash" button. The main content area displays the text "Generated Salted Hash:" followed by a long alphanumeric hash string: "c297ec88a129915ff2a85e342c702d1cea763ba38e4033e843cab64e5e7e".

### 1. Functionality:

The Salted Hash Generation page allows users to generate secure, salted hashes for text or file inputs. Users can optionally provide a custom salt and choose from hashing algorithms (MD5, SHA-1, SHA-256, or SHA-512). The server processes the input, adds the salt, and generates a unique hash displayed on the page.

---

## 2. Implementation:

```
<div class="elevated-box">
  <div class="logo">
    
  </div>
  <h1>Custom Salted Hash Generation</h1>
  <form method="POST" action="/salted-hash" enctype="multipart/form-
data">
    <label for="input_type">Input Type:</label>
    <select name="input_type" id="input_type"
onchange="toggleInputFields()">
      <option value="text">Text</option>
      <option value="file">File</option>
    </select>
    <div id="text-input">
      <label for="text">Enter Text:</label>
      <textarea name="text" id="text" placeholder="Enter text
here"></textarea>
    </div>
    <div id="file-input" class="file-input">
      <label for="file">Upload File:</label>
      <input type="file" name="file" id="file">
    </div>
    <label for="salt">Enter Custom Salt:</label>
    <input type="text" name="salt" id="salt" placeholder="Enter a
custom salt (optional)">
    <label for="algorithm">Select Hashing Algorithm:</label>
    <select name="algorithm" id="algorithm" required>
      <option value="">--Select Algorithm--</option>
      <option value="md5">MD5</option>
      <option value="sha1">SHA-1</option>
      <option value="sha256">SHA-256</option>
      <option value="sha512">SHA-512</option>
    </select>
    <button type="submit">Generate Salted Hash</button>
  </form>
  {% if salted_hash %}
```

```
<h2>Generated Salted Hash:</h2>
<p><strong>{{ salted_hash }}</strong></p>
{% endif %}
</div>
```

---

### 3. Explanation:

Users select "Text" or "File" for input, with corresponding fields displayed. They can add a custom or random salt for added security, choose a hashing algorithm (MD5, SHA-1, SHA-256, SHA-512), and submit the form. The backend processes the input and salt, generating and returning the salted hash.

---

### 4. Purpose:

- **Enhancing Security:** Demonstrates the role of salting in making hashes more secure and unique.
- **Educational:** Teaches users the concept of salted hashes and their importance in cryptographic practices.

---

### 5. Benefits:

- **Increased Security:** Prevents hash collisions with unique salt values.
- **Customizability:** Users can define custom or random salts.
- **Versatility:** Supports text, file inputs, and multiple hashing algorithms.
- **Ease of Use:** Simplifies salted hash generation with an intuitive interface.

---

### 6. Backend (Flask) Code for Handling Salted Hash Generation:

The backend combines the user-provided text or file content with the salt, hashes the input using the selected algorithm, and returns the result to the frontend.

```
@app.route('/salted-hash', methods=['GET', 'POST'])
def salted_hash():
    salted_hash = None
    if request.method == 'POST':
```

```

text = request.form.get('text', '') # Get text input
salt = request.form.get('salt', '') # Get custom salt
file = request.files.get('file') # Get uploaded file
algorithm = request.form.get('algorithm', '')

# If no salt is provided, generate a random one
if not salt:
    import os
    salt = os.urandom(16).hex()

# Read file content if a file is uploaded
file_content = ''
if file and file.filename:
    try:
        file_content = file.read().decode('utf-8') # Decode file content
to string
    except Exception as e:
        file_content = ''
        print(f"Error reading file: {e}")

# Combine text and file content
combined_input = text + file_content + salt

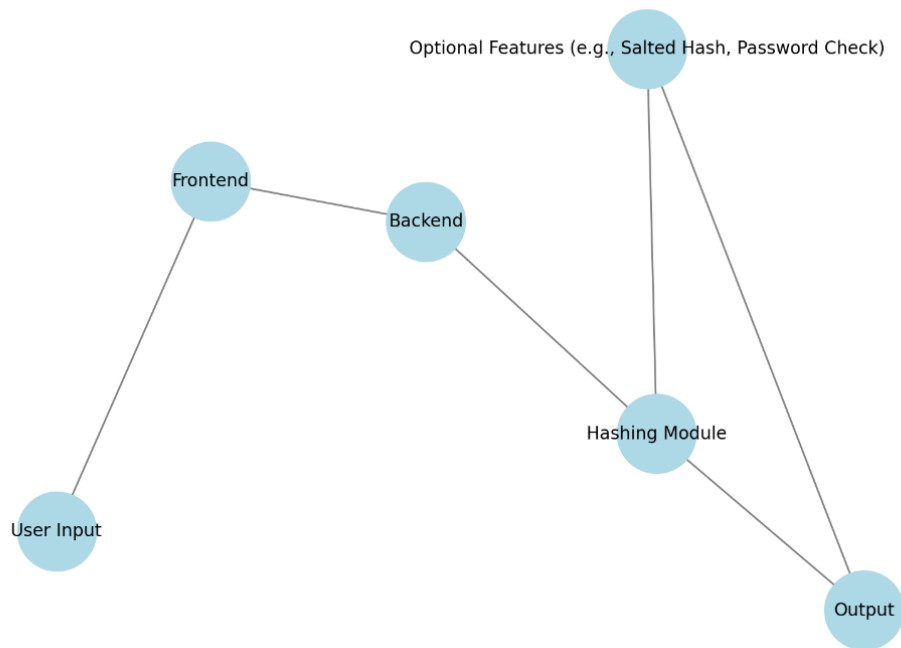
# Hash the combined input based on the selected algorithm
if algorithm == 'md5':
    salted_hash = hashlib.md5(combined_input.encode()).hexdigest()
elif algorithm == 'sha1':
    salted_hash = hashlib.sha1(combined_input.encode()).hexdigest()
elif algorithm == 'sha256':
    salted_hash = hashlib.sha256(combined_input.encode()).hexdigest()
elif algorithm == 'sha512':
    salted_hash = hashlib.sha512(combined_input.encode()).hexdigest()

return render_template('salted-hash.html', salted_hash=salted_hash)

```

## SYSTEM ARCHITECTURE DIAGRAM

System Architecture Diagram: Data Hashing Tool



## CONCLUSION

The **Data Hashing Tool** is a web application that offers hash generation, comparison, salted hashes, and password strength evaluation, providing valuable insights into cryptographic techniques and secure data practices. Its user-friendly interface and robust backend ensure an efficient, accessible experience. The tool highlights the importance of secure data management, with potential future enhancements like advanced hashing techniques, performance metrics, and external security API integration.

---