Project Report: Password Cracking and Hashing Algorithms

# 1. Introduction

Passwords are one of the most widely used authentication mechanisms, yet they are often vulnerable to attacks such as brute force, dictionary attacks, and credential stuffing. Weak passwords, combined with insecure hashing algorithms, can lead to major data breaches.  
  
This project demonstrates the implementation of password hashing using algorithms like MD5, SHA256, and bcrypt, followed by simulating password cracking using brute force and dictionary attacks. The goal is to understand how strong hashing algorithms protect passwords and why weak algorithms should be avoided.

# 2. Objectives

- To understand common password cracking techniques.  
- To implement password hashing in Python using different algorithms.  
- To simulate password cracking using brute force and dictionary attacks.  
- To analyze the strength of different hashing algorithms.  
- To recommend secure practices for password storage.

# 3. Tools & Environment

- Programming Language: Python 3.10+  
- Libraries: hashlib, bcrypt  
- Tools: Hashcat or John the Ripper (for cracking simulation)  
- Platform: Kali Linux / Windows

# 4. Implementation

## 4.1 Hashing Passwords

We implemented hashing in Python using hashlib and bcrypt.  
  
Example Code (SHA256):  
  
import hashlib  
password = 'mypassword'.encode()  
hash\_object = hashlib.sha256(password)  
print(hash\_object.hexdigest())  
  
Example Code (bcrypt):  
  
import bcrypt  
password = b'mypassword'  
hashed = bcrypt.hashpw(password, bcrypt.gensalt())  
print(hashed)  
  
[Insert Screenshot: Python terminal showing hashed passwords]

## 4.2 Password Cracking Simulation

We simulated cracking using brute force and dictionary attacks with tools like Hashcat and John the Ripper.  
  
Example Command (Hashcat):  
hashcat -m 0 -a 0 hashes.txt wordlist.txt  
  
[Insert Screenshot: Hashcat cracking results]  
  
Results showed that weak passwords such as '12345' or 'password' were cracked instantly, while strong passwords with special characters and long lengths resisted cracking.

# 5. Results

The following table shows the cracking times for different passwords:

|  |  |  |
| --- | --- | --- |
| Password | Hash Algorithm | Cracking Time |
| 12345 | MD5 | Instant (<1s) |
| password | SHA1 | Instant (<1s) |
| MyPass@123 | SHA256 | Few minutes |
| StrongPass!2024 | bcrypt | Not cracked (resistant) |

# 6. Mitigation

1. Use strong hashing algorithms like bcrypt, scrypt, or Argon2.  
2. Apply salts to prevent rainbow table attacks.  
3. Enforce strong password policies (minimum 12+ characters, mix of symbols, numbers, cases).  
4. Implement multi-factor authentication (MFA).  
5. Apply account lockouts and rate limiting against brute force attacks.

# 7. Conclusion

This project demonstrated how password cracking works and why strong hashing is essential for security. Simple algorithms like MD5 and SHA1 are vulnerable to fast cracking, while modern algorithms like bcrypt provide stronger resistance. By enforcing strong password policies and using advanced hashing, organizations can significantly improve their security posture.

# 8. Skills Learned

- Password hashing implementation (MD5, SHA, bcrypt)  
- Brute force and dictionary attack simulation  
- Password security analysis  
- Secure password storage practices  
- Cryptography fundamentals