**DIGITAL FORENSICS**

**Project - 1 Password Cracking**

**Run the code as**:

python password\_cracker.py

It will prompt to choose the method you want to use for cracking password.

Enter the option number only. Error handling is done for invalid inputs.

**For option 1:**

It will prompt you to enter a dictionary file. If the file in the same directory enter the name directly, else enter the full path name.

Error handling is been done to handle invalid path names.

Next, Enter the MD5 hash to be cracked. This code is tested against the 10 passwords from Lab-1.

The code handles errors if the hash length does not match the standard md5 hash length (32 characters)

**Explanation:**

Here, the function dict\_method () is called on selecting option 1.

After validating the input length of the hash, match\_hash () function is called. I am reading each word from the file and computing its hash and matching with the input hash. It prints the cracked password, time taken in seconds and no of attempts.

If the password is not in the wordlist it will prompt that password is not found.

**For Option 2:**

Enter the MD5 hash to be cracked

The code handles errors if the hash length does not match the standard md5 hash length (32 characters)

**Explanation:**

After validating the input length of the hash, generate\_hash () function is called. I have created a list of lowercase, and uppercase and tests only for password length up to 6 characters.

This list could be extended to adding numbers and special characters however the computational time will increase.

Here I am checking the input hash against the list in the iterative manner starting from single character to 6 characters from the list. It calls function compare\_hash () for the comparison.

It prints the computation time, no of attempts and the Password that is cracked.

I have tested the code for 4-5 character words with a mix and match of upper and lower case.