IMPLEMENTATION ASSIGNMENT

Working of the project:

It is developed as a command line execution using Ubuntu virtual machine and python language with Pycrypto library for cryptography.

Execute configset.py:

```
from Crypto import Random
from Crypto.Cipher import AES
import ConfigParser
import os

#Creating config file
cfgfile = open("appconfig.cfg",'w')

#initializing value of key and IV's
IV = 32 * '\x00'
IVhalf = 16 * '\x00'
keyrand = os.urandom(32)
masterkey = '7e472acf339432f257068efb56baf925cea5eeba675552aba34142ec616340ac'
msecret = 'a00bec422eb3e93cfc8315ba2fff0367b394be0d166bdd4b7be0734cfeb7e4bc'
enckey = msecret[:32]
encryptor = AES.new(enckey, AES.MODE_CBC, IV=IVhalf)
key = encryptor.encrypt(keyrand)

#Adding data to config file
config = ConfigParser.ConfigParser()

config.add_section('keys')
config.set('keys','key',key)
config.set('keys','mkey',masterkey)
config.set('keys','secret',msecret)
config.add_section('init_vector')
config.set('init_vector','IV', IV)
config.set('init_vector','IV2', IVhalf)
config.write(cfgfile)
```

NOTE: This script must only be run once. It is for setting up the necessary configurations in order to run the application.

This script creates a configuration file called “appconfig.cfg” and stores all the needed data in this file. This configuration file stores the initialization vectors needed for encryption, the encryption key which is generated randomly, the master key and master secret are hardcoded and stored in their sha-256 hashed forms. The encryption key is stored in an encrypted form which is encrypted using a key derived from the master secret.

Execute test2.py:

This script is the application script. Once the configset.py is executed and all the settings are in place the application script can be executed.

When the script is executed it first imports all the necessary libraries. **Pycrypto library is used for the encryption process.**

test2.py

```
#import crypto functions
from Crypto import Random
from Crypto.Cipher import AES
import os
import ConfigParser
import hashlib
import sys
```

After this, it extracts all the necessary information from the configuration file.

```
#Extract info from config file
config = ConfigParser.ConfigParser()
config.read("appconfig.cfg")
mkey = config.get('keys','mkey')
msecret = config.get('keys','secret')
IV1 = config.get('init_vector','IV')
IV2 = config.get('init_vector','IV2')

#Extracts encrypted key
enckey = config.get('keys', 'key')

#Get key from master secret to decrypt key
dkey = msecret[:32]

#decrypt key
decryptor = AES.new(dkey, AES.MODE_CBC, IV=IV2)
key = decryptor.decrypt(enckey)
```

Once these steps are complete, we are prompted for the master key. **By default the master key is "Open12saysme" and the master secret is "This is the secret data".**

```
preethi@preethi-VirtualBox:~$ python test2.py
Please enter the passcode: Open12saysme
```

The program code to check the master key is as follows.

```
#check masterkey
n = 3
while n > 0:
        #ASk for masterkey and check
        Masterkey = raw_input("Please enter the passcode: ")
        if hashlib.sha256(Masterkey).hexdigest() == mkey:
```

If the **correct master key** is entered the application lists the options we can perform.

```
        while option != '5':
                option = raw_input("\n1. Register \n2. Authentication \n3. Retreive password \n4. Change passcode \n5
Exit \n\n Enter choice:")

                if option == '1':
                        register()

                elif option == '2':
                        authentication()

                elif option == '3':
                        retrieve(None)

                elif option == '4':
                        changekey()

                elif option == '5':
                        sys.exit()

                else:
                        print "\n\n   Invalid option! Please enter the valid option"
```

```
preethi@preethi-VirtualBox:~$ python test2.py
Please enter the passcode: Open12saysme

1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:
```

If the **master key entered is wrong** then the application provides us with three chances to enter the correct key. If we don't enter the correct key by the third attempt then the application is terminated.

```
        else:
                print "Incorrect passcode!"
                n = n-1
if n == 0:
        print "Too many wrong attempts....Application terminated"
```

```
preethi@preethi-VirtualBox:~$ python test2.py
Please enter the passcode: hello1
Incorrect passcode!
Please enter the passcode: hello2
Incorrect passcode!
Please enter the passcode: hello3
Incorrect passcode!
Too many wrong attempts....Application terminated
preethi@preethi-VirtualBox:~$
```

Specification: Given a (Username, Password) pair in ASCII; store the pair to a file.
Specification: Using a flag the user should be able to choose ECB, CTR or CBC modes.

Register():

- It gets the username, password, secret data and the mode of encryption as input from the user. The secret data is used for password recovery.
- It checks if username already exists and if so asks the user to enter another username.

```
Enter username: henry
Username already exists
```

- It checks if the password is 8 characters long (minimum length) and checks to see if it has atleast one capital letter, atleast one small letter and atleast one number.

```
Enter password: henry
Password must be atleast 8 characters long

Enter username: henry

Enter password: Henryhenry
Password must contain one capital letter, one small letter and one number
```

- It concatenates all the information using a ":" to the form **username:password:secretdata**. As a result, it ensures that this special character along with # and $ are not used in the input. The # and $ are used just to confuse the attacker.

```
Enter username: rog:er

Enter password: Rogerroger2

Enter secret phrase incase of password retrieval: rogersecret

 One of the inputs contain #, $ or :
```

```
Enter username: roger

Enter password: Roge#rroger2

Enter secret phrase incase of password retrieval: rogersecret

 One of the inputs contain #, $ or :
```

```
Enter username: roger

Enter password: Rogerroger2

Enter secret phrase incase of password retrieval: roger$secret

 One of the inputs contain #, $ or :
```

- Once all these checks are complete, the information is stored into one of the three files (ECB file, CBC file and CTR file) based on the mode of encryption and is then encrypted.
- The encrypted data is stored in another file and the plaintext file is removed from the system.
- If this is not the first data to be stored in file then the encryption file is decrypted into a temporary plaintext file and then the information is appended to the file after which the file is encrypted. The same process of storing the encrypted data into another file and the plaintext file being removed is carried out.

```
 Enter choice:1

#, $ and : special characters are not allowed

Password must contain atleast 8 characters and one capital letter, one small let
ter and one number

Enter username: henry

Enter password: Henryhenry2

Enter secret phrase incase of password retrieval: henrysecret

Please enter the mode of encryption
1. e:ECB
2. r:CTR
3. c:CBC
Enter choice: e

1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:
```

```
Enter username: caroline

Enter password: Carolinecaroline2

Enter secret phrase incase of password retrieval: carolinesecret

Please enter the mode of encryption
1. e:ECB
2. r:CTR
3. c:CBC
Enter choice: r

1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:
```

```
Enter username: julian

Enter password: Julianjulian2

Enter secret phrase incase of password retrieval: juliansecret

Please enter the mode of encryption
1. e:ECB
2. r:CTR
3. c:CBC
Enter choice: c

1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:
```

Invalid mode option:

```
Please enter the mode of encryption
1. e:ECB
2. r:CTR
3. c:CBC
Enter choice: i

 Enter a valid option

Please enter the mode of encryption
1. e:ECB
2. r:CTR
3. c:CBC
Enter choice:
```

Specification: Given a (Username, Password) pair in ASCII; check if the username exists and if the password matches the one stored in a file.

Authentication():

- It accepts a username and password as input.
- It decrypts each mode file and checks to see if there is a match after which the decrypted file is deleted from the system.
- If **authentication succeeds**, a success message is displayed and control is returned back to options.

```
1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:2

Enter username: henry

Enter password: Henryhenry2

Authenticated
```

- If **authentication fails** then two more attempts are provided before completely terminating the application.

```
1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:2

Enter username: kjdsbg

Enter password: lsrkjbg

Authentication failed
Try Again!

Enter username: Henry

Enter password: henryhenry2

Authentication failed
```

```
Try Again!

Enter username: Roger

Enter password: ksebg

Authentication failed
Too many wrong attempts....Application terminated
preethi@preethi-VirtualBox:~$
```

Specification: Given a username, retrieve the password.

Retrieve():
This function is also used for checking if the username exists.

Check if username exists:

- Takes username as argument and is called from register() function.
- Checks each file to see if username exists.
- If match found then the function returns True. Otherwise it returns false.

Retrieve password:

- When calling the function a “None” value is passed in order to make the username value, used for checking if the username exists, a null value.
- It takes in the username and secret data as input.
- Checks each file for a match.
- If a match is found then the corresponding password is displayed. If not we are brought back to the list of options.

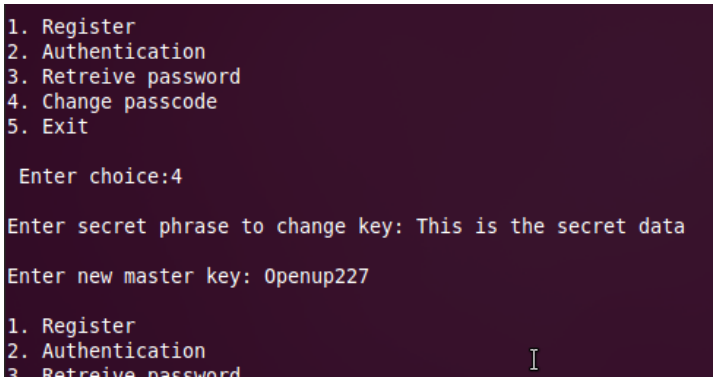
```
1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:3

Enter the username: roger

Enter secret phrase: rogersecret
Your password is Rogerroger2
```

```
1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:3

Enter the username: henry

Enter secret phrase: don'tknow

 There is no match for the given information. Please try again!!
```

Changing masterkey

Changekey():

- It accepts the master secret as input from the user and checks with the master secret stored in the configuration file.
- If there is a match then the user is prompted to enter a new master which will be overwritten in the configuration file.




```
preethi@preethi-VirtualBox:~$ python test2.py
Please enter the passcode: Openup227

1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:
```

Finally choosing to exit the program:

```
1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:5
preethi@preethi-VirtualBox:~$
```

If any other option other than that provided is given as input then the following occurs.

```
1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:h


   Invalid option! Please enter the valid option

1. Register
2. Authentication
3. Retreive password
4. Change passcode
5. Exit

 Enter choice:
```

If user1 and user2 have the same password, why do they show up differently?

In this application the files that store the password are encrypted rather than the user information. A check is put in place to ensure that usernames are unique and there is no repetition. As a result, even if two users have the same password they will have different usernames. Attached to the username and password pair is some secret information that the user enters when they register this pair. The chances of the secret data being the same for two users with the same passwords are very less. Due to these constraints, if user1 and user2 have the same password they will show up differently.

How can we make a bruteforcers' job harder if they had the masterkey?

If the attacker obtains the masterkey, they can gain access into the application. However, the attacker cannot do anything more than that. Brute-force attack is an attack where all possible combinations are tried to find the correct combination. The key length used in the encryption algorithm

determines the practical feasibility of performing a brute-force attack. The bigger the key the harder it is to crack it. This application uses the AES-256 algorithm for encryption and decryption. Therefore, it uses a 256-bit key. The number of possible combinations available for this key is 2^{256} which is $1.15 * 10^{77}$. Even with the use of a supercomputer it will take billions of years to crack the key. As a result, brute-force attack is not possible on a 256-bit AES encryption.

Now, consider the case where the master key can be changed by the user. In this situation, as illustrated in the previous explanation of how the application works, the master secret information is required to change the master key. Therefore, the attacker cannot change the master key. This provision is provided so that if the user detects any compromise then the user can change the master key.

Security features:

- The master key and the secret data required to change the master key is hardcoded and saved in their hashed forms. Therefore, even if the attacker gains access to the configuration file they will not be able to find the master key or the master secret.
- The master key and the master secret are only known to the developer and the user.
- The encryption key is encrypted using a key derived from the master secret and stored in the configuration file. Master secret is used instead of master key because if the master key is compromised then there is no need to worry about the encryption key. This is because the encryption key will not have any relation to the master key.
- The master key is allowed to be changed because if there is any detection of a breach then the user can improve security by changing the master key.
- In order to change the master key the master secret must be known to the attacker which is again stored in its hashed form.
- Password constraints are placed to make brute forcing hard.
- Number of attempts on the master key as well as authentication is restricted to allow limited chances for the attacker to gain access.