بخش اول:

در این قسمت از کتابخانه های زیر استفاده شده است به منظور گفته شده :

# Used libraries:  
import argparse # For command line argument processing  
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC # For deriving key  
from cryptography.hazmat.backends import default\_backend # For selecting cryptography backend  
from cryptography.hazmat.primitives import hashes # For using hash functions  
from base64 import urlsafe\_b64encode, urlsafe\_b64decode # For performing Base64 operations  
import os # For generating random numbers and interacting with files

از تابع های زیر به منظور های گفته شده است:

یک کلاس به منظور عملکرد کلی سیستم و تعریف تابع ها:

class PasswordManager:  
 *"""  
 PasswordManager class for managing encrypted passwords.  
 """*

تعریف کلید اصلی برای رمز کردن رمز عبور ها و ذخیره آنها در زمان شی سازی از کلاس :

def \_\_init\_\_(self, key):  
 *"""  
 Initializes the PasswordManager with a master key.  
  
 Parameters:  
 - key (str): The master key for password encryption and decryption.  
 """* self.key = key

تابع رمزنگاری با استفاده از AES و PBKDF2 :

def encrypt\_password(self, password):  
 *"""  
 Encrypts a password using PBKDF2 key derivation and AES encryption.  
  
 Parameters:  
 - password (str): The password to be encrypted.  
  
 Returns:  
 - str: The encrypted password.  
 """* kdf = PBKDF2HMAC(  
 algorithm=hashes.SHA256(),  
 length=32,  
 salt=os.urandom(16),  
 iterations=100000,  
 backend=default\_backend()  
 )  
 key = urlsafe\_b64encode(kdf.derive(self.key.encode()))  
 cipher\_text = urlsafe\_b64encode(password.encode())  
 return key + cipher\_text

تابع رمزگشایی:

def decrypt\_password(self, encrypted\_password):  
 *"""  
 Decrypts an encrypted password using the master key.  
  
 Parameters:  
 - encrypted\_password (str): The encrypted password.  
  
 Returns:  
 - str: The decrypted password.  
 """* key = urlsafe\_b64encode(self.key.encode())  
 cipher\_text = urlsafe\_b64decode(encrypted\_password)  
 return urlsafe\_b64decode(cipher\_text[len(key):]).decode()

تابع ذخیره سازی:

این تابع نام و کامنت مربوط به رمز رمزنگاری شده را به همراه خودرمزنگاری رمز در فایل تکستی ذخیره میکند.

def save\_password(self, name, comment, password):  
 *"""  
 Saves a new password entry to a text file.  
  
 Parameters:  
 - name (str): The name associated with the password.  
 - comment (str): A comment or description for the password.  
 - password (str): The password to be saved.  
 """* encrypted\_password = self.encrypt\_password(password)  
 with open('passwords.txt', 'a') as file:  
 file.write(f"{name}:{comment}:{encrypted\_password}\n")

تابع های نمایش کل رمزهای ذخیره شده یا نمایش یک رمز با نام داده شده:

@staticmethod  
def show\_passwords():  
 *"""  
 Shows all passwords stored in the text file.  
 """* with open('passwords.txt', 'r') as file:  
 for line in file:  
 data = line.strip().split(':')  
 if len(data) >= 3:  
 name, comment, \_ = data[:3]  
 print(f"Name: {name}, Comment: {comment}")  
 else:  
 print(f"Invalid line: {line}")

def show\_password(self, name):  
 *"""  
 Shows details of a specific password.  
  
 Parameters:  
 - name (str): The name associated with the password.  
 """* with open('passwords.txt', 'r') as file:  
 for line in file:  
 data = line.strip().split(':')  
 if data[0] == name:  
 decrypted\_password = self.decrypt\_password(data[2])  
 print(f"Name: {data[0]}, Comment: {data[1]}, Password: {decrypted\_password}")

تابع های بروزرسانی و حذف رمز در فایل ذخیره سازی:

def update\_password(self, name, new\_password):  
 *"""  
 Updates the password for a specific entry.  
  
 Parameters:  
 - name (str): The name associated with the password.  
 - new\_password (str): The new password to be updated.  
 """* with open('passwords.txt', 'r') as file:  
 lines = file.readlines()  
 with open('passwords.txt', 'w') as file:  
 for line in lines:  
 data = line.strip().split(':')  
 if data[0] == name:  
 updated\_line = f"{name}:{data[1]}:{self.encrypt\_password(new\_password).decode()}\n"  
 file.write(updated\_line)  
 else:  
 file.write(line)  
  
@staticmethod  
def delete\_password(names):  
 *"""  
 Deletes password entries for specified names.  
  
 Parameters:  
 - names (list): A list of names for passwords to be deleted.  
 """* with open('passwords.txt', 'r') as file:  
 lines = file.readlines()  
 with open('passwords.txt', 'w') as file:  
 for line in lines:  
 data = line.strip().split(':')  
 if data[0] not in names:  
 file.write(line)

تابع اصلی برای تعریف نحوه گرفتن ورودی در cmd:

def main():  
 *"""  
 Main function for handling command-line interface operations.  
 """* parser = argparse.ArgumentParser(description="Password Manager CLI")  
 parser.add\_argument("--newpass", help="Create a new password", nargs=3)  
 #parser.add\_argument("--c", help="Comment for the password", nargs=1)  
 #parser.add\_argument("--key", help="User simple password", nargs=1)  
 parser.add\_argument("--showpass", help="Show all passwords", action="store\_true")  
 parser.add\_argument("--sel", help="Show a specific password", nargs=1)  
 parser.add\_argument("--update", help="Update a password", nargs=1)  
 parser.add\_argument("--delete", help="Delete a password", nargs='+') # Allow one or more names

parser.add\_argument("--test", help="run test part", action="store\_true")

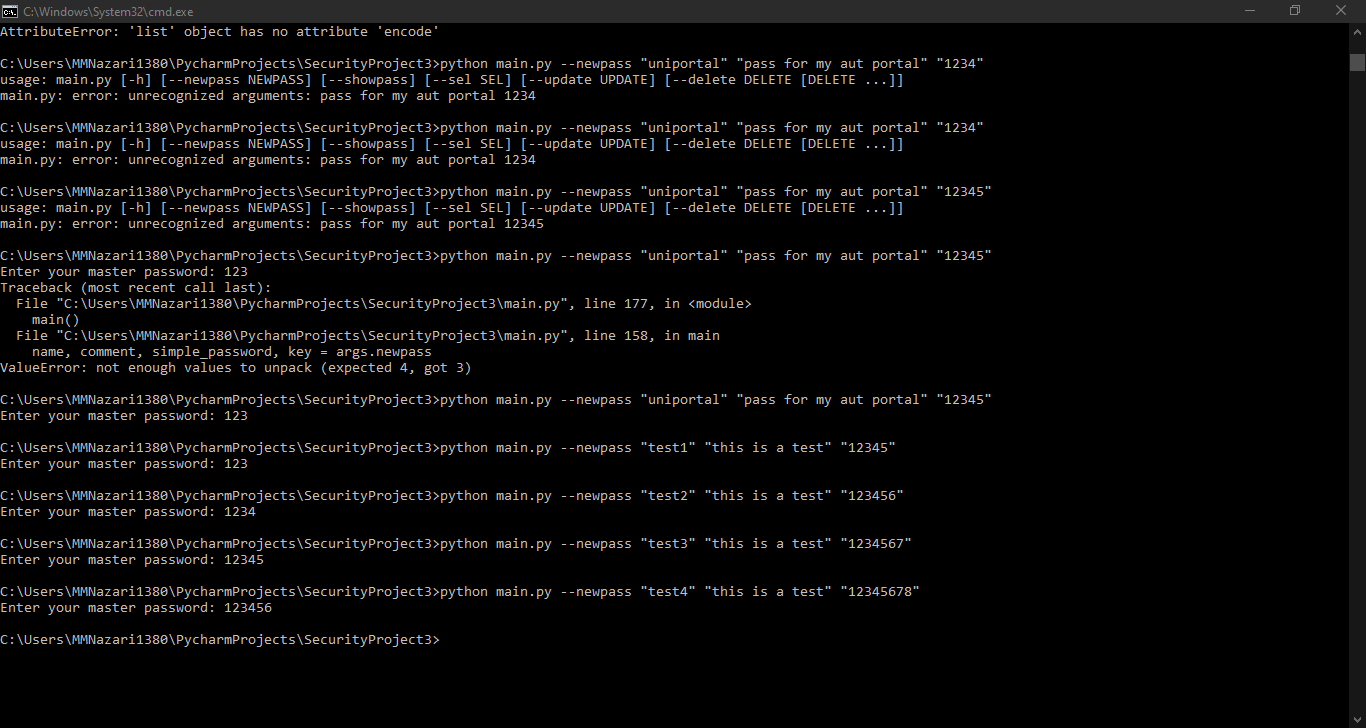
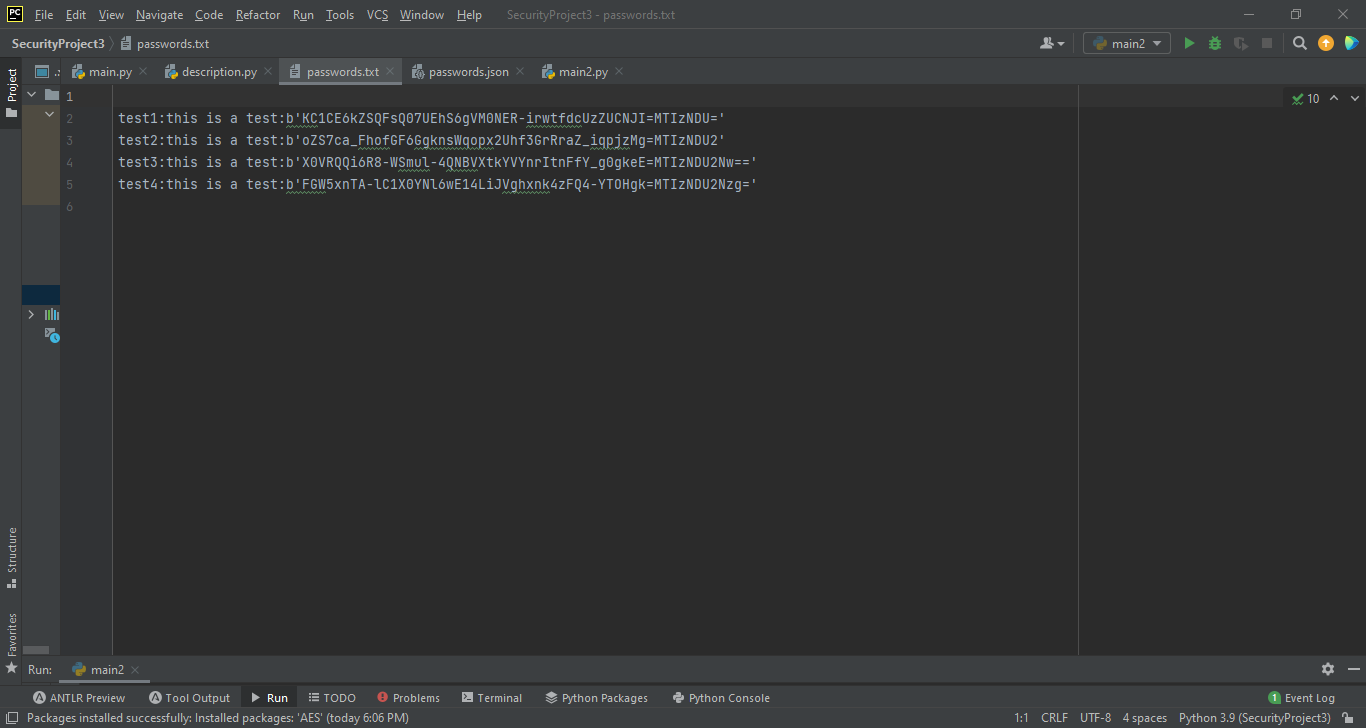
args = parser.parse\_args()  
  
 key = input("Enter your master password: ")  
 password\_manager = PasswordManager(key)  
  
 '''  
 if args.newpass and args.c and args.key:  
 name = args.newpass  
 comment = args.c  
 user\_password = args.key  
 password\_manager.save\_password(name, comment, user\_password)  
 '''  
  
 if args.newpass:  
 name, comment, simple\_password = args.newpass  
 password\_manager.save\_password(name, comment, simple\_password)  
 elif args.showpass:  
 password\_manager.show\_passwords()  
 elif args.sel:  
 name = args.sel[0]  
 password\_manager.show\_password(name)  
 elif args.update:  
 name = args.update[0]  
 new\_password = input("Enter the new password: ")  
 password\_manager.update\_password(name, new\_password)  
 elif args.delete:  
 names = args.delete  
 password\_manager.delete\_password(names)

elif args.test:  
 password\_manager.save\_test()  
 else:  
 print("Invalid command. Use --help for usage information.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

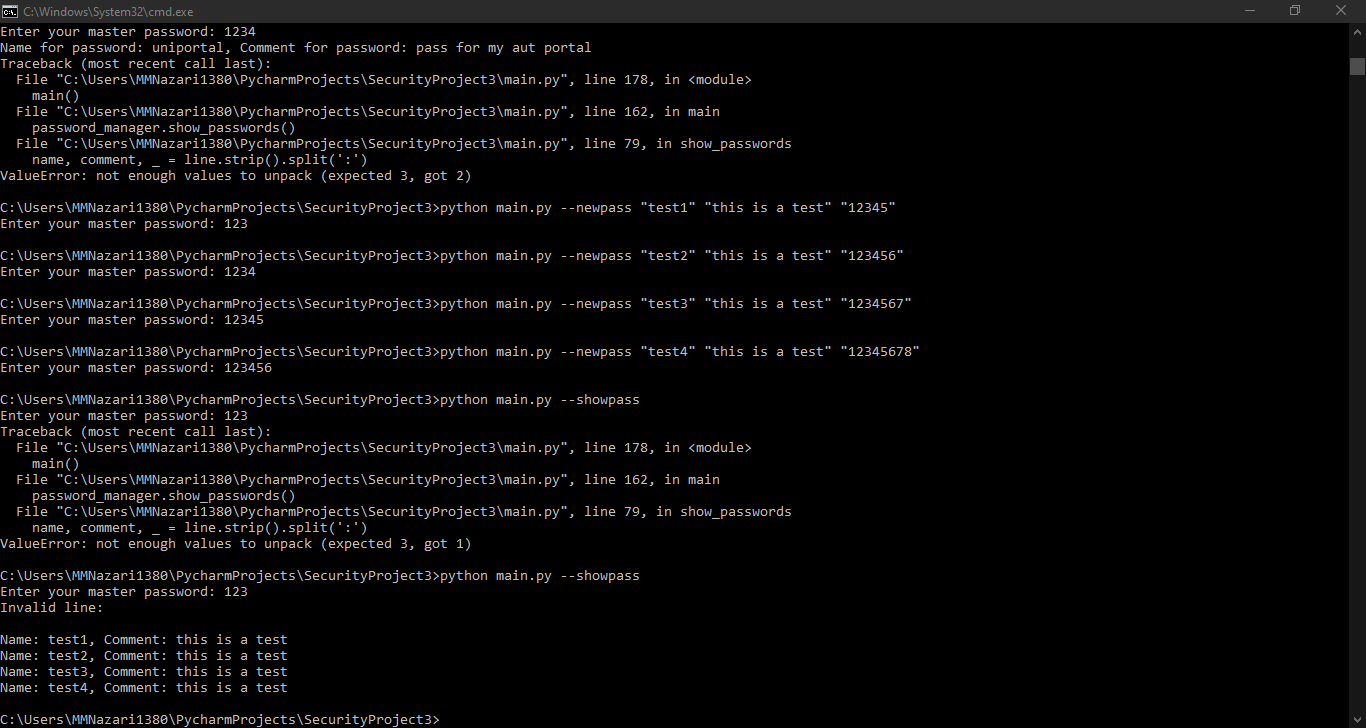
تست سناریو های مختلف:

ساخت رمز جدید:

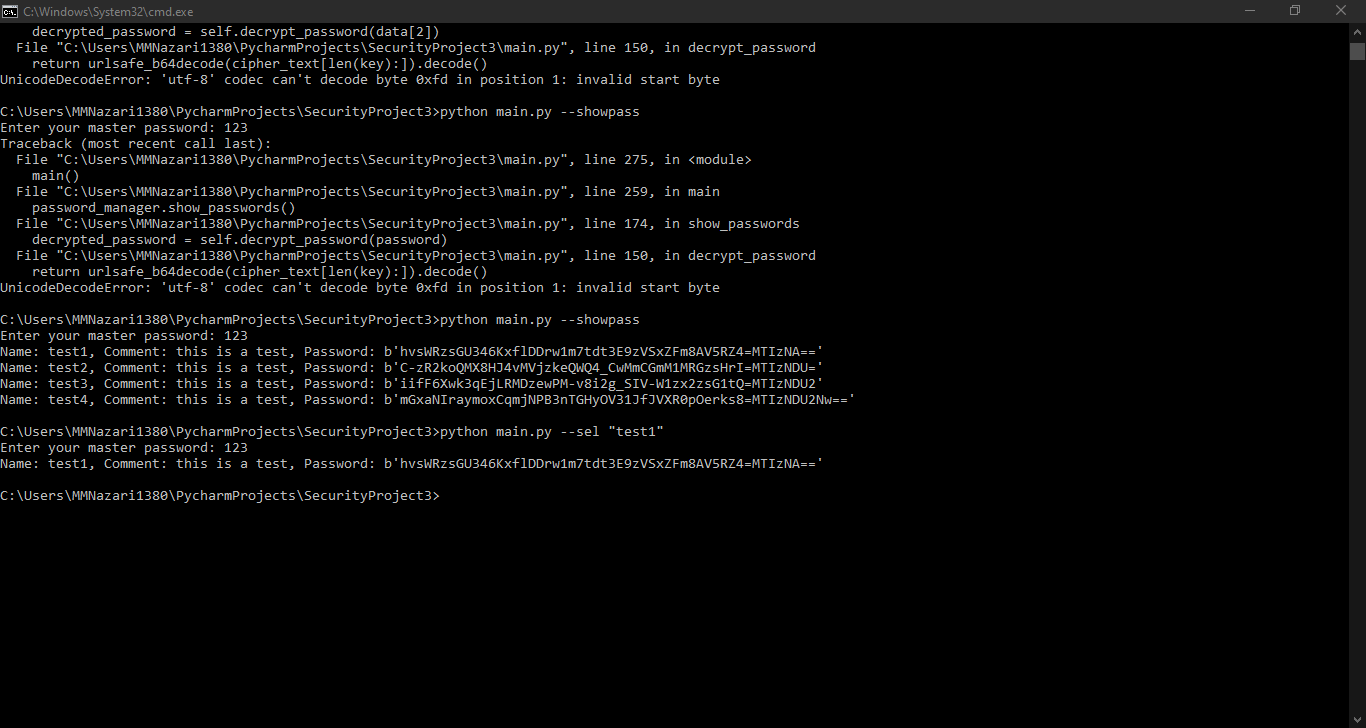
لازم به ذکر است که در کامند لاین دیگر سوییچ های --c و --key پیاده سازی نشده اند و هر سه پارامتر به سوییچ –newpass پاس داده میشوند.



نمایش همه رمزها:

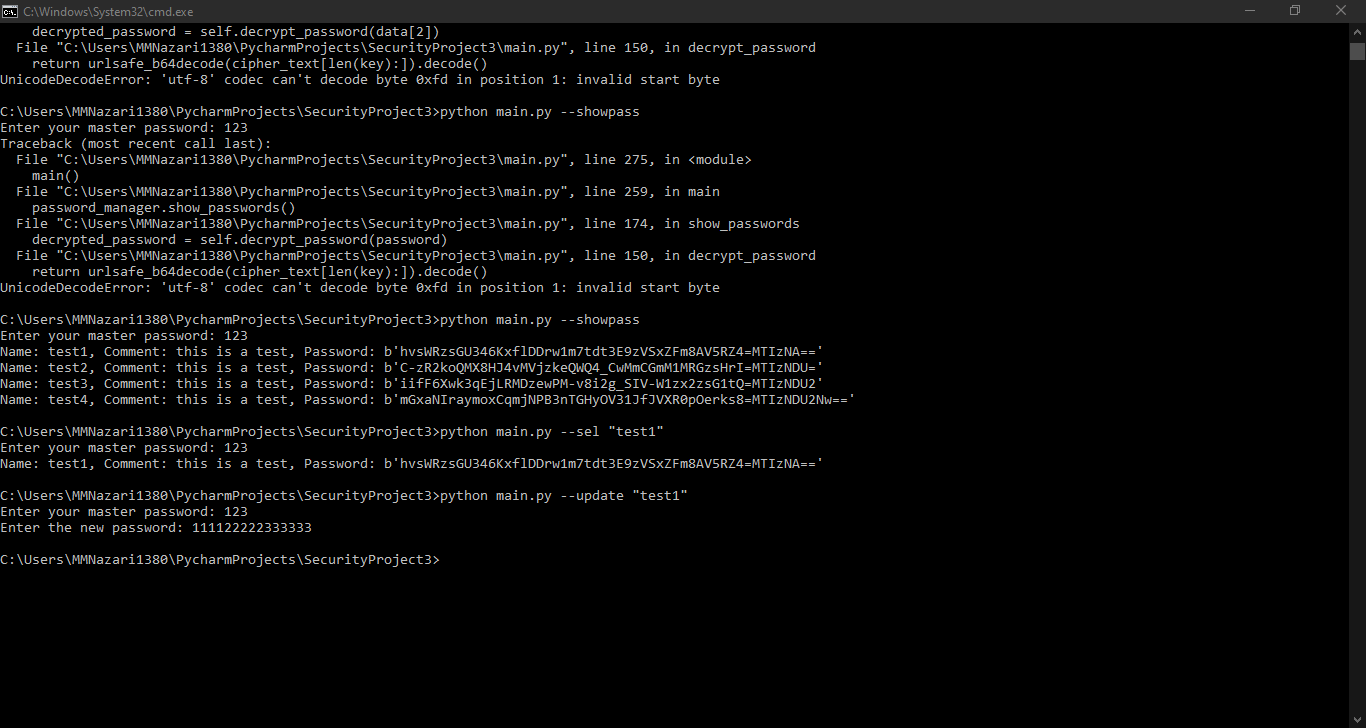


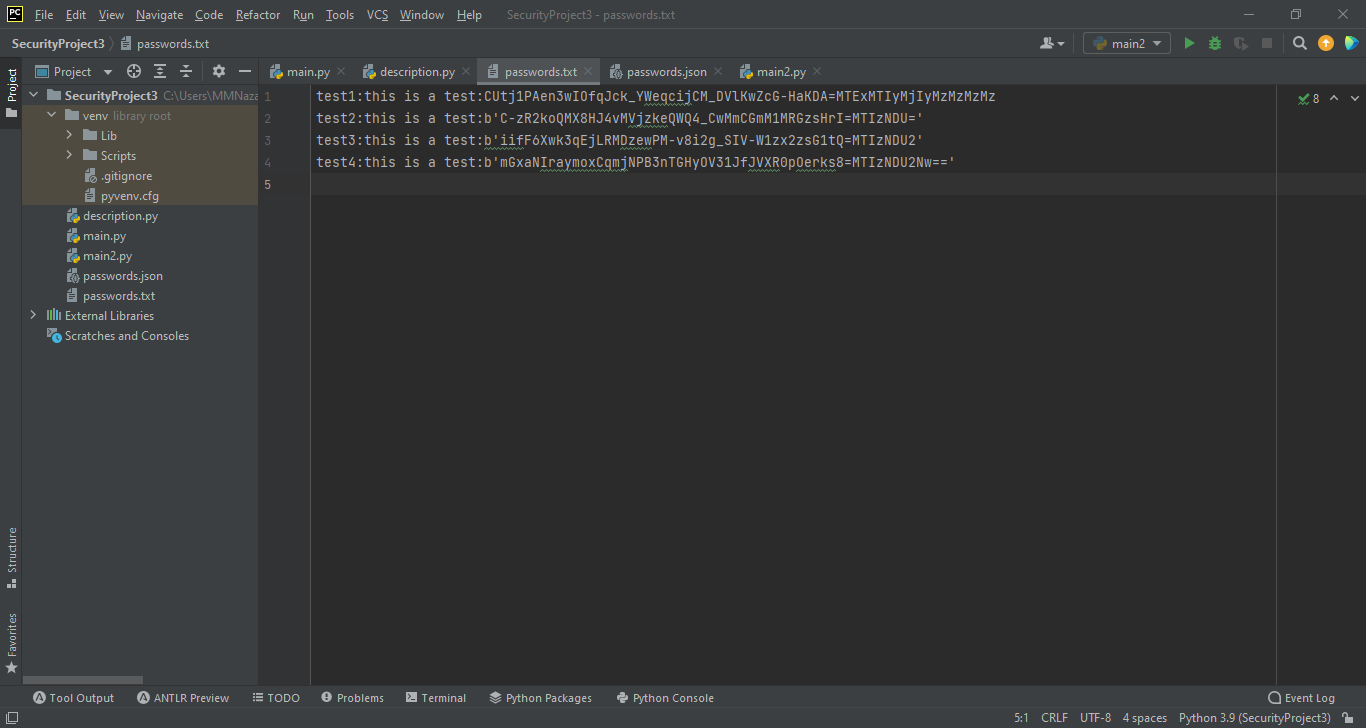
نمایش یک رمز با اسم خاص:



\* در رابطه با نمایش رمز بصورت رمز شده این نکته باید لحاظ شود که میدانیم که رمز باید بصورت رمز شده در فایل ذخیره شود که این امر رعایت شده است و در زمان نمایش باید رمزگشایی شود و نمایش شود و برای این امر تابع رمزگشایی هم نوشته شده اما چونکه به مشکلات زاد و حل نشدنی از جهت انکودینگ و دیکودینگ، کار با فرمت utf-8 و ... برخوردم نتوانستم فرم رمزگشایی شده را نمایش دهم و همان فرم رمز شده نمایش داده میشود. تابع رمزنگاری ونحوه نمایش را هم تغییر دادم اما مشکل را حل نکرد. این کدها کامنت شده اند.

بروزرسانی یک رمز:

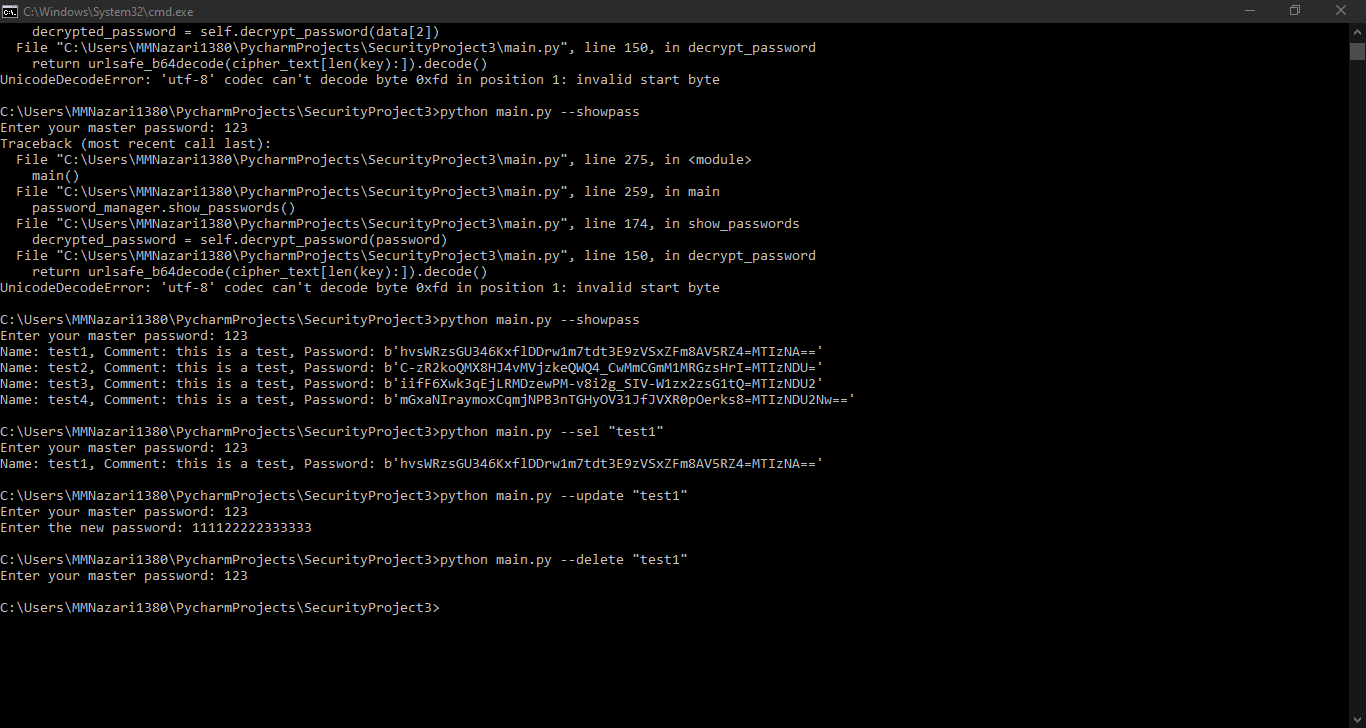


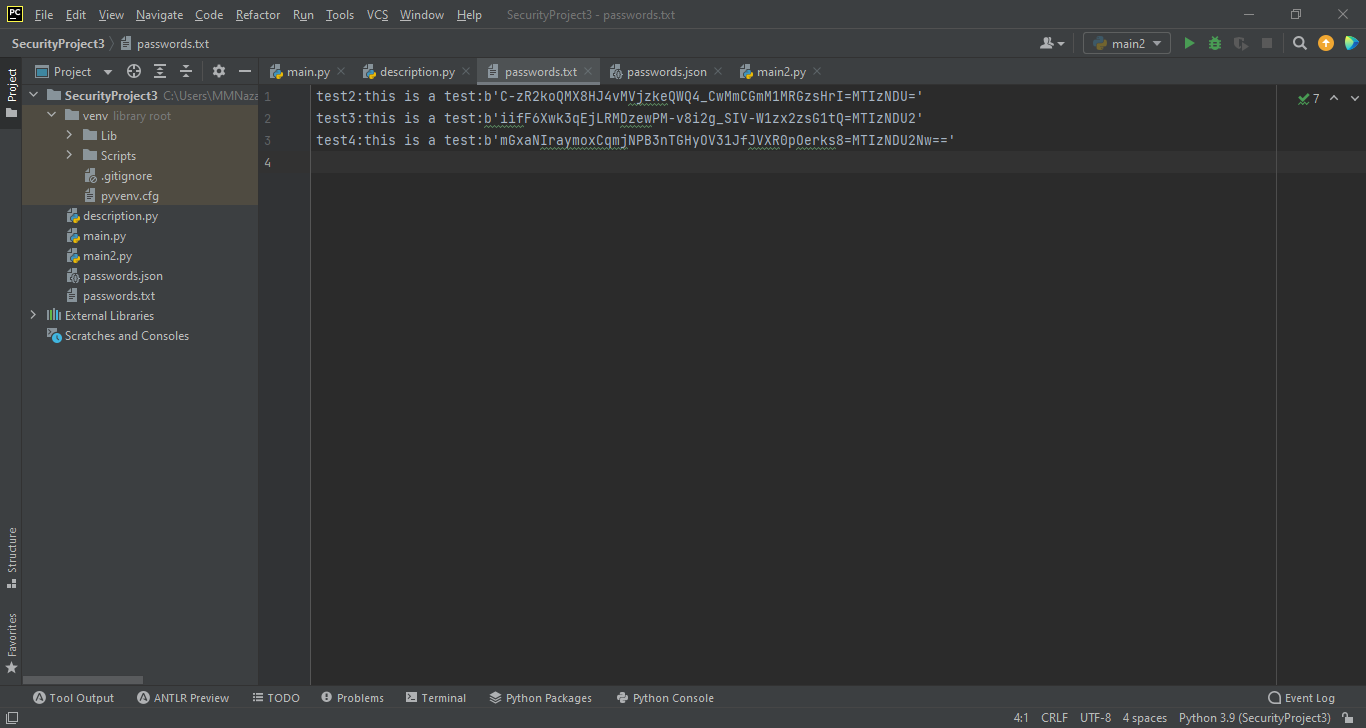




همانطور که میبینیم رمز رمزنگاری شده تغییر کرده است.

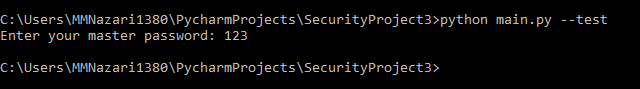
حذف یک نمونه:



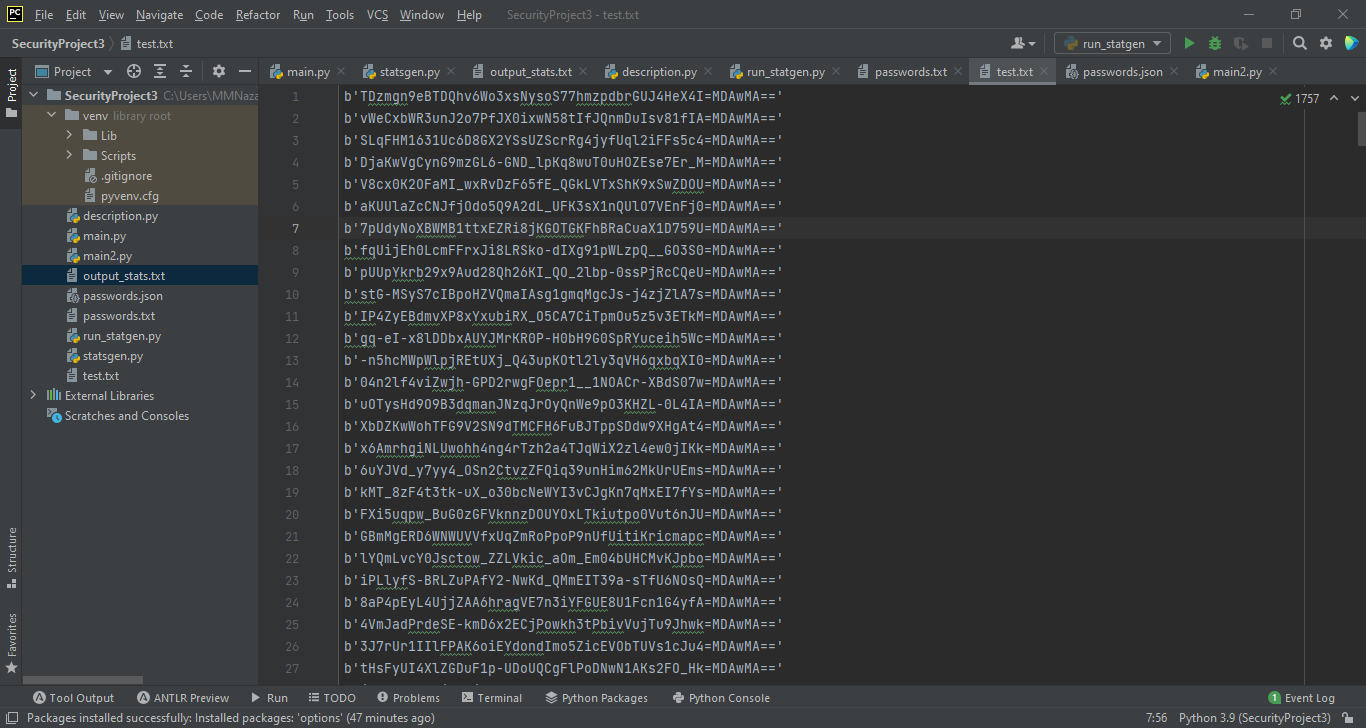


بخش دوم:

در این قسمت یک تابع دیگر پیاده سازی کردیم که در هر بار پیمایش کلید رمزنگاری را شماره متغیر داخل حلقه قرار داده و ورودی "0000" را با همان الگوریتم رمزنگاری قسمت قبل رمز میکند. سوییچ –test برای این قسمت تعریف شده است.



محتوای فایل رمزها:



اما با توجه به فرمت رمزهای ذخیره شده دربرنامه برای statsgen قابل ارزیابی نیستند اما برنامه اجرا شده و فایل خروجی ساخته شده است. لازمبه ذکر است که فایل statsgen ارایه شده دارای مشکل بود و مشکلات آن بررف شد و سپس برنامه تست شد.

