گزارش پروژه سوم – امنیت اطلاعات

* بخش اول : پیاده سازی یک ابزار مدیریت گذرواژه و ساخت گذرواژه های پیچیده

در این بخش از برنامه ما قصد داریم کلید و اطلاعات و نام یک گذرواژه را گرفته، با استفاده از کلید آن را رمز کنیم و ذخیره کنیم. برای این منظور دستور newpass بصورت زیر پیاده سازی شد:

if args.newpass:  
 name, comment, key = args.newpass  
 if os.path.exists(passwords\_file):  
 decrypt\_passwords\_file(key)  
 enc = save\_password(name, comment, key)  
 print('New Password added!')  
 print(f'Encrypted Password : {enc}')  
 encrypt\_passwords\_file()

ابتدا چک میکنیم اگر فایل رمزها وجود دارد با استفاده از کلید دریافتی آن را رمزگشایی میکنیم. اگر هم وجود نداشت با فراخوانی تابع save\_passwords این فایل ایجاد میشود. در این تابع که بصورت زیر است:

def save\_password(name, comment, key):  
 generated\_password = generate\_password(name, comment, key)  
 encrypted\_password = encrypt(generated\_password, key)  
 entry = {"name": name, "password": encrypted\_password, "comment": comment, "key": key}  
 with open(passwords\_file, "a") as file:  
 file.write(json.dumps(entry) + "\n")  
  
 return encrypted\_password

ابتدا یک فایل متنی با استفاده از تابع generate\_passwords میسازیم:

def generate\_password(name, comment, key):  
 # Generating a random salt  
 salt = ''.join(random.choices(string.ascii\_letters + string.digits, k=16))  
  
 # Combining name, comment, key, and salt for password generation  
 password\_input = f"{name}\_{comment}\_{key}\_{salt}"  
 generated\_password = hashlib.sha256(password\_input.encode()).hexdigest()  
  
 return generated\_password

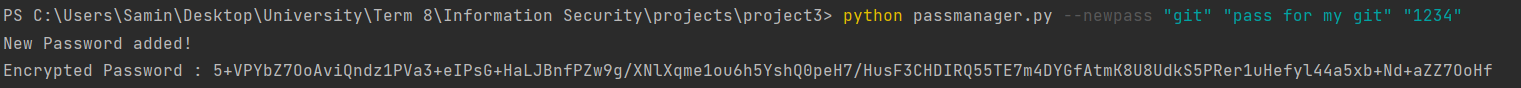
که همانطور که مشاهده میشود با انتخاب رندوم یک سالت و ترکیب بخش های مختلف رمز و در نهایت هش کردن آن متن رمز را میسازد و برمیگرداند.

در گام بعدی این متن و کلید به تابع encrypt فرستاده میشود:

def encrypt(text, key):  
 key = derive\_key(key)  
 cipher = AES.new(key, AES.MODE\_EAX)  
 ciphertext, tag = cipher.encrypt\_and\_digest(text.encode())  
 return base64.b64encode(cipher.nonce + tag + ciphertext).decode()

در این تابع ابتدا از کلید ورودی یک کلید که مناسب رمزنگاری AES باشد با استفاده از تابع derive\_key ایجاد کرده و با استفاده از الگوریتم AES آن را رمز میکنیم و برمیگردانیم. تابع مشتق گیرنده کلید نیز به صورت زیر است:

def derive\_key(password, salt=b'some\_salt', iterations=100000, key\_length=32):  
 key = hashlib.pbkdf2\_hmac('sha256', password.encode(), salt, iterations, key\_length)  
 return key

در نهایت save\_passwords رمز را برگردانده و به کاربر نمایش داده میشود، سپس فایل رمزها رمز و ذخیره میشود. خروجی این بخش بصورت زیر است:

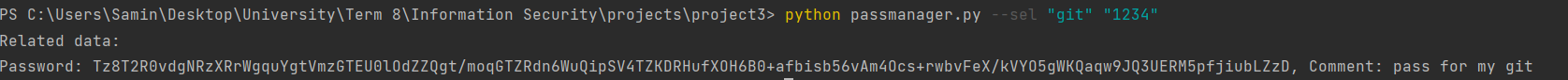
در بخش بعدی میخواهیم اطلاعات یک رمز مشخص را ببینیم. برای این منظور دستور sel پیاده سازی شده است:

elif args.sel:  
 name, key = args.sel  
 decrypt\_passwords\_file(key)  
 print('Related data: ')  
 select\_password(name)  
 encrypt\_passwords\_file()

در این بخش پس از دریافت نام رمز و کلید اصلی با فراخوانی تابع select\_password پس از رمزگشایی اطلاعات برگردانده شده سپس فایل مجددا رمز نگاری میشود.

def select\_password(name):  
 with open(passwords\_file, "r") as file:  
 for line in file:  
 entry = json.loads(line)  
 if entry["name"] == name:  
 print(f"Password: {entry['password']}, Comment: {entry['comment']}")  
 break

در این تابع به سادگی پس از لود کردن داده های جیسون اطلاعات مربوط به رمز مدنظر برگردانده میشود.

خروجی این بخش بصورت زیر است:

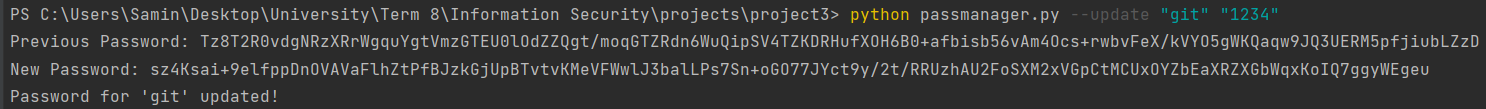
در بخش بعدی میخواهیم رمز را اپدیت کنیم به این منظور دستور update تعریف شده است:

elif args.update:  
 name, key = args.update  
 decrypt\_passwords\_file(key)  
 update\_password(name)  
 encrypt\_passwords\_file()

در این دستور پس از رمز گشایی فایل اسم آن را به تابع update\_passwords برمیگردانیم.

def update\_password(name):  
 global key  
 with open(passwords\_file, "r") as file:  
 entries = [json.loads(line) for line in file]  
  
 updated\_entries = []  
  
 for entry in entries:  
 if entry["name"] == name:  
 print(f'Previous Password: {entry["password"]}')  
 # Generate a new password for the existing entry  
 new\_generated\_password = generate\_password(entry["name"], entry["comment"], entry["key"])  
  
 # Update the password field in the entry  
 entry["password"] =encrypt(new\_generated\_password, entry["key"])  
  
 print(f'New Password: {entry["password"]}')  
  
 updated\_entries.append(entry)  
  
 with open(passwords\_file, "w") as file:  
 for entry in updated\_entries:  
 file.write(json.dumps(entry) + "\n")  
  
 print(f"Password for '{name}' updated!")

در این بخش پس از شناسایی رمز خواسته شده به شیوه تولید رمز اولیه رمز جدیدی برای آن ساخته میشود که همانطور که قبلا گفتیم بعلت اینکه بر اساس سالت رندوم و عملکرد هش در گام اول و ساختن کلید مشتق برای AES که با سالت رندوم است جلو میرود هرگز تکراری نخواهد بود.

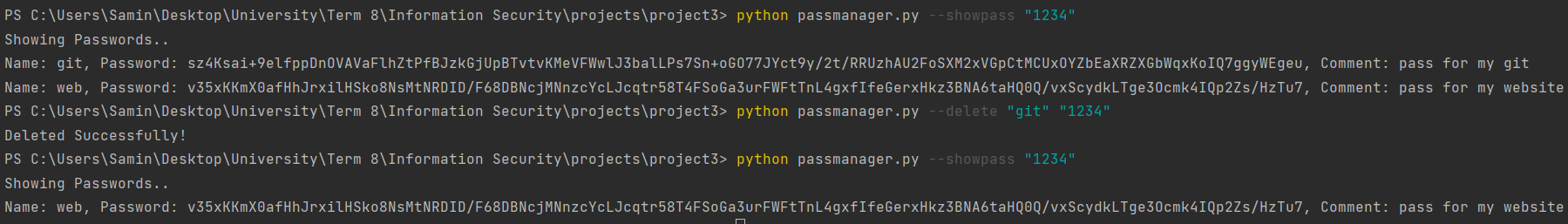
خروجی این بخش:

در بخش بعدی ما قصد حذف کردن یک رمز را داریم برای این منظور دستور delete پیاده سازی شده است:

elif args.delete:  
 name, key = args.delete  
 decrypt\_passwords\_file(key)  
 delete\_password(name)  
 encrypt\_passwords\_file()  
 print('Deleted Successfully!')

پس از رمزگشایی تابع delete\_passwords فراخوانی شده و پس از حذف فایل مجددا رمزنگاری میشود.

def delete\_password(name):  
 with open(passwords\_file, "r") as file:  
 entries = [json.loads(line) for line in file]  
 with open(passwords\_file, "w") as file:  
 for entry in entries:  
 if entry["name"] != name:  
 file.write(json.dumps(entry) + "\n")

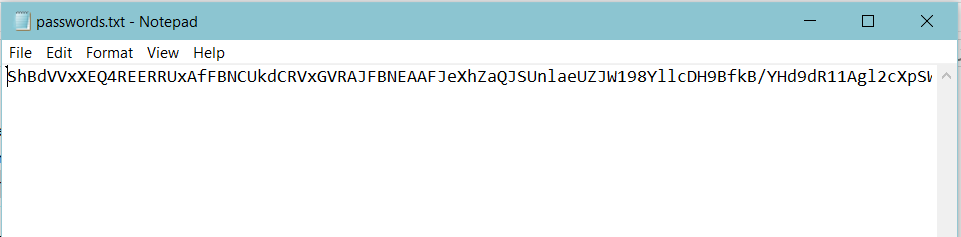
در این تابع پس از شناسایی رمز خواسته شده آن را حذف میکنیم. خروجی این بخش:

لازم به ذکر است که الگوریتم رمزنگاری XOR برای فایل درنظر گرفته شده:

# Encrypt passwords.txt using XOR encryption  
def encrypt\_passwords\_file():  
 with open(passwords\_file, "r") as file:  
 plaintext = file.read()  
 encrypted\_data = xor\_encrypt(plaintext, key)  
 with open(passwords\_file, "w") as file:  
 file.write(encrypted\_data)

def xor\_encrypt(data, key):  
 # Convert data and key to bytes  
 data\_bytes = data.encode()  
 key\_bytes = key.encode()  
  
 # Repeat the key to match the length of data  
 repeated\_key = key\_bytes \* (len(data\_bytes) // len(key\_bytes)) + key\_bytes[:len(data\_bytes) % len(key\_bytes)]  
  
 # Perform XOR operation  
 encrypted\_bytes = bytes([data\_byte ^ key\_byte for data\_byte, key\_byte in zip(data\_bytes, repeated\_key)])  
  
 # Encode the result as base64 for storage  
 return base64.b64encode(encrypted\_bytes).decode()

که با استفاده از کلید داده های فایل رمزهارا رمزنگاری میکنید و خروجی بصورت زیر است:



برای رمزگشایی هم:

# Decrypt passwords.txt using XOR decryption  
def decrypt\_passwords\_file(key):  
 with open(passwords\_file, "r") as file:  
 encrypted\_data = file.read()  
 decrypted\_data = xor\_decrypt(encrypted\_data, key)  
   
 with open(passwords\_file, "w") as file:  
 file.write(decrypted\_data)

که از تابع زیر استفاده میکند.

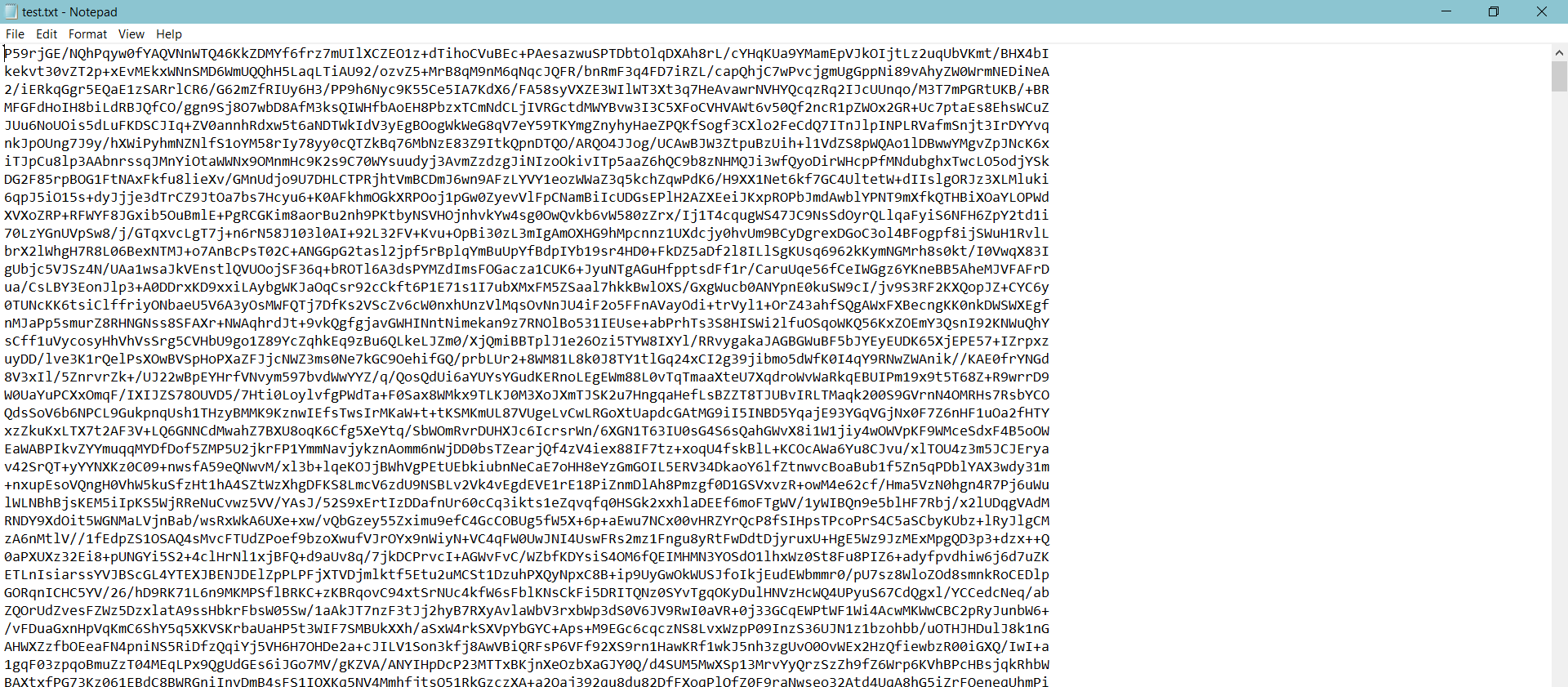
def xor\_decrypt(data, key):  
 encrypted\_bytes = base64.b64decode(data)  
 key\_bytes = key.encode()  
  
 # Repeat the key to match the length of data  
 repeated\_key = key\_bytes \* (len(encrypted\_bytes) // len(key\_bytes)) + key\_bytes[  
 :len(encrypted\_bytes) % len(key\_bytes)]  
  
 # Perform XOR operation  
 decrypted\_bytes = bytes(  
 [encrypted\_byte ^ key\_byte for encrypted\_byte, key\_byte in zip(encrypted\_bytes, repeated\_key)])  
  
 # Decode the result to get the plaintext  
 decrypted\_data = decrypted\_bytes.decode()  
  
 return decrypted\_data

بخش دوم - استفاده از ابزار statsgen برای تحلیل گذرواژه های تولید شده توسط ابزار پیاده سازی شده

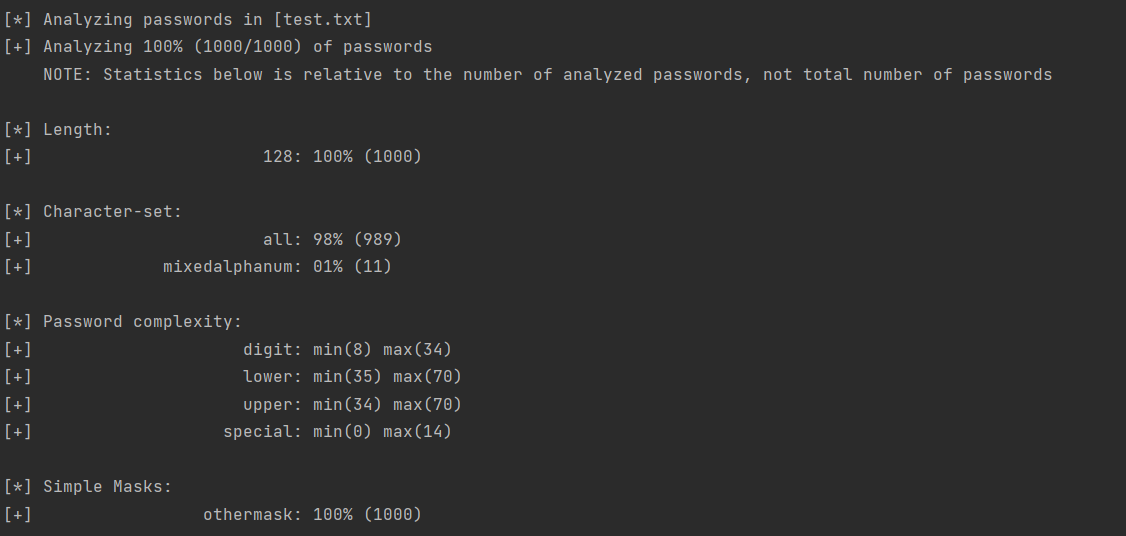
توسط ماژول generator.py رمزهای خواسته شده را تولید و در فایل تست ذخیره میکنیم:

# generator.py  
import string  
import random  
from passmanager import \*  
  
def generate\_passwords():  
 # Set the common parameters  
 name = "generated"  
 comment = "auto-generated password"  
 key = "0000"  
  
 # Create a list to store passwords  
 passwords = []  
  
 # Generate 1000 passwords  
 for i in range(0, 1000):  
 name = f'generated-{i}'  
 comment = f'auto-generated password-{i}'  
 generated\_password = generate\_password(name, comment, key)  
 encrypted\_password = encrypt(generated\_password, key)  
 passwords.append(encrypted\_password)  
 print(f'Password{i} generated!')  
 print('Generation is Done!')  
  
 # Write passwords to the test.txt file  
 with open("test.txt", "w") as file:  
 for password in passwords:  
 print(f'writing password: {password}')  
 file.write(f"{password}\n")  
 print('Done')  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 generate\_passwords()

فایل تست بصورت زیر است:



در گام بعدی خروجی بصورت زیر است:



تحلیل آماری از فایل `test.txt` مطابق با خروجی اسکریپت StatsGen به صورت زیر است:

طول رمزها:

- همه رمزها در فایل `test.txt` دارای طول 128 کاراکتر هستند.

مجموعه کاراکترها:

- 98٪ از رمزها شامل تمامی کاراکترها (عددی، حروف کوچک و بزرگ، ویژه) هستند.

- 1٪ از رمزها شامل ترکیبی از حروف کوچک و عدد می‌باشند.

پیچیدگی رمزها:

- تعداد اعداد در رمزها حداقل 8 و حداکثر 34 می‌باشد.

- تعداد حروف کوچک در رمزها حداقل 35 و حداکثر 70 می‌باشد.

- تعداد حروف بزرگ در رمزها حداقل 34 و حداکثر 70 می‌باشد.

- تعداد کاراکترهای ویژه در رمزها حداقل 0 و حداکثر 14 می‌باشد.

الگوهای ساده:

- 100٪ از رمزها الگوهای پیچیده‌ای دارند که با `othermask` نمایش داده می‌شوند. این الگوها از ترکیب‌های مختلف از اعداد، حروف کوچک و بزرگ، و کاراکترهای ویژه تشکیل شده‌اند.

نکته:

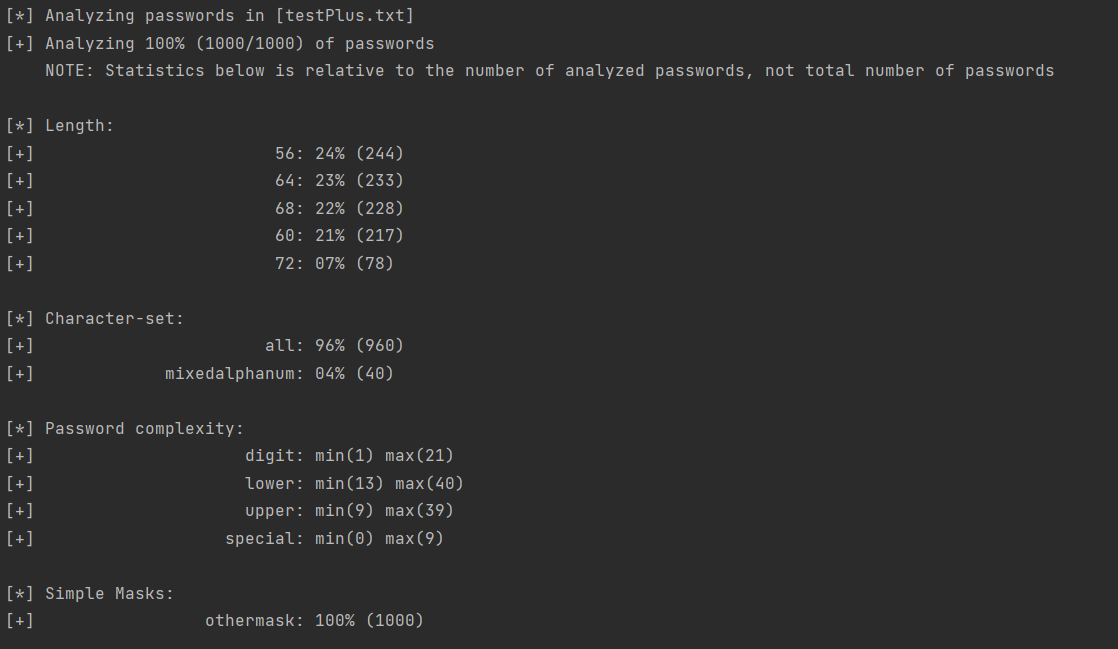
این تحلیل بر اساس رمزهای موجود در فایل `test.txt` انجام شده است. اگر تعداد کل رمزها در فایل بیشتر از 1000 می‌باشد، آمارها نسبت به تعداد آن رمزها محاسبه شده‌اند.

بخش امتیازی – رمز با اندازه متفاوت

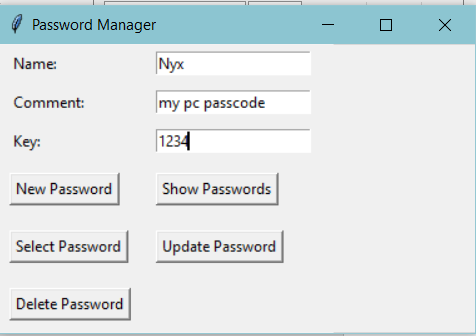
با تغییر کد به شکل زیر در فایل passmanagerPlus.py رمزها با طول متفاوت تولید میشوند:

def generate\_password(name, comment, key):  
 # Generating a random salt  
 salt = ''.join(random.choices(string.ascii\_letters + string.digits, k=16))  
  
 # Generating a random password length between 8 and 20 characters  
 password\_length = random.randint(8, 20)  
  
 # Generating a random password with the specified length  
 password\_input = f"{name}\_{comment}\_{key}\_{salt}"  
 generated\_password = hashlib.sha256(password\_input.encode()).hexdigest()[:password\_length]  
  
 return generated\_password

حالا 1000 رمز را توسط فایل generatorPlus.py تولید میکنیم و به فایل stategen میدهیم:



همانطور که مشاهده میشود رمزها در طولهای مختلف تولید شده اند و پیچیدگی بالاتر رفته است.

با استفاده از کد بخش اول رابط کاربری زیر پیاده سازی شد:

