# 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:

```
PS C:\Users\Samin\Desktop\University\Term 8\Information Security\projects\project3> python passmanager.py --newpass "git" "pass for my git" "1234"
New Password added!
Encrypted Password : 5+VPYbZ7OoAviQndz1PVa3+eIPsG+HaLJBnfPZw9g/XNlXqme1ou6h5YshQ0peH7/HusF3CHDIRQ55TE7m4DYGfAtmK8U8UdkS5PRer1uHefyl44a5xb+Nd+aZZ7OoHf
```

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:

```
PS C:\Users\Samin\Desktop\University\Term 8\Information Security\projects\project3> python passmanager.py --sel "git" "1234"
Related data:
Password: Tz8T2R0vdgNRzXRrWgquYgtVmzGTEU0lOdZZQgt/moqGTZRdn6WuQipSV4TZKDRHufXOH6B0+afbisb56vAm4Ocs+rwbvFeX/kVYO5gWKQaqw9JQ3UERM5pfjiubLZzD, Comment: pass for my git
```

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:

```
PS C:\Users\Samin\Desktop\University\Term 8\Information Security\projects\project3> python passmanager.py --update "git" "1234"
Previous Password: Tz8T2R0vdgNRzXRrWgquYgtVmzGTEU0lOdZZQgt/moqGTZRdn6WuQipSV4TZKDRHufXOH6B0+afbisb56vAm4Ocs+rwbvFeX/kVYO5gWKQaqw9JQ3UERM5pfjiubLZzD
New Password: sz4Ksai+9elfppDnOVAVaFlhZtPfBJzkGjUpBTvtvKMeVFWwlJ3balLPs7Sn+oGO77JYct9y/2t/RRUzhAU2FoSXM2xVGpCtMCUxOYZbEaXRZXGbWqxKoIQ7ggyWEgeu
Password for 'git' updated!
```

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:

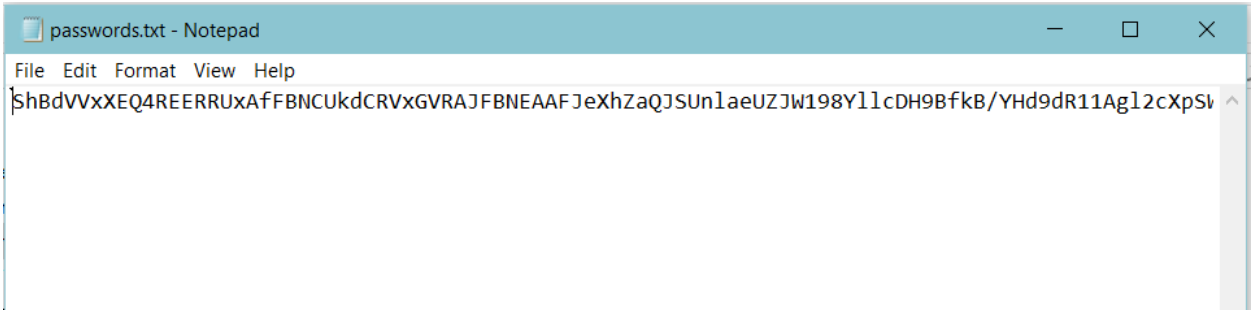
```
PS C:\Users\Samin\Desktop\University\Term 8\Information Security\projects\project3> python passmanager.py --showpass "1234"
Showing Passwords..
Name: git, Password: sz4Ksai+9elfppDnOVAVaFlhZtPfBJzkGjUpBTvtvKMeVFWwlJ3balLPs7Sn+oGO77JYct9y/2t/RRUzhAU2FoSXM2xVGpCtMCUxOYZbEaXRZXGbWqxKoIQ7ggyWEgeu, Comment: pass for my git
Name: web, Password: v35xKKmX0afHhJrxilHSko8NsMtNRDID/F68DBNcjMNnzcYcLJcqtr58T4FSoGa3urFWFtTnL4gxfIfeGerxHkz3BNA6taHQ0Q/vxScydkLTge3Ocmk4IQp2Zs/HzTu7, Comment: pass for my website
PS C:\Users\Samin\Desktop\University\Term 8\Information Security\projects\project3> python passmanager.py --delete "git" "1234"
Deleted Successfully!
PS C:\Users\Samin\Desktop\University\Term 8\Information Security\projects\project3> python passmanager.py --showpass "1234"
Showing Passwords..
Name: web, Password: v35xKKmX0afHhJrxilHSko8NsMtNRDID/F68DBNcjMNnzcYcLJcqtr58T4FSoGa3urFWFtTnL4gxfIfeGerxHkz3BNA6taHQ0Q/vxScydkLTge3Ocmk4IQp2Zs/HzTu7, Comment: pass for my website
```

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)])
```

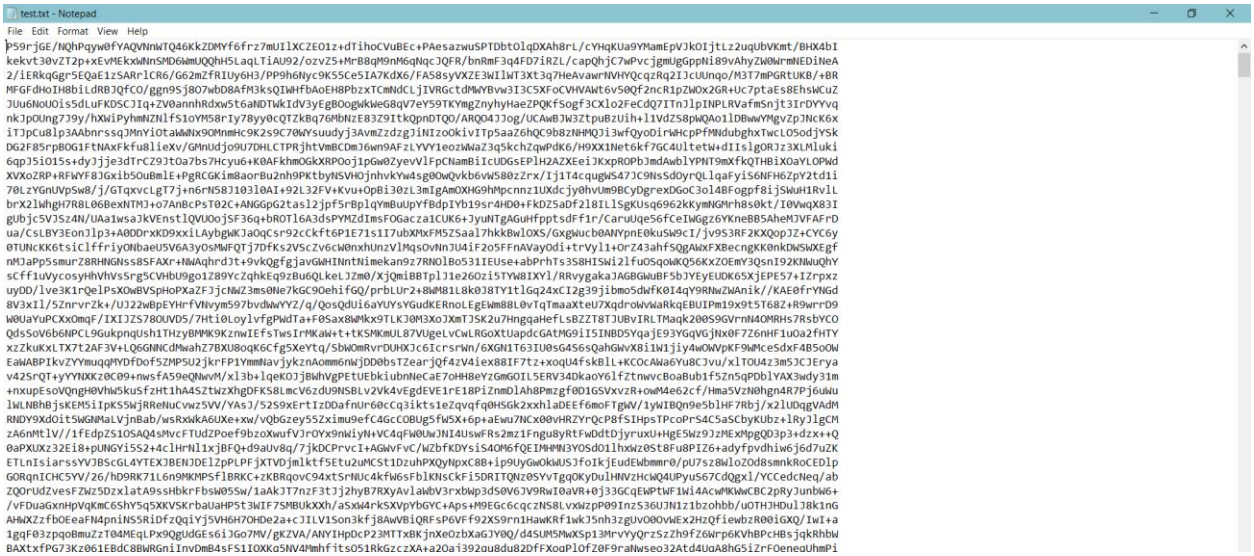
```
    # Decode the result to get the plaintext
    decrypted_data = decrypted_bytes.decode()

    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:

```
[*] Analyzing passwords in [test.txt]
[+] Analyzing 100% (1000/1000) of passwords
    NOTE: Statistics below is relative to the number of analyzed passwords, not total number of passwords

[*] Length:
[+]                       128: 100% (1000)

[*] Character-set:
[+]                       all: 98% (989)
[+]             mixedalphanum: 01% (11)

[*] Password complexity:
[+]                     digit: min(8) max(34)
[+]                     lower: min(35) max(70)
[+]                     upper: min(34) max(70)
[+]                   special: min(0) max(14)

[*] Simple Masks:
[+]                 othermask: 100% (1000)
```

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 ۱۲۸ 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 ۸ و حداکثر ۳۴ می‌باشد.
- تعداد حروف کوچک در رمزها حداقل ۳۵ و حداکثر ۷۰ می‌باشد.
- تعداد حروف بزرگ در رمزها حداقل ۳۴ و حداکثر ۷۰ می‌باشد.
- تعداد کاراکترهای ویژه در رمزها حداقل ۰ و حداکثر ۱۴ 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 ۱۰۰۰ 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 ۱۰۰۰ رمز را توسط فایل generatorPlus.py تولید میکنیم و به فایل stategen میدهیم:

```
[*] Analyzing passwords in [testPlus.txt]
[+] Analyzing 100% (1000/1000) of passwords
    NOTE: Statistics below is relative to the number of analyzed passwords, not total number of passwords

[*] Length:
[+]                      56: 24% (244)
[+]                      64: 23% (233)
[+]                      68: 22% (228)
[+]                      60: 21% (217)
[+]                      72: 07% (78)

[*] Character-set:
[+]                     all: 96% (960)
[+]           mixedalphanum: 04% (40)

[*] Password complexity:
[+]                   digit: min(1) max(21)
[+]                   lower: min(13) max(40)
[+]                   upper: min(9) max(39)
[+]                 special: min(0) max(9)

[*] Simple Masks:
[+]               othermask: 100% (1000)
```

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

## بخش امتیازی – طراحی رابط کاربری

با استفاده از کد بخش قبلی (بخصوص برای این بخش passmanagerGui.py) طراحی شد. نمونه: