Project

Nama : Fawwaz Zulfaa

NIM : 2301955844

Github : https://github.com/fawwazzulfaa/project

Project : enkripsi dan dekripsi message pada server dan client

Penjelasan project :

Pada project yang saya buat yakni membuat lapisan keamanan pada message client dan server menggunakan enkripsi dan dekripsi.

- Metode enkripsi dekripsi yang digunakan yakni AES CFB 256
  AES-CFB adalah mode operasi block cipher yang digunakan untuk enkripsi. Untuk AES-256-CFB menggunakan iv yang diperlukan untuk enkripsi. Jika tidak ada iv yang diberikan, kemungkinan besar itu hanya nol (yang berarti 16 0x00 byte, karena iv sama dengan ukuran blok, yaitu 128bit).

- Dalam program enkripsi dan dekripsi pada key nya saya menggunakan generate key yang menggunakan *library import random.* Jadi pada proses generate key diambil dari string huruf + string angka yang di randomize - 16 karakter pada generate keynya.

code :

demo.py

```python
# fawwaz zulfaa - 2301955844
import sys
from getopt import getopt
import socket
import base64
import hashlib
from Crypto import Random
from Crypto.Cipher import AES
import random, string

# generate key dari string huruf + string angka yang di randomize - 16 karakter
def generate_key():
    key = str()

    for _ in range(16):
        key += random.choice(string.ascii_lowercase + string.digits)
    return key

```

```python
ip =""
port = 0
is_server= False
```

demo.py > ...

```python
    return key


# menggunakan aes 256 cfb
def encrypt(password, message):
    # hashing key pakai sha256 (hashing)
    private_key = hashlib.sha256(password.encode()).digest()
    # dapetin iv (initialization vector) value IV yang random dari block size aes.
    iv = Random.new().read(AES.block_size)
    # buat aes enkripsi baru dengan privite key yang di input
    # dengan mode aes CFB, iv yang baru saja di randomize, ...
    cipher = AES.new(private_key, AES.MODE_CFB, iv, segment_size=128)
    # kemudian inisialissai digunkaan untuk meng encripyt message
    enc = cipher.encrypt(message.encode())
    # lalu IV + hasil dari enkripsi di encode menggunakan base64
    return base64.b64encode(iv + enc).decode()


def decrypt(password, message):
    # hashing dulu pakai sha256
    private_key = hashlib.sha256(password.encode()).digest()
    # disini message di decode menggunakan base 64
    message = base64.b64decode(message)
    # disini mendapatkan IV dan message enkripsi
    iv, enc = message[:16], message[16:]
    # kita inisialissai kembali aes enkripsi dengan password mode iv segmen yang sama
    cipher = AES.new(private_key, AES.MODE_CFB, iv, segment_size=128)
    # mengemnalikan hasil decripsi nya
    return (cipher.decrypt(enc)).decode()
```

```python
def run_server():
    print("server is trying to listening")
    #bikin socket
    #parameternya : tipe address(Ipv4) , protokol(Tcp)
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((ip,port))
    #listen
    server.listen()
    print("server is listening")
    connection, client_address =server.accept()
    print(f"connected with client at{client_address}")
    while True:
    #accept
        message = input("input message['exit' to close connection]: ")
        #kalau exit -> dead
        if message == "exit":
            server.close()
            break
        # intial pass dengan generate key yang sudah di randomize
        password = generate_key()
        # pada message_enc = berisi key yang sudah di randomize + pesan dan key password yang di encrypt
        message_enc = password + ':' + encrypt(password, message)
        print(message_enc)
        connection.send(message_enc.encode())
        #terima response/result dari targetnya
        password, result = connection.recv(1024).decode().split(':')
        result_dec = decrypt(password, result)
        print(result_dec)
```

```python
def run_client():
    print("client is trying to listening")
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    client.connect((ip,port))
    while True:
        # result = client.recv(1024).decode()
        # print(result)
        password, result = client.recv(1024).decode().split(':')
        result_dec = decrypt(password, result)
        print(result_dec)
        message = input("input message['exit' to close connection]: ")
        if message == "exit":
            client.close()
            break
        # initial pass dengan func generate_key yang sudah di randomize
        password = generate_key()
        # pada message_enc = berisi key yang sudah di randomize + pesan dan key password yang di encrypt
        message_enc = password + ':' + encrypt(password, message)
        print(message_enc)
        client.send(message_enc.encode())
```

```python
def main():
    global ip , port, is_server

    print(sys.argv[0:])
    print(sys.argv[1:])
    opts, _ = getopt(sys.argv[1:], "i:p:s",["ip=","port=", "server"] )
    print(opts)
    for opt, value in opts:
        print(f"{opt}:{value}")
        if opt == "-i" or opt == "--ip":
            ip=value
        elif opt == "-p" or opt =="--port":
            port = int(value)
        elif opt == "-s" or opt =="--server":
            is_server = True
    if ip == "":
        print("Ip must be filled !")
        exit()
    if port >2500 :
        print("Port must be between 2000 and 2500!")
        exit()
    if port <2000 :
        print("Port must be between 2000 and 2500!")
        exit()
    if is_server :
        run_server()
    else:
        run_client()
main()
```

demo run code :

```
(.venv) PS C:\Users\Faw\Desktop\pntest\faw\exercise1> python demo.py -i
 127.0.0.1 -p 2000 -s
['demo.py', '-i', '127.0.0.1', '-p', '2000', '-s']
['-i', '127.0.0.1', '-p', '2000', '-s']
[('-i', '127.0.0.1'), ('-p', '2000'), ('-s', '')]
-i:127.0.0.1
-p:2000
-s:
server is trying to listening
server is listening
```

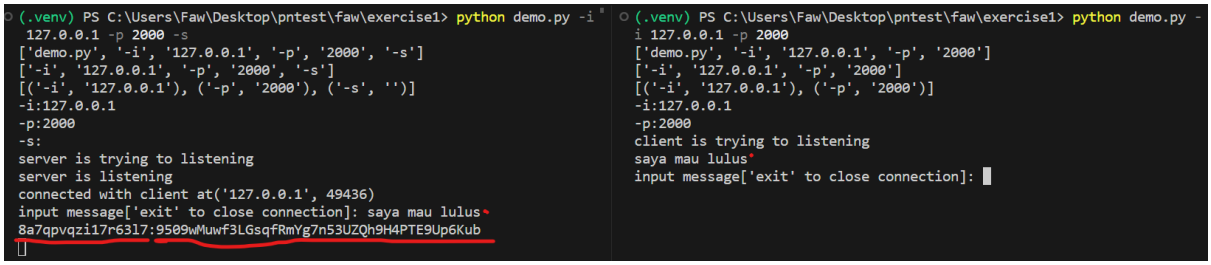
Jalankan sebagai server dengan command “python demo.py -i 127.0.0.1 -p 2000 -s”

```
(.venv) PS C:\Users\Faw\Desktop\pntest\faw\exercise1> python demo.py -
i 127.0.0.1 -p 2000
['demo.py', '-i', '127.0.0.1', '-p', '2000']
['-i', '127.0.0.1', '-p', '2000']
[('-i', '127.0.0.1'), ('-p', '2000')]
-i:127.0.0.1
-p:2000
client is trying to listening
```

Lalu jalankan sebagai client “python demo.py -i 127.0.0.1 -p 2000”

```
(.venv) PS C:\Users\Faw\Desktop\pntest\faw\exercise1> python demo.py -i
 127.0.0.1 -p 2000 -s
['demo.py', '-i', '127.0.0.1', '-p', '2000', '-s']
['-i', '127.0.0.1', '-p', '2000', '-s']
[('-i', '127.0.0.1'), ('-p', '2000'), ('-s', '')]
-i:127.0.0.1
-p:2000
-s:
server is trying to listening
server is listening
connected with client at('127.0.0.1', 49436)
input message['exit' to close connection]:
```

Setelah terkoneksi server dengan client maka tampilan nya seperti diatas, disini kita sudah bisa mengirim pesan pada client dan server.



Kemudian saya melakukan aktivitas mengirim pesan dari server ke client.

Pada gambar diatas saya mengirim pesan “saya mau lulus” sebagai server lalu pesan tersebut di enkripsi dan key (dari string huruf + string angka yang di randomize - 16 karakter) yang digenerate di enkrispsi juga . Lalu pesan di dekripsi oleh client untuk dapat melihat isi pesan tersebut, aktifitas mengirim pesan enkripsi dan dekripsi dengan server dan client atau sebaliknya berhasil dilakukan.

- Pada gambar diatas yang **digaris bawahi merah** berisi key yang sudah di randomize : + pesan dan key password yang di encrypt.

Manfaat dari penggunaan enkripsi dan dekripsi pada pesan

- Menjaga Kerahasiaan Data
  Dengan menerapkan enkripsi dan dekripsi, dapat memastikan bahwa pesan tidak akan dapat diakses oleh pihak yang tidak berwenang sehingga memberikan kemanan
- Menghindari Penyadapan
  Dengan enkripsi dan dekripsi, pesan yang terbaca hanya berupa teks acak yang tidak memiliki arti bahkan jika informasi tersebut disadap oleh orang yang tidak bertanggung jawab.
- Keamanan dalam Transfer Data
  enkripsi dan dekripsi dapat menjaga keamanan selama proses transfer atau pengiriman pesan. Hal ini mencegah data dari potensi peretasan atau pemalsuan yang bisa terjadi dalam serangan Man-in-the-Middle.

Kekurangan dari AES-CFB

- Data yang corrupt tidak dapat dipulihkan karena setiap ciphertext bergantung pada ciphertext lainnya untuk didekripsi.
- Sequential Processing: AES-CFB memproses data secara berurutan, artinya setiap blok bergantung pada enkripsi blok sebelumnya. Hal ini dapat membatasi kemampuan pemrosesan paralel, sehingga kurang cocok untuk skenario komputasi performa tinggi tertentu.
- Overhead performance: Sifat berurutan dari AES-CFB dapat menimbulkan overhead performence dibandingkan dengan mode lain yang memungkinkan pemrosesan paralel, terutama dalam implementasi hardware
- Sensitivitas terhadap Bit Flipping: Mode CFB sensitif terhadap serangan bit-flipping. Jika penyerang dapat memanipulasi ciphertext, mereka mungkin dapat mengontrol atau memodifikasi output yang didekripsi.