Lab Assignment 3

AP21110010302

Krish Srivastava

CSE – E
Network Security – CSE 315L

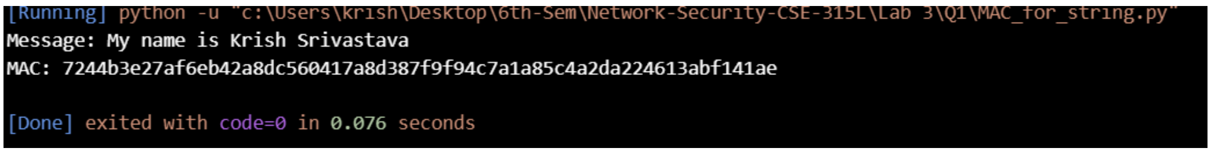
Q1. Write a program to obtain Message Authentication Code (MAC) for a given string [Use HMAC].

```python
import hmac
import hashlib

def generate_mac(secret_key, message):
    secret_key_bytes = bytes(secret_key, 'utf-8')
    message_bytes = bytes(message, 'utf-8')
    hmac_object = hmac.new(secret_key_bytes, message_bytes, hashlib.sha256)
    return hmac_object.hexdigest()

secret_key = "eenameenadeeka"
message = "My name is Krish Srivastava"
mac = generate_mac(secret_key, message)

print(f"Message: {message}")
print(f"MAC: {mac}")
```



```
[Running] python -u "c:\Users\krish\Desktop\6th-Sem\Network-Security-CSE-315L\Lab 3\Q1\MAC_for_string.py"
Message: My name is Krish Srivastava
MAC: 7244b3e27af6eb42a8dc560417a8d387f9f94c7a1a85c4a2da224613abf141ae

[Done] exited with code=0 in 0.076 seconds
```

Q2. Write a program to ensure integrity and confidentiality in client server communication using symmetric key based mechanism [Use Internal Error Control using HMAC].

Server.py

```python
import socket
import hmac
import hashlib

SECRET_KEY = "eenameenadeeka"

def generate_mac(secret_key, message):
    secret_key_bytes = bytes(secret_key, 'utf-8')
    message_bytes = bytes(message, 'utf-8')
    hmac_object = hmac.new(secret_key_bytes, message_bytes, hashlib.sha256)
    return hmac_object.hexdigest()

def verify_mac(secret_key, message, received_mac):
    calculated_mac = generate_mac(secret_key, message)
    return hmac.compare_digest(calculated_mac, received_mac)

def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 12345))
    server_socket.listen(1)

    print("Server listening...")

    while True:
        conn, addr = server_socket.accept()
        print('Connected to', addr)

        while True:
            data = conn.recv(1024)
            if not data:
                break

            data = data.decode()
            message, received_mac = data.split("|||")

            if verify_mac(SECRET_KEY, message, received_mac):
                print("Message received from client:", message)
                conn.sendall(message.encode())
            else:
                print("Integrity check failed. Message may have been 
tampered.")
        conn.close()

if __name__ == "__main__":
    main()
```

Client.py

```
import socket
import hmac
import hashlib

SECRET_KEY = "eenameenadeeka"

def generate_mac(secret_key, message):
    secret_key_bytes = bytes(secret_key, 'utf-8')
    message_bytes = bytes(message, 'utf-8')
    hmac_object = hmac.new(secret_key_bytes, message_bytes, hashlib.sha256)
    return hmac_object.hexdigest()

def main():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 12345))

    while True:
        message = input("Enter message: ")
        mac = generate_mac(SECRET_KEY, message)
        message_with_mac = f"{message}|||{mac}"
        client_socket.sendall(message_with_mac.encode())
        data = client_socket.recv(1024)
        print("Response from server:", data.decode())
    client_socket.close()

if __name__ == "__main__":
    main()
```

Q3. Modify the program 2 to implement External Error Control.

Server.py

```
import socket
import hmac
import hashlib
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

SECRET_KEY = b'eenameenadeeka'

def generate_mac(secret_key, message):
    secret_key_bytes = bytes(secret_key, 'utf-8')
    message_bytes = bytes(message, 'utf-8')
    hmac_object = hmac.new(secret_key_bytes, message_bytes, hashlib.sha256)
    return hmac_object.hexdigest()

def verify_mac(secret_key, message, received_mac):
    calculated_mac = generate_mac(secret_key, message)
    return hmac.compare_digest(calculated_mac, received_mac)

def decrypt_message(secret_key, encrypted_message):
    cipher = AES.new(secret_key, AES.MODE_CBC, iv=encrypted_message[:16])
    decrypted_message = unpad(cipher.decrypt(encrypted_message[16:]),
AES.block_size)
    return decrypted_message.decode()

def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 12345))
    server_socket.listen(1)

    print("Server listening...")

    while True:
        conn, addr = server_socket.accept()
        print('Connected to', addr)

        while True:
            # Receive data from client
            data = conn.recv(1024)
            if not data:
                break

            encrypted_message = data[:-64]
            received_hmac = data[-64:]
            if verify_mac(SECRET_KEY, encrypted_message, received_hmac):
                decrypted_message = decrypt_message(SECRET_KEY,
encrypted_message)
                print("Message received from client:", decrypted_message)

                conn.sendall(encrypted_message)
            else:
                print("Message may have been tampered.")

        conn.close()

if __name__ == "__main__":
    main()
```

Client.py

```
import socket
import hmac
import hashlib
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

SECRET_KEY = b'eenameenadeeka'

def generate_mac(secret_key, message):
    secret_key_bytes = bytes(secret_key, 'utf-8')
    message_bytes = bytes(message, 'utf-8')
    hmac_object = hmac.new(secret_key_bytes, message_bytes, hashlib.sha256)
    return hmac_object.hexdigest()

def encrypt_message(secret_key, message):
    cipher = AES.new(secret_key, AES.MODE_CBC)
    padded_message = pad(message.encode(), AES.block_size)
    encrypted_message = cipher.iv + cipher.encrypt(padded_message)
    return encrypted_message

def main():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 12345))

    while True:
        message = input("Enter message: ")
        encrypted_message = encrypt_message(SECRET_KEY, message)
        mac = generate_mac(SECRET_KEY, encrypted_message)
        message_with_mac = encrypted_message + mac.encode()
        client_socket.sendall(message_with_mac)
        data = client_socket.recv(1024)
        print("Response from server:", data.decode())
```

```python
    client_socket.close()

if __name__ == "__main__":
    main()
```