**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].**



**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**

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())

    client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

    main()