**Secure Hash Algorithm 512-bit**.

**CSE 459: Cryptography & Network Security**

Submitted by

Name: D. Tejaswi

Roll No: AP22110011503

Section: CSE Y

Lab Date: 17/03/2025

Submission Date: 23/03/2025

**A picture containing text

Description automatically generated**

**Department Computer Science and Engineering**

**School of Engineering and Sciences**

**SRM University–AP**

**Amaravati, Andhra Pradesh – 522 240, India**

1. **Question**
2. **Write a program to generate Hash code for a given text using SHA 512**
3. **Write a program to send a message ensuring confidentiality and integrity  
   Hint for 2:  
   i) perform Diffie Hellman exchange (from previous lab) to generate a session key  
   ii) Compute Hash code of the message text  
   iii) use AES to encrypt the (text + hash code)  
   iv) encrypt the (message + Hash Code)  
   v) send it to client/server  
   vi) decrypt the message  
   vii) check the integrity.**
4. **Algorithm Description**
5. **Solution**
6. **Write a program to generate Hash code for a given text using SHA 512**

**Code:**

1. import hashlib
2. def generate\_sha512\_hash(text):
3. sha512\_hash = hashlib.sha512(text.encode()).hexdigest()
4. return sha512\_hash
5. text = input("Enter the text: ")
6. print("SHA-512 Hash:", generate\_sha512\_hash(text))

**output:**

**Enter the text: Hello**

**SHA-512 Hash: 3615f80c9d293ed7402687f94b22d58e529b8cc7916f8fac7fddf7fbd5af4cf777d3d795a7a00a16bf7e7f3fb9561ee9baae480da9fe7a18769e71886b03f315**

1. **Write a program to send a message ensuring confidentiality and integrity Hint for 2: i) perform diffie hellman exchange (from previous lab) to generate a session key  
   ii) Compute Hash code of the message text  
   iii) use AES to encrypt the (text + hash code)  
   iv) encrypt the (message + Hash Code)  
   v) send it to client/server  
   vi) decrypt the message  
   vii) check the integrity.**

**Code:**

import hashlib

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.primitives import padding

from cryptography.hazmat.backends import default\_backend

private\_A = ec.generate\_private\_key(ec.SECP256R1())

private\_B = ec.generate\_private\_key(ec.SECP256R1())

public\_A = private\_A.public\_key()

public\_B = private\_B.public\_key()

shared\_secret\_A = private\_A.exchange(ec.ECDH(), public\_B)

shared\_secret\_B = private\_B.exchange(ec.ECDH(), public\_A)

assert shared\_secret\_A == shared\_secret\_B, "Key exchange failed!"

aes\_key = shared\_secret\_A[:16]

def compute\_hash(msg):

    return hashlib.sha512(msg.encode()).hexdigest()

def encrypt\_message(msg, key):

    msg\_hash = compute\_hash(msg)

    combined\_data = (msg + msg\_hash).encode()

    padder = padding.PKCS7(algorithms.AES.block\_size).padder()

    padded\_data = padder.update(combined\_data) + padder.finalize()

    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default\_backend())

    encryptor = cipher.encryptor()

    ciphertext = encryptor.update(padded\_data) + encryptor.finalize()

    return ciphertext

def decrypt\_message(ciphertext, key):

    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default\_backend())

    decryptor = cipher.decryptor()

    decrypted\_data = decryptor.update(ciphertext) + decryptor.finalize()

    unpadder = padding.PKCS7(algorithms.AES.block\_size).unpadder()

    unpadded\_data = unpadder.update(decrypted\_data) + unpadder.finalize()

    message = unpadded\_data[:-128].decode().strip()

    received\_hash = unpadded\_data[-128:].decode().strip()

    return message, "Integrity Verified!" if compute\_hash(message) == received\_hash else "Integrity Check Failed!"

message = "Cryptography!"

ciphertext = encrypt\_message(message, aes\_key)

print("\nCiphertext (Confidentiality Ensured):", ciphertext.hex())

decrypted\_msg, status = decrypt\_message(ciphertext, aes\_key)

print("\nDecrypted Message:", decrypted\_msg)

print("Integrity Status:", status)

**Output:**

**Ciphertext (Confidentiality Ensured): 44e0e113a164febb2dce785328469bc16f6f4899411deb81a8ef4f716c5d5e44139e950741cbd2a8e9d331c6adfeea24260e3bf0e1ebbaf717216bc3522468a1716aa72e27abc31e9cbafa972d516f89af27e29f4d733a848f96e6bb47dd5b00ceb0c76728fa2bcc8030ac8dda0634e346961a1579a26cd797a09c441609bc32a6b8cc9e70349615d1fa50128f69b1c0**

**Decrypted Message: Cryptography!**

**Integrity Status: Integrity Verified!**

1. **Code Repository:**