**Practical 6**

**Aim:** Refer to the figure (a) attached here. Bob (Source A) is preparing to send message to Alice (Destination B). Bob applies SHA256 hash algorithm on prepared message and append with original message (M) which is further encrypted by single secret key. Alice will receive bundle of encrypted H(M) and original message (M). Alice will first apply single secret key to decrypt the entire bundle and collect H(M) and original message (M). Furthermore, Alice will apply the same algorithm SHA256 which was used by Bob and produce hash of received message (H). Lastly, Alice will verify the computed hash with received H(M) to make sure message is not altered by any attackers.

Task to perform:

1. Use any algorithm to implement encryption function only. Give proper justification of choosing the algorithm.
2. Implementation can be done using any programming language such as c, c++, java, python, c#, javascript, etc.
3. For SHA256 hashing, you may use library compatible as per your programming language. [This task is optional for implement]

**Code:**

import hashlib

from Crypto import Random from Crypto.Cipher import AES

from base64 import b64encode, b64decode SECRETKEY = 'mysecretpassword'

class AESCipher:

def \_\_init (self, key):

self.blockSize = AES.block\_size

self.key = hashlib.sha256(key.encode()).digest()

def addPadding(self, plainText):

bytesToPad = self.blockSize - len(plainText) % sel f.blockSize

padding = bytesToPad \* chr(bytesToPad) paddedPlainText = plainText + padding return paddedPlainText

def removePadding(self, plainText):

lastCharacter = plainText[len(plainText) - 1:]

bytesToRemove = ord(lastCharacter)

return plainText[:-bytesToRemove]

def encrypt(self, plainText):

plainText = self.addPadding(plainText)

iv = Random.new().read(self.blockSize)

cipher = AES.new(self.key, AES.MODE\_CBC, iv)

encryptedText = cipher.encrypt(plainText.encode())

return b64encode(iv + encryptedText).decode("utf-8")

def decrypt(self, encryptedText):

encryptedText = b64decode(encryptedText)

iv = encryptedText[:self.blockSize]

cipher = AES.new(self.key, AES.MODE\_CBC, iv)

plainText = cipher.decrypt(encryptedText[self.blockSize:]).decode("utf-8")

return self.removePadding(plainText)

if \_\_name\_\_ == "\_\_mai\_\_":

# Source Side

message = 'Practical Task-1'

print('Message:', message, sep='\n')

# Calculate Hash of message

hash = hashlib.sha256(message.encode()).hexdigest()

print('Message Hash:', hash, sep='\n')

# Append Hash to the message

message = message + str(hash)

# Encrypt new message using AES cipher

cipher = AESCipher(SECRETKEY)

cipherText = cipher.encrypt(message)

print('CipherText:', cipherText, sep='\n')

# Send cipherText and Key to destination

# Destination Side

# Decrypt the message using AES cipher

cipher = AESCipher(SECRETKEY) decryptedText = cipher.decrypt(cipherText)

print('DecryptedText:', decryptedText, sep='\n')

# Extract message and hash of message from decryptedText

extMessage, extHash = decryptedText[:-64], decryptedText[-64:]

print('Extracted Message:', extMessage, sep='\n')

print('Extracted Hash:', extHash, sep='\n')

# Calcluate hash of extracted message

destHash = hashlib.sha256(extMessage.encode()).hexdigest()

# if extracted hash and calculated hash matches i.e. a re same then the message is not tempered

if(destHash == extHash):

print('Message is not tempered')

else:

print('Message is tempered.')

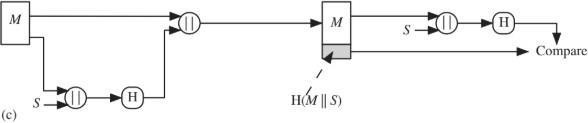
**Output:**



**Conclusion:** By performing this practical we learned about AES and SHA256 algorithms, also studied how it works with a simple scenario.

**Practical 7**

**Aim:** Refer to the attached figure here. Bob is preparing to send message to Alice. Bob and Alice both secretly computes the code(s) without sharing on any communication channel. Suggest key exchange algorithm to Bob and Alice for securely exchange information without sharing actual key. Once they form secret code, Bob applies SHA256 hash algorithm on original message (M) plus code (s) and send hash of original message and code (M||s) to Alice. Alice will receive bundle of H(M||s) and first append code (s) with received message (M) and produce hash of the message (H) that compare with H(M||s) to make sure that message is not altered by any attackers.



Task to perform:

* 1. Use some key exchange algorithm to calculate value of s (secret code) which must be unique at sender and receiver side.
  2. Implementation can be done using any programming language such as c, c++, java, python, c#, javascript, php etc.
  3. Apply SHA256 on message and secret code and display it on output screen. Verify the hash value at receiver end.

**Code:**

import hashlib

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

# public keys G and P

P = int(input('P : '))

G = int(input('G : '))

# get the generated key for sender

a = int(input('a : '))

x = int(pow(G, a, P))

# get the generated key for reciever

b = int(input('b : '))

y = int(pow(G, b, P))

# Secret key for sender

ka = int(pow(y, a, P))

# Secret key for reciever

kb = int(pow(x, b, P))

# Source Side

message = 'Practical-7'

print('Message:', message, sep='\n')

# Calculate Hash of message + private key

hash = hashlib.sha256((message+str(ka)).encode()).hexd igest()

print('Message + Private Key Hash:', hash, sep='\n')

# Append Hash to the message cipherText = message + str(hash)

print('Message + Hash:', cipherText, sep='\n') # Destination Side

# Extract message and hash of message from decryptedText

extMessage, extHash = cipherText[:-64], cipherText[-64:]

print('Extracted Message:', extMessage, sep='\n')

print('Extracted Hash:', extHash, sep='\n')

# Calcluate hash of extracted message

destHash = hashlib.sha256((extMessage+str(ka)).encode()).hexdigest()

# if extracted hash and calculated hash matches i.e. a re same then the message is not tempered

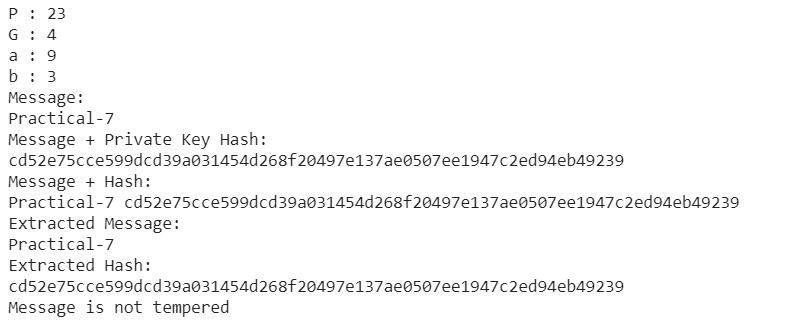
if(destHash == extHash):

print('Message is not tempered')

else:

print('Message is tempered.')

**Output:**



**Conclusion:** By performing this practical we learned about how secret keys can be exchanged between two communicating parties using Diffie-Hellman key exchange algorithm. We also implemented and tested working of Diffie- Hellman key exchange algorithm.