**CS6008 Cryptography and Network Security.**

**Assignment No.9**

**Name:** G Vignesh **Reg No:**2018103078

**Date:**12.6.2022

**MODULE VIII: HASHES**

**External Learning-** Computing MACs, Hashes and HMACs for messages.

**Aim:**

To compute message authentication code(Mac), Hashes, Hashes + Mac(Hmac) for the mesaages in order to provide authentication, integrity of the messages passed, so that intentional modifications can be detected and avoided.

**Tools used:**

i) Ubuntu terminal

ii) python compiler

iii )hashing packages

**Description:**

**MACs**

A message authentication code (MAC) is a [cryptographic checksum](https://www.techtarget.com/searchsecurity/definition/cryptographic-checksum) on data that uses a [session key](https://www.techtarget.com/searchsecurity/definition/session-key) to detect both accidental and intentional modifications of the data.

A MAC requires two inputs: a message and a [secret key](https://www.techtarget.com/searchsecurity/definition/private-key) known only to the originator of the message and its intended recipient(s). This allows the recipient of the message to verify the integrity of the message and authenticate that the messege's sender has the shared secret key. If a sender doesn’t know the secret key, the hash value would then be different, which would tell the recipient that the message was not from the original sender.

**Sender side:**

import random

keys=[]

**def** Gen(klen):

**global** keys

    for i in range(0,2\*\*klen):

        key=str(bin(i).replace("0b",""))

        key='0'\*(klen-len(key))+key

        keys.append(key)

    k=random.choice(keys)

    return k

**def** Mac(k,m,tlen):

    mk=m+k

    mk=int(mk,2)

    t=mk%(2\*\*tlen)

    t=str(bin(t).replace("0b",""))

    t='0'\*(tlen-len(t))+t

    return t

**def** Sender():

**global** m,k,t,klen,tlen,keys

    keys=[]

    f="-"\*50

    s=" "\*18

    print()

    print(f,"\n")

    print(s,"Sender's side",s)

    m=input("\nEnter the message:\t")

    klen=int(input("Enter length of key:\t"))

    k=Gen(klen)

    tlen=int(input("Enter length of mac-code:\t"))

    t=Mac(k,m,tlen)

    print()

    print("Msg:",m,"\t|\tKey:",k,"\t|\tmac-code:",t)

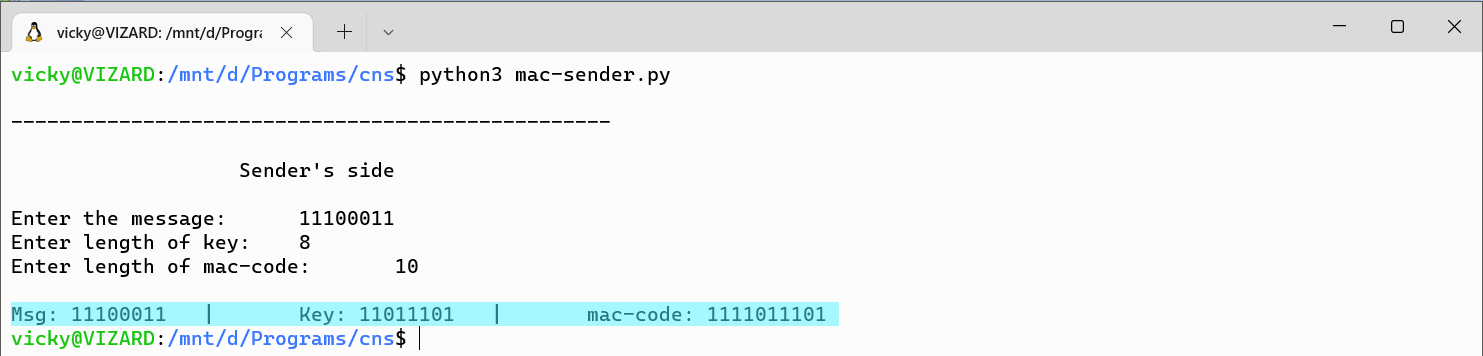
Sender()

In sender side the message to be send , the mac-code generated by the program both are combined and sent to the receiver.

The message: **11100011**

Key size-8 ; key-**11011101**

Mac-code size-10, Mac-code generated: **11111011101**



**Receiver side:**

**def** Mac(k,m,tlen):

    mk=m+k

    mk=int(mk,2)

    t=mk%(2\*\*tlen)

    t=str(bin(t).replace("0b",""))

    t='0'\*(tlen-len(t))+t

    return t

**def** verifier(m,t,keys):

    print("Msg:",m,"\t|\tKey:",keys,"\t|\tmac-code:",t)

    if Mac(keys,m,len(t))==t:

        print("\t\tSuccess")

    else:

        print("\t\tFailed")

**def** receiver():

**global** m,t

    f="-"\*50

    s=" "\*17

    print()

    print(f,"\n")

    print(s,"Receiver's side",s)

    print()

    m=input("\nEnter the message:\t")

    k=input("\nEnter the key:\t")

    t=input("\nEnter the mac-code:\t")

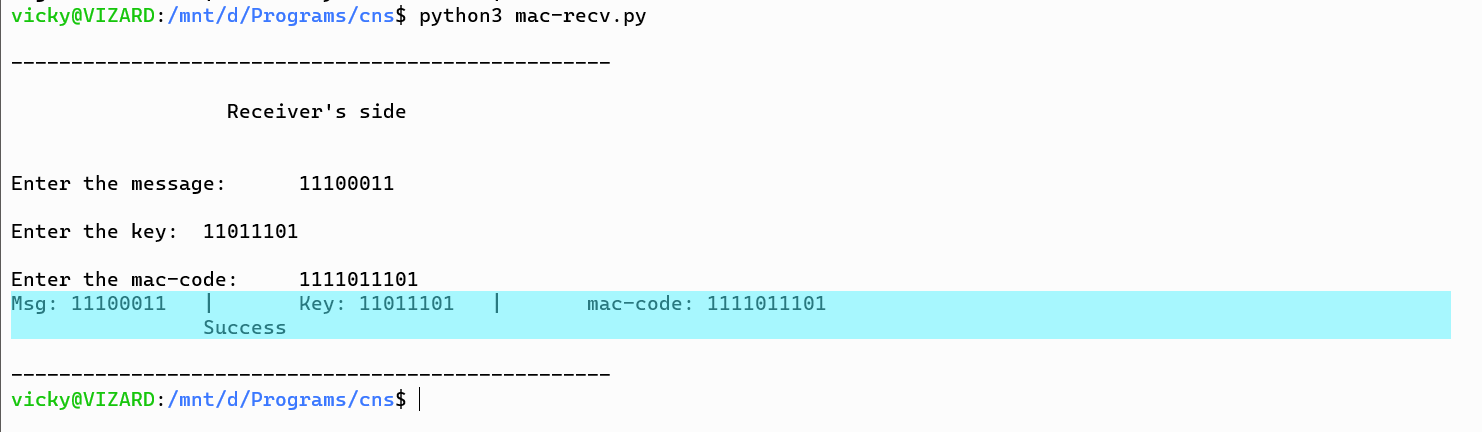
    verifier(m,t,k)

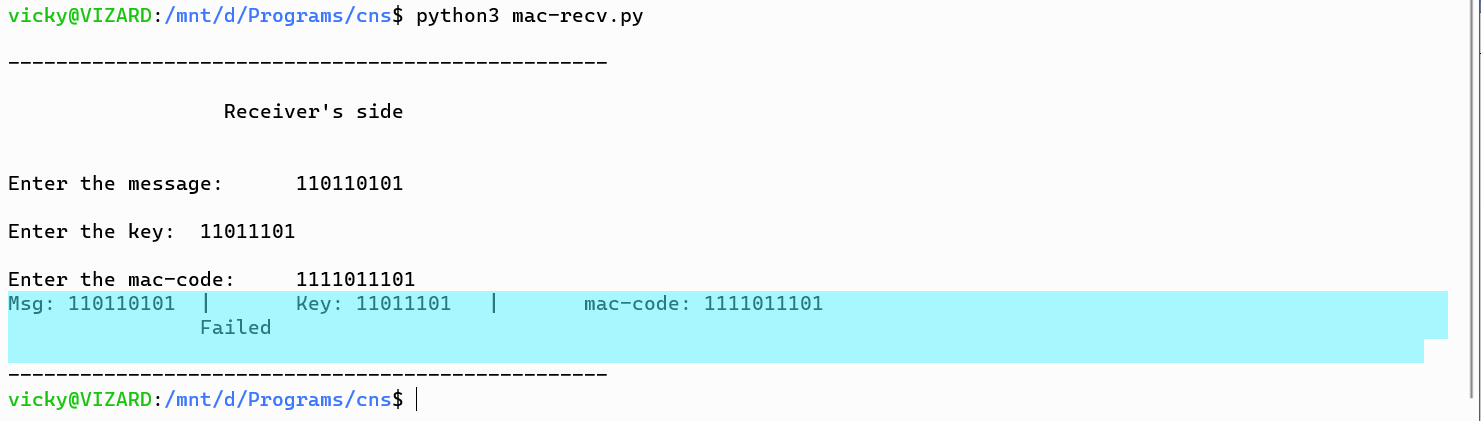
    print()

    print(f)

receiver()

In receiver side the message and mac code is passed if correct key value is entered then we get the **Success message** else if the data is modified then we can get to know by the **False Message .**

****

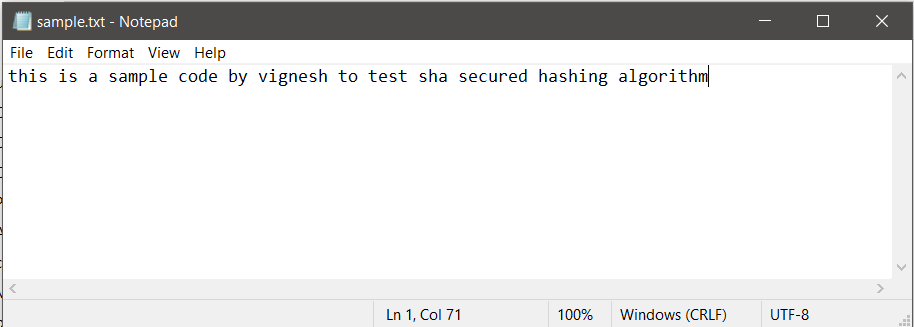
Now we entered the modified data, so we get failed message, ****

Thus Mac code are generated and sent along with the message to check if there are any modification happened in the data or not.

**Hashes**

Hashes are the output of a hashing algorithm like MD5 (Message Digest 5) or SHA (Secure Hash Algorithm). These algorithms essentially aim to produce a unique, fixed-length string – the hash value, or “message digest” – for any given piece of data or “message”. As every file on a computer is, ultimately, just data that can be represented in binary form, a hashing algorithm can take that data and run a complex calculation on it and output a fixed-length string as the result of the calculation. The result is the file’s hash value or message digest.

Sample file passed to sha256 to find hash code.

****

**Program:**

import hashlib

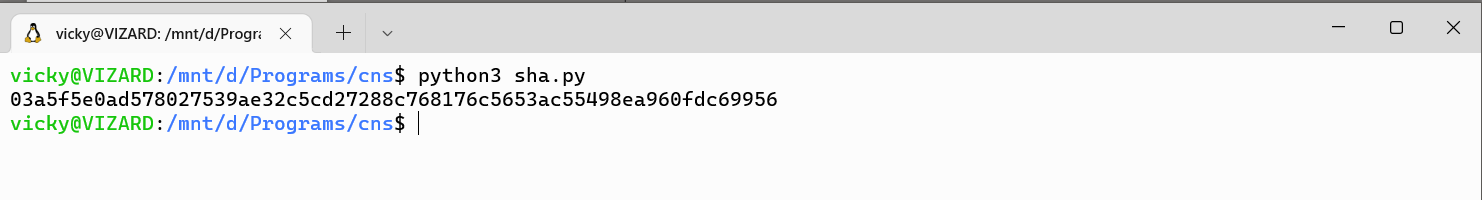
filename = "sample.txt"

with open(filename,"rb") as f:

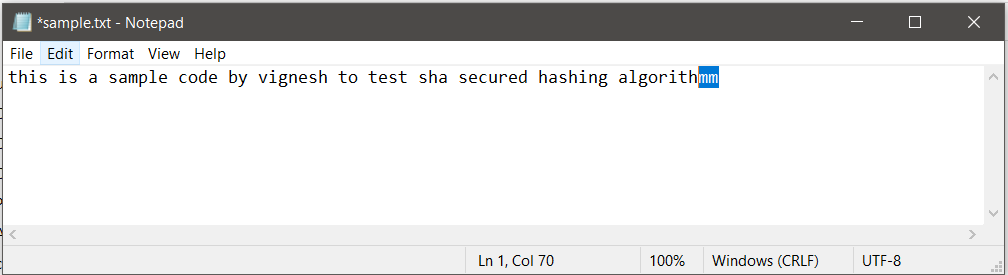
    bytes = f.read()

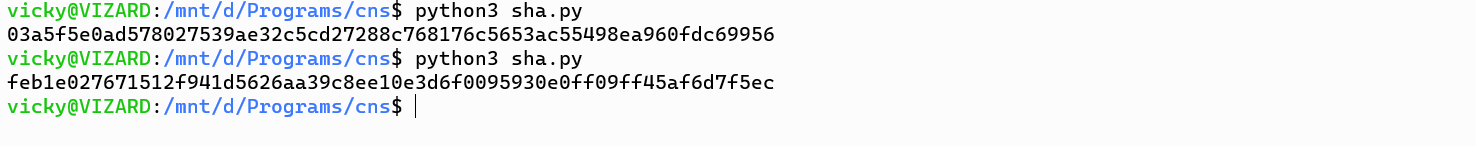
    hash = hashlib.sha256(bytes).hexdigest();

    print(hash)

****

Even if we change one word in the message the hash code changes, that means no two messages can have same hash code which can be used to send along the mesaage , so that if there are any modifications in the text , it can be identified easily.





Since we changed one word the hashcode has changes this can be used to identify modifications in data.

**HMAC**

HMAC stands for **keyed-hash message authentication code**.

It is a type of message authentication code, that includes a cryptographic hash function along with a secret cryptographic key. HMAC, you can achieve authentication and verify that data is correct and authentic with shared secrets, as opposed to approaches that use signatures and asymmetric cryptography.

import hmac

import hashlib

import binascii

**def** hmacs(key, msg):

    key = binascii.unhexlify(key)

    msg = msg.encode()

    return hmac.new(key, msg, hashlib.sha256).hexdigest().upper()

print(hmacs("ABCDEF", "hmac used for authentication and integerity"))

**mesaage:** “hmac used for authentication and integrity”

**password:** ABCDEF



Here the message is encrypted using key(ABCDEF) and then hashed by **SHA256,** so if the message is changed it can be identified by variation in sha hashed code and integrity can be maintained by encryption by the key.

**Conclusion:**

All the hashes and Mac, HMacs are used to maintain authentication of messages send and to maintain integrity in the message any modification by the attackers can be identified, hashes along cannot provide any authentication that’s why hashes are combined with Message authentication code to produce HMACS.