```
pip install pycryptodome

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Collecting pycryptodome
       Downloading pycryptodome-3.17-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 20.8 MB/s eta 0:00:00
     Installing collected packages: pycryptodome
     Successfully installed pycryptodome-3.17
```

```
import operator
import math
import hashlib
import random
import time
import secrets
leaf_count = 32
def prng(seed):
        return int(hashlib.sha3_224(b'\xff\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01' + seed).hexdigest(), 16)

class node:
    def __init__(self, node_id, hashval, left_child, right_child, parent):
        self.hashval = hashval
        self.node_id = node_id
        self.left_child = left_child
        self.right_child = right_child
        self.left_child_hash = []
        self.right_child_hash = []
        self.parent = parent
        self.parent_hash = []



class tree:
    def __init__(self, n, leaves, layers, cntlayers):
        self.n = 32 #leaf count
        self.layers = []
        self.leaves = []
        self.cntlayers = 0
        # create leaf nodes from the input string
        for i in range(0,32):
            node_id = i
            si = [prng(secrets.token_bytes(16))]
            IDi = f"{node_id+900}j{si}"
            hashval = hashlib.sha3_224(f"{IDi}{input_string}".encode()).hexdigest() # create the hash of d with its identifier IDi and si
            self.leaves.append(node(node_id, hashval, None, None, None))
            #self.n+=1
        self.cntlayers = int(math.log(self.n,2)+ 1) #node count
        # update the hashval calculation for each leaf node

    def fill_leaf(self, node):
        self.leaves.append(node)


    def construct_layers(self):
        i = 1
        for i in range(self.cntlayers):
            self.layers.append([])

    def fill_leaves(self):
        self.layers = self.layers + [self.leaves]

    def build_tree(self):
        self.fill_leaves()
        self.construct_layers()
        #build_tree


        len_layer = int(self.n)
        ctr = self.n #Node ID counter
        for i in range(1,self.cntlayers):
            if(len_layer==1):# Root Node
                ctr = ctr +1 #Node ID counter
                self.layers[i].append(node(ctr, 0, 0, 0, 0))
                hashv1 = int(''.join(format(ord(x), 'b') for x in self.layers[i-1][0].hashval))
                hashv2 = int(''.join(format(ord(x), 'b') for x in self.layers[i-1][1].hashval))

                temp = operator.xor(hashv1,hashv2)
                self.layers[i][0].hashval = hashlib.sha3_224(str(temp).encode()).hexdigest()
                self.layers[i][0].left_child = self.layers[i-1][0].node_id
                self.layers[i][0].left_child_hash=hashlib.sha3_224(str(hashv1).encode()).hexdigest()
                self.layers[i][0].right_child = self.layers[i-1][1].node_id
                self.layers[i][0].right_child_hash= hashlib.sha3_224(str(hashv2).encode()).hexdigest()

                self.layers[i-1][0].parent = ctr
                self.layers[i-1][0].parent_hash = hashlib.sha3_224(str(ctr).encode()).hexdigest()
                self.layers[i-1][1].parent = ctr
                self.layers[i-1][1].parent_hash = hashlib.sha3_224(str(ctr).encode()).hexdigest()
            else: # Inner nodes
                k=0
                rng = len(self.layers[i-1]) # Child layer length
                for j in range(len_layer):
                    if k < rng:
                        ctr = ctr +1 #Node ID counter
                        self.layers[i].append(node(ctr, 0, 0, 0, 0))
                        hashv1 = int(''.join(format(ord(x), 'b') for x in self.layers[i-1][k].hashval))   #left child
                        hashv2 = int(''.join(format(ord(x), 'b') for x in self.layers[i-1][k+1].hashval))  #right child
                        temp = operator.xor(hashv1,hashv2)
                        # (SHA256.new(str(temp).encode())).hexdigest()
                        self.layers[i][j].hashval = hashlib.sha3_224(str(temp).encode()).hexdigest()  #parent
                        self.layers[i][j].left_child = self.layers[i-1][k].node_id
                        self.layers[i][j].left_child_hash= hashlib.sha3_224(str(hashv1).encode()).hexdigest()
                        self.layers[i][j].right_child = self.layers[i-1][k+1].node_id
                        self.layers[i][j].right_child_hash= hashlib.sha3_224(str(hashv2).encode()).hexdigest()
                        self.layers[i-1][k].parent = ctr
                        self.layers[i-1][k].parent_hash = hashlib.sha3_224(str(ctr).encode()).hexdigest()
                        self.layers[i-1][k+1].parent = ctr
                        self.layers[i-1][k+1].parent_hash = hashlib.sha3_224(str(ctr).encode()).hexdigest()

                        k = k+2

                len_layer = int(len_layer/2)


    def find_root(self):
        return self.layers[self.cntlayers-1][0].hashval

    def print_tree(self):
        for i in range(self.cntlayers):
            for j in range(len(self.layers[i])):
                print('Node id:',self.layers[i][j].node_id, 'Node Hash value:',self.layers[i][j].hashval,'| Left child  value:', (self.layers[i][j].lef
            print('*****')

    def find_node_from_id(self, node_id):
        for i in range(self.cntlayers):
            for j in range(len(self.layers[i])):
                temp = self.layers[i][j].node_id
                if (temp == node_id):
                    return self.layers[i][j]

    def find_path(self, node_id):
        path = []
        pivot = self.find_node_from_id(node_id)
        for i in range(self.cntlayers):
            parent = 0
            for j in range(len(self.layers[i])):
                if(pivot.parent == self.layers[i][j].node_id):  ############ not sure if good code
                    parent = self.layers[i][j]
                    if parent.left_child == pivot.node_id:
                        path.append(self.find_node_from_id(parent.right_child).hashval)
                    else:
                        path.append(self.find_node_from_id(parent.left_child).hashval)
                    pivot = parent
        return path




def root_from_path(path, node_hash):
    root = node_hash
    for i in range(len(path)):
        hashv1 = int(''.join(format(ord(x), 'b') for x in path[i]))
        hashv2 = int(''.join(format(ord(x), 'b') for x in root))
        temp = operator.xor(hashv1, hashv2)
        root = hashlib.sha3_224(str(temp).encode()).hexdigest()
    return root

input_string = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce vel ex sapien. Integer commodo, mauris non vestibulum euismod, nunc libe
start_time = time.time()
t = tree(input_string,[],[],0)
t.build_tree()
end_time = time.time()
merkle_time = end_time - start_time
root = t.find_root()
node_id=7
node_hash = t.find_node_from_id(node_id).hashval
path = t.find_path(node_id)
found_root = root_from_path(path,node_hash)

print("MERKLE TREE efficiency: {} seconds".format(merkle_time))

print("\n*************************************************************************************")
print("***************************** Merkle tree using SHA "+ str(leaf_count) +" leaves *****************************")
print("*************************************************************************************")
t.print_tree()
print("*************************************************************************************")
print("********************************************************************************")
print("********************************************************************************\n")
print("Root of the Merkle tree ************************************************************")
print(root)
print("********************************************************************************")
print("********************************************************************************\n")
print("Node hash of the node with id "+str(node_id)+ " ***********************************************")
print(node_hash)
print("********************************************************************************")
print("********************************************************************************\n")
print("Path of the node with id "+str(node_id)+ " ****************************************************")
for i in path:
    print(i)
print("********************************************************************************")
print("********************************************************************************\n")
print("Found root with path and hash of the node with id "+str(node_id)+ " ********************************")
print(found_root)
print("********************************************************************************")
print("********************************************************************************\n")



if(found_root == root):
    print("DATA NOT TAMPERED")
else:
    #the user will reject d and search d from some other node
    print("DATA  TAMPERED")
```

```
MERKLE TREE efficiency: 0.004303932189941406 seconds

************************************************************************************
***************************** Merkle tree using SHA 32 leaves ****************************
************************************************************************************
Node id: 0 Node Hash value: 1b1d246b0a2cf4d02a2c58ec5890f876b0a31b0e41256dbaabc98e32 | Left child  value: [] | Right  child  value: [] | Parent
Node id: 1 Node Hash value: e0b3a83b3ba192824a80b16fa32b3e82b97dfbb4c535eb8c55f916bc | Left child  value: [] | Right  child  value: [] | Parent
Node id: 2 Node Hash value: 1dfff2465b836657641d6514f10ad9e7a2d1305438420827a17a2c93 | Left child  value: [] | Right  child  value: [] | Parent
Node id: 3 Node Hash value: ca48dc545728a2243d3372359fda3104a95fc600902dc65469cc1c39 | Left child  value: [] | Right  child  value: [] | Parent
Node id: 4 Node Hash value: 7fc895ae71cef2031c9fea24ce66288613636b493101c4b83eeed1d4 | Left child  value: [] | Right  child  value: [] | Parent
Node id: 5 Node Hash value: 225eaa0c4453b9cd87146c0c75a152911b33888d97016c3908e2be36 | Left child  value: [] | Right  child  value: [] | Parent
Node id: 6 Node Hash value: 198f469fdae03bdab80af93ddcc8c57e7fac181ebb6d87130dca4efc | Left child  value: [] | Right  child  value: [] | Parent
Node id: 7 Node Hash value: 717466a56f80d99e2aa74f5a2d3ccfb2dafdce8622031faae361dbf5 | Left child  value: [] | Right  child  value: [] | Parent
Node id: 8 Node Hash value: 51ce8fce6219e9dea6a838eef6393c31dcf81f3ced588568dcbf2d7a | Left child  value: [] | Right  child  value: [] | Parent
Node id: 9 Node Hash value: 098b5d55046874553fc6172080f25901b705f3f8cf31b8ad58a12081 | Left child  value: [] | Right  child  value: [] | Parent
Node id: 10 Node Hash value: e3054d42f32279a1b98f7c951d4d1daa2297d0e28301e1f36c199118 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 11 Node Hash value: f2a4cecb79d9446a632612f7e87187f326cfaef1ca39912244a6ce3f | Left child  value: [] | Right  child  value: [] | Paren
Node id: 12 Node Hash value: 449b2a6ea2223ea34e29b64be6f90d0571595c235e80e505f8781df1 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 13 Node Hash value: b34bca61a3ed1b95a137edb0f2b01cf8b4c93722b10f8d7742fe00c9 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 14 Node Hash value: f78aacfeac08cfcf972387e70ef615ff8e0011c29f171080bf6816d3 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 15 Node Hash value: 5373445b98be02357d7d30cda0db51500bb7c156a7b4ce11f5914ee3 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 16 Node Hash value: 1c7f4e20b250b5e7628eab9138d433a90843ba7737d7ed8f3aab818c | Left child  value: [] | Right  child  value: [] | Paren
Node id: 17 Node Hash value: 7636f7aab600cf6cc4f5ba8f945ca4e2313e3e6a1d8a728731ed16ba | Left child  value: [] | Right  child  value: [] | Paren
Node id: 18 Node Hash value: ab3253068bdc2300840529f62c6fa74e9a8c4e927ceeb9b31f7b12fc | Left child  value: [] | Right  child  value: [] | Paren
Node id: 19 Node Hash value: d03dea12eac0433006810e8f1f5355e8e8a5cb391037bb06c9a4d56f | Left child  value: [] | Right  child  value: [] | Paren
Node id: 20 Node Hash value: a6793ccde96897a4a444b1e97402fee4a02416139569f55ec7af642d | Left child  value: [] | Right  child  value: [] | Paren
Node id: 21 Node Hash value: 737ca93c2f1d6f8bf47d65982bf399dc0933fbbe10db9db3df7e852f | Left child  value: [] | Right  child  value: [] | Paren
Node id: 22 Node Hash value: e90c701d0164040ac03c7544508409bf9b7ab78a8852ddf78f13affc | Left child  value: [] | Right  child  value: [] | Paren
Node id: 23 Node Hash value: bca6fc5847b56c2a8409d98442054ce46b877fba3569ea80c4c0c4a3 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 24 Node Hash value: 338db7e1a8bf9e178f547f7ebec7f3737c8a516c67c55c88bc7e7141 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 25 Node Hash value: c97928f5e4fb85b432f96d93709b73567db1df745ba8d369a53c6a63 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 26 Node Hash value: bf8f47fd87fc160c34ff7229fa1cf30677ea37db375d3d652f3a28f9 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 27 Node Hash value: dcb223b75c4c671296dfa89d8a0032d863545d24c8a3569eb1ed3f52 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 28 Node Hash value: 82d146a6514f7065f348e8301a66bd05ebea3864e976af088fa61055 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 29 Node Hash value: 1f88bc5e1b9cec34f375e52c92e91745041a8aa44410db0608871825 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 30 Node Hash value: a7f0a721ee7e0d9e53768357da14dba80c5b22ddca5735cd829d77d9 | Left child  value: [] | Right  child  value: [] | Paren
Node id: 31 Node Hash value: 129c0868a7445c25f238473c8ee1de1daae75c32305dc2b00bbbb2ec | Left child  value: [] | Right  child  value: [] | Paren
*****
Node id: 33 Node Hash value: 5fa8c157e06517ebf0ab39b6f1de1d44ba313c989e18d349cd144c89 | Left child  value: 9fb55c650f330351ef23de3f5e3850118a25
Node id: 34 Node Hash value: 3be5ac69b7bd963bb7a730f725c264b30ccaaac43ea8fa830c44cf40 | Left child  value: 7a4d4002503ba30183ad091ac31af4bfd20c
Node id: 35 Node Hash value: e38e799f901917aa45ce9688397eac1bdd3f425b8645e0177873bbe9 | Left child  value: e7363f3a074f5f8da529e7a7d3d4e710820c
Node id: 36 Node Hash value: f49de71dcd1439804f947099c603036a1f48c0c5f9a6d1c2368e0378 | Left child  value: 68fe9a0e8a07bc42f1663621630f9f6790fb
Node id: 37 Node Hash value: 4953cf030917dfb2f9846a6c8c8859a20d2f00773038562f2e26fab9 | Left child  value: 2fc447c0936a128f96d90c2f3817912db3f0
Node id: 38 Node Hash value: 4ccb970774b3daa38d538654aa4bd08b10529dee9947c44deee4d063 | Left child  value: 9678193d089f735f87612978442bd1f38449
Node id: 39 Node Hash value: de696f98a4b89625431b8c51dc28ebf3cf23f8650f03d8ed11d88e65 | Left child  value: 576f1334f146a67cc732a9069653a20f70e2
Node id: 40 Node Hash value: eff5e3c3c83046986d1581f135a3c7ffb530410a7d9b4b66ab233743 | Left child  value: 6db1167043eef2ed182850fae61c662bf276
Node id: 41 Node Hash value: e930ff73ee6e6373fa1670631a5ea4ee5a0e1f3eb8154be9967812f7 | Left child  value: 3f2d605d272856178bfd2a20f9076188a07b
Node id: 42 Node Hash value: e88d1a9df826da0e356bdaa9a19ec86515fbb04e3f59fc03c4de7c42 | Left child  value: 0cfa2564f839a3c84340e6959bc32bfd6f3f
Node id: 43 Node Hash value: c310e64eec2f00eaa92c8887ece4c47df532ff9216dcd8ad72f322ac | Left child  value: ff113898591dee20c0679aedffea4cca9126
Node id: 44 Node Hash value: 1107e50e56f66320a1f2caa88bc7fd2b49f8d278c10ddbe8c6591de6 | Left child  value: 23302af8c0e15152f38376f5601ac3defbfa
Node id: 45 Node Hash value: f4f7e9ef02fa1a3c26a5d30bfe398bc51892992d767166cbcc9e9447 | Left child  value: f81a6323aa002ec278775867b68ec1dc24e7
Node id: 46 Node Hash value: d6621e28363fd4f96e949f2fcd44a824c56696a945b5e0a371cf3192 | Left child  value: 156a76a81597c872c0a3282ce1f7fa418fee
Node id: 47 Node Hash value: f04538a895b563a3930da4003caed177dea59344b70d6ec7b9786e79 | Left child  value: f52e8d2980d378a31552f43a2a3f7d1ef2ca
Node id: 48 Node Hash value: b542d53df47a556ab892df330c9a8a34f5a64874bfcf6755524a0f61 | Left child  value: dee6ce979de69350988c78ac8fae7866e60b
*****
Node id: 49 Node Hash value: 5a580eb75fda76e0dce104e44528ace56981e4214141f4929b2559f4 | Left child  value: 4b4fe68cb1b14477aadfad4b7ed9b8c33bc2
Node id: 50 Node Hash value: 3894a3e52e01b06b435251c6e04365ab1d96e5102050850ec1329aa2 | Left child  value: b4aafdf3319206e54c639b86bd5dbcb57fe0
```

```
from·Crypto.PublicKey·import·DSA
from·Crypto.Signature·import·DSS
from·Crypto.Hash·import·SHA256

#·Generate·a·new·DSA·key·pair
```

```
key·=·DSA.generate(2048)··#112·bit·security
start_time·=·time.time()

#·Sign·a·message
def·sign(msg):
····h·=·SHA256.new(msg)
····signer·=·DSS.new(key,·'fips-186-3')
····signature·=·signer.sign(h)
····return·signature

#·Verify·a·message
def·verify(msg,·signature):
····h·=·SHA256.new(msg)
····verifier·=·DSS.new(key,·'fips-186-3')
····try:
········verifier.verify(h,·signature)
········return·True
····except·ValueError:
········return·False

#·Example·usage
msg·=·b"Lorem·ipsum·dolor·sit·amet,·consectetur·adipiscing·elit.·Fusce·vel·ex·sapien.·Integer·commodo,·mauris·non·vestibulum·euismod,·nunc·libero·eleme
signature·=·sign(msg)
end_time·=·time.time()
end1_time·=·end_time·-·start_time

is_valid·=·verify(msg,·signature)
print("Signature:",·signature)
print("Is·valid:",·is_valid)
#security·=·key.size()·//·2


print("DSA·efficiency:·{}·seconds".format(end1_time))
#print("DSA·security:·{}·bits".format(security))
```

```
Signature: b'\xdd\xbf\x10\xcaq\r\x0b\xf7)\xc0\x91#\xb7\xc24\xb1\x08\x8b\x9dz\xb2#vg{#t\x8517\x0er\xe7\xc4\xae\xa2\x18\x8f\xcdmG\xdc\xb0\x83\xfc\xe
Is valid: True
DSA efficiency: 0.002287626266479492 seconds
```