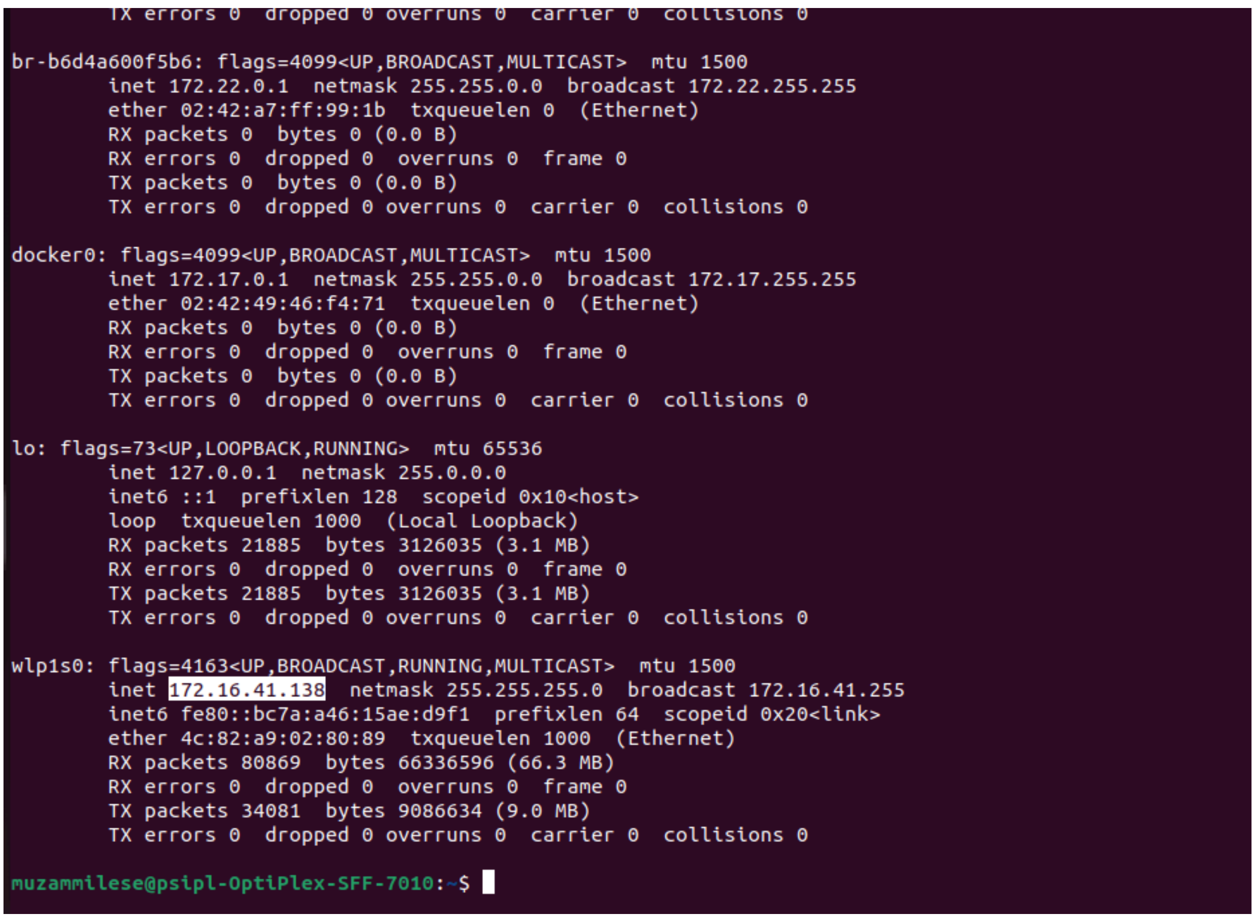
OEIT1- BCT (2023-2024)
Lab ESE

Date: 06/05/2024
Student Name: Muzammil Ansari
UCID: 2022701001
Branch: CSE-DS
Sem: VI

Lab title: **Blockchain Primitives- Cryptosystems**
Brief description of each task with screenshots (caption)



```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-b6d4a600f5b6: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.22.0.1  netmask 255.255.0.0  broadcast 172.22.255.255
        ether 02:42:a7:ff:99:1b  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:49:46:f4:71  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 21885  bytes 3126035 (3.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 21885  bytes 3126035 (3.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.41.138  netmask 255.255.255.0  broadcast 172.16.41.255
        inet6 fe80::bc7a:a46:15ae:d9f1  prefixlen 64  scopeid 0x20<link>
        ether 4c:82:a9:02:80:89  txqueuelen 1000  (Ethernet)
        RX packets 80869  bytes 66336596 (66.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 34081  bytes 9086634 (9.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

muzammilese@psipl-OptiPlex-SFF-7010:~$
```

```
muzammilese@psipl-OptiPlex-SFF-7010:~$ w
 10:51:31 up  1:40,  2 users,  load average: 0.59, 0.60, 0.65
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
psipl    tty2     tty2             09:11    1:40m  0.00s  0.00s /usr/libexec/gdm-wayland-session env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --session=u
muzammil tty3     tty3             09:27    1:40m  0.01s  0.01s /usr/libexec/gnome-session-binary --session=ubuntu
muzammilese@psipl-OptiPlex-SFF-7010:~$ date
Monday 06 May 2024 10:52:17 AM IST
muzammilese@psipl-OptiPlex-SFF-7010:~$
```

Task 1: Perform the cryptographic hash functions md5 and SHA

Code:

import hashlib

Data to be hashed

data = b"Hello, World!"

print(data)

MD5 hash

md5_hash = hashlib.md5(data).hexdigest()

print("MD5 Hash:", md5_hash)

SHA-256 hash

sha256_hash = hashlib.sha256(data).hexdigest()

print("SHA-256 Hash:", sha256_hash)

Output:

```
muzammilese@psipl-OptiPlex-SFF-7010:~$ python3 task1.py
MD5 Hash: 65a8e27d8879283831b664bd8b7f0ad4
SHA-256 Hash: dffd6021bb2bd5b0af676290809ec3a53191dd81c7f70a4b28688a362182986f
muzammilese@psipl-OptiPlex-SFF-7010:~$ python3 task1.py
b'Hello, World!'
MD5 Hash: 65a8e27d8879283831b664bd8b7f0ad4
SHA-256 Hash: dffd6021bb2bd5b0af676290809ec3a53191dd81c7f70a4b28688a362182986f
muzammilese@psipl-OptiPlex-SFF-7010:~$
```

Here I used the python function for MD5 and SHA-256 hashing, i passed the input string “Hello, World!”. Both functions' hashed value is generated and displayed as output..

Task 2: Demonstrate the properties of Cryptographic hash functions.

Code:

import hashlib

Deterministic property

```
hash1 = hashlib.sha256(b"Hello, World!").hexdigest()

hash2 = hashlib.sha256(b"Hello, World!").hexdigest()

print("Deterministic property:", hash1 == hash2)


# Avalanche effect

hash1 = hashlib.sha256(b"Hello, World!").hexdigest()

hash2 = hashlib.sha256(b"Hello, World!!").hexdigest()

print("Avalanche effect:", hash1 != hash2)
```

Output:

```
muzammilese@psipl-OptiPlex-SFF-7010:~$ python3 task2.py
String a= Hello, World!
String b= Hello, World!!
Deterministic property: True
Avalanche effect: True
muzammilese@psipl-OptiPlex-SFF-7010:~$
```

Here i am demonstrating two property which is deterministic and avalanche effect, in deterministic both the input is same as “Hello World” and then their comparison is done, as deterministic property states that same input always produces same output. Then, in avalanche effect states small change in input can cause a large difference in the output so for input string a i given “Hello, World!” and for b”Hello, World!!” adding one more exclamation point in the end makes the both hash different from each other.

Task 3: Use the Pymerkel library to compute the hash of transactions.

Code:

```
def compute_merkle_tree(transactions):

# Convert transactions to leaf nodes and compute hashes

leaves = []

for transaction in transactions:

transaction_hash = hashlib.sha256(transaction.encode()).digest()

leaves.append(transaction_hash)

print(f"Hash for transaction '{transaction}': {transaction_hash.hex()}")
```

```
# Construct tree

root_hash = _compute_root(leaves)


return root_hash


def _compute_root(nodes):

if len(nodes) == 1:

return nodes[0]


parents = []

for i in range(0, len(nodes), 2):

left_child = nodes[i]

right_child = nodes[i + 1] if i + 1 < len(nodes) else nodes[i]

parent = hashlib.sha256(left_child + right_child).digest()

parents.append(parent)

return _compute_root(parents)

# Example usage

transactions = ["transaction1", "transaction2", "transaction3", "transaction4", "transaction5",

"transaction6", "transaction7", "transaction8", "transaction9", "transaction10", "transaction11"]

root_hash = compute_merkle_tree(transactions)

print("Root Hash:", root_hash.hex())
```

Output:

```python
        right_child = nodes[i + 1] if i + 1 < len(nodes) else nodes[i]
        parent = hashlib.sha256(left_child + right_child).digest()
        parents.append(parent)

    return _compute_root(parents)

# Example usage
transactions = ["transaction1", "transaction2", "transaction3", "transaction4", "transaction5",
                "transaction6", "transaction7", "transaction8", "transaction9", "transaction10", "trans
root_hash = compute_merkle_tree(transactions)
# print("Root Hash:", root_hash.hex())
```

```
Hash for transaction 'transaction1': bde4693e55a336ff81ab238ce20cae1dd9c8ba03b9b8f43963f5569bf3cf5229
Hash for transaction 'transaction2': 4beace8bdcf9b5b74630eaee2e7f501180e46025ca89b05e7e041fbe953d817a
Hash for transaction 'transaction3': 293755ab6384e02d9202d483f2f0250100d786e75fdab1b6f3925b2800ece3cb
Hash for transaction 'transaction4': 2435dc0372e12b3f7684fb7093fbe6f6dee79dbff96cc28b1687839ef526e02f
Hash for transaction 'transaction5': 26af218ec2912ce97d8dac14ce69b73e54d935139ad0186cf2cc99f927acfd96
Hash for transaction 'transaction6': 1fd4bbf0a645748595e05cc79a980bf10d65a17c50f63226e7f39993bc09a9f8
Hash for transaction 'transaction7': 221fb44023518da3908a8cefe5b97d1a42218e5a8316e35a7c4077302f8ed9a2
Hash for transaction 'transaction8': 4f029880ea69372ae4d659e002cce9b6e87dfc608400d6e93f27a3862214980e
Hash for transaction 'transaction9': 0620fac861009d4d5b44b18c95ee63eefefbb5a0178276f6850c9c65da7455ab
Hash for transaction 'transaction10': 0fbe4ca079c883dd26d53295ee2d091ed9a8551958f003508d1a13851d418fec
Hash for transaction 'transaction11': 244cbb4c15aba1b98e8d754c5b47d85944187c5c29e5d57d4af42025c8d05167
```

i created merkle tree then used sha256 algorithm to calculate hash of each transactions, i am adding node to the tree as transaction then calculating its hash value then again repeating the same till reaching end node.

Task 4: Compute the Root hash (Merkle Root) for 11 transactions.

Code:

```
import hashlib


def compute_root_hash(transactions):

# Convert transactions to leaf nodes

leaves = [hashlib.sha256(transaction.encode()).digest() for transaction in transactions]


# Construct tree

tree = _compute_tree(leaves)


return tree[0]


def _compute_tree(nodes):

if len(nodes) == 1:
```

```
return nodes

parents = []
for i in range(0, len(nodes), 2):
left_child = nodes[i]
right_child = nodes[i+1] if i+1 < len(nodes) else nodes[i]
parent = hashlib.sha256(left_child + right_child).digest()
parents.append(parent)

return _compute_tree(parents)

# Example usage
transactions = ["transaction1", "transaction2", "transaction3", "transaction4", "transaction5",
"transaction6", "transaction7", "transaction8", "transaction9", "transaction10", "transaction11"]
root_hash = compute_root_hash(transactions)
print("Root Hash:", root_hash.hex())
```

Output:

```
import hashlib

def compute_root_hash(transactions):
    # Convert transactions to leaf nodes
    leaves = [hashlib.sha256(transaction.encode()).digest() for transaction in

    # Construct tree
    tree = _compute_tree(leaves)

    return tree[0]

def _compute_tree(nodes):
    if len(nodes) == 1:
        return nodes

    parents = []
    for i in range(0, len(nodes), 2):
        left_child = nodes[i]
        right_child = nodes[i+1] if i+1 < len(nodes) else nodes[i]
        parent = hashlib.sha256(left_child + right_child).digest()
        parents.append(parent)

    return _compute_tree(parents)

# Example usage
transactions = ["transaction1", "transaction2", "transaction3", "transaction4"
                "transaction6", "transaction7", "transaction8", "transaction9"
root_hash = compute_root_hash(transactions)
print("Root Hash:", root_hash.hex())
```

```
Root Hash: f257096a3380cedfd926cd079c89a2187cec1082a65b5485e0f8e8023e7fee97
```

Used same merkle tree created in before task then calculated root hash by combining transaction and calculating its hash value then again repeating the same till i reach root. Used sha256 algo for calculating hash.

All 4 tasks are completed!.

History:

```
muzammilese@psipl-OptiPlex-SFF-7010:~$ history
    1  su psipl
    2  who am i
    3  whoami
    4  ifconfig
    5  nano task1.py
    6  python
    7  python3
    8  python task1.py
    9  python3 task1.py
   10  python3 task2.py
   11  python3 task3n4.py
   12  pip install pymerkle
   13  python3 task3n4.py
   14  pip install --upgrade pymerkle
   15  python3 task3n4.py
   16  cd /home/muzammilese/.local/lib/python3.10/site-packages/pymerkle/
   17  ls
   18  cd
   19  cd /home/muzammilese/.local/lib/python3.10/site-packages/pymerkle/
   20  ls
   21  cd concrete
   22  ls
   23  cd ..
   24  ls
   25  cd pymerkle
   26  ls
   27* pip3 install pymerkle
   28  cd
   29  python3 task3n4.py
   30  history
   31  python3 task2.py
   32  history
muzammilese@psipl-OptiPlex-SFF-7010:~$
```