Bhartiya Vidya Bhavan’s
Sardar Patel Institute of Technology, Mumbai-400058
Department of Electronics and Telecommunication Engineering
IT424:Blockchain Technology and Applications

Lab4: Blockchain Programming-II

Merkle-tree cryptographic library for generation and validation of Proofs

Objective: Merkle Tree Implementation, generation and validation proofs

Outcomes: After successful completion of lab students should be able to
Implement Merkle Tree and demonstrate
Write a code in python to build a Merkle Tree
Demonstrate the Merkle Tree as a fundamental part of Blockchain.

System Requirements:
PC (C2D, 4GB RAM, 100GB HDD space and NIC)
Ubuntu Linux 14.04/20.04
Internet connectivity
Pymerkle
Python Cryptography and Pycrypto

Part-4A: Implementing Merkle Tree using Python

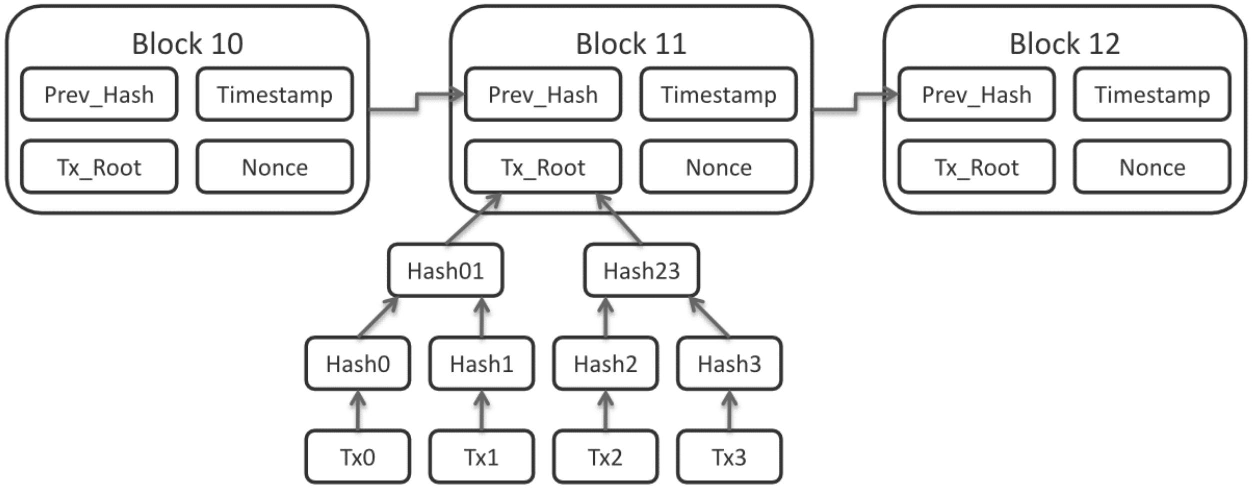
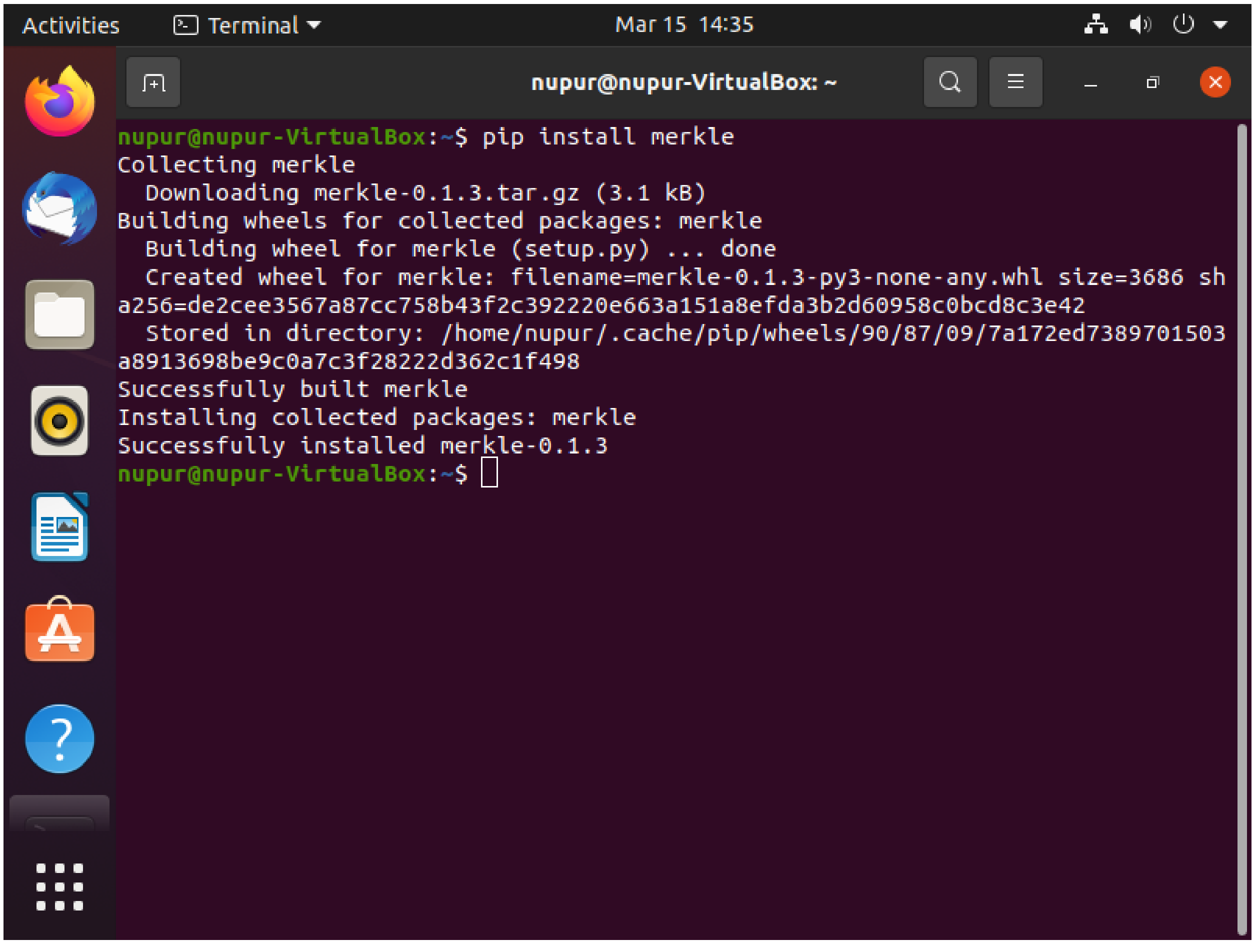


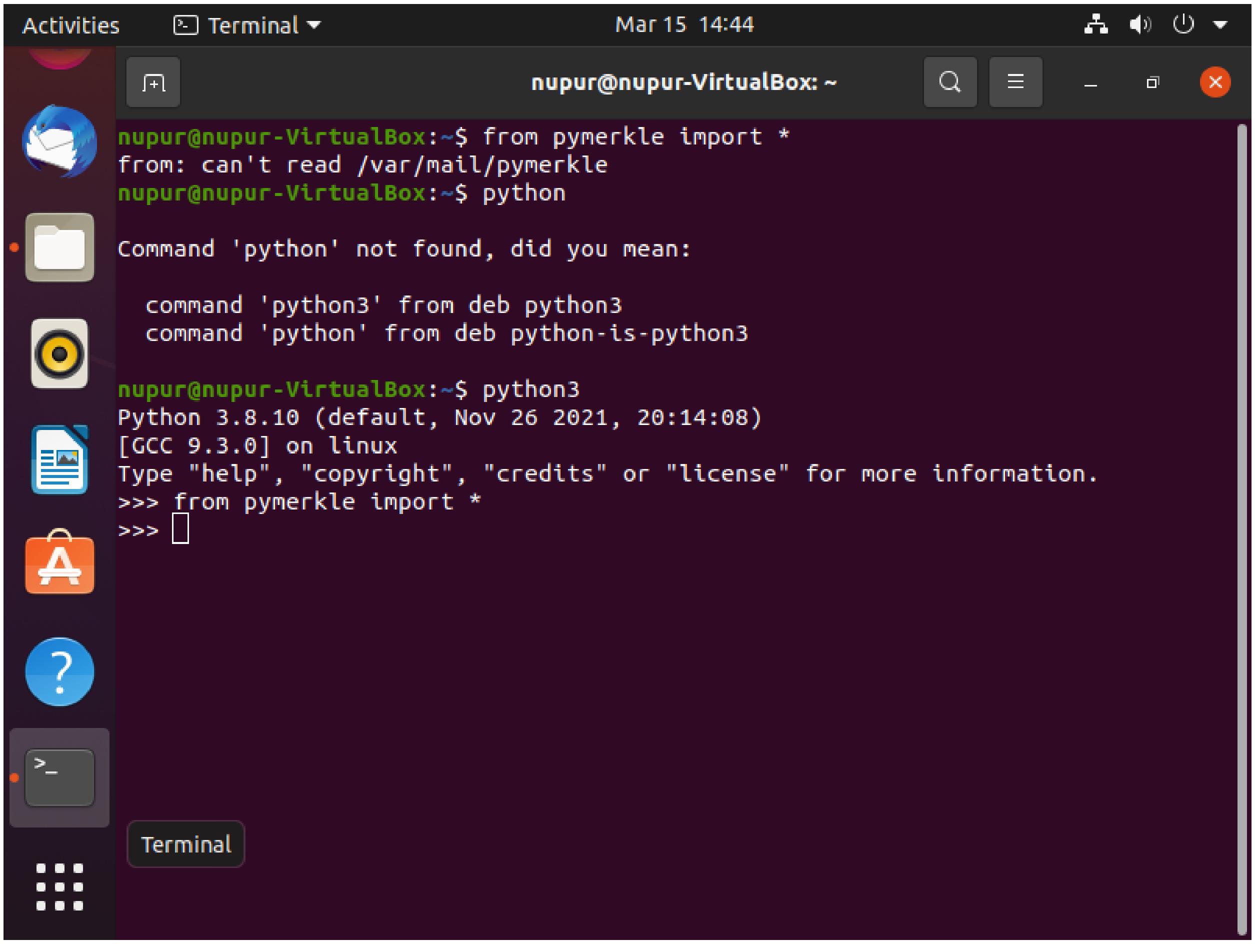
Figure-1: Merkle Tree

Procedure:
[1] Install merkel [1]:
pip install merkle



```
nupur@nupur-VirtualBox:~$ pip install merkle
Collecting merkle
  Downloading merkle-0.1.3.tar.gz (3.1 kB)
Building wheels for collected packages: merkle
  Building wheel for merkle (setup.py) ... done
  Created wheel for merkle: filename=merkle-0.1.3-py3-none-any.whl size=3686 sh
a256=de2cee3567a87cc758b43f2c392220e663a151a8efda3b2d60958c0bcd8c3e42
  Stored in directory: /home/nupur/.cache/pip/wheels/90/87/09/7a172ed7389701503
a8913698be9c0a7c3f28222d362c1f498
Successfully built merkle
Installing collected packages: merkle
Successfully installed merkle-0.1.3
nupur@nupur-VirtualBox:~$ 
```

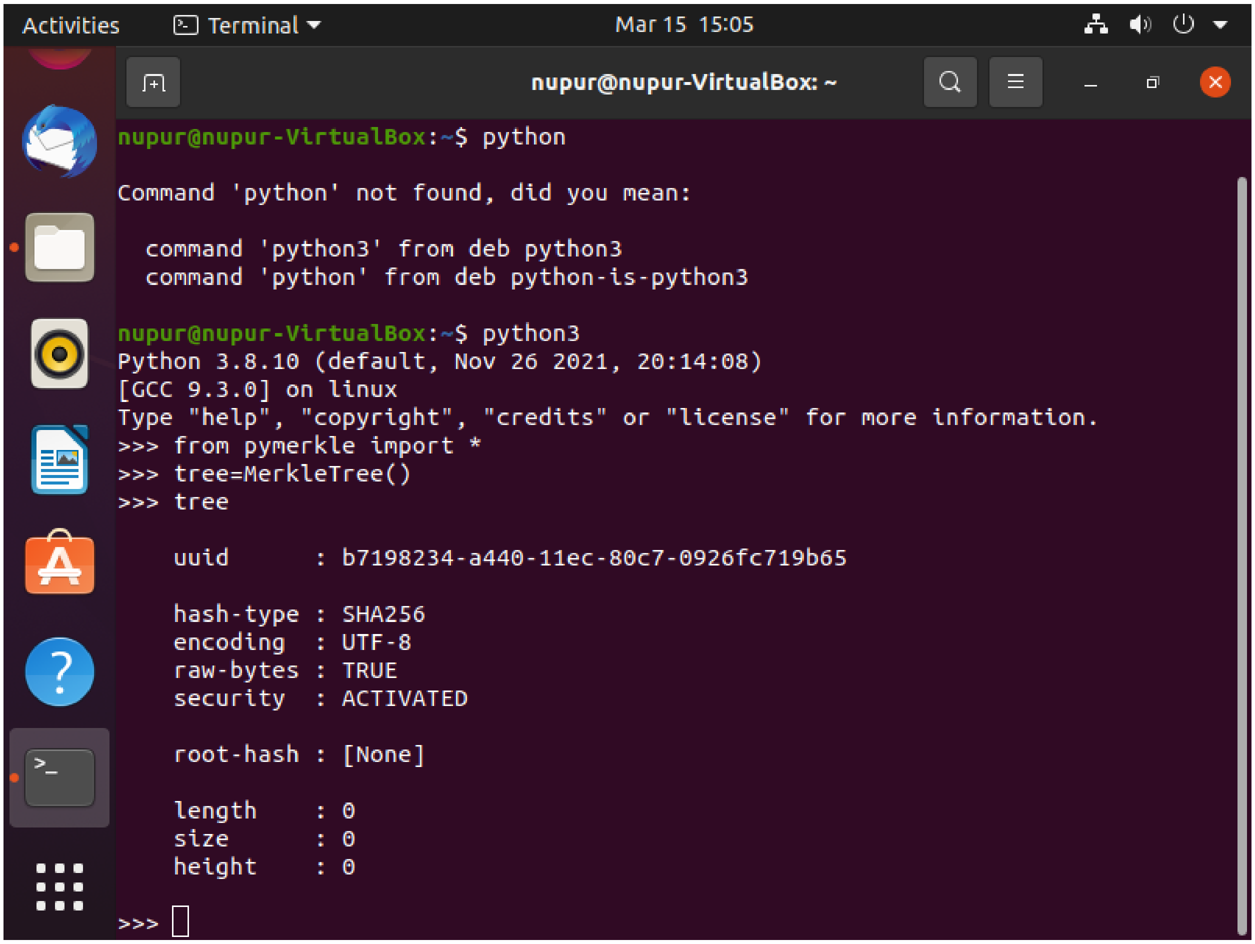
[2] Import merkle
from pymerkle import *



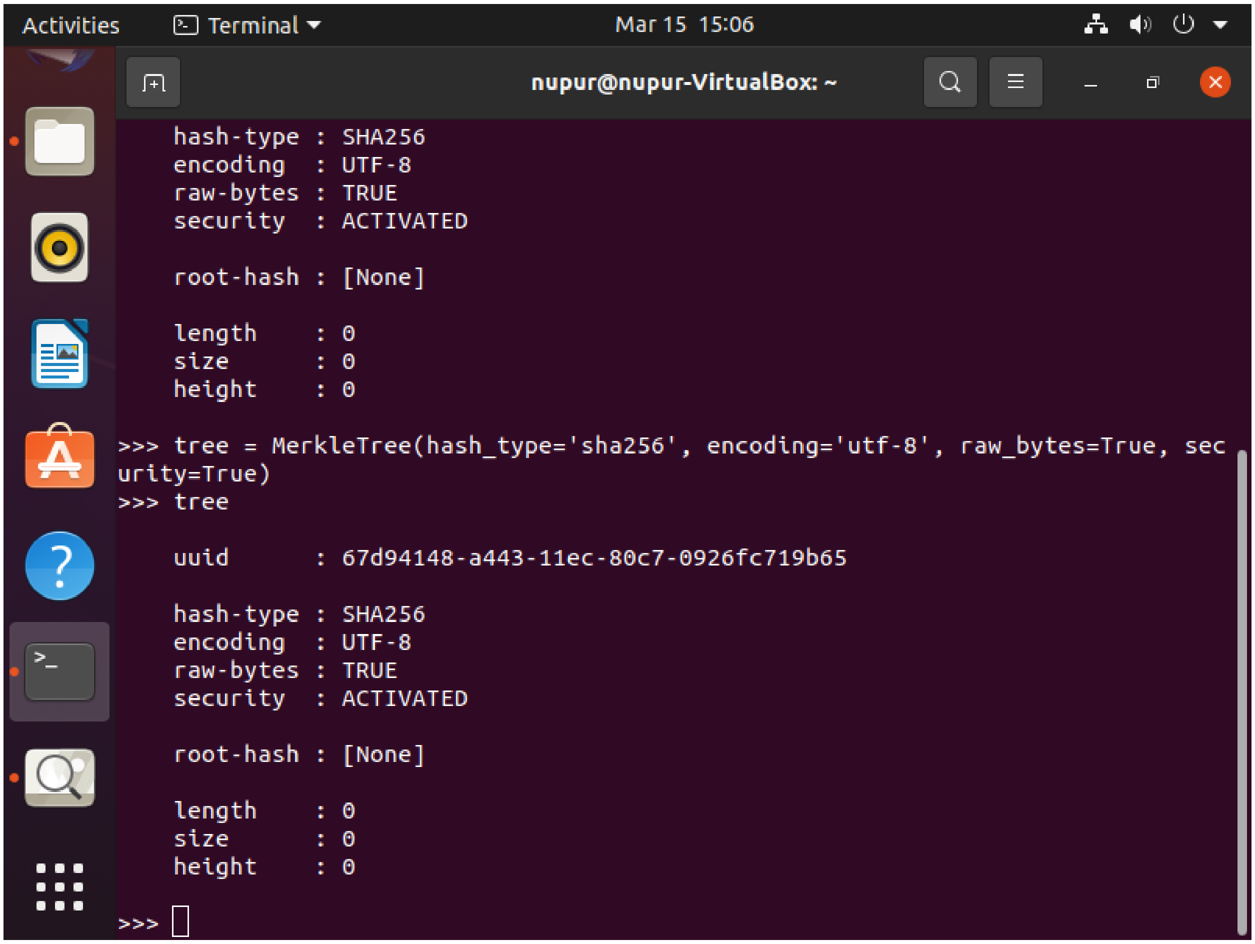
```
nupur@nupur-VirtualBox:~$ from pymerkle import *
from: can't read /var/mail/pymerkle
nupur@nupur-VirtualBox:~$ python

Command 'python' not found, did you mean:

  command 'python3' from deb python3
  command 'python' from deb python-is-python3

nupur@nupur-VirtualBox:~$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pymerkle import *
>>> 
```

[3] Merkle tree object:
tree=MerkleTree()

[4] Explore configuration of tree and Attributes and properties
Refer to the official documentation of Merkle Tree implementation (pymerkle)



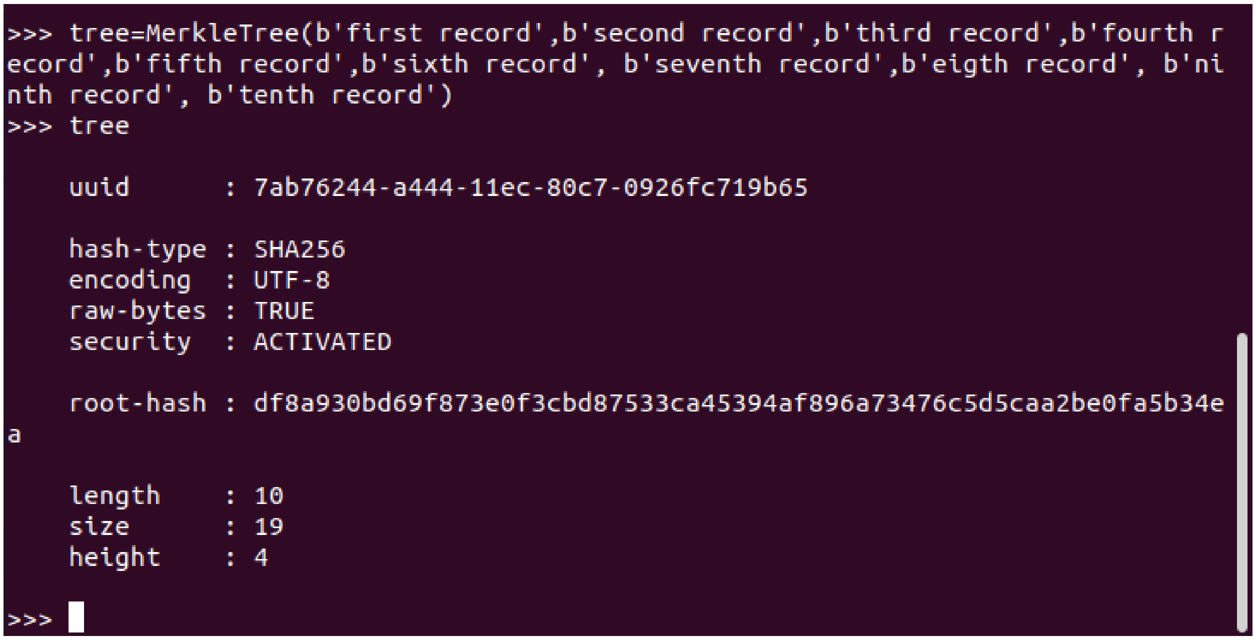
```
nupur@nupur-VirtualBox:~$ python

Command 'python' not found, did you mean:

  command 'python3' from deb python3
  command 'python' from deb python-is-python3

nupur@nupur-VirtualBox:~$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pymerkle import *
>>> tree=MerkleTree()
>>> tree

    uuid      : b7198234-a440-11ec-80c7-0926fc719b65

    hash-type : SHA256
    encoding  : UTF-8
    raw-bytes : TRUE
    security  : ACTIVATED

    root-hash : [None]

    length    : 0
    size      : 0
    height    : 0

>>> 
```



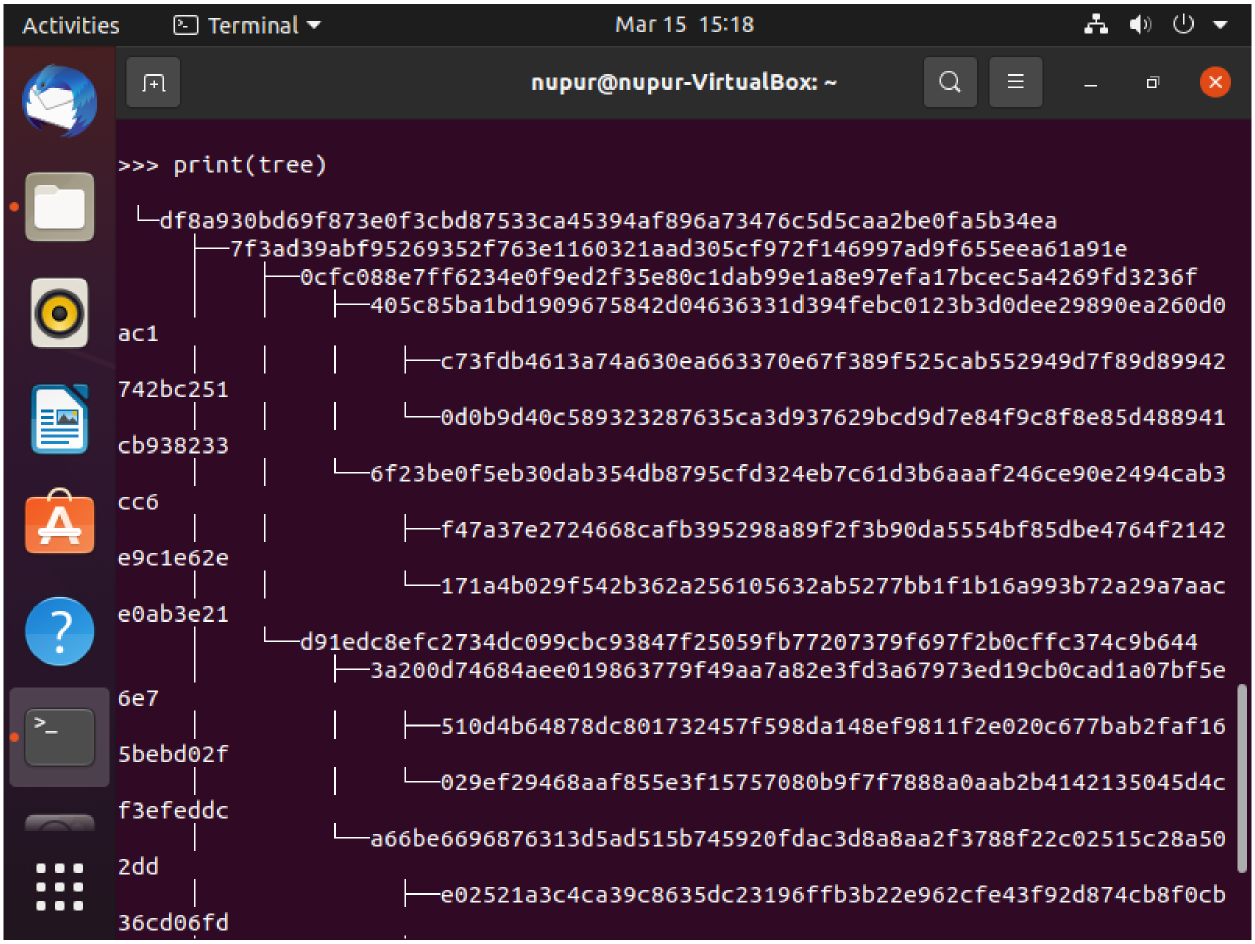
```
    hash-type : SHA256
    encoding  : UTF-8
    raw-bytes : TRUE
    security  : ACTIVATED

    root-hash : [None]

    length    : 0
    size      : 0
    height    : 0

>>> tree = MerkleTree(hash_type='sha256', encoding='utf-8', raw_bytes=True, sec
urity=True)
>>> tree

    uuid      : 67d94148-a443-11ec-80c7-0926fc719b65

    hash-type : SHA256
    encoding  : UTF-8
    raw-bytes : TRUE
    security  : ACTIVATED

    root-hash : [None]

    length    : 0
    size      : 0
    height    : 0

>>> 
```

[5] Create 10 transactions record and build the Merkle tree and verify

Add screenshot with brief description.



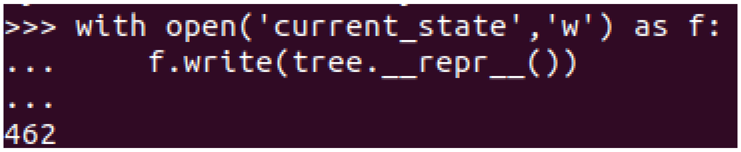
```
>>> tree=MerkleTree(b'first record',b'second record',b'third record',b'fourth r
ecord',b'fifth record',b'sixth record', b'seventh record',b'eigth record', b'ni
nth record', b'tenth record')
>>> tree

    uuid      : 7ab76244-a444-11ec-80c7-0926fc719b65

    hash-type : SHA256
    encoding  : UTF-8
    raw-bytes : TRUE
    security  : ACTIVATED

    root-hash : df8a930bd69f873e0f3cbd87533ca45394af896a73476c5d5caa2be0fa5b34e
a

    length    : 10
    size      : 19
    height    : 4

>>> 
```



```
>>> print(tree)

 └─df8a930bd69f873e0f3cbd87533ca45394af896a73476c5d5caa2be0fa5b34ea
     ├──7f3ad39abf95269352f763e1160321aad305cf972f146997ad9f655eea61a91e
     │    ├──0cfc088e7ff6234e0f9ed2f35e80c1dab99e1a8e97efa17bcec5a4269fd3236f
     │    │    ├──405c85ba1bd1909675842d04636331d394febc0123b3d0dee29890ea260d0
ac1
     │    │    │    ├──c73fdb4613a74a630ea663370e67f389f525cab552949d7f89d89942
742bc251
     │    │    │    └──0d0b9d40c589323287635ca3d937629bcd9d7e84f9c8f8e85d488941
cb938233
     │    │    └──6f23be0f5eb30dab354db8795cfd324eb7c61d3b6aaaf246ce90e2494cab3
cc6
     │    │         ├──f47a37e2724668cafb395298a89f2f3b90da5554bf85dbe4764f2142
e9c1e62e
     │    │         └──171a4b029f542b362a256105632ab5277bb1f1b16a993b72a29a7aac
e0ab3e21
     │    └──d91edc8efc2734dc099cbc93847f25059fb77207379f697f2b0cffc374c9b644
     │         ├──3a200d74684aee019863779f49aa7a82e3fd3a67973ed19cb0cad1a07bf5e
6e7
     │         │    ├──510d4b64878dc801732457f598da148ef9811f2e020c677bab2faf16
5bebd02f
     │         │    └──029ef29468aaf855e3f15757080b9f7f7888a0aab2b4142135045d4c
f3efeddc
     │         └──a66be6696876313d5ad515b745920fdac3d8a8aa2f3788f22c02515c28a50
2dd
     │              ├──e02521a3c4ca39c8635dc23196ffb3b22e962cfe43f92d874cb8f0cb
36cd06fd
```

[6] Save as a file:

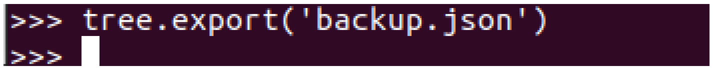
```
with open('current_state', 'w') as f:
    f.write(tree.__repr__())
```



```
>>> with open('current_state','w') as f:
...     f.write(tree.__repr__())
...
462
```

[7] Export and save as JSON

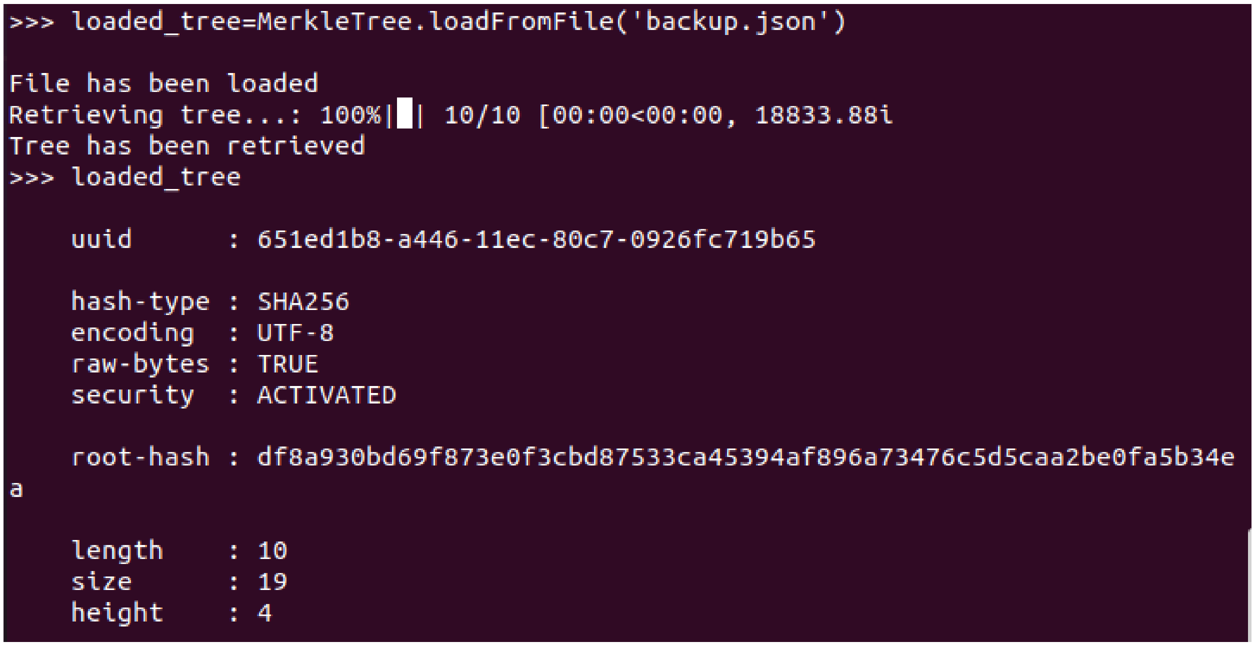
```
tree.export('backup.json')
```



```
>>> tree.export('backup.json')
>>> █
```

[8] Recover the tree by means of the .loadFromFile classmethod:

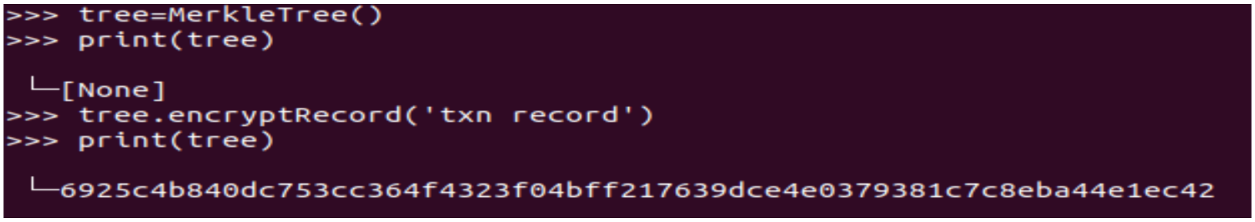
```
loaded_tree = MerkleTree.loadFromFile('backup.json')
```



```
>>> loaded_tree=MerkleTree.loadFromFile('backup.json')

File has been loaded
Retrieving tree...: 100%|█| 10/10 [00:00<00:00, 18833.88i
Tree has been retrieved
>>> loaded_tree

    uuid      : 651ed1b8-a446-11ec-80c7-0926fc719b65

    hash-type : SHA256
    encoding  : UTF-8
    raw-bytes : TRUE
    security  : ACTIVATED

    root-hash : df8a930bd69f873e0f3cbd87533ca45394af896a73476c5d5caa2be0fa5b34e
a

    length    : 10
    size      : 19
    height    : 4
```

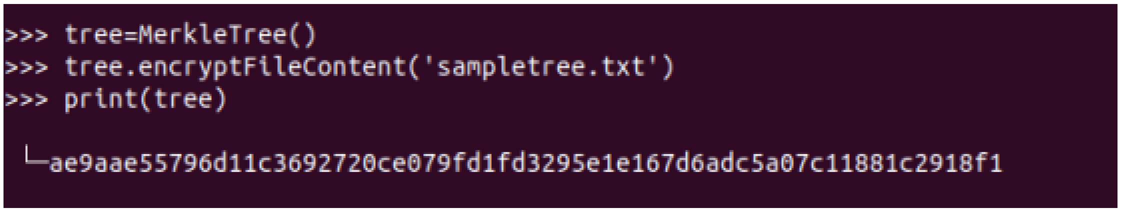
[9] Explore the Encryption modes

single record encryption



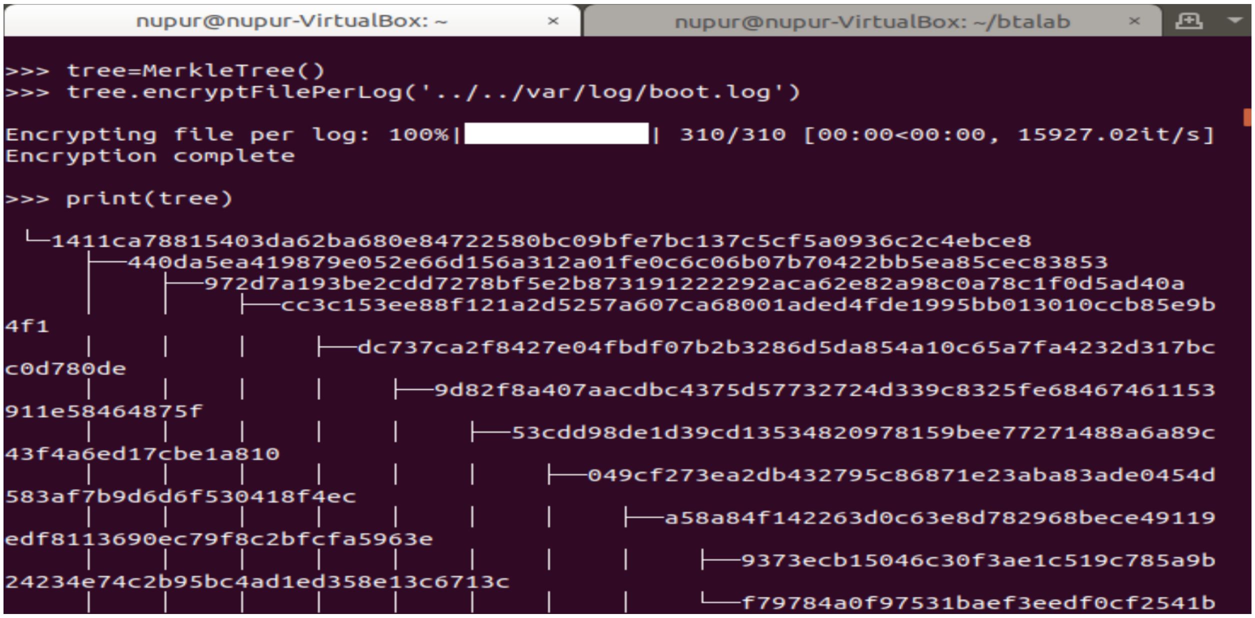
```
>>> tree=MerkleTree()
>>> print(tree)

 └─[None]
>>> tree.encryptRecord('txn record')
>>> print(tree)

 └─6925c4b840dc753cc364f4323f04bff217639dce4e0379381c7c8eba44e1ec42
```

Bulk file encryption

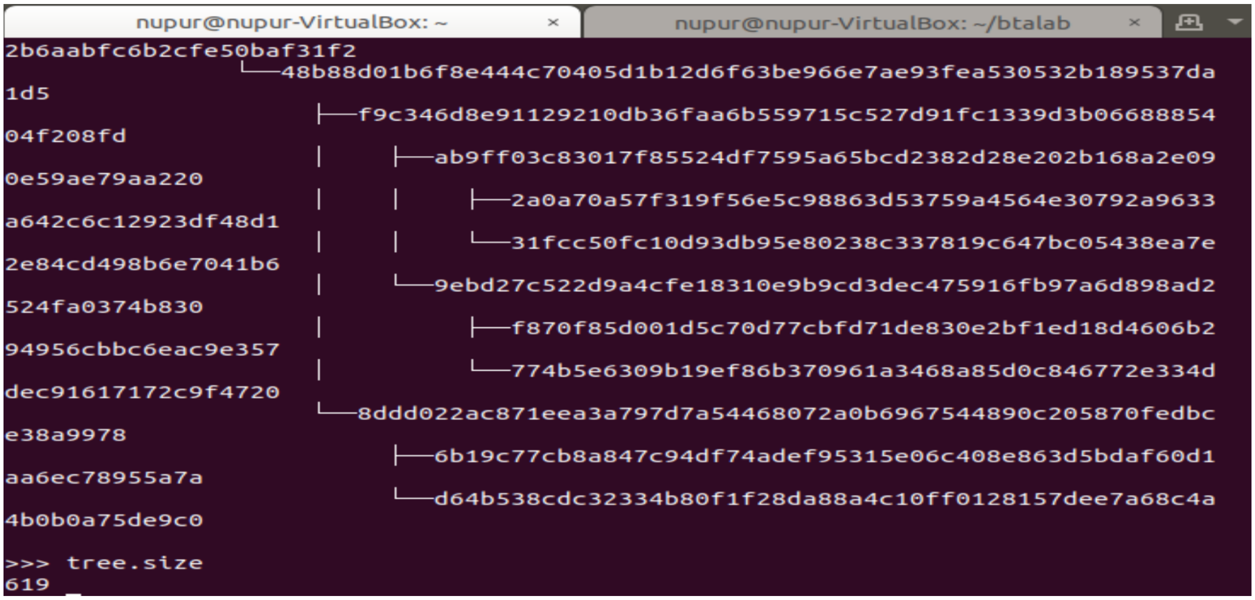


```
>>> tree=MerkleTree()
>>> tree.encryptFileContent('sampletree.txt')
>>> print(tree)

 └─ae9aae55796d11c3692720ce079fd1fd3295e1e167d6adc5a07c11881c2918f1
```

Per Log file encryption

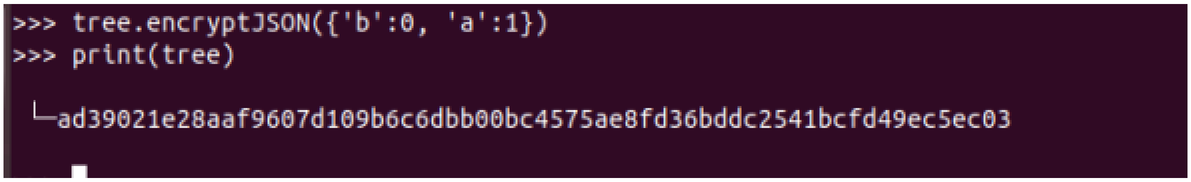


```
>>> tree=MerkleTree()
>>> tree.encryptFilePerLog('../../var/log/boot.log')

Encrypting file per log: 100%|████████████| 310/310 [00:00<00:00, 15927.02it/s]
Encryption complete

>>> print(tree)

 └─1411ca78815403da62ba680e84722580bc09bfe7bc137c5cf5a0936c2c4ebce8
     ├──440da5ea419879e052e66d156a312a01fe0c6c06b07b70422bb5ea85cec83853
     │    ├──972d7a193be2cdd7278bf5e2b873191222292aca62e82a98c0a78c1f0d5ad40a
     │    │    ├──cc3c153ee88f121a2d5257a607ca68001aded4fde1995bb013010ccb85e9b
4f1
     │    │    │    ├──dc737ca2f8427e04fbdf07b2b3286d5da854a10c65a7fa4232d317bc
c0d780de
     │    │    │    │    ├──9d82f8a407aacdbc4375d57732724d339c8325fe68467461153
911e58464875f
     │    │    │    │    │    ├──53cdd98de1d39cd13534820978159bee77271488a6a89c
43f4a6ed17cbe1a810
     │    │    │    │    │    │    ├──049cf273ea2db432795c86871e23aba83ade0454d
583af7b9d6d6f530418f4ec
     │    │    │    │    │    │    │    ├──a58a84f142263d0c63e8d782968bece49119
edf8113690ec79f8c2bfcfa5963e
     │    │    │    │    │    │    │    │    ├──9373ecb15046c30f3ae1c519c785a9b
24234e74c2b95bc4ad1ed358e13c6713c
     │    │    │    │    │    │    │    │    └──f79784a0f97531baef3eedf0cf2541b
```



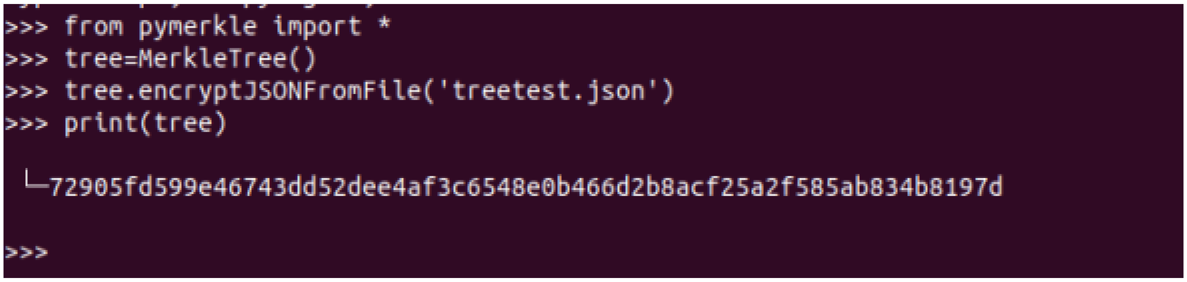
```
2b6aabfc6b2cfe50baf31f2
               └──48b88d01b6f8e444c70405d1b12d6f63be966e7ae93fea530532b189537da
1d5
                    ├──f9c346d8e91129210db36faa6b559715c527d91fc1339d3b06688854
04f208fd
                    │    ├──ab9ff03c83017f85524df7595a65bcd2382d28e202b168a2e09
0e59ae79aa220
                    │    │    ├──2a0a70a57f319f56e5c98863d53759a4564e30792a9633
a642c6c12923df48d1
                    │    │    └──31fcc50fc10d93db95e80238c337819c647bc05438ea7e
2e84cd498b6e7041b6
                    │    └──9ebd27c522d9a4cfe18310e9b9cd3dec475916fb97a6d898ad2
524fa0374b830
                    │         ├──f870f85d001d5c70d77cbfd71de830e2bf1ed18d4606b2
94956cbbc6eac9e357
                    │         └──774b5e6309b19ef86b370961a3468a85d0c846772e334d
dec91617172c9f4720
                    └──8ddd022ac871eea3a797d7a54468072a0b6967544890c205870fedbc
e38a9978
                         ├──6b19c77cb8a847c94df74adef95315e06c408e863d5bdaf60d1
aa6ec78955a7a
                         └──d64b538cdc32334b80f1f28da88a4c10ff0128157dee7a68c4a
4b0b0a75de9c0

>>> tree.size
619
```

Direct JSON encryption



```
>>> tree.encryptJSON({'b':0, 'a':1})
>>> print(tree)

 └─ad39021e28aaf9607d109b6c6dbb00bc4575ae8fd36bddc2541bcfd49ec5ec03
```

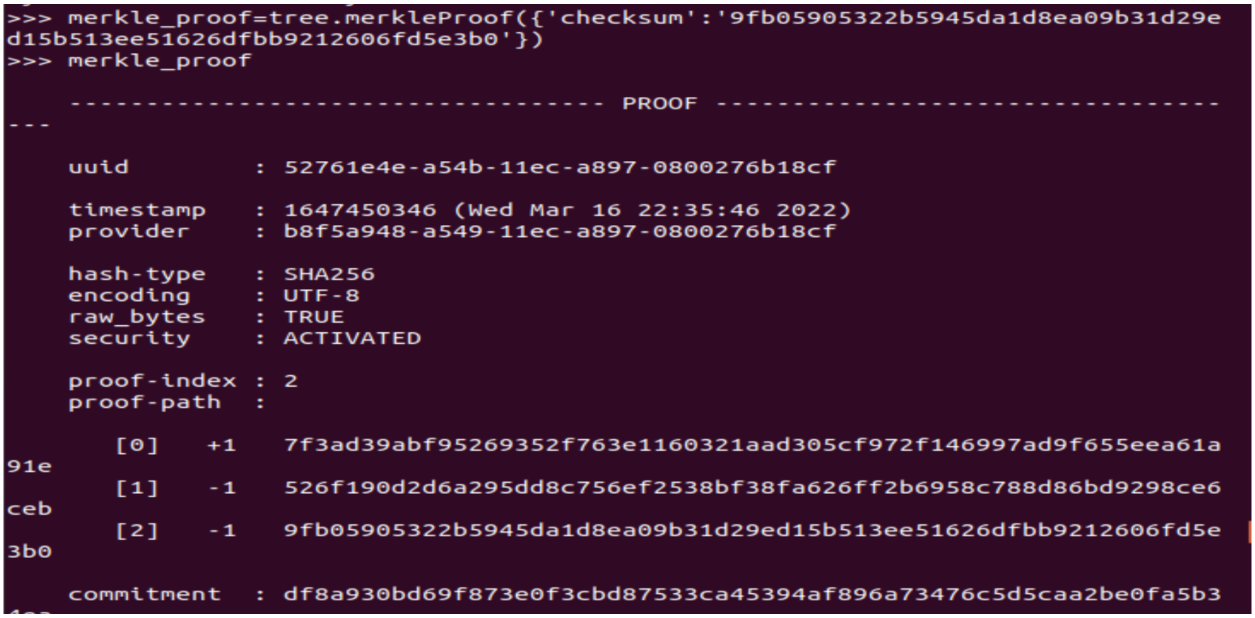
File based JSON encryption



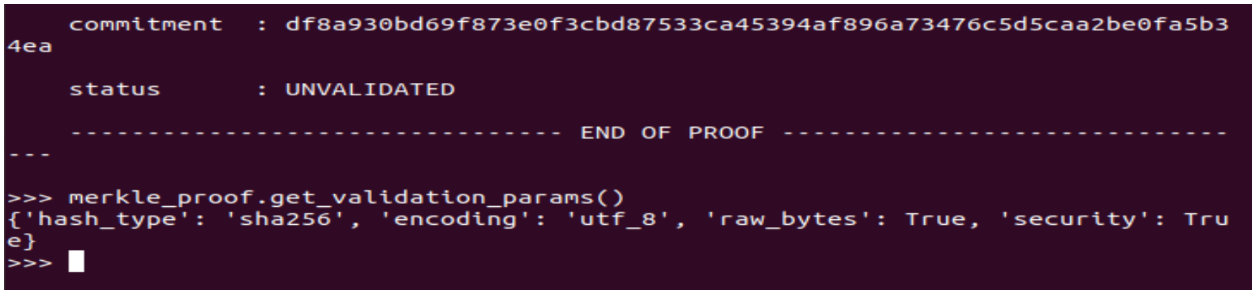
```
>>> from pymerkle import *
>>> tree=MerkleTree()
>>> tree.encryptJSONFromFile('treetest.json')
>>> print(tree)

 └─72905fd599e46743dd52dee4af3c6548e0b466d2b8acf25a2f585ab834b8197d

>>>
```

[10] Explore proof generation and validation

Proof generation
checksum is the digest value stored at one of the merkle tree’s leaves

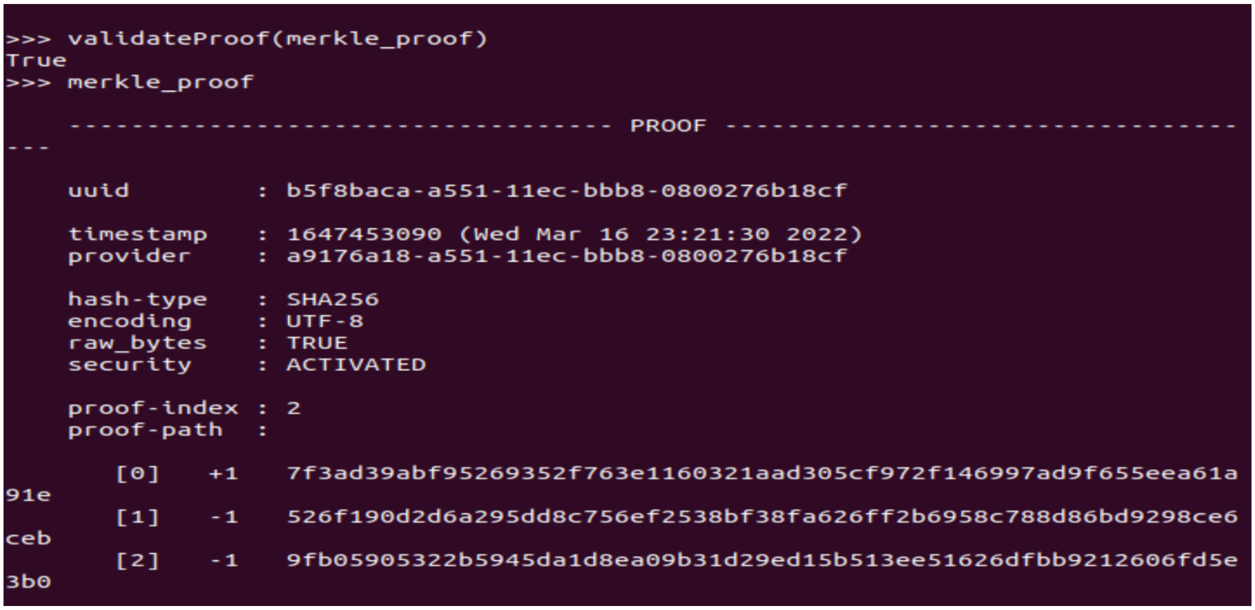


```
>>> merkle_proof=tree.merkleProof({'checksum':'9fb05905322b5945da1d8ea09b31d29e
d15b513ee51626dfbb9212606fd5e3b0'})
>>> merkle_proof

    ---------------------------------- PROOF --------------------------------
---

    uuid        : 52761e4e-a54b-11ec-a897-0800276b18cf

    timestamp   : 1647450346 (Wed Mar 16 22:35:46 2022)
    provider    : b8f5a948-a549-11ec-a897-0800276b18cf

    hash-type   : SHA256
    encoding    : UTF-8
    raw_bytes   : TRUE
    security    : ACTIVATED

    proof-index : 2
    proof-path  :

       [0]   +1   7f3ad39abf95269352f763e1160321aad305cf972f146997ad9f655eea61a
91e
       [1]   -1   526f190d2d6a295dd8c756ef2538bf38fa626ff2b6958c788d86bd9298ce6
ceb
       [2]   -1   9fb05905322b5945da1d8ea09b31d29ed15b513ee51626dfbb9212606fd5e
3b0

    commitment  : df8a930bd69f873e0f3cbd87533ca45394af896a73476c5d5caa2be0fa5b3
```

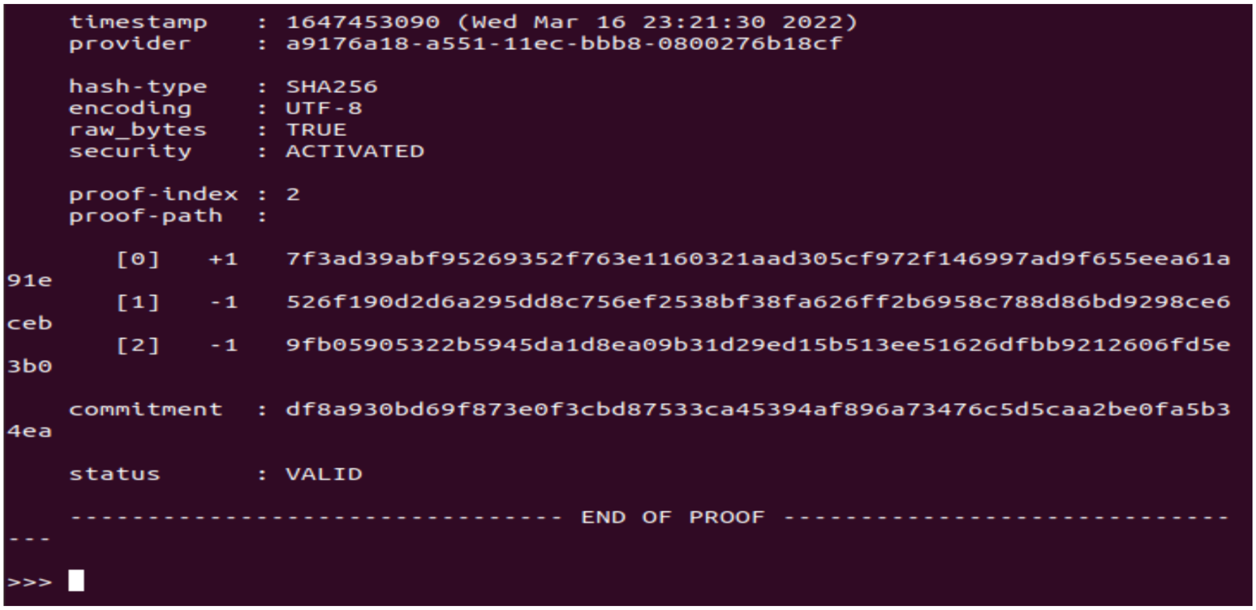


```
    commitment  : df8a930bd69f873e0f3cbd87533ca45394af896a73476c5d5caa2be0fa5b3
4ea

    status      : UNVALIDATED

    -------------------------------- END OF PROOF --------------------------------
---

>>> merkle_proof.get_validation_params()
{'hash_type': 'sha256', 'encoding': 'utf_8', 'raw_bytes': True, 'security': Tru
e}
>>> 
```

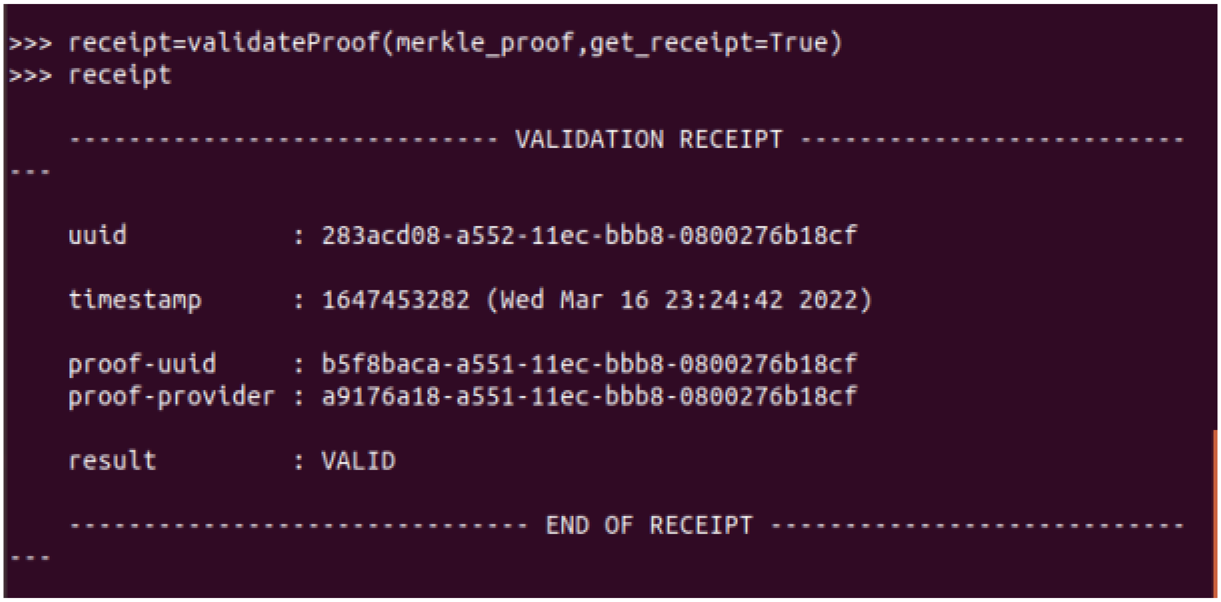
Proof validation



```
>>> validateProof(merkle_proof)
True
>>> merkle_proof

    ----------------------------------- PROOF -----------------------------------
---

    uuid        : b5f8baca-a551-11ec-bbb8-0800276b18cf

    timestamp   : 1647453090 (Wed Mar 16 23:21:30 2022)
    provider    : a9176a18-a551-11ec-bbb8-0800276b18cf

    hash-type   : SHA256
    encoding    : UTF-8
    raw_bytes   : TRUE
    security    : ACTIVATED

    proof-index : 2
    proof-path  :

       [0]   +1   7f3ad39abf95269352f763e1160321aad305cf972f146997ad9f655eea61a
91e
       [1]   -1   526f190d2d6a295dd8c756ef2538bf38fa626ff2b6958c788d86bd9298ce6
ceb
       [2]   -1   9fb05905322b5945da1d8ea09b31d29ed15b513ee51626dfbb9212606fd5e
3b0
```

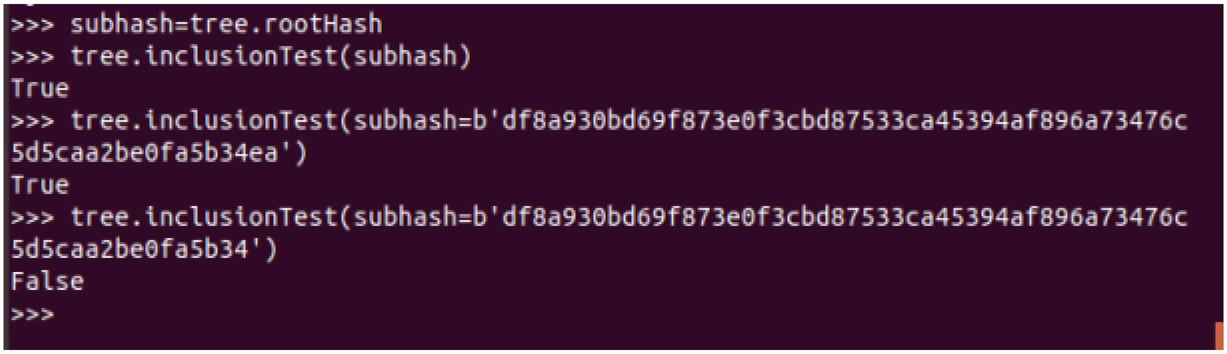


```
    timestamp   : 1647453090 (Wed Mar 16 23:21:30 2022)
    provider    : a9176a18-a551-11ec-bbb8-0800276b18cf

    hash-type   : SHA256
    encoding    : UTF-8
    raw_bytes   : TRUE
    security    : ACTIVATED

    proof-index : 2
    proof-path  :

       [0]   +1   7f3ad39abf95269352f763e1160321aad305cf972f146997ad9f655eea61a
91e
       [1]   -1   526f190d2d6a295dd8c756ef2538bf38fa626ff2b6958c788d86bd9298ce6
ceb
       [2]   -1   9fb05905322b5945da1d8ea09b31d29ed15b513ee51626dfbb9212606fd5e
3b0

    commitment  : df8a930bd69f873e0f3cbd87533ca45394af896a73476c5d5caa2be0fa5b3
4ea

    status      : VALID

    -------------------------------- END OF PROOF --------------------------------
---

>>> 
```



```
>>> receipt=validateProof(merkle_proof,get_receipt=True)
>>> receipt

    ----------------------------- VALIDATION RECEIPT --------------------------
---

    uuid           : 283acd08-a552-11ec-bbb8-0800276b18cf

    timestamp      : 1647453282 (Wed Mar 16 23:24:42 2022)

    proof-uuid     : b5f8baca-a551-11ec-bbb8-0800276b18cf
    proof-provider : a9176a18-a551-11ec-bbb8-0800276b18cf

    result         : VALID

    ------------------------------- END OF RECEIPT ----------------------------
---
```

[11] Explore the Inclusion Tests



```
>>> subhash=tree.rootHash
>>> tree.inclusionTest(subhash)
True
>>> tree.inclusionTest(subhash=b'df8a930bd69f873e0f3cbd87533ca45394af896a73476c
5d5caa2be0fa5b34ea')
True
>>> tree.inclusionTest(subhash=b'df8a930bd69f873e0f3cbd87533ca45394af896a73476c
5d5caa2be0fa5b34')
False
>>>
```

Conclusion:

Thus implemented the Merkle tree successfully with the help of pymerkle library. Checked the configuration and inserted records into the tree. Explored different encryption nodes with the help of which we can store data from different file types into the tree. Performed proof generation, validation and inclusion tests. Exported the tree into a JSON file. Hence learnt about the merkle tree which is fast, efficient and requires less storage space and is verifiable.

References:

[1] Merkle-tree cryptographic library for generation and validation of Proofs

https://pymerkle.readthedocs.io/en/latest/index.html?highlight=install#installation