Digital certification using Blockchain

Marulasidda Swamy K S and Naveenkumar Adi Raman

University of Montreal

Proof of Concept

<u>Issuing Phase:</u>

Get the source code and install Required packages

1. Run

```
git clone https://github.com/blockchain-certificates/cert-issuer.git && cd cert-issuer
```

2. Make below changes to fix compilation issues:

```
Go to : ~/.local/lib/python3.6/site-packages/merkletools-1.0.2.dist-info and edit METADATA and metadata.json
```

3. Downgrade jsonschema to the version 2.6

```
pip uninstall jsonschema
pip install jsonschema==2.6
```

4. Run

```
python setup.py experimental –blockchain=ethereum
python setup.py install
```

Create an Ethereum issuing address

1. In Ethereum a public/private key pair is the same across all test/main networks. So, we don't need to have a new pair if later we need to run on the main network.
2. Go to https://www.myetherwallet.com/

3. Download and save issuer credentials (Keystore file).

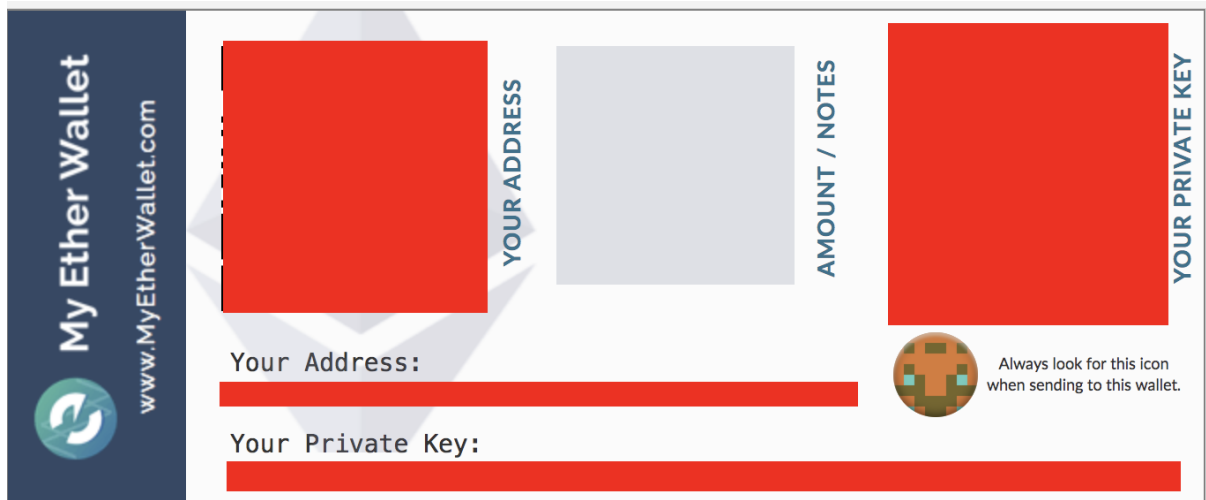# Save your Keystore File.

Download Keystore File (UTC / JSON)

Do not lose it! It cannot be recovered if you lose it.

**Do not share it!** Your funds will be stolen if you use this file on a malicious/phishing site.

Make a backup! Secure it like the millions of dollars it may one day be worth.

I understand. Continue.

4. We can print a paper wallet.

5. Your address can be known as your Public Key. It is what you share with people so they can send you Ether or Tokens.
6. You may use the Keystore file or your Private key to continue.
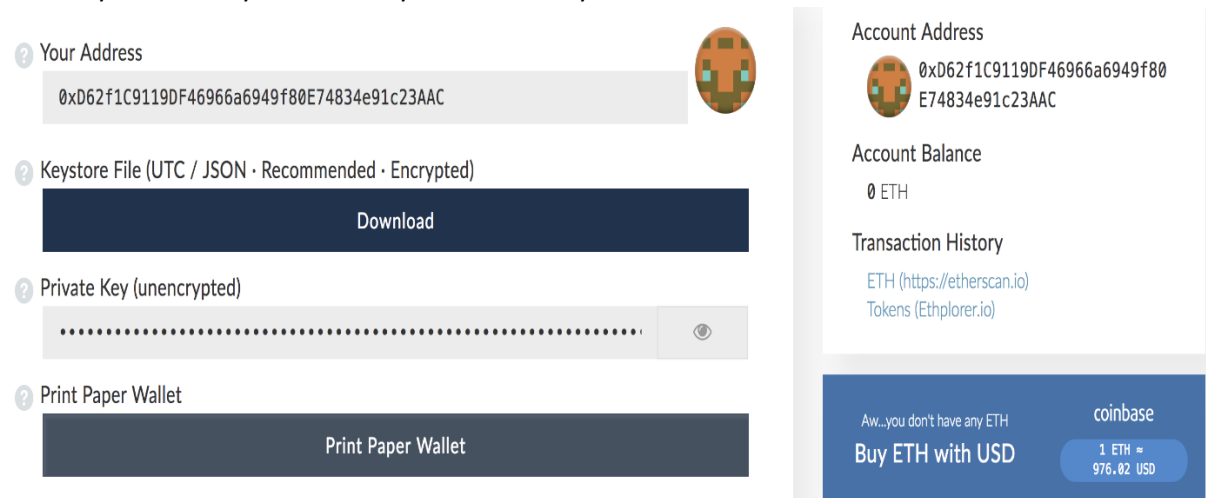
7. Before we can issue certificates, we need to have sufficient funds to cover the transaction fee. Why transaction fee? It is paid to miners for their work. We can get some free coins from the network. You can request test coins by searching for "TESTNET Faucet" and entering your issuing public address.
8. Go to http://faucet.ropsten.be:3001/. Enter your public key and click "Send me 3 test ether!".

9. Check the balance in your account.

Configuring cert-issuer

1. We need to save you the private key to somewhere. It is recommended to store the private key on a USB stick and unplug it when not used. But to save time plugging in and out, we will save the private key locally. For example, `/home/siddu/siddu-pk` folder and `pk` file.

2. Then, you should see a `conf_ethtest.ini` under the `cert-issuer` root folder. Open it using any of your favorite editors. Fill in the issuing_address, choose ethereum_ropsten as the blockchain, enter the usb_name and key_file as the private key storage location, and set the three dirs. Remember to rename the config to conf.ini.

Issuing a certificate

1. Let us use some sample certificates:

```
siddu@LAPTOP-T777GQUT:~/cert-issuer$
siddu@LAPTOP-T777GQUT:~/cert-issuer$
siddu@LAPTOP-T777GQUT:~/cert-issuer$ cp ./examples/data-testnet/unsigned_certificates/* ./data/unsigned_certificates/
```

2. Here you need to edit the unsigned certificates, replace "msBCHdwaQ7N2ypBYupkp6uNxtr9Pg76imj" with your public key.

3. Then, you need to find a place to host the issuer profile. From Open Badges, a Profile is a collection of information that describes the entity or organization using Open Badges. The main usage is to verify the cert is issued from a proved person. Since we are using our owned public key, we need to host the profile file by ourselves.

4. Download two sample files, one is issuer profile and another one is revocation list.
   a. https://www.blockcerts.org/samples/2.0/issuer-testnet.json
   b. https://www.blockcerts.org/samples/2.0/revocation-list-testnet.json

5. Then, open the two files, and you will see a compressed JSON. What you need to do is first replace "msBCHdwaQ7N2ypBYupkp6uNxtr9Pg76imj" with your public key in issuer-testnet.json file, and upload it to somewhere, like GitHub, where you need to get a stable link for direct download. Then, replace the links in the two JSON files with the correct one.

📓 creative-swamy / **blockchain_cert**

🔔 Watch ▾   0   ★ Star   0   🍴 Fork   0

‹› Code   ⓘ Issues 0   ⑂ Pull requests 0   🗒 Projects 0   📖 Wiki   📊 Insights   ⚙ Settings

blockchain certificates   Edit

Manage topics

ⓟ **5 commits**	⑂ **1 branch**	◇ **0 releases**	👥 **1 contributor**

Branch: master ▾   New pull request   Create new file   Upload files   Find File   Clone or download ▾

🟩 **creative-swamy** Update revocation-list.json   Latest commit d8e9c97 on Mar 21

| 📄 issuer.json | Update issuer.json | a month ago |
| 📄 revocation-list.json | Update revocation-list.json | a month ago |

🟩 **creative-swamy** Update issuer.json   b3f46c0 on Mar 21

1 contributor

21 lines (20 sloc) | 85.6 KB   Raw   Blame   History   🖥 ✏ 🗑

```
 1    {
 2       "@context": [
 3         "https://w3id.org/openbadges/v2",
 4         "https://w3id.org/blockcerts/v2"
 5       ],
 6       "type": "Profile",
 7       "id": "https://github.com/creative-swamy/blockchain_cert/blob/master/1ssuer.json",
 8       "name": "University of Learning",
 9       "url": "https://www.issuer.org",
10       "introductionURL": "https://www.issuer.org/intro/",
11       "publicKey": [
12         {
13           "id": "ecdsa-koblitz-pubkey:0x4e031f6f0e6879260301893d29207f54598f3f34",
14           "created": "2017-06-29T14:48:03.814936+00:00"
15         }
```

Branch: master ▾   **blockchain_cert** / revocation-list.json   Find file   Copy path

🟩 **creative-swamy** Update revocation-list.json   d8e9c97 on Mar 21

1 contributor

17 lines (16 sloc) | 533 Bytes   Raw   Blame   History   🖥 ✏ 🗑

```
 1    {
 2       "@context": "https://w3id.org/openbadges/v2",
 3       "id": "https://github.com/creative-swamy/blockchain_cert/blob/master/revocation-list.json",
 4       "type": "RevocationList",
 5       "issuer": "https://github.com/creative-swamy/blockchain_cert/blob/master/issuer.json",
 6       "revokedAssertions": [
 7         {
 8           "id": "urn:uuid:93019408-acd8-4420-be5e-0400d643954a",
 9           "revocationReason": "Honor code violation"
10         },
11         {
12           "id": "urn:uuid:eda7d784-c03b-40a2-ac10-4857e9627329",
13           "revocationReason": "Issued in error."
14         }
15       ]
16    }
```

6. Run `cert-issuer -c conf.ini` to see if everything is working.

7. Read the debug info, it is useful. You will find a string of txid, which is your transaction ID. Now you can check this transaction using Etherscan.
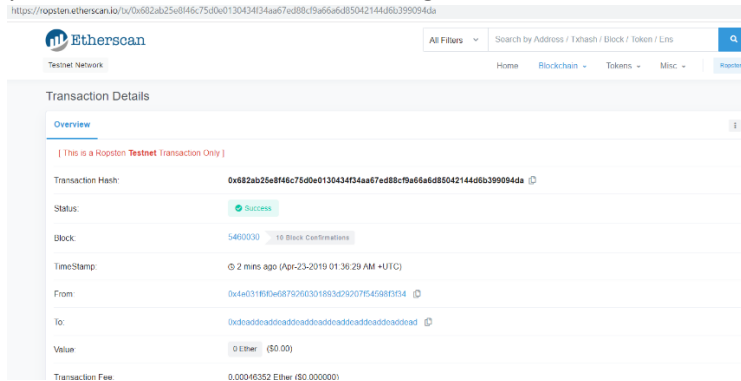
8. Now you get a blockchain-based certificate under data/blockchain_certificates.

Verification Phase

Traditional way of verifying the certificate using Blockcerts verifier code

1. Clone cert-verifier repo from GitHub:

```
git clone https://github.com/blockchain-certificates/cert-verifier.git
&& cd cert-verifier
```

2. Use certificates which are issued by cert-issuer (from certificate issuing step) and run :

3. If the certificate issued is genuine and validated, then all verification steps show the result as "passed" (As depicted in the above screenshot).

Observations made with the traditional way of verifying the certificate

1. Steps followed to complete verification of a given certificate:
   a. Check if the certificate has been modified/tampered.
   b. Check if the certificate has been expired.
   c. Check if the certificate has been revoked by the issuer.
   d. Check if the issued certificate is authentic and genuine.

2. The complete verification process is implemented in **checks.py** file

```
siddu@LAPTOP-T777GQUT:~/.local/lib/python3.6/site-packages/cert_verifier$ ls
__init__.py   __pycache__   checks.py   connectors.py   errors.py   verifier.py
```

3. The function which implements verification :

```python
def create_verification_steps(certificate_model, transaction_info, issuer_info, chain):
    steps = []

    v2ish = certificate_model.version == BlockcertVersion.V2 or certificate_model.version == BlockcertVersion.V2_ALPHA

    # embedded signature: V1.1. and V1.2 must have this
    if not v2ish:
        embedded_signature_group = create_embedded_signature_verification_group(certificate_model.signatures,
                                                                                transaction_info, chain)
        if not embedded_signature_group:
            raise InvalidCertificateError('Did not find signature verification info in certificate')
        steps.append(embedded_signature_group)

    # transaction-anchored data. All versions must have this. In V2 we add an extra check for unmapped fields
    detect_unmapped_fields = v2ish
    transaction_signature_group = create_anchored_data_verification_group(certificate_model.signatures,
                                                                          chain,
                                                                          transaction_info,
                                                                          detect_unmapped_fields)
    if not transaction_signature_group:
        raise InvalidCertificateError('Did not find transaction verification info in certificate')
    steps.append(transaction_signature_group)

    # expiration check. All versions have this as an option.
    expired_group = ExpiredChecker(certificate_model.expires)
    steps.append(VerificationGroup(steps=[expired_group],
                                   name='Checking certificate has not expired'))

    # revocation check. All versions have this
    revocation_group = create_revocation_verification_group(certificate_model, issuer_info, transaction_info)
    steps.append(revocation_group)

    # authenticity check
    if chain != Chain.mockchain and chain != Chain.bitcoin_regtest:
        key_map = {k.public_key: k for k in issuer_info.issuer_keys}
        authenticity_checker = AuthenticityChecker(transaction_info.signing_key, transaction_info.date_time_utc,
                                                   key_map)
        steps.append(VerificationGroup(steps=[authenticity_checker],
                                       name='Checking authenticity'))

    if chain == Chain.mockchain or chain == Chain.bitcoin_regtest:
```

4. Merkle root verification is also one of the steps involved in the verification process:

```python
class MerkleRootIntegrityChecker(VerificationCheck):
    def __init__(self, expected_merkle_root, actual_merkle_root):
        self.expected_merkle_root = expected_merkle_root
        self.actual_merkle_root = actual_merkle_root
    def do_execute(self):
        merkle_root_matches = hashes_match(self.expected_merkle_root,
                                           self.actual_merkle_root)
        return merkle_root_matches
```

Where in Merkle root present in the shared certificate is compared with the Merkle root present in the transaction details.

5. Merkle root reference in the certificate:

yyMvE8G7HQpCC8HY1dJvx5Pg7AcCyy5AgihvyDrbkkUgAb0gbwjG26rkijyY9NiETSuaADOawD3Az5siEtLwoFrS5i9BnYRdwxWiqr9ZdQA0ubfOcKY2hFFJI8HCzYEDw6p24aFA4f4JSA3Es1hv5A11EL3S3BPOkxQz1EjkDEs8JqfZxgb
dKxYCucK+9ZOFXDiqKxzYULCnoujtU3xDqTc7uBoXf/zYXjB3yrhowWYAbwuDVc+GVcYms0hM6AYH5sL5SlpqfeLJHgwVtHpFsAPdtHVD77HWutVR8UvfpPlP3IE93lFYsj1WjgIW0YbFe4RpkmQx99CZKR/ob5Cjc0SO0BsRMF5cueNANt
BRJspirF+i0Hnx1KUQc+lgRW0RgSOBDKh9mls80hwpy+zlMksXowxEHYa1CCREL/jQ5tagEd2tBVkI3li6dTXk9sFtJshjRFfzOkzc3wG48OAM3pDa3tfFFJda4rIE/01FH4KTJYjTflylRzMdZRxBw6iwHW4cUWbTrmzbWVVmaLGbxkzie
hff/bANjHvypdSJ6autnorcbC1Y1tsuhnv9HixIejaRT85oYzFBBsgQEZy6Oti9UzwrZZMFWXcOjz/jg/uPhVMw2Mc+monJwh4ki37Rn5kmwCz1FKxtVjueoJXDRxOpcfj26xn04CkWTPRSkK572R2CjZIJ6f/oOcMwJAcw+6JY9s5BMmF7wU
RRzfWgP8qrQnJuOiLu5z+6VvmrLPE/QBGLxZwAPNJO6aGWULHK3XY1Nct1qpfydDijaXE3s5Wk6mGTvFcuFVFWy6PHgRZK5hm0r4G7+X7JOzvxVZvAyNVXoxW0OjXaD5SqFEVEBLE2JSs79VtLvFBhFSqK19Akx1kxjWCWGoQIOpPC0tbM6
ZqlatWrVqlfqP9Q9WKn4v31JdhQAAAABJRU5ErkJggg=="}]}, "id": "urn:uuid:3bcla96a-3501-46ed-8f75-49612bbac257", "recipientProfile": {"publicKey":
"ecdsa-koblitz-pubkey:mtr98kany9GiXYNU74nEnfBCmaCr2FEImc", "name": "Eularia Landroth", "type": ["RecipientProfile", "Extension"]}, "signature": {"type": ["MerkleProof2017",
"Extension"], "merkleRoot": "5f22fb346f255986bc4322bc8c1aaf1c7d03adf6dc20db76984a6fb80c90012e", "targetHash": "23257af6f91fa03c547f24aba9aeed4016038ea745a48f8e327d997eec78a0ac",
"proof": [{"right": "1558d49c9bfcc26dcbcfd88a61e4ffdc64fDb97/399a55ieea62/cdd/De9b5c8"}], "anchors": [{"sourceId":
"0x30b79b7c482136150e74398d9d765addfe61e1e7c85264b78d2f596e52284666", "type": "ETHData", "chain": "ethereumRopsten"}]})

6. Merkle root reference in the blockchain transaction :

Input Data:

`0x5f22fb346f255986bc4322bc8c1aaf1c7d03adf6dc20db76984a6fb80c90012e`

View Input As ⌄

7. Hash of the Merkle root present in the certificate and the Blockchain transaction should match.
8. Please note that the Merkle root hash is shared in the certificate can be exploited by hackers. So, in our proposal, we are demonstrating about how to securely share Merkle root details to the verifier.
9. This way of sharing the Merkle root can facilitate implementing the version management for revoked certificates.

Securely Sharing Merkle root of the certificate to verifier - Proposed Work

1. The cert-issuer need to modify the issued certificate by changing the Merkle root entry in the certificate.
2. Merkle root entry is replaced with IPFS reference link, where the Merkle root of the certificate is saved in an encrypted format.
3. Steps to encrypt the Merkle root: (It must be done by cert-issuer. For POC, we have done it manually replacing the Merkle root with IPFS reference link).
   a. Follow this for installing IPFS on Ubuntu terminal: https://blog.siderus.io/how-to-get-ipfs-on-ubuntu-debian-linux-d7920c1a42b7
   b. Run IPFS daemon:

   c. Save Merkle root in a file and encrypt using recipient public key : (Please follow steps guided here to generate public and secure keys: https://medium.com/@mycoralhealth/learn-to-securely-share-files-on-the-blockchain-with-ipfs-219ee47df54c).
   d. When we encrypt using verifier public key, the only verifier will be able to decrypt the file and hence it is secure.

e. When the file containing Merkle root of the certificate is encrypted and uploaded to IPFS, a reference link is returned by IPFS.

```
siddu@LAPTOP-T777GQUT:~$ ipfs add tran_list
added QmU6SXLad4498YhvynvvJmSfvvfZhLcFg8Jn9VdttXvhJw tran_list
 65 B / ? [-------------------------------------=-----------
siddu@LAPTOP-T777GQUT:~$
```

f. Blockcerts code can only handle hexadecimal values and hence we need to convert the IPFS reference link to the hexadecimal form: (Sample code is given below)

```
siddu@LAPTOP-T777GQUT: ~

import binascii
x = b'QmU6SXLad4498YhvynvvJmSfvvfZhLcFg8Jn9VdttXvhJw'
x = binascii.hexlify(x)
y = str(x, 'ascii')
print(y)
~
```

```
siddu@LAPTOP-T777GQUT:~$ python convert_hex.py
516d553653584c61643434393835968676796e76764a6d53667676665a684c634667384a6e39566474745876684a77
```

g. Replace hexadecimal formatted IPFS reference link in the certificate :

"signatureLines": [{"jobTitle": "University Issuer", "name": "Your signature", "type": ["SignatureLine", "Extension"], "image":
"data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAJcAAABVBAMAAAC2kWv1AAAAMFBMVEX///8AAAC/v79/f39fX1+fn5/f398/Pz8fHx/Dw8NTU1Onp6cvLy93d3cnJydvb28isqigAAAACXBIWXMAAA7EAAAOxAGVKw4bAAADQE1
SQVRYhe2WS27bMBCGp5HIudsCRbasX8iqUJ3Y3hIO7HZJSJayKAqw8StLIYnTLpNFnCv0BLiNT9MLdEhJzkNSEkfaFNAPmKQl6tNwNDMkQKlatV4nyqtjsbPqWODwCmF+hSyIKmTZvELYvELYWXFbIIgJgaUZMlobZwfAiphwHZVnz6/NuPCJRq
yyMvE8G7HQpCC8HYIdJvx5Pg7AcCyy5AgihvyDrbkkUgAb0gbwjG26rkijyY9NiETSuaAD0awD3Az5siEtLwoFr85i9BnYRdwxWiqr9EdQA0ubf0cKY2hfFJI8HCzYEDw6p24aFA4f4JSA3Eslhv5Al1EL3S3BP0kxQz1EjkDEs8JqfIxgb62+
3KxYCucK+9ZOFXDiqKxzYULCnoujtU3xDqTc7uBoXf/zYXjB3yrhowWYAbwuDVc+GVcYms0hM6AYH5sL5SlpqfeLJHgwVtHpFsAPdtHVD77HWutVR80vfpF1F3IE931FYsj1WjgIW0YbFe4RpkmQx99CZKR/ob5Cjc0SO0BsRMF5cueNANtXTb
3RJspirF+i0Hnx1KUQc+LgRW0RgSOBDKh9mls80hwpy+z1MksXowxEHYa1CCREL/jQ5tagEd2tBVkI3li6dTXk9sFtJshjRPfzOkzc3wG480AM3pDa3tFFFJda4rIE/01FH4KTJVjtf1y1RzMdZRxBw6iwHW4cUwbTrmzbWvVmaLGbxkzieiGc
1ff/bANjHvypdbJ6autnorcbClYltsuhnvHixIejaRTS5oYzFBSzQZ2y60ti9UzwrZZMFWXcOjz/jg/uPhVMw2Mc+monJwh4ki37RnSkmwCz1FKxtVjueoJXDRxOpcfj26xn04CkWTPRZkK572R2CjZIJ6f/oOcMwJAcw+6JY9s5BMmF7wU9sL
1RzfWg98grQnJu0iLu5z+6VvmrLFE/QBGLxZwAPnJO6aGWULHK3XY1Nct1qpfydDijaXZ3s5Wk6mGTvFCuFVFWy6PHgREK5hm0r4G7+X7J0zvxVZvAyNVXOxW0OjXaD58gFEVEBLE2JSs79VtLvFBhFSqK19Akx1kxjWCWGoQIOpPC0tbM6gVc
2qlatWrVqlfqP9Q9Wkn4v31JdhQAAAABJRU5ErkJggg==}]}, "id": "urn:uuid:3bcla96a-3501-46ed-8f75-49612bbac257", "recipientProfile": {"publicKey":
"ecdsa-koblitz-pubkey:mtr98kanv9G1XYNU74pBnfBOmaCp2FSImc", "name": "Eularia Landroth", "type": ["RecipientProfile", "Extension"]}, "signature": {"type": ["MerkleProof2017",
"Extension"], "merkleRoot": "516d553653584c616434343938596876796e76764a6d53667676665a684c634667384a6e39566474745876684a77", "targetHash":
"23257af6f91fau3c547f24aba9aeed4016d3dea745a48f8e327a997eec78avac", "proof": [{"right": "1556d49c5b1cc2cdcbcfd88ab1e4f1f0cc4fbb97399a55leea627cdd7be9b5c8"}], "anchors": [{"sourceId"
: "0x30b79b7c482136150e74398d9d765addfe61ele7c85264b78d2f596e52284666", "type": "ETHData", "chain": "ethereumRopsten"}]}}

h. Issue the modified certificate to the student.
i. At the verifier side, verifier code needs to fetch the actual Merkle root from IPFS.
j. To fetch the actual hash of the Merkle root, verifier needs to convert the hexadecimal IPFS link to actual IPFS ink.

k. Using actual IPFS link, need to fetch the actual hash of the Merkle root. (**Main code: verifier.py)**

```python
def get_merklehash_from_ipfs(hexa_ipfs_link):
    asci_ipfs_link = binascii.unhexlify(hexa_ipfs_link)
    asci_ipfs_link = str(asci_ipfs_link, 'ascii')
    command = "ipfs "+"cat "+asci_ipfs_link
    output = subprocess.check_output(command, shell=True)
    actual_merkle_root = str(output, 'ascii')
    actual_merkle_root = actual_merkle_root[:-1]
    print("Actual merkle root ", actual_merkle_root)
    return actual_merkle_root

def verify_certificate_file(certificate_file_name, transaction_id=None, options={}):
    with open(certificate_file_name, 'rb') as cert_fp:
        certificate_bytes = cert_fp.read()
        certificate_json = json.loads(certificate_bytes.decode('utf-8'))
        certificate_json["signature"]["merkleRoot"]=  get_merklehash_from_ipfs(certificate_json["signature"]["merkleRoot"])
        certificate_model = to_certificate_model(certificate_json=certificate_json,
                                                 txid=transaction_id,
                                                 certificate_bytes=certificate_bytes)
        #print(certificate_json)
        result = verify_certificate(certificate_model, options)
    return result
```

l.   With the actual hash of the Merkle root inserted, the verification process will be executed, and the result should be a pass.

m.   Screen shot of the output :

Fs9XZSowXsS3DRgCEctfLmDshRFg3J5aQEz/ID8xx8xzbR1ZBjwb6ZDq5ux+L0CVcZXgTCbS2jVUStUZtfEj/Doo3YmLsrQFQIv1EXbqqolKG26JFm+qP1xdxz+50VRcrBJUXKm7U4tCzwN8RswvB9gi3YZDF+x4rZ/QM2QRA3iDWaQiYIBeOE5Wp/K60gao3afJKwA5Mv3Jcwdgss
rm1E93vEf8je4RDNRXit2vk3JRevC0NeJfw4/J7hA9lSfgLHD4hLaKmAIgrHBk/PyVQZmu4K+MaWkHUGpew3BZeeRpSQzA7Eb6oGteJ7xjJLVJ3h2RJfGda3ywighTCS7MChC0X2e7BO/Q+BBEOn5+pnur4AJ3eEdyFYvGQwEerjyv9R7SCqDVqq+ye2T5YIQuU0hr3WC1RVcDHyzZv
yHltLq9PeMJQ/H58SwmRQQqpg7LHfg4+Uup7dLn5zkdhUrlCOK3a55ACRh8bCHul+vGqjZigRq0Rdwp72K7D5Ns/t7BuUu6aVMDTuwQkKvu/Zbe1F036DM9/IN0O4+BqQG5QGkAQOlMFtw5bE5y43MJ+lbgWDs7hmwf1QnIk2yFaPdQavQ11Wa7ABESE//ZxecqEx+pKKP2Pr1x4SC9
R6VuvQG3FfyADRHw+VishRZQscTuK7MbuDnF4TZEaA9gt3NzwSpZ2qEwDVRFhQQsG5TCpO3C+120bQauM4uNLWK70i+oVba0QP/C5bdIAV6tLIt5p9Agz7w3xObGtWFnlzeWuAad3EiGFw843+dyn69UFjthSE1fl1tqvOt4S6fvlltt+IUv2uYy4vlbnideVkUWrh1rTkJYk/tEvla
f6yjO6GwVJh53Qbksh1vHrC6Nzb882+oTHvq9yvWdaAZFSGEfZyZqq4PCQPx5FpRucP7YnVwePxzJP4Jnul2xT2Xz5d7841iyoUYurOXG1bxPP+QO73Hk1Aq9R2jwS/6RPNzvZlz5PjfY/7ivP/DWq+07vm52UTqjJAjj88iwciMWTx8VMFT839hE73xyJ7NzhWQ+fOyYx9YnnfUcTF
01NWnJKaW5hH9TCUAsH/j+vLyzU7DoKa3p1sVwaON/2FP6OKjzxLOqjG51x2XgShY/XO+MmY9QP1D3CH3v1slyIMfOVZTgZaf0O44YdvU83hX7u3ZPyaz3UO95BM5ZaCyeszC5YDMc2Gy8a/H3REJ4D6TtGVn+Su1rb1jvfcv2lskf8N0JnYieFYr4Dtnl0nmn1F83oBwVK4aiJ+4o7
uPnQ0sLaHGdi33qbHoy8GNlvYuN6Dr+/sajQ8Br4kATcEcT/koRErcWJK1eHG1SnQ23RAyweBCfXOp2eYwtuRbA+FJJteKxly5B1FTtZ/gr7uUjz3IfzBkca34NLCuO0O3XibgGQKEH4KL4QC2IN7ZOt+CXxm2DuJVXd7r6ZEf4/cI3Sz+QoJ7cVG8b/wXhOs5Uatb5085i68k8scnV
8ntpWosx6ujaURxL/NKyb9baQtrSyj3/j2Jl8S1kgRJ7uxCN87i3taG0L9ktkbYir0Gftbhqkkk82xq5ivjTbUSsZ2cb4JPsU/Hv6k9FFBiC7aMDVRKgMNX5wMWGq+RyeIYKR90uzvydbxEaCzAYyYqOpSo1aGlI2WwdgsWCBYftdLXzhKxcGy12uvzRCW9gUkM/JbZFJLxtGqMZ74V
M1to2Rw2P42+bRsTmzbhqVr/a5jC3yz5HaN5asBmd3dlLG9cwCURzv4bjlh/AM95UJPdH3pAj7xDPOqK1yJEuR5Y+Su1vu8nvsRwOZOLK5Oo4YUBCWXGszpEatxbeny9tei0WBUlO1C/7jSpetY1Sk5+54B85dOCg7rAG/zqfII+zTOrgGHGkXyJL4EjYBjm2JTjpCIr1AcjphB0jGV
24tYarWd7FIbFD/uM9Sua2gnupbavON/o70Nh+FUX8hpdbez6JoxhfxfnK5zYfrqB801sf7NEupUVO/7NwJVfy2dut9aDTOp/X6kYSdUGlOB5FAkFGA24UQiO22zpcdjgv0FQznUoZ4Lhr1EimJeekqj/jJsHvb/aSOYNw8qQi19CLt6531RQGicQszjZ4Rjpd+j/BfXO/1RUV8U/ks
qwyQ3rAoHD3rXcedaWHj+R7Pv16W4aRZSo2aVnqQbGMr59u3jUu1dLVvgwW2aVTD3T5UquZ2MY/9G0YG/pD34ha4ioriseEI5shGZ60dTPxHVZcONzSwz0m2Mehr/ci8sD5Pp5n0h+pFjJD+ObzrXVcFuz8Yyv2S+Af0+8J0R4MKzXAKrfVVew33EOt4YldQdtsG0eykRi0Rklpa2g+
YmndEn/bm3ZVFPCjbXyF+zlS1h8P2A7uRWzgGIjoeoiLxsS01vPgD6/frYD5KKKDgk6wbVI/8YD5VILN1NQG7WnW2IRHuHgdktbpE9tgOcoIencfJTu0LAeWQWAd2yyBDOTVqydyQMQAq3wvTC+p2uUZzqUX8qyxZOkXaz7Bu7W7j1YgHZhu6EyD7wMx+AYlteFn7vHAqLyPSj7i/0t
JjdwdkLA3Y1WzPRSEVMQlO3IbnXV6Vy43xx76Xrs5SGhdq1JK8IT8T1DMs2r0zA4Zwpj79cXhphzujVlPd4jSQASru6FW3ld5c7rvlc9uejpRc4KIRKG8b3/Z0akcJz3bBFv8iFudMprbNCoQ4ZTMpzsGlK1Q2otlAjVoqSFlu66/ULJhMsvldKHtQOPLz1/qIqcHCUN22j8sPbPW7N
VVREYsr/UeMlP6N3bejm4rr3A+em+Lg/Xk9tgd1Voc16xDECKopfmgWUKOWQo2pan+WK8R35fT0i+Lp2Sgrszw8zdfKebW8xA7VofXlmIjF1bEjiAUuHKiKc3iNs9Z8VPg43mf2PWD7Ne+rLKOCZgA1ainUqh7Iy09lCQT7xk0EHnE7dygdkxdQMT3C+/HE1TF1rUfAZyyH30pj0tX/
Q03EU6RLqkhDjVribtg1ZFiW/9TGNF5hUNF2P4ub8N320Wp+KsRC0ihTo0bNtOYrF7vLHvfzb2lGB1NIEC4YkrgCJc/SaDePGjVqpkta3513mdnfoE7XXgY3D8pZapG2/w9KMUT7yswRhQAAAABJRU5ErkJggg==', 'id': 'urn:uuid:82a4c9f2-3588-457b-80ea-da69557d
b8fc', 'description': 'Lorem ipsum dolor sit amet, mei docendi concludaturque ad, cu nec partem graece. Est aperiam consetetur cu, expetenda moderatius neglegentur ei nam, suas dolor laudem eam an.', 'signaturel
ines': [{'jobTitle': 'University Issuer', 'name': 'Your signature', 'type': ['SignatureLine', 'Extension'], 'image': 'data:image/png;base64, iVBORw0KGgoAAAANSUhEUgAAAJcAAABVBAMAAAC2kWv1AAAAMFBMVEX///8AAAC/v79/f39
fX1+fn5/f398/Pz8FHx/Dw8NTU1Onp6cvLy93d3cnJydvb281sqiqAAAACXBIWXMAAA7EAAAOxAGVKw4bAAAADQElEQVRYhe2WS27hMBCGp5RIudsCRbasX8iqUJ3Y3h1O7HZJSJayKAqw8StLIYnTLpNFnCv0BL1NT9MLdEhJzkNSEkfaFMAPmKQl6tNwNDMkQK1atV4nyqtjsbPqWD
DwCmF+h5yIKmTZvELYvEIWXFbIIgJgaUZMlobZwfAiphwHZVnz6/NuPCJRqyyMvE8G7HQpCC8HY1dJvx5Pg7AcCyy5AgihvyDrbkkUqAb0gbwjG26rkijyY9NiETSuaADOawD3Az5siEtLwoFrS519BnYRdwxWiqr9ZdQA0ubf0cKY2hFFJI8HCzYEDw6p24aFA4f4JSA3Zslhv5AlT
EL3S3BPOkxQz1EjkDZs8JqfZxgb62+dKxYCucK+9Z0FXDiqKxzYULCnoujtU3xDgTc7uBoXf/zYXjB3yrhowWYAbwuDVc+GVcYms0hM6AYH5sL5SlpqfeLJHgwWtHpFsAPdtHVD77HWutVR8UvfpP1P3IE93IFYsjlWjglW0YbFe4RpkmQx99CZXR/ob5Cjc0SO0BsRMF5cueNANtX7
bBRJspirF+i0Hnx1KUQc+lgRWORgSOBDKh9mls80hwpy+zlMksXowxEHYa1CCREL/jQ5tagEd2tBVkI3li6dTXk9sFtJshjRPfzOkzc3wG480AM3pDa3tFFFJda4rIE/01FH4KTJYjTf1y1RzMdZRxBw6iwHW4cUwbTrmzbWvVmaLGbxkzieiGchff/bANjHvypdSJ6autnorcbC1Y/
tsuhnvHixIejaRTS5oYzFBSzQZZy6Oti9UzwrZZMFWXcOjz/jg/uPhVMw2Mc+monJwh4ki37RnSkmwCz1FXxtVjueoJXDRxOpcfj26xn04CkWTPRZKK572R2CjZIJ6f/oOcMwJAcw+6JY9s5BMmF7wU9sLRRzfWgPBqrQnJuOiLu5z+6VvmrLPE/QBGLxZwAPNJO6aGWULHK3XY1Nct
lqpfydDijaXZ3sSWk6mGTvFcuFvFWy6PHgRZK5hm0r4G7+X7JOzvxVZvAyNVXOxW0OjXaD5SqFEVEBLE2JSs79VtLvFBhFSqK19Akx1kxjWCWGoQIOpPC0tbM6gVcZq1atWrVq1fqP9Q9Wkn4v3lJdhQAAAABJRU5ErkJggg==']}, 'id': 'urn:uuid:3bc1a96a-3501-46ed-
8f75-49612bbac257', 'recipientProfile': {'publickey': 'ecdsa-koblitz-pubkey:mtr98kamv9G1XYNU7ApRnfBQmaCp2FZLmc', 'name': 'Eularia Landroth', 'type': ['RecipientProfile', 'Extension']}, 'signature': {'type': ['Me
rkleProof2017', 'Extension'], merkleRoot: '5f22fb346f255986bc4322bc8c1aaf1c7d03adf6dc20db76984a6fb80c90012e' 'targetHash': '23257af6f91fa03c547f24aba9aeed4016038ea745a48f8e327d997eec78a0ac', 'proof': [{'right
': '1558d49c9bfcc26dcbcfd88a81ed4fdc04fbb9759563516ea62fcab7be905ca'}], 'anchors': [{'sourceId': '0x360f907c4b2136150e74398d9d765addfe61e1e7c85264b78d2f596e52284666', 'type': 'ETHData', 'chain': 'ethereumRopste
n'}]}}
Checking certificate has not been tampered with,passed
Checking certificate has not expired,passed
Checking not revoked by issuer,passed
Checking authenticity,passed
Validation,passed
[{'name': 'Checking certificate has not been tampered with', 'status': 'passed'}, {'name': 'Checking certificate has not expired', 'status': 'passed'}, {'name': 'Checking not revoked by issuer', 'status': 'passe
d'}, {'name': 'Checking authenticity', 'status': 'passed'}, {'name': 'Validation', 'status': 'passed'}]
siddu@LAPTOP-T777GQUT:~/cert-verifier/cert_verifier$

n.   We can push all confidential information which is kept in the certificate(ex: Merkle root and transaction ID) to IPFS and let the verifier fetch the details from IPFS securely.

o.   We will be sharing verifier.py (the only changed file and need to be replaced with actual verifier.py from the repository.