Blockchain Cryptography 2

Exercise

In this exercise, you will write code to play with popular **cryptographic algorithms** using crypto libraries from various programming languages. You will write code to sign **Ethereum and Bitcoin** messages, derive **blockchain addresses**, and more.

Currently supported languages:

- **Python**
- **JavaScript**

You may complete in a different language of your choice (C#, Java), but we don't have preconfigured environments for those languages.

A **preconfigured project** can be found in the resources folder; you may use that as a starting point for the supported languages.

1. Ethereum Signature Creator

Write a program to calculate an **Ethereum signature** by given **message** and **private key**.

Input: 256-bit private key + input text message.
Output: signature + message.

Refer to the provided resources for sample inputs and outputs.

Suggested Python library: **eth_keys**
Suggested JavaScript library: **eth-crypto**

2. Ethereum Signature to Address

Write a program to find the **signer's Ethereum address** by given **message** + **Ethereum signature**.

Input: message + signature
Output: address

Refer to the provided resources for sample inputs and outputs.

Suggested Python library: **eth_keys**
Suggested JavaScript library: **eth-crypto**

3. Ethereum Signature Verifier

Write a program to **verify** the **Ethereum signature** of given **message** by given Ethereum **address**.

Input: message + signature + address
Output: valid / invalid.

Refer to the provided resources for sample inputs and outputs.

Suggested Python library: **eth_keys**
Suggested JavaScript library: **eth-crypto**

4. Private Key to Bitcoin Address

Write a program to generate a **Bitcoin address** by given **Bitcoin private key** (WIF-encoded).

Input: BTC Private Key
Output: address

Refer to the provided resources for sample inputs and outputs.

Suggested Python library: **bitcoin**
Suggested JavaScript library: **bitcoinjs-lib**

5. (Optional) Merkle Tree

Required Files

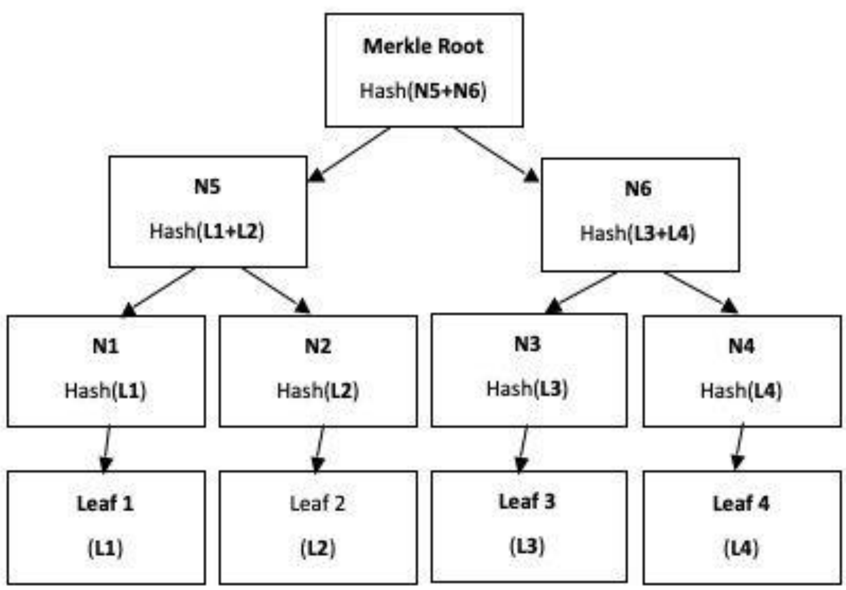
- **merkletree.py – The code template with TODOs that you need to complete**
- **merkletree-test.py – Tests that you may run to check your solution**
- **merkleproof-test.py – Tests that you may run to check your solution**

Complete the **implementation** of a **Merkle Tree** using **Python** (you can use **other** languages too). You will be **provided** with a **merkletree.py (Python Class)** and **two test files** with which you can **test** the newly implemented **functionality** of the tree. Some of the **methods** in the class will be **implemented**. Your job is to **try implementing** the **build_root method by yourself** and the **request_proof method** by **following** the **step by step tutorial** below.

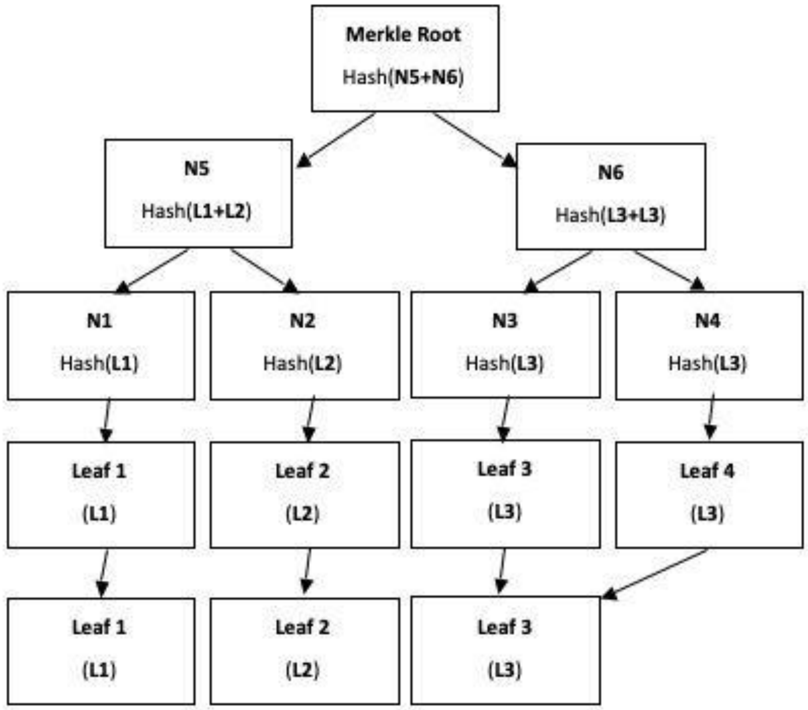
After completing the exercise, you will be **able to use** it in your **team project (optional).** The **test files** you are given are to **test** the two functionalities that you have to implement: **merkletree-test.py** and **merkleproof-test.py**.

Merkle Tree Specification

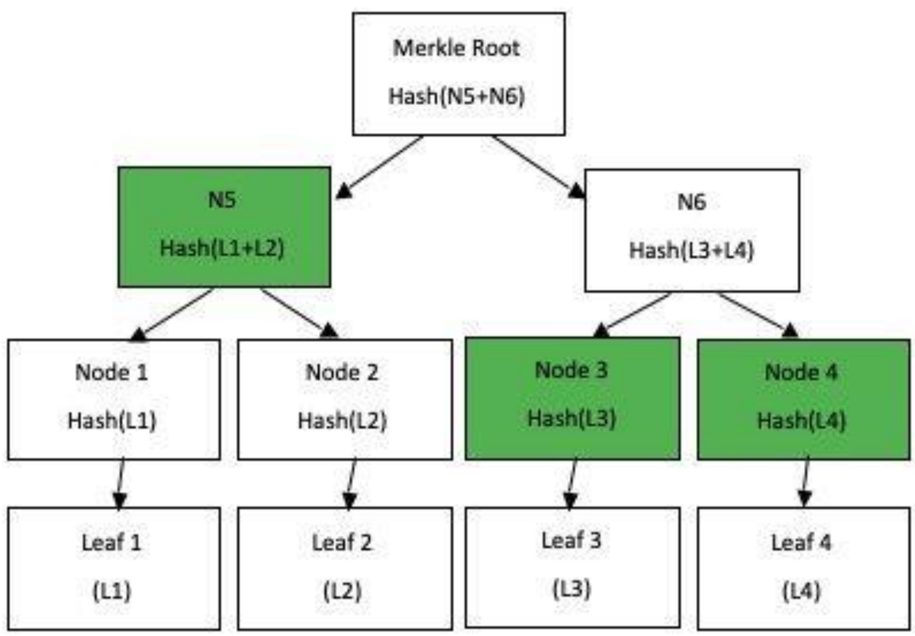
1. **Building** a Merkle Tree
    a. **Even** number of **elements (Hash can be any cryptographically secure hash function)**



2. **Odd number of elements**



3. **Building a Merkle Proof**
    a. **Verify that L3 was indeed inside the tree and took part in the construction of the Root Hash**
    b. **The nodes that are colored in green are required for the proof**



Request Proof Solution (build_root should be implemented)

We are going to use a **modified** version of the **Depth-First-Search** Algorithm

1. Write the **pre-conditions** of the **algorithm**
   1.1. **Hash** the **passed in** value using **self.digest** delegate since the values in the leaves are **not kept** in plaintext
   1.2. If the hash of the value that was passed in is not contained in the tree, raise an Exception.
   1.3. Create the **__build_valid_proof** method that we will use to implement the algorithm for the proof
   1.4. Pass the **root**, the **hashed value** and an empty list to the **__build_valid_proof** method
   1.5. Add the value itself as a part of the proof

```
def request_proof(self, value):
    hashed_value = self.digest(value)

    if self.__find(self.root, hashed_value) is None:
        raise Exception('This argument is not contained in the tree. Therefore is not part of the root')

    proof = []
    self.__build_valid_proof(self.root, hashed_value, proof)

    if len(proof) != 0:
        proof.insert(0, (0 if proof[1][0] else 1, hashed_value))

    return proof
```

2. Implement the **__build_valid_proof method**

```python
def __build_valid_proof(self, node, value, proof_list):

    if node is None:
        return False

    if node.value == value:
        return True

    found_left = self.__build_valid_proof(node.left, value, proof_list)
    found_right = self.__build_valid_proof(node.right, value, proof_list)

    if not found_left and not found_right:
        return False

    if found_left and found_right and (0, node.left.value) in proof_list:
        return False

    n = (0, node.right.value) if found_left else (1, node.left.value)

    proof_list.append(n)

    return True
```

Testing your solution

After **completing** the **exercise** you can **test** your **solutions** with the **provided** python **test files**.

```
merkletree-exercise>python merkleproof-test.py
```

The **following** output signalizes that **all the tests passed**.

```
test_merkle_should_contain_items (__main__.MerkleProofTest) ... ok
test_merkle_should_return_correct_proof_on_edge_case (__main__.MerkleProofTest) ... ok
test_merkle_should_return_correct_proof_when_leaf_is_left_child (__main__.MerkleProofTest) ... ok
test_merkle_should_return_correct_proof_when_leaf_is_right_child (__main__.MerkleProofTest) ... ok
test_merkle_should_throw_on_requesting_proof_for_non_existing_element (__main__.MerkleProofTest) ... expected failure
```

```
merkletree-exercise>python merkletree-test.py
```

The **following** output indicates that **all the tests passed**.

```
test_merkle_no_digest (__main__.MerkleTest) ... ok
test_merkle_with_cryptographic_digest (__main__.MerkleTest) ... ok
test_unable_to_build_from_empty_collection (__main__.MerkleTest) ... expected failure
```

What to Submit?

Create a ZIP file (e.g. **your-name-blockchain-cryptography-exercise.zip**) holding your source code for all problems:

Submit your ZIP file as homework at the course Web site.