Current layout: 1 Panel with tree

**1. Creating Merkle Proofs**

**Overview**

In this assignment, you’ll use Python to generate a Merkle Tree proof of inclusion, and submit the proof to an on-chain contract for verification.

The Merkle Tree in question has213=8192leaves, and the leaves are the first 8192 prime numbers. So the leaves are 2,3,5,7,11,13,17,19,…,84017

With knowledge of the leaves, you should be able to create a Merkle Proof of inclusion for *any* leaf in the tree. When you create a proof of inclusion, the contract assigns that leaf to you. The contract will *not* accept a proof for a leaf that has already been assigned to someone else, so you must find an unclaimed leaf when you make your proof.

**Assignment**

We’ve deployed [a contract](https://testnet.bscscan.com/address/0xaA7CAaDA823300D18D3c43f65569a47e78220073#code) to the BNB testnet (run by Binance). This is the same contract you’ve connected to in Module 1 (“Connecting to the Blockchain”) and Module 2 (“Reading Ethereum Blocks and Contracts”). The difference is that now you’ll have to *write* to the contract, not just read from it.

You’ll have to call the “submit” function on the contract and provide a merkle proof as well as the leaf, whose inclusion you’re proving. You can do this any way you’d like – I would recommend doing it python following the steps outlined in the “submitProof.py” starter code though.

Our autograder will check that you have indeed claimed a prime from the contract. In order to do that, we’ll need the address you used to claim the prime. If, however, we just asked you to provide an address, you could find an address that had already claimed a prime and forgo generating a Merkle Proof. So, our autograder is going to ask you to *prove* that you control the address in question by asking you to [sign a challenge message](https://web3py.readthedocs.io/en/stable/web3.eth.account.html#sign-a-message) using the private key corresponding to that address.

So your assignment has two parts

1. Claim a prime from [the contract](https://testnet.bscscan.com/address/0xaA7CAaDA823300D18D3c43f65569a47e78220073#code) by submitting a Merkle proof. You can do this any way you like, but if you choose to do it using python (recommended), the file [submitProof.py](https://shampoopoint-panamaclassic.codio.io/submitProof.py) has some helper functions and a method named “merkle\_assignment” that will guide you.
2. Finish the sign\_challenge function in [submitProof.py](https://shampoopoint-panamaclassic.codio.io/submitProof.py). This function takes a challenge string and signs that challenge using **the private key corresponding to the address that claimed the prime**. The function should return the address, and the signature (in hexadecimal).

Our autograder will check the signature (to make sure you have control of the private key), then check the on-chain contract to make sure the address you provided has claimed a prime. If you claimed a prime without using the starter code provided you will still need to complete the “sign\_challenge” method so that the grader can verify you claimed a prime

Since claiming a prime requires writing to the blockchain, you will need to claim funds from the [BNB testnet faucet](https://www.bnbchain.org/en/testnet-faucet) (if you have not done so already)

Our contract uses the [verify](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/MerkleProof.sol#L32) function provided by openZeppelin, so if you have any uncertainty about how to format your proof, you can look at how the proof will be verified.

The [openZeppelin Merkle Tree Validator sorts every pair before hashing](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/MerkleProof.sol" \l "L217).

When you generate a Merkle Proof, you **must** sort pairs before hashing otherwise your proof will not validate.

If you are trying to submit a proof via python, and you have an integer (a prime), you will need to convert the integer to bytes before submitting it to the contract.

The way to do this is using int.to\_bytes()

prime = 7

pb = int.to\_bytes(prime,'big')

Calling bytes(prime) will **not** cast prime to bytes, but instead give you a string of 0 bytes of length prime.

Mark as CompletedBack to dashboard