

# ZKU.ONE Week1 Assignment

---

Email – stanleychiu@protonmail.com

Discord – HKerStanley#4125

GitHub – <https://github.com/HKerStanley/zk-uni> (./asset/week\_1)

## Question 1 - Intro to circom

1. Takes a 4 leaves input and get the Merkle Root

`get_merkle_root.circom`

```
pragma circom 2.0.0;

include "mimcsponge.circom";

template GetMerkleRoot(nLeaves) {
    signal input leaves[nLeaves];
    signal output root;

    var nRuns = nLeaves / 2;

    component hashFunction;
    component hashFunctions[2];

    if (nRuns == 1) {
        hashFunction = MiMCSponge(2, 220, 1);
        hashFunction.k <== 0;
        hashFunction.ins[0] <== leaves[0];
        hashFunction.ins[1] <== leaves[1];
        root <== hashFunction.outs[0];
    } else {
        hashFunction = MiMCSponge(2, 220, 1);
        hashFunctions[0] = GetMerkleRoot(nRuns);
        hashFunctions[1] = GetMerkleRoot(nRuns);
        for (var i = 0; i < nRuns; i++) {
            hashFunctions[0].leaves[i] <== leaves[i];
            hashFunctions[1].leaves[i] <== leaves[nRuns + i];
        }
        hashFunction.k <== 0;
        hashFunction.ins[0] <== hashFunctions[0].root;
        hashFunction.ins[1] <== hashFunctions[1].root;
        root <== hashFunction.outs[0];
    }
}

component main {public [leaves]} = GetMerkleRoot(4);
```

public.json

```
[
  "6464294476958346139385024074008223400825166653076969388043746597957512245037",
  "1",
  "2",
  "3",
  "4"
]
```

Verified with the deployed Verifier.sol on RemixIDE

The screenshot shows the Remix IDE interface with the following components:

- Left Sidebar:**
  - ENVIRONMENT:** JavaScript VM (London)
  - ACCOUNT:** 0x5B3...eddC4 (99.9999999)
  - GAS LIMIT:** 3000000
  - VALUE:** 0 Wei
  - CONTRACT:** Verifier - contracts/Verifier.sol
  - Deploy:** Button to deploy the contract.
  - Transactions recorded:** 2
  - Deployed Contracts:** List of deployed contracts.
  - PAIRING AT 0XD91...39138 (MEMORY):** Section for pairing data.
  - Low level interactions:** Section for low-level interactions.
  - VERIFIER AT 0XD8B...33FA8 (MEMORY):** Section for verifier data.
  - verifyProof:** Function call with arguments [0x1317576f008ad9e75dc].
  - Low level interactions:** Section for low-level interactions.
- Main Editor:**

```
178 function verifyingKey() internal pure returns (VerifyingKey memory vk) {
179   vk.alfa1 = Pairing.G1Point(
180     8245231266457507221989169579407850281531147643213534261270460004267952729,
181     8016430114778418348769634463517984077561117844758492762176729191043249420475
182   );
183
184   vk.beta2 = Pairing.G2Point(
185     [14386746519434246450197825466214347848998539513867728267747305141090851650570,
186       16272461181845294662284808858574157345246108516039957958509969561551233412141],
187     [7685287379219825389121906653617196692236838132175947833147474372154020942558,
188       17135981744473827253989378357729921957881506217486675104324024264800249406103]
189   );
190
191   vk.gamma2 = Pairing.G2Point(
192     [11559732032986387107991004021392285783925812861821192530917403151452391805634,
193       10857046999023057135944570762232829481370756359578518086990519993285655852781],
194     [4082367875863433681332203403145435568316851327593401208105741076214120093531,
195       8495653923123431417604973247489272438418198587263600148770280649306958101930]
196   );
197
198   vk.delta2 = Pairing.G2Point(
199     [1987633809733942317467338198488149719892402784840942014867927013989626409440,
200       4111493926412865242697224780070366072184946255042330488826587350573741365464],
201     [21837171612253186382055484247899287684649080343581611413862152916269426397693,
202       1823942251854120440171823499908981388888001994156122439310614336751985087178]
203   );
204
205   vk.IC = new Pairing.G1Point[](6);
206
207   vk.IC[0] = Pairing.G1Point(
208     11764720794985328700453991122047826734113924101414082614201655027781509204964,
209     2094067015266290944855310448175638267590115764121736341322750267893151566541
210   );
211
212   vk.IC[1] = Pairing.G1Point(
213     8466676841764219155183539070177762756714018649936801662862813130317087592775,
214     20292593698397327126650956744268930440207784698884707238607652119866733969822
215   );
216
217   vk.IC[2] = Pairing.G1Point(
218     20972098420879516186120791348088274375476852167752961140338317983286079971585,
219     6976955394903904903718027841068324827111741594412692909166618661568235610278
220   );
221
222   vk.IC[3] = Pairing.G1Point(
223     182539732819700208822010510734120608506206544929712308692903243087221377663,
224     16168058237293039964984069051423718190363287658562572418498450416866668778027
225   );
226
227   vk.IC[4] = Pairing.G1Point(
228     10786195391518384281236295461480510697688221444858477853238501298475306449623,
229     20347646584028309633831503250873316627520075557493775647977618165868576840306
230   );
231
232   vk.IC[5] = Pairing.G1Point(
233     1489613389485875734505347210472027004323845440716897863170922571655954641587,
234     1109135231990809525327231982982336295505824307469685370306928836413659893813
235   );
236
237   }
238
239   function verify(uint[] memory input, Proof memory proof) internal view returns (uint) {
240     uint256 snark_scalar_field = 2188824287183927522246405745257275088548364400416034343698204186575808495617;
241     VerifyingKey memory vk = verifyingKey();
242     require(input.length + 1 == vk.IC.length, "verifier-bad-input");
243     // Compute the linear combination vk_x
244     Pairing.G1Point memory vk_x = Pairing.G1Point(0, 0);
245     for (uint i = 0; i < input.length; i++) {
246       require(input[i] < snark_scalar_field, "verifier-gte-snark-scalar-field");
247       vk_x = Pairing.addition(vk_x, Pairing.scalar_mul(vk.IC[i + 1], input[i]));
248     }
249     vk_x = Pairing.addition(vk_x, vk.IC[0]);
250     if (!Pairing.pairingProd4(
251       Pairing.negate(proof.A), proof.B,
252       vk.alfa1, vk.beta2,
253       vk_x, vk.gamma2,
254       proof.C, vk.delta2
255     )) return 0;
256     return 1;
257   }
258
259   /// @return r: bool true if proof is valid
```
- Right Sidebar:**
  - Transactions:** List of transactions.
  - Deployed Contracts:** List of deployed contracts.



```
] "8"
```

- Yes, it allows verifier to verify data in the Merkle Tree without knowing the actual input. Publicly verifiable smart contract cannot be used to construct the merkle tree as all the inputs will be public. Tornado Cash is using Merkle Tree technology, depositors use a random nullifier and their secret to compute a hash value stored in the Tornado Cash Merkle Tree, and when withdrawal they only have to provide that hash and nullifier for verification, no identity will be stored anywhere.
- Thanks to the documentation of circom and snarkjs, each steps are well explained and I am able to put all steps to `execute.sh` for much faster and easier execution.

## Question 2 - Minting an NFT and committing the mint data to a Merkle Tree

NFT Contract - [https://github.com/HKerStanley/zk-uni/blob/main/contracts/1\\_MerkleNFT.sol](https://github.com/HKerStanley/zk-uni/blob/main/contracts/1_MerkleNFT.sol)

**DEPLOY & RUN TRANSACTIONS**

**ENVIRONMENT**  
JavaScript VM (London)

**ACCOUNT**  
0x5B3...eddC4 (99.9999999%)

**GAS LIMIT**  
3000000

**VALUE**  
0 Wei

**CONTRACT**  
MerkleNFT - contracts/1\_MerkleNFT.sol

**Deploy**

**Transactions recorded**  
All transactions (deployed contracts and function executions) in this environment can be saved and replayed in another environment. e.g Transactions created in Javascript VM can be replayed in the Injected Web3.

**Deployed Contracts**

**MERKLE NFT AT 0x7EF...8CB47 (MEMO)**

- approve address to, uint256 tokenId
- mint 0x5B38Da6a701c56854dCfC8B75F56beddC4
- safeTransferFr... address from, address to, uint256 tokenId
- safeTransferFr... address from, address to, uint256 tokenId
- setApprovalFo... address operator, bool approve
- transferFrom address from, address to, uint256 tokenId
- balanceOf address owner
- getApproved uint256 tokenId
- getLeaveValue uint256 index
- getMerkleRoot
- getMerkleTree... uint256 index
- isApprovedFor... address owner, address operator
- name
- ownerOf uint256 tokenId

```

27
28 // An array to store the leaves
29 bytes32[] private _leaves = new bytes32[](_totalSupply);
30
31 // Stores the current Merkle Root
32 bytes32 private _merkleRoot;
33
34 constructor() ERC721("MerkleNFT", "MNFT") {}
35
36 /**
37  * @dev Mint an NFT to any address
38  * @param to receiver address of the NFT
39  */
40 function mint(address to) public returns (uint256) {
41     require(to != address(0), "Receiver address is 0");
42     require(_tokenIds.current() < _totalSupply, "Sold out");
43 }

```

**Transaction Details:**

- from: 0x5B38Da6a701c56854dCfC8B75F56beddC4
- to: MerkleNFT.mint(address) 0x7EF2e0048f5bAe046f6BF797943daF4ED8CB47
- gas: 80000000 gas
- transaction cost: 387527 gas
- execution cost: 387527 gas
- hash: 0x79d40880559ee2c3dbe4b5ba300978c749335074a7410ca980cd69a39ff95ca
- input: 0x6a6...eddC4
- decoded input: {"address to": "0x5B38Da6a701c56854dCfC8B75F56beddC4"}
- decoded output: {"0": "uint256: 0"}
- logs: [{"from": "0x7EF2e0048f5bAe046f6BF797943daF4ED8CB47", "topic": "0xddf25ad1be2c89669c2b0888fc378daa952ba7f1e3c4a11628f55a4df523b3ef", "event": "Transfer", "args": {"0": "0x00", "1": "0x5B38Da6a701c56854dCfC8B75F56beddC4", "2": "0", "from": "0x00", "to": "0x5B38Da6a701c56854dCfC8B75F56beddC4", "tokenId": "0"}}]
- val: 0 wei

transact to MerkleNFT.mint pending ...

**[vm] from: 0x5B3...eddC4 to: MerkleNFT.mint(address) 0x7EF...8CB47 value: 0 wei data: 0x6a6...eddC4 logs: 1 hash: 0x97a...c5254**

**status** true Transaction mined and execution succeed

**transaction hash** 0x97a7b6f10cf1926eaae7458265f7af547e88ee6ad9b903e97f5c9a92aac5254

**from** 0x5B38Da6a701c56854dCfC8B75F56beddC4

**to** MerkleNFT.mint(address) 0x7EF2e0048f5bAe046f6BF797943daF4ED8CB47

**gas** 80000000 gas

**transaction cost** 189647 gas

**execution cost** 189647 gas

**hash** 0x97a7b6f10cf1926eaae7458265f7af547e88ee6ad9b903e97f5c9a92aac5254

**input** 0x6a6...eddC4

The screenshot displays a blockchain explorer interface with a sidebar on the left and a main content area on the right. The sidebar includes sections for 'supportsInterf...', 'symbol', 'tokenURI', and 'Low level interactions' with a 'Transact' button. The main area shows transaction details for a 'MerkleNFT.mint' operation. It includes a status bar indicating success, a transaction hash, and various fields like 'from', 'to', 'gas', 'transaction cost', 'execution cost', 'hash', 'input', 'decoded input', 'decoded output', and 'logs'. The 'decoded input' and 'decoded output' are shown in JSON format, and the 'logs' section contains a detailed log entry for the 'Transfer' event.

```

{
  "address to": "0x5B38D6a701c568545dCfcB03FcB875f56beddC4"
}

{
  "0": "uint256: 1"
}

[
  {
    "from": "0x7Ef2e0048f5bAe0e046f68F797943daf4E08C847",
    "topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
    "event": "Transfer",
    "args": {
      "0": "0x0000000000000000000000000000000000000000000000000000000000000000",
      "1": "0x5B38D6a701c568545dCfcB03FcB875f56beddC4",
      "2": "1",
      "from": "0x0000000000000000000000000000000000000000000000000000000000000000",
      "to": "0x5B38D6a701c568545dCfcB03FcB875f56beddC4",
      "tokenId": "1"
    }
  }
]

```

val 0 wei

transact to MerkleNFT.mint pending ...

[vm] from: 0x5B3...eddC4 to: MerkleNFT.mint(address) 0x7Ef...8CB47 value: 0 wei data: 0x6a6...eddC4 logs: 1 hash: 0xca5...28a35

status true Transaction mined and execution succeed

transaction hash 0xca53178f8f3ba534d9821406f84fb52ead0bf9943721c625238883eff328a35

from 0x5B38D6a701c568545dCfcB03FcB875f56beddC4

to MerkleNFT.mint(address) 0x7Ef2e0048f5bAe0e046f68F797943daf4E08C847

gas 80000000 gas

transaction cost 189647 gas

execution cost 189647 gas

hash 0xca53178f8f3ba534d9821406f84fb52ead0bf9943721c625238883eff328a35

input 0x6a6...eddC4

decoded input { "address to": "0x5B38D6a701c568545dCfcB03FcB875f56beddC4" }

decoded output { "0": "uint256: 2" }

logs [ { "from": "0x7Ef2e0048f5bAe0e046f68F797943daf4E08C847", "topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef", "event": "Transfer", "args": { "0": "0x00", "1": "0x5B38D6a701c568545dCfcB03FcB875f56beddC4", ... } } ]

### Question 3 - Understanding and generating ideas about ZK technologies

1. SNARKs comes out earlier than STARKs so there are better and more mature support from the community. As STARKs relies on hash function it requires a much larger proof size and hence cost more gas. But that also makes STARKs secure even after quantum computing available and STARKs does not need a Trusted setup.
2. Groth16 requires a per circuit trusted setup, meaning that any changes to the circuit requires a new trusted setup. On the other hand PLONK has a secure Multi Party Computation Ceremony, providing a universal trusted setup that can be independent to the circuit.
3. Inspired by the Merkle Tree technology, we can take unique token id of NFTs as input and "hash" a new NFTs from them, its like breeding but the output will be verifiable yet no need to its reveal parents.
4. I am thinking if ZK can improve the communication or chain of command in a DAO. Think of a DAO where members are all truly anonymous, they can use ZK to proof that they have the power to request others on some task, while the receiver of the request can verify the request is valid instead of some random outsider.