

ZKU Assignment 1

Name: Shyam Patel

Discord: L_guy#4297

Email: shyampkira@gmail.com

GitHub Repo: [[Link](#)]

Question:1 Intro to circom

1. Construct circuit for merkleTree

merkleProof.circom: Link to GitHub [[merkleProof.circom](#)]

```
pragma circom 2.0.0;
include "node_modules/circomlib/circuits/mimc.circom";
// Helper template that computes hashes of the next tree layer
template branch(height) {
    var items = 1 << height;
    // input array
    signal input ins[items * 2];
    // output array
    signal output outs[items];
    component hash[items];
    for(var i = 0; i < items; i++) {
        hash[i] = MultiMiMC7(2,91);
        hash[i].in[0] <== ins[i * 2];
        hash[i].in[1] <== ins[i * 2 + 1];
        hash[i].k <== 0;
        hash[i].out ==> outs[i];
    }
}
// Builds a merkle tree from leaf array
template merkleProof(levels) {
    signal input leaves[1 << levels];
    signal output root;
    component layers[levels];
    for(var level = levels - 1; level >= 0; level--) {
        layers[level] = branch(level);
        for(var i = 0; i < (1 << (level + 1)); i++) {
            layers[level].ins[i] <== level == levels - 1 ? leaves[i] : layers[level + 1].outs[i];
        }
    }
    root <== levels > 0 ? layers[0].outs[0] : leaves[0];
}
component main{public [leaves]} = merkleProof(3);
```

2. Errors encountered and solution:

This was the first error I encountered with the input array of bigger size(8):

```
[ERROR] snarkJS: circuit too big for this power of tau ceremony.  
5096*2 > 2**12
```

I encountered this while starting a new “power of tau” ceremony :

```
snarkjs powersoftau new bn128 12 pot12_0000.ptau -v
```

I resolved it by increasing the “power of tau” such that it is greater than the required number. So increasing the power to 14 resolved this since $2^{**14} > 5096*2$.

3. Theory Question:

We do not need zero-knowledge proof for this same functionality. A publicly verifiable smart contract that computes Merkle root will achieve the same. We are still able to construct a Merkle Tree without zero knowledge proofs. Zero-knowledge proofs allow us to improve the security nature if heightened security is needed. It also allows transactions and interactions to be noninteractive, as well as making transactions cheaper and resource efficient. A scenario where this ZKP can be useful is if we want to achieve the same merkleTree functionality without revealing other unnecessary data like inputs and such. One such application: ZKRollup that uses ZKPs where many transactions (leaves) are bundled up together and submitted on an L1 for a verifier contract to verify that they are valid.

=> ScreenShots and public.json

I used a shell script to run the commands simultaneously: Updated Github Link for script.sh

[\[Link\]](#)

```
snarkjs powersoftau new bn128 14 pot12_0000.ptau -v  
snarkjs powersoftau contribute pot12_0000.ptau pot12_0001.ptau --name="First contribution"  
-v -e="kira"  
  
snarkjs powersoftau prepare phase2 pot12_0001.ptau pot12_final.ptau -v  
snarkjs groth16 setup merkleProof.r1cs pot12_final.ptau merkleProof_0000.zkey  
  
snarkjs zkey contribute merkleProof_0000.zkey merkleProof_0001.zkey --name="1st Contributor  
Name" -v -e="kira"  
snarkjs zkey export verificationkey merkleProof_0001.zkey verification_key.json  
  
cp merkleProof_js/witness.wtns .  
  
snarkjs groth16 prove merkleProof_0001.zkey witness.wtns proof.json public.json
```

```

snarkjs groth16 verify verification_key.json public.json proof.json

#snarkjs zkey export solidityVerifier merkleProof_0001.zkey verifier.sol
#snarkjs generatecall

```

Screenshots for running the script.sh

```

> ./script.sh
[DEBUG] snarkJS: Calculating First Challenge Hash
[DEBUG] snarkJS: Calculate Initial Hash: tauG1
[DEBUG] snarkJS: Calculate Initial Hash: tauG2
[DEBUG] snarkJS: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkJS: Calculate Initial Hash: betaTauG1
[DEBUG] snarkJS: Blank Contribution Hash:
    786a02f7 42015903 c6c6fd85 2552d272
    912f4740 e1584761 8a86e217 f71f5419
    d25e1031 afee5853 13896444 934eb04b
    903a685b 1448b755 d56f701a fe9be2ce
[INFO]  snarkJS: First Contribution Hash:
    bc0bde79 80381fa6 42b20975 91dd83f1
    ed15b003 e15c3552 0af32c95 eb519149
    2a6f3175 215635cf c10e6098 e2c612d0
    ca84f1a9 f90b5333 560c8af5 9b9209f4
[DEBUG] snarkJS: Calculating First Challenge Hash
[DEBUG] snarkJS: Calculate Initial Hash: tauG1
[DEBUG] snarkJS: Calculate Initial Hash: tauG2
[DEBUG] snarkJS: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkJS: Calculate Initial Hash: betaTauG1
[DEBUG] snarkJS: processing: tauG1: 0/32767
[DEBUG] snarkJS: processing: tauG1: 16384/32767
[DEBUG] snarkJS: processing: tauG2: 0/16384
[DEBUG] snarkJS: processing: tauG2: 8192/16384
[DEBUG] snarkJS: processing: alphaTauG1: 0/16384
[DEBUG] snarkJS: processing: betaTauG1: 0/16384
[DEBUG] snarkJS: processing: betaTauG2: 0/1
[INFO]  snarkJS: Contribution Response Hash imported:
    be997877 bd19890d ecdb28eba 54f43881
    8158d77b 97f01b52 6351bdcc 662c19b9
    a0bb394e fba51c50 fb28de85 f23a00ef
    f8a0fb72 508cac57 0cdaddef 24afe0ff
[INFO]  snarkJS: Next Challenge Hash:
    6528f31a db606697 edcbd656 ba77eb61
    eb06b133 6cae0def 6560619f 6ae246a8
    da770da9 c40a809f 6cd1dccc 62f41613
    f08fff680 f5b10218 a930eab6 7b2ac45c
[DEBUG] snarkJS: Starting section: tauG1
[DEBUG] snarkJS: tauG1: fft 0 mix start: 0/1
[DEBUG] snarkJS: tauG1: fft 0 mix end: 0/1

```

```

[DEBUG] snarkJS: tauG1: fft 15 join: 15/15
[DEBUG] snarkJS: tauG1: fft 15 join 15/15 1/1 2/4
[DEBUG] snarkJS: tauG1: fft 15 join 15/15 1/1 0/4
[DEBUG] snarkJS: tauG1: fft 15 join 15/15 1/1 1/4
[DEBUG] snarkJS: tauG1: fft 15 join 15/15 1/1 3/4
[DEBUG] snarkJS: Starting section: tauG2
[DEBUG] snarkJS: tauG2: fft 0 mix start: 0/1
[DEBUG] snarkJS: tauG2: fft 0 mix end: 0/1
[DEBUG] snarkJS: tauG2: fft 1 mix start: 0/1

[DEBUG] snarkJS: tauG2: fft 14 join 14/14 1/1 1/4
[DEBUG] snarkJS: tauG2: fft 14 join 14/14 1/1 2/4
[DEBUG] snarkJS: Starting section: alphaTauG1
[DEBUG] snarkJS: alphaTauG1: fft 0 mix start: 0/1
[DEBUG] snarkJS: alphaTauG1: fft 0 mix end: 0/1
[DEBUG] snarkJS: alphaTauG1: fft 1 mix start: 0/1

```

```
[DEBUG] snarkJS: alphaTauG1: fft 14 join 14/14 1/1 0/4
[DEBUG] snarkJS: alphaTauG1: fft 14 join 14/14 1/1 2/4
[DEBUG] snarkJS: alphaTauG1: fft 14 join 14/14 1/1 1/4
[DEBUG] snarkJS: Starting section: betaTauG1
[DEBUG] snarkJS: betaTauG1: fft 0 mix start: 0/1
[DEBUG] snarkJS: betaTauG1: fft 0 mix end: 0/1
[DEBUG] snarkJS: betaTauG1: fft 1 mix start: 0/1
```

```
[DEBUG] snarkJS: betaTauG1: fft 14 join 14/14 1/1 3/4
[DEBUG] snarkJS: betaTauG1: fft 14 join 14/14 1/1 1/4
[DEBUG] snarkJS: betaTauG1: fft 14 join 14/14 1/1 0/4
[DEBUG] snarkJS: betaTauG1: fft 14 join 14/14 1/1 2/4
[INFO] snarkJS: Reading r1cs
[INFO] snarkJS: Reading tauG1
[INFO] snarkJS: Reading tauG2
[INFO] snarkJS: Reading alphatauG1
[INFO] snarkJS: Reading betatauG1
[INFO] snarkJS: Circuit hash:
          0dfa5c33 c3c38e1e ab636b2a fa7db1a8
          650a5d9c 57826d28 fe777603 b63f1bad
          fbe45f72 7ff71140 b5a94ddd 5639e2cb
          73870f26 3247e087 1fcddcc8 85c3a298
[DEBUG] snarkJS: Applying key: L Section: 0/5103
[DEBUG] snarkJS: Applying key: H Section: 0/8192
[INFO] snarkJS: Circuit Hash:
          0dfa5c33 c3c38e1e ab636b2a fa7db1a8
          650a5d9c 57826d28 fe777603 b63f1bad
          fbe45f72 7ff71140 b5a94ddd 5639e2cb
          73870f26 3247e087 1fcddcc8 85c3a298
[INFO] snarkJS: Contribution Hash:
          e73e3316 9b410f39 e8c80321 a4daebfa
          ac12a754 e0417571 1346a257 e2580a99
          791538c0 84494dc5 101718a7 f12473fa
          705fe4f3 c1c0d854 147c51d8 277e1098
[INFO] snarkJS: OK!
```

public.json: GitHub Link to public.json: [\[Link\]](#)

```
[
"14075477420338855047846876569733296091014866770910171554054559505431936618782",
"1",
"2",
"3",
"4",
"5",
"6",
"7",
"8"
]
```

=> SmartContract Deployment and verification on remix: Link to [Verifier.sol](#)

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is visible, showing network selection (Rinkeby), account (0xdA7...da9B6), gas limit (3000000), value (0 Wei), and a deployed contract named 'Verifier'. Below these are buttons for 'Deploy', 'Publish to IPFS', and 'At Address'. The main area displays the Solidity code for 'Verifier.sol'. The code defines a constructor that initializes variables with values from memory. It includes a function 'verifyProof' that takes four parameters (uint[2] memory a, uint[2][2] memory b, uint[2] memory c, uint[9] memory input) and returns a boolean. Inside this function, it creates a 'Proof' memory variable, initializes 'proof.A' and 'proof.B' with specific G1 and G2 points, and 'proof.C' with a G1 point. It then loops through the 'input' array, setting 'inputValues[i]' to 'input[i]'. Finally, it checks if 'verify(inputValues, proof)' returns 0, indicating a successful verification. The bottom right corner of the code editor has a 'Debug' button. At the bottom of the interface, there is a transaction history section showing a pending creation of the Verifier contract and a recent call to its verifyProof function.

```
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
```

```
contract Verifier {
    uint[2] vk_alpha1, vk_beta2;
    uint[2] vk_x, vk_gamma2;
    Proof C, vk_delta2;

    function () public payable {
        vk_alpha1 = msg.value;
        vk_beta2 = msg.value;
        vk_x = msg.value;
        vk_gamma2 = msg.value;
        C = Proof();
        C.A = Pairing.G1Point(a[0], a[1]);
        C.B = Pairing.G2Point([b[0][0], b[0][1]], [b[1][0], b[1][1]]);
        C.C = Pairing.G1Point(c[0], c[1]);
        uint[] memory inputValues = new uint[](input.length);
        for(uint i = 0; i < input.length; i++){
            inputValues[i] = input[i];
        }
        if (verify(inputValues, proof) == 0) {
            return true;
        } else {
            return false;
        }
    }

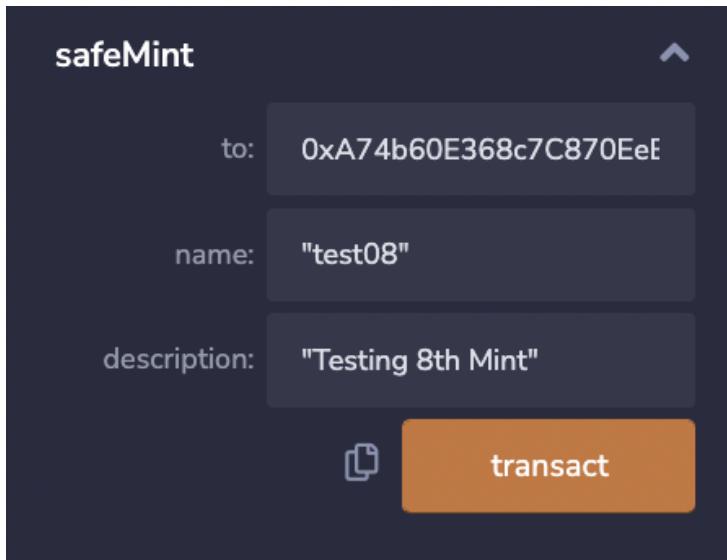
    function verifyProof(
        uint[2] memory a,
        uint[2][2] memory b,
        uint[2] memory c,
        uint[9] memory input
    ) public view returns (bool r) {
        Proof proof;
        proof.A = Pairing.G1Point(a[0], a[1]);
        proof.B = Pairing.G2Point([b[0][0], b[0][1]], [b[1][0], b[1][1]]);
        proof.C = Pairing.G1Point(c[0], c[1]);
        uint[] memory inputValues = new uint[](input.length);
        for(uint i = 0; i < input.length; i++){
            inputValues[i] = input[i];
        }
        if (verify(inputValues, proof) == 0) {
            return true;
        } else {
            return false;
        }
    }
}
```

Question:2

1. NFT Contract and merkle Tree:

- Link to code for ERC721 contract that can mint NFTs: [[NFT.sol](#)].
- Link to code for merkle tree used within NFT.sol : [[merkleTree.sol](#)].
- The token URI is stored on-chain and also includes name and description field
- Screenshots for methods of deployed Contracts are attached below and mintDetails screenshots are also included below them.

- SafeMint() Method used for minting the NFT to specific address:



- Retrieval of metaData from the token URI:



2 & 3. Screenshots and Gas Used:

can be Verified in screenshots below and on this [[link](#)]

No.	tokenURI	gasUsed
1	0	296789
2	1	194129
3	2	211313
4	3	194165
5	4	226313
6	5	213765
7	6	211713
8	7	196665

Minting screenShots for 1st and 2nd NFTs

```
transact to ZKMint.safeMint pending ...

view on etherscan

✓ [block:10288350 txIndex:8] from: 0xdA7...da9B6 to: ZKMint.safeMint(address,string,string) 0x416...F4C39 value: 0 wei data: 0xeca...00000 logs: 1
hash: 0xff6...b94a6

status true Transaction mined and execution succeed

transaction hash 0x790cblb3d9ecaldb97384d536c00341aa9b09bf3965b5eb27c6d9fe565d647d1 ⓘ

from 0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6 ⓘ

to ZKMint.safeMint(address,string,string) 0x416b214868417EF9C08624b430d6024f7F6F4C39 ⓘ

gas 296789 gas ⓘ

transaction cost 296789 gas ⓘ

hash 0x790cblb3d9ecaldb97384d536c00341aa9b09bf3965b5eb27c6d9fe565d647d1 ⓘ

input 0xeca...00000 ⓘ

decoded input {
    "address to": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
    "string name": "Shyam",
    "string description": "Testing 1st NFT"
} ⓘ

decoded output - ⓘ

logs [
    {
        "from": "0x416b214868417EF9C08624b430d6024f7F6F4C39",
        "topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
        "event": "Transfer",
        "args": {
            "0": "0x0000000000000000000000000000000000000000000000000000000000000000",
            "1": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
            "2": "0",
            "from": "0x0000000000000000000000000000000000000000000000000000000000000000",
            "to": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
            "tokenId": "0"
        }
    }
] ⓘ ⓘ

transact to ZKMint.safeMint pending ...

view on etherscan

✓ [block:10288359 txIndex:3] from: 0xdA7...da9B6 to: ZKMint.safeMint(address,string,string) 0x416...F4C39 value: 0 wei data: 0xeca...00000 logs: 1
hash: 0x770...918aa

status true Transaction mined and execution succeed

transaction hash 0x57915e6f7bd025a0ab1fe4d06f9fb61d35d339d16ebb47729bbcd9662d2ec4ad ⓘ

from 0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6 ⓘ

to ZKMint.safeMint(address,string,string) 0x416b214868417EF9C08624b430d6024f7F6F4C39 ⓘ

gas 194129 gas ⓘ

transaction cost 194129 gas ⓘ

hash 0x57915e6f7bd025a0ab1fe4d06f9fb61d35d339d16ebb47729bbcd9662d2ec4ad ⓘ

input 0xeca...00000 ⓘ

decoded input {
    "address to": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
    "string name": "kira",
    "string description": "Testing 2nd NFT"
} ⓘ

decoded output - ⓘ

logs [
    {
        "from": "0x416b214868417EF9C08624b430d6024f7F6F4C39",
        "topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
        "event": "Transfer",
        "args": {
            "0": "0x0000000000000000000000000000000000000000000000000000000000000000",
            "1": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
            "2": "1",
            "from": "0x0000000000000000000000000000000000000000000000000000000000000000",
            "to": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
            "tokenId": "1"
        }
    }
] ⓘ ⓘ
```

ScreenShot 3rd and 7th NFT Mint:

```

view on etherscan
[block:10288694 txIndex:9] from: 0xdA7...da9B6 to: ZKMint.safeMint(address,string,string) 0x416...F4C39 value: 0 wei data: 0xec...00000 logs: 1 Debug ▾
✓ [block:10288694 txIndex:9] from: 0xdA7...da9B6 to: ZKMint.safeMint(address,string,string) 0x416...F4C39 value: 0 wei data: 0xec...00000 logs: 1 Debug ▾
status true Transaction mined and execution succeed
transaction hash 0x237b32f92767137042c3353f40787cfb01a4e899be9878bf585241a998420218 ⓘ
from 0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6 ⓘ
to ZKMint.safeMint(address,string,string) 0x416b214868417EF9C08624b430d6024f7F6F4C39 ⓘ
gas 211713 gas ⓘ
transaction cost 211713 gas ⓘ
hash 0x237b32f92767137042c3353f40787cfb01a4e899be9878bf585241a998420218 ⓘ
input 0xec...00000 ⓘ
decoded input {
    "address to": "0xA74b60E368c7C870EeB581dc899cFA8158058bb8",
    "string name": "test07",
    "string description": "Testing 7th Mint"
} ⓘ
decoded output -
logs [
{
    "from": "0x416b214868417EF9C08624b430d6024f7F6F4C39",
    "topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
    "event": "Transfer",
    "args": {
        "0": "0x0000000000000000000000000000000000000000000000000000000000000000",
        "1": "0xa74b60E368c7C870EeB581dc899cFA8158058bb8",
        "2": "6",
        "from": "0x0000000000000000000000000000000000000000000000000000000000000000",
        "to": "0xA74b60E368c7C870EeB581dc899cFA8158058bb8",
        "tokenId": "6"
    }
}
]
val 0 wei ⓘ

transact to ZKMint.safeMint pending ...
view on etherscan
[block:10288492 txIndex:10] from: 0xdA7...da9B6 to: ZKMint.safeMint(address,string,string) 0x416...F4C39 value: 0 wei data: 0xec...00000 logs: 1 Debug ▾
✓ [block:10288492 txIndex:10] from: 0xdA7...da9B6 to: ZKMint.safeMint(address,string,string) 0x416...F4C39 value: 0 wei data: 0xec...00000 logs: 1 Debug ▾
status true Transaction mined and execution succeed
transaction hash 0xc8f5075cbdb89014ce7101913256544fa689b41c8cad478b438685f29753eab7 ⓘ
from 0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6 ⓘ
to ZKMint.safeMint(address,string,string) 0x416b214868417EF9C08624b430d6024f7F6F4C39 ⓘ
gas 211313 gas ⓘ
transaction cost 211313 gas ⓘ
hash 0xc8f5075cbdb89014ce7101913256544fa689b41c8cad478b438685f29753eab7 ⓘ
input 0xec...00000 ⓘ
decoded input {
    "address to": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
    "string name": "test03",
    "string description": "Testing 3rd Mint"
} ⓘ
decoded output -
logs [
{
    "from": "0x416b214868417EF9C08624b430d6024f7F6F4C39",
    "topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
    "event": "Transfer",
    "args": {
        "0": "0x0000000000000000000000000000000000000000000000000000000000000000",
        "1": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
        "2": "2",
        "from": "0x0000000000000000000000000000000000000000000000000000000000000000",
        "to": "0xdA7773E91a396d592AD33146164dA6d7d2Fda9B6",
        "tokenId": "2"
    }
}
]
val 0 wei ⓘ
>

```

Last minted NFT:

```
transact to ZKMint.safeMint pending ...

view on etherscan

✓ [block:10288697 txIndex:28] from: 0xda7...da9B6 to: ZKMint.safeMint(address,string,string) 0x416...F4C39 value: 0 wei data: 0xeeca...00000 logs: 1 Debug ▾

status true Transaction mined and execution succeed

transaction hash 0xee762a98bfadb54242fd1ca96439e28b0f1145c5115e9ac33ab4cb1d511e81e ⓘ

from 0xdA7773B91a396d592AD33146164dA6d7d2Pda9B6 ⓘ

to ZKMint.safeMint(address,string,string) 0x416b214868417EF9C08624b430d6024f7F6F4C39 ⓘ

gas 196665 gas ⓘ

transaction cost 196665 gas ⓘ

hash 0xee762a98bfadb54242fd1ca96439e28b0f1145c5115e9ac33ab4cb1d511e81e ⓘ

input 0xeeca...00000 ⓘ

decoded input {
    "address to": "0xA74b60E368c7C870EeB581dc899cFA8158058bb8",
    "string name": "test07",
    "string description": "Testing 7th Mint"
} ⓘ

decoded output - ⓘ

logs [
    {
        "from": "0x416b214868417EF9C08624b430d6024f7F6F4C39",
        "topic": "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
        "event": "Transfer",
        "args": {
            "0": "0x0000000000000000000000000000000000000000000000000000000000000000",
            "1": "0xA74b60E368c7C870EeB581dc899cFA8158058bb8",
            "2": "7",
            "from": "0x0000000000000000000000000000000000000000000000000000000000000000",
            "to": "0xA74b60E368c7C870EeB581dc899cFA8158058bb8",
            "tokenId": "7"
        }
    }
] ⓘ ⓘ
```

Transactions for all Minted NFT can found in this [[Link](#)] with SafeMint Method:

Question 3: Understanding and generating ideas about ZK technologies

1. Differences between SNARKs and STARKs

The proof size of SNARKs are much smaller than STARKs meaning it would take less storage on-chain. Therefore SNARKs are estimated to require much more gas than SNARKs and that makes STARKs economically expensive for end users compared to SNARKs.

Zk-SNARK proofs are dependent on an initial trusted setup between a prover and verifier and this could create a potential centralization issue. They can be used in blockchain based applications much easily as they are smaller in size and can be verified on-chain.

Zk-STARKs don't require a trusted setup but disadvantage is that a prover with enough computational power could create fake proofs. They can be used in advanced p2p systems where there is no trusted setup (Binance p2p).

2. Differences between the trusted setup process between groth16 and PLONK.

The setup process in groth16 has to be repeated for each circuit. For example, we need to create a unique setup for every different circuit. Whereas in PLONK, one trusted setup can be used to validate all the circuits.

3. Give an idea of how we can apply ZK to create unique usage for NFTs.

ZK can be used to check if you have any NFT of a particular collection without revealing the tokenID. It can also be used to hide certain attributes/metadata of an NFT.

4. Give a novel idea on how we can apply ZK for Dao Tooling.

We know voting is a very popular idea but we can take one step further by making the voter identity secret. ZK can be used in hiding the identity of a proposer and voter.

ZK can help a Grants DAO to hide its abstract details. It can also be used for running a freelancers' network. One futuristic idea can be verifying the integrity of pharmaceutical products like medicines, vaccines, even vaccine certificates.