ZKU Assignment 3

Question 1

Question 1.1

> Consider a hypothetical move called the ‘triangle jump’, a player hops from planet A to B then to C and returns to A all in one move, such that A, B, and C lie on a triangle.

> Write a Circom circuit that verifies this move. The coordinates of A, B, and C are private inputs. You may need to use basic geometry to ascertain that the move lies on a triangle. Also, verify that the move distances (A → B and B → C) are within the energy bounds.

The circuit link is the following. Here!

Question 2

Question 2.1

> In a card game, how can a player commit to a card without revealing what the card is? A naive protocol would be to map all cards to a number between 0 and 51 and then hash this number to get a commitment. This won’t actually work as one could easily brute force the 52 hashes. To prevent players from changing the card we need to store some commitment on-chain. How would you design this commitment? Assume each player has a single card that needs to be kept secret. Modify the naive protocol so that brute force doesn’t work.

One can apply additional attributes to the card. The example of mapping is (color, suite, number). For example, the card has own number 28, and it is diamond while color is red. We can hash all of given attributes, and concatenate those hashed attributes, and finally hash the string again. It would render a large enough psuedorandom enumeraton.

Question 2.2

> Now assume that the player needs to pick another card from the same suite. Design a circuit that can prove that the newly picked card is in the same suite as the previous one. Can the previous state be spoofed? If so, what mechanism is needed in the contracts to verify this?

> Design a contract, necessary circuits, and verifiers to achieve this. You may need to come up with an appropriate representation of cards as integers such that the above operations can be done easily.

Let us say, we have 52 cards [0-51] that each consecutive set of 13 numbers has its own single suit. In a 52-card deck, it preserves the four original French suits of centuries ago: clubs (♧), diamonds (♢), hearts (♥), and spades (♤). For example, from 0 to 12 is spades, while 13-25 clubs, 26-38 diamonds, and 39-51 hearts. We are going to prove that two cards passed belong to the same suit. For example, 17 to 23 belongs to the same suit while 37 and 47 does not belong to the same suit.

Question 3: MACI and VDF

Question 3.1

> What problems in voting does MACI not solve? What are some potential solutions?

A key-selling attack would not be solved according to the current MACI version. To specify, it is possible for a briber to bribe voters to insert their public key during the signup process. This would be solved by migrating to other zk solutions other than zk-SNARKS. The candidate solutions should not involve any outer trusted setup per unique circuit.

In addition, a vote-buying attack where a user may be bribed immediately after the voting period starts should be solved in a later MACI version. The first message from a voter may be influenced by a briber. After the signup period ends after a fixed amount of time, each user sign their command using the key which they had signed up with and then use a random but ephemeral key to generate a shared key to encrypt it.

In this phase, however, if users are bribed, they should sign it using an old public key which has already been replaced with a new one. Otherwise, the user should use the most current public key they have registered. To solve this problem, render a contract to mandate that the first message that each participant propagates is to change their keys.

Question 3.2

> How can a pseudorandom dice roll be simulated using just Solidity? What are the issues with this approach?

```
function randomHash(uint256 entrophy) external returns (bytes32) {
    return keccak256(abi.encodePacked(entrophy, block.hash));
}
```

Yes, Randomness in Ethereum blockchain solely is notoriously difficult problem. Since the blockchain has a deterministic nature, it is inherently no randomness involved. To solve this problem, we can take an approach to entertain some random entrophies and hash it to get a pseudo-random hash.

```
function randomNumber() internal view returns (uint) {
    return uint(blockhash(block.number - 1));
}
```

You are also able to use block timestamps for randomNumber.

Question 3.2.1

> What are the issues with this approach?

To begin with, entertaining randonNumber from an outer scope of the function is susceptible to "man-in-the-middle" attack. In addition, the last of the aforementioned approach could be strongly influenced by miners. If a miner has an incentive to manipulate timestamps, one can use its mining power to manipulate the randomNumber. To specify, a miner could hide a found block if one believes that the random number derived from the block is disadvantage. In addition, from the code above (randomNumber function), `block.number` is available to all users so that one can use accurately (not involved randomness) it anytime.

Question 3.2.2

> How would you design a multi-party system that performs a dice roll?

To begin with, I have assigned the map for participating members who are eligible to put each randomNumbers. After all members have put their randomNumbers, the contract hash them with utilizing block.difficulty. Use all them to keccack256 for making its output.

The pseudocode would be like the following.

```
contract randomDice {

  uint256 maxInputsNum;
  bytes32 hash;
  mapping(address => bool) isEnterInput;
  uint256 memberInputsNum;

  constructor(address[] members) {
      maxInputs = members.length;
  }

  function rollDice(uint256 randomNumber) external returns (bytes32) {
    require(isEnterInput[msg.sender] == 0, "already voted");
    isEnterInput[msg.sender] = true;

    hash = abi.encodePacked(block.hash, randomNumber);
    if (memberInputsNum == maxInputsNum) {
        return keccak256(abi.encodePacked(hash, block.hash));
    }
  }
}
```

Question 3.2.3

> Compare both techniques and explain which one is fairer and why.

It is clear that the second approach is much fairer, since more participants put their own input data. The more inputs, the more complexity involves. This is similiar to the notion that zCash argues when they have entertained more participants when doing powersofTau in 2018 than before.

Question 3.2.4

> Show how the multi party system is still vulnerable to manipulation by malicious parties and then elaborate on the use of VDF's in solving this.

A multi-party system, MPC, has innate possibilities to be manipulated when those who are responsible for putting the inputs have colluded at once. In the first contract setup, the last participant has control of outputs since the final partaker knows the previous inputs from other members.

Thus, we need a verifiable delay function, VDF, to ensure that requires a moderate amount of sequential computation to evaluate, but once a solution is found, it is easy for anyone to verify that it is correct. When using a VDF for the purpose of delaying the time imposed on the output of some pseudorandom generator, it could prevent malicious actors from influenceing the output of the generator. This could be achieved when all inputs will be finalized before anyone can finish computing the VDF.

Practically, one can use Chainlink VDF as the following.

Contract Link is the following.

```
  );
```

```
pragma solidity 0.6.2;

import "https://raw.githubusercontent.com/smartcontractkit/chainlink/develop/contracts/src/v0.6/VRFConsumerBa

contract Verifiable6SidedDiceRoll is VRFConsumerBase {
  using SafeMath for uint;

  bytes32 internal keyHash;
  uint256 internal fee;

  event RequestRandomness(
    bytes32 indexed requestId,
    bytes32 keyHash,
    uint256 seed
  );

  event RequestRandomnessFulfilled(
    bytes32 indexed requestId,
    uint256 randomness
  );

  /**
   * @notice Constructor inherits VRFConsumerBase
   * @dev Ropsten deployment params:
   * @dev   _vrfCoordinator: 0xf720CF1B963e0e7bE9F58fd471EFa67e7bF00cfb
   * @dev   _link:           0x20fE562d797A42Dcb3399062AE9546cd06f63280
   */
  constructor(address _vrfCoordinator, address _link)
```

```
        VRFConsumerBase(_vrfCoordinator, _link) public
    {
        vrfCoordinator = _vrfCoordinator;
        LINK = LinkTokenInterface(_link);
        keyHash = 0xced103054e349b8dfb51352f0f8fa9b5d20dde3d06f9f43cb2b85bc64b238205; // hard-coded fo
        fee = 10 ** 18; // 1 LINK hard-coded for Ropsten
    }

    /**
     * @notice Requests randomness from a user-provided seed
     * @dev The user-provided seed is hashed with the current blockhash as an additional precaution.
     * @dev   1. In case of block re-orgs, the revealed answers will not be re-used again.
     * @dev   2. In case of predictable user-provided seeds, the seed is mixed with the less predictable blockhash.
     * @dev This is only an example implementation and not necessarily suitable for mainnet.
     * @dev You must review your implementation details with extreme care.
     */
    function rollDice(uint256 userProvidedSeed) public returns (bytes32 requestId) {
        require(LINK.balanceOf(address(this)) > fee, "Not enough LINK - fill contract with faucet");
        uint256 seed = uint256(keccak256(abi.encode(userProvidedSeed, blockhash(block.number)))); // Hash use
        bytes32 _requestId = requestRandomness(keyHash, fee, seed);
        emit RequestRandomness(_requestId, keyHash, seed);
        return _requestId;
    }

    function fulfillRandomness(bytes32 requestId, uint256 randomness) external override {
        uint256 d6Result = randomness.mod(6).add(1);
        emit RequestRandomnessFulfilled(requestId, randomness);
    }

}
```

Question 4: InterRep

Question 4.1

> How does InterRep use Semaphore in their implementation? Explain why InterRep still needs a centralized server.

Interep is a centralized service that verifies the reputation of users and provides special group mapping for the usage of dApps and other social media services. The suggested program utilizes Semaphore to allow users to prove that their own identifier is satisfied to be valid for membership in a certain group. By doing so, the user does not in need to expose their identities, but to create an unique identifier with an Ethereum account. This mean is that we need some centralization for the sake of more privacy.

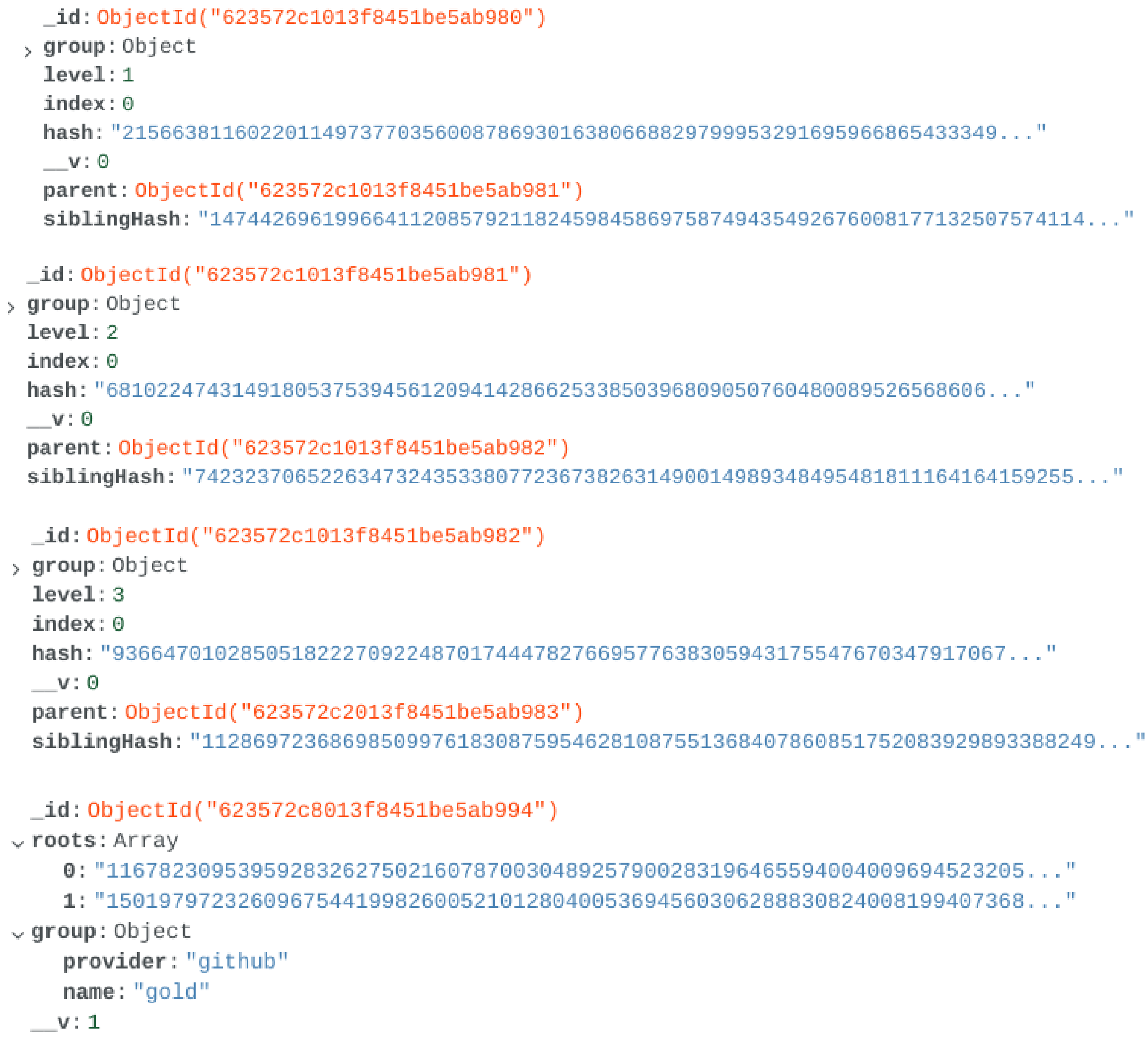
Question 4.2

> Clone the InterRep repos: contracts and reputation-service. Follow the instructions on the Github repos to start the development environment. Try to join one of the groups, and then leave the group. Explain what happens to the Merkle Tree in the MongoDB instance when you decide to leave a group.

I had deployed the contract to here, Ropsten Network.

According to deleteLeaf.ts, the database prunes the matching leaf from the merkle tree on which it maintains your commitment when you leave a group. The function takes 1) the provider of the group, 2) the name of the group, 3) and identityCommitment for the leaf of the tree. It returns new merkle tree root that is pruned the corresponding leaf.

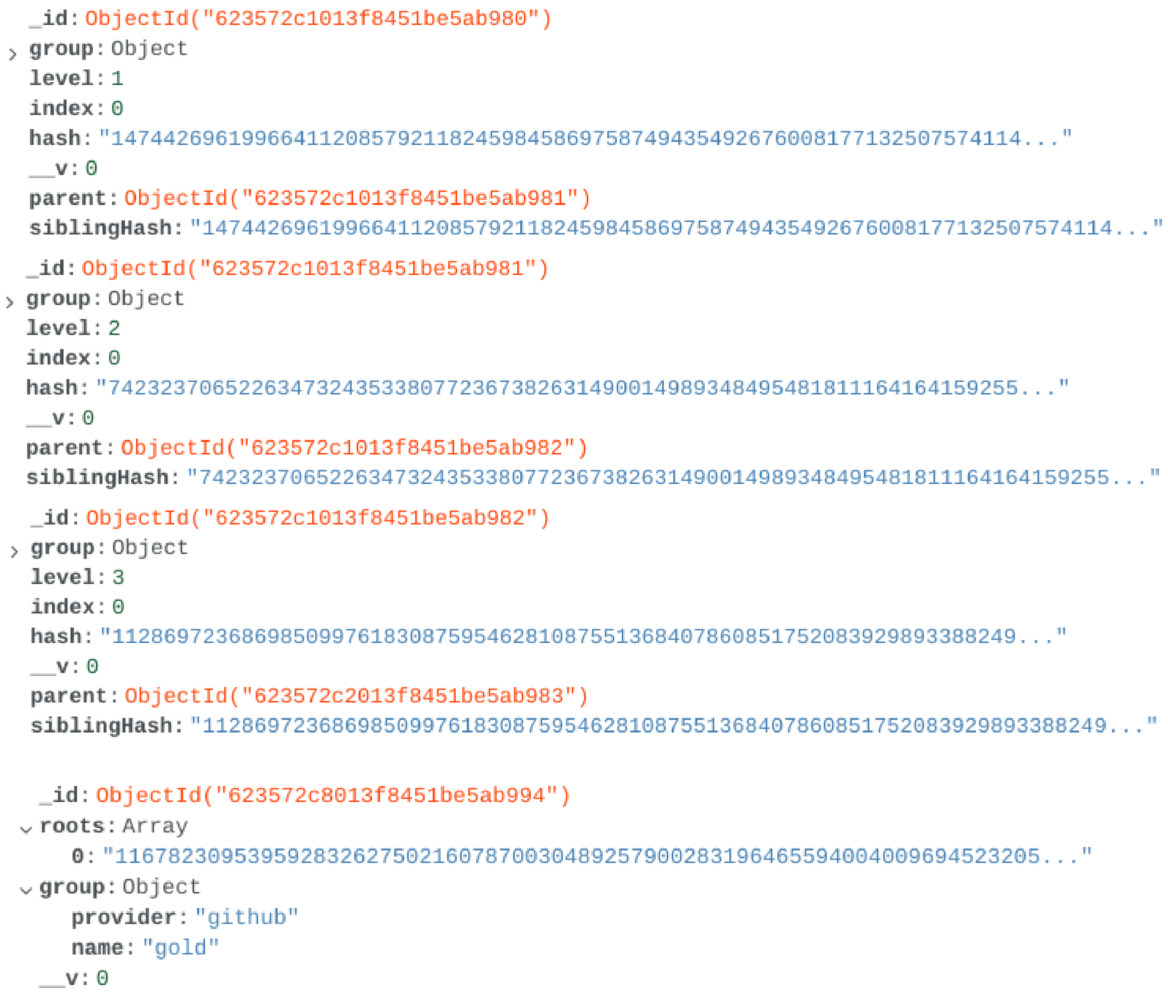
> before leaving the group (three levels from total 20 levels)



```
  _id: ObjectId("623572c1013f8451be5ab980")
› group: Object
  level: 1
  index: 0
  hash: "215663811602201149737703560087869301638066882979995329169596686543334..."
  __v: 0
  parent: ObjectId("623572c1013f8451be5ab981")
  siblingHash: "147442696199664112085792118245984586975874943549267600817713250757411..."

 _id: ObjectId("623572c1013f8451be5ab981")
› group: Object
 level: 2
 index: 0
 hash: "681022474314918053753945612094142866253385039680905076048008952656860..."
 __v: 0
 parent: ObjectId("623572c1013f8451be5ab982")
 siblingHash: "742323706522634732435338077236738263149001498934849548181116416415925..."

 _id: ObjectId("623572c1013f8451be5ab982")
› group: Object
 level: 3
 index: 0
 hash: "936647010285051822270922487017444782766957763830594317554767034791706..."
 __v: 0
 parent: ObjectId("623572c2013f8451be5ab983")
 siblingHash: "112869723686985099761830875954628108755136840786085175208392989338824..."

 _id: ObjectId("623572c8013f8451be5ab994")
⌄ roots: Array
    0: "116782309539592832627502160787003048925790028319646559400400969452320..."
    1: "150197972326096754419982600521012804005369456030628883082400819940736..."
⌄ group: Object
    provider: "github"
    name: "gold"
 __v: 1
```

This picture above depicts the mongoDB instance after joining the group.

> after leaving the group (three levels from total 20 levels)



```
 _id: ObjectId("623572c1013f8451be5ab980")
› group: Object
 level: 1
 index: 0
 hash: "147442696199664112085792118245984586975874943549267600817713250757411..."
 __v: 0
 parent: ObjectId("623572c1013f8451be5ab981")
 siblingHash: "147442696199664112085792118245984586975874943549267600817713250757411..."
 _id: ObjectId("623572c1013f8451be5ab981")
› group: Object
 level: 2
 index: 0
 hash: "742323706522634732435338077236738263149001498934849548181116416415925..."
 __v: 0
 parent: ObjectId("623572c1013f8451be5ab982")
 siblingHash: "742323706522634732435338077236738263149001498934849548181116416415925..."
 _id: ObjectId("623572c1013f8451be5ab982")
› group: Object
 level: 3
 index: 0
 hash: "112869723686985099761830875954628108755136840786085175208392989338824..."
 __v: 0
 parent: ObjectId("623572c2013f8451be5ab983")
 siblingHash: "112869723686985099761830875954628108755136840786085175208392989338824..."

  _id: ObjectId("623572c8013f8451be5ab994")
⌄ roots: Array
    0: "116782309539592832627502160787003048925790028319646559400400969452320..."
⌄ group: Object
    provider: "github"
    name: "gold"
  __v: 0
```

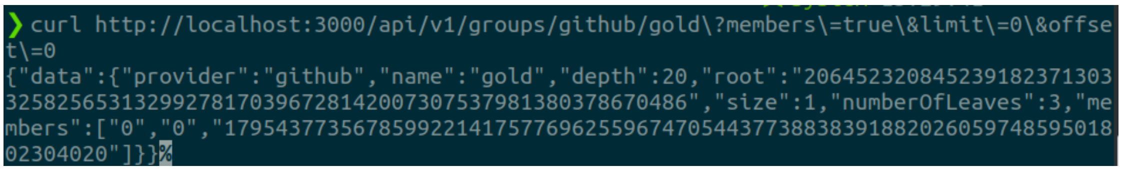
The picture above depicts the mongoDB instance after leaving the group. The hash of the those leaves, and the sibilingHash changes after leaving the group.

In addition, the treeRootBatches also changes from one elements to two elements.

Question 4.3

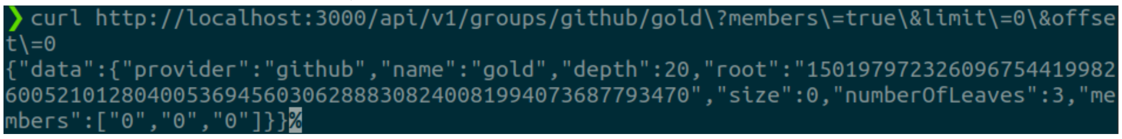
> Use the public API (instead of calling the Kovan testnet, call your localhost) to query the status of your own identityCommitment in any of the social groups supported by InterRep before and after you leave the group. Take the screenshots of the responses and paste them to your assignment submission PDF.

- Number of leaves before leaving the group



```
❯ curl http://localhost:3000/api/v1/groups/github/gold\?members\=true\&limit\=0\&offse
t\=0
{"data":{"provider":"github","name":"gold","depth":20,"root":"206452320845239182371303
32582565313299278170396728142007307537981380378670486","size":1,"numberOfLeaves":3,"me
mbers":["0","0","179543773567859922141757769625596747054437738838391882026059748595018
02304020"]}}%
```

- Number of leaves diminshes by 1 after leaving the group

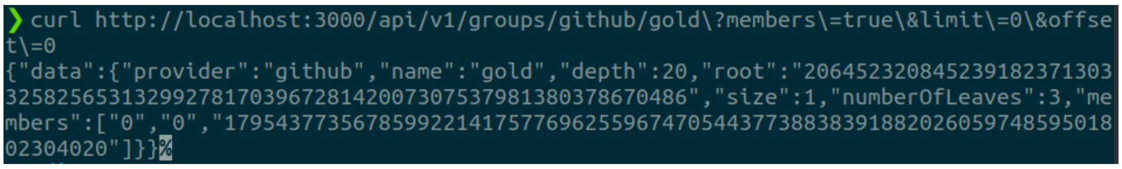


```
❯ curl http://localhost:3000/api/v1/groups/github/gold\?members\=true\&limit\=0\&offse
t\=0
{"data":{"provider":"github","name":"gold","depth":20,"root":"150197972326096754419982
60052101280400536945603062888308240081994073687793470","size":0,"numberOfLeaves":3,"me
mbers":["0","0","0"]}}%
```

We can notice from the above images that the hash of the merkle tree changes, and the size also changes, which is diminished by 1.

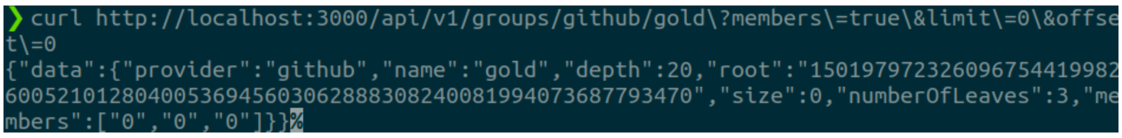
However, the number of leaves does not change even after leaving the group. The leaves changes its value to 0, and the size from API also shows 0, which seems that if the value of leaf is 0, it does not count to value output. I suspects that the reason why the number of leaves is three is that I have tried to join and leave the groups for myself for three times.

- Before leaving the group



```
❯ curl http://localhost:3000/api/v1/groups/github/gold\?members\=true\&limit\=0\&offse
t\=0
{"data":{"provider":"github","name":"gold","depth":20,"root":"206452320845239182371303
32582565313299278170396728142007307537981380378670486","size":1,"numberOfLeaves":3,"me
mbers":["0","0","179543773567859922141757769625596747054437738838391882026059748595018
02304020"]}}%
```

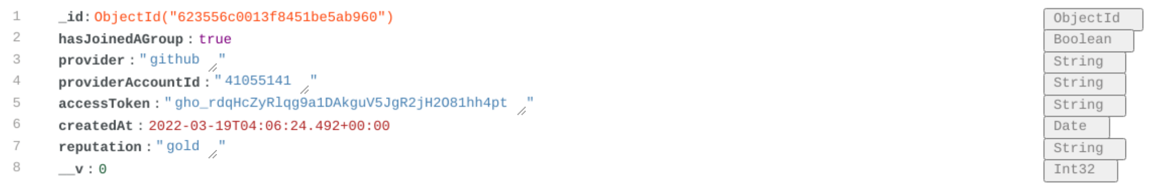
- After leaving the group



```
❯ curl http://localhost:3000/api/v1/groups/github/gold\?members\=true\&limit\=0\&offse
t\=0
{"data":{"provider":"github","name":"gold","depth":20,"root":"150197972326096754419982
60052101280400536945603062888308240081994073687793470","size":0,"numberOfLeaves":3,"me
mbers":["0","0","0"]}}%
```

Question 5.

> If you have a chance to meet with the people who built DarkForest and InterRep, what questions would you ask them about their protocols?



```
_id: ObjectId("623556c0013f8451be5ab960")                    ObjectId
hasJoinedAGroup : true                                        Boolean
provider : "github "                                          String
providerAccountId : "41055141 "                               String
accessToken : "gho_rdqHcZyRlqg9a1DAkguV5JgR2jH2081hh4pt "     String
createdAt : 2022-03-19T04:06:24.492+00:00                     Date
reputation : "gold "                                          String
__v : 0                                                       Int32
```

To InterRep, How can we make this awesome web3 identity linking service to be decentralized, while maintaining the similiar level of privacy? Definitely, as we tested above, the mongoDB from the centralized verifier has the account information, as you see above. The database has a providerAccountId.

In addition, the protocol requires a method to prevent users from minting more than a single reputation NFT from the same Twitter account.

To prevent this, we can use centralized database to be linked so that it is impossible to mint another NFT for that account. However, in this approach, we cannot say that is decentralized.

In addition, is it possible to allow users to change the association, like switching the address that is already linked to the twitter account while maintaining decentralization and privacy? If it is linked to the database, the approach is clear: but it is not decentralization. Lastly but not least, could the platform have the ability to revoke reputation badges if conditions are changed? If then, how in what way?