Question 1: Dark Forest

> In DarkForest the move circuit allows a player to hop from one planet to another.

> Consider a hypothetical extension of DarkForest with an additional 'energy' parameter. If the energy of a player is 10, then the player can only hop to a planet at most 10 units away. The energy will be regenerated when a new planet is reached.

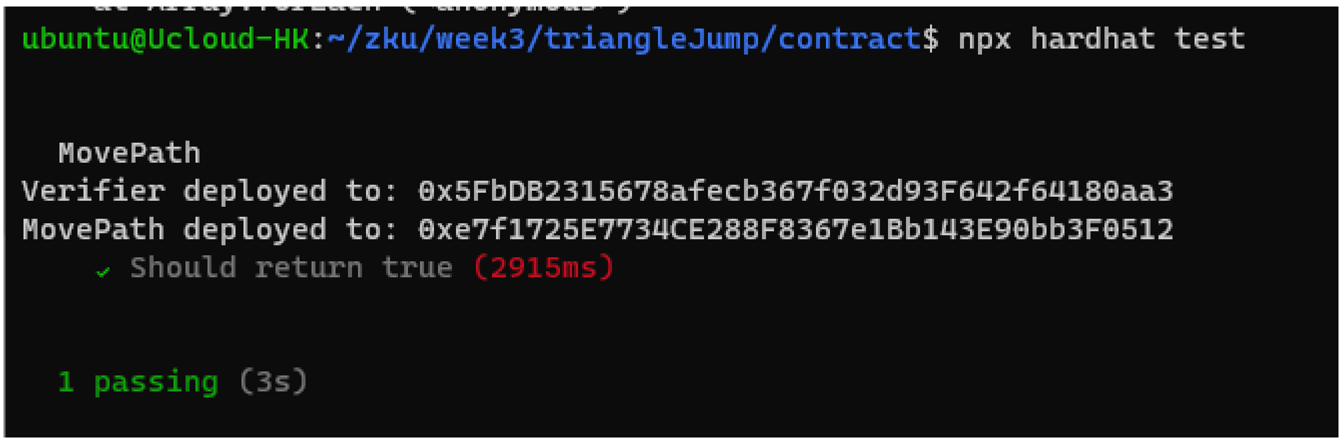
> Consider a hypothetical move called the 'triangle jump', a player hops from planet A to B then to C and returns to A all in one move, such that A, B, and C lie on a triangle.

1. 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.

Code is here

2. [Bonus] Make a Solidity contract and a verifier that accepts a snark proof and updates the location state of players stored in the contract.

contract



```
ubuntu@Ucloud-HK:~/zku/week3/triangleJump/contract$ npx hardhat test


  MovePath
Verifier deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
MovePath deployed to: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
    ✓ Should return true (2915ms)


  1 passing (3s)
```

Question 2: Fairness in card games

1. Card commitment - In DarkForest, players commit to a location by submitting a location hash. It is hard to brute force a location hash since there can be so many possible coordinates.

> 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.

when create commitment, just add a password for a card. this is same to add salt when hash password.

The code is here.

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.

The code is here.

Question 3: MACI and VDF

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

The problems in voting which MACI does not solve:

1. No fully Anonymisation. In MACI,While all votes are encrypted, the coordinator is able to decrypt and read them. We can use rerandomizable encryption to process votes.
2. No an official trusted setup. Every application needs to do a different trusted setup with a specific set of circuit parameters.We can build general circle lib for trusted setup.

2. How can a pseudorandom dice roll be simulated using just Solidity?

1. What are the issues with this approach?

solution A: using blocktime for pseudorandom dice roll.

```
    function randomNum() internal view returns (uint256) {
        return (uint256(blockhash(block.timestamp)) % 6) + 1;
    }
```

The issues:

1. according to the blocktime, Attacker can predict the dice roll
2. Miners can manipulate the blocktime

2. How would you design a multi party system that performs a dice roll?

solution B:

```
//SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.4;

contract diceRoll {
    mapping(address => bool) private submitRecords;
    uint256[] private nums;
    uint256 minSubmit = 5;
    uint256 currentSubmit = 0;

    function submitRandom(uint256 num) public {
        require(!submitRecords[msg.sender]);
        currentSubmit++;
        nums.push(num);
    }

    function roll() public view returns (uint256) {
        require(currentSubmit >= minSubmit);
        uint256 sum = 0;
        for (uint256 i = 0; i < currentSubmit; i++) {
            sum += uint256(blockhash(nums[i]));
        }
        sum += uint256(blockhash(block.timestamp));
        return (sum % 6) + 1;
    }
}
```

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

solution B is fair. In solution B, the result of roll is depend on some inputs:

1. more than 5 random numbers from different address
2. block.timestamp

The more depends on , the more difficult to predict and manipulate

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.

In solution B, if Users share random numbers with miners, so the miners will get all information to manipulate the result of roll. So, we need VDF to fairly generate the random number. The random number generated by VDF can not be predicted before the deadline, it is true random numbers.

The code is as follows:

```

pragma solidity 0.6.2;

import "https://raw.githubusercontent.com/smartcontractkit/chainlink/develop/evm-
contracts/src/v0.6/VRFConsumerBase.sol";

contract VRFTestnetD20 is VRFConsumerBase {
    using SafeMath_Chainlink for uint;

    uint256[] public d20Results;

    bytes32 internal keyHash;
    uint256 internal fee;

    /**
     * @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;
        fee = 10 ** 18;
    }

    /**
     * @notice Requests randomness from a user-provided seed
     * @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");
        bytes32 _requestId = requestRandomness(keyHash, fee, userProvidedSeed);
        return _requestId;
    }

    /**
     * @notice Modifier to only allow updates by the VRFCoordinator contract
     */
    modifier onlyVRFCoordinator {
        require(msg.sender == vrfCoordinator, 'Fulfillment only allowed by
VRFCoordinator');
        _;
    }

    /**
     * @notice Callback function used by VRF Coordinator
     * @dev Important! Add a modifier to only allow this function to be called by
the VRFCoordinator
     * @dev This is where you do something with randomness!
     * @dev The VRF Coordinator will only send this function verified responses.
     * @dev The VRF Coordinator will not pass randomness that could not be
verified.
```

```
     */
    function fulfillRandomness(bytes32 requestId, uint256 randomness) external
override onlyVRFCoordinator {
        uint256 d20Result = randomness.mod(20).add(1);
        d20Results.push(d20Result);
    }

    /**
     * @notice Convenience function to show the latest roll
     */
    function latestRoll() public view returns (uint256 d20result) {
        return d20Results[d20Results.length - 1];
    }
}
```

Question 4: InterRep

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

InterRep uses Semaphore:

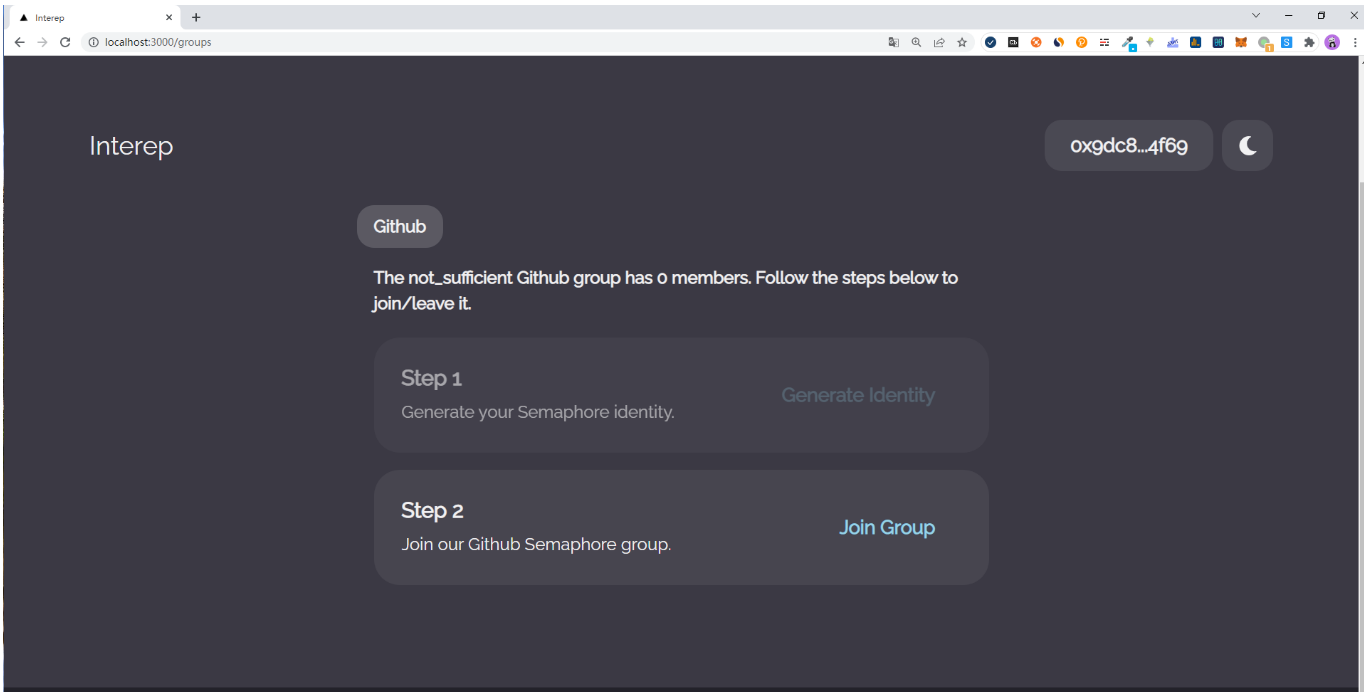
1. users use Semaphore to re-generate their Semaphore identity
2. users use Semaphore identity to create Semaphore proofs to hide their private information
3. with Semaphore proofs, users can use use the anonymous user signals

The reasons for why InterRep still needs a centralized server:

1. a centralized server is need to bridge the web2.0 account to InterRep
2. a centralized server is need to maintain group's data and state such as joining/leaving group

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.

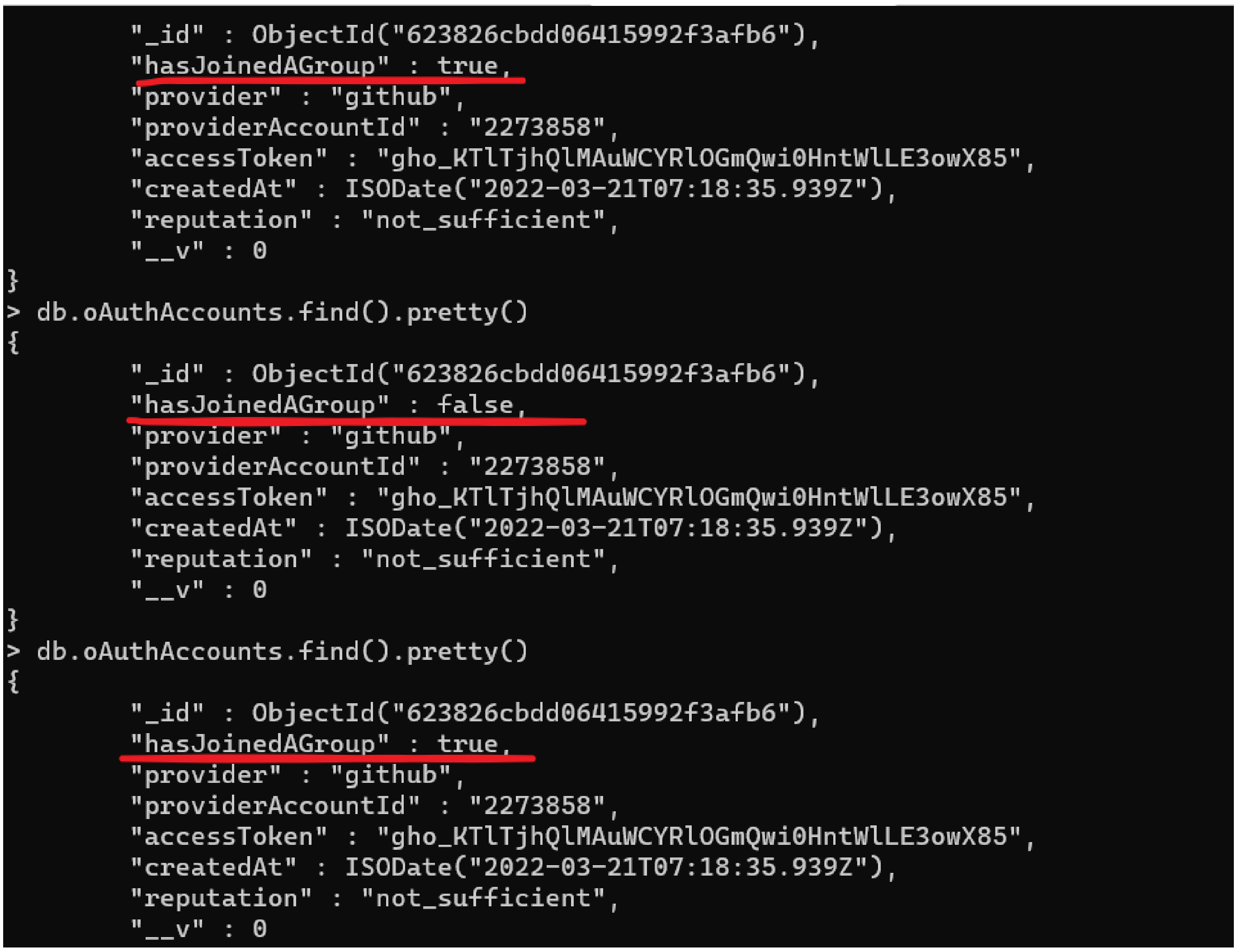
join group:



The not_sufficient Github group has 0 members. Follow the steps below to join/leave it.

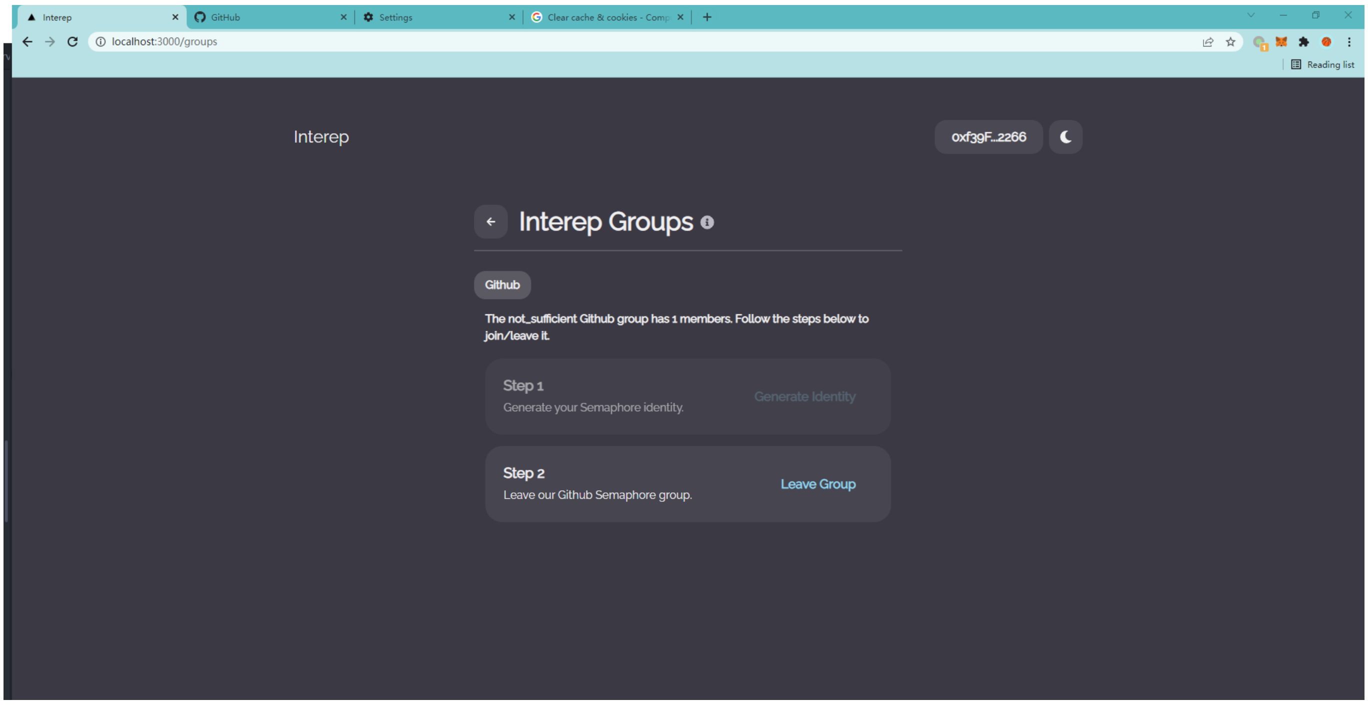
Step 1
Generate your Semaphore identity.
Generate Identity

Step 2
Join our Github Semaphore group.
Join Group



```
        "_id" : ObjectId("623826cbdd06415992f3afb6"),
        "hasJoinedAGroup" : true,
        "provider" : "github",
        "providerAccountId" : "2273858",
        "accessToken" : "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",
        "createdAt" : ISODate("2022-03-21T07:18:35.939Z"),
        "reputation" : "not_sufficient",
        "__v" : 0
}
> db.oAuthAccounts.find().pretty()
{
        "_id" : ObjectId("623826cbdd06415992f3afb6"),
        "hasJoinedAGroup" : false,
        "provider" : "github",
        "providerAccountId" : "2273858",
        "accessToken" : "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",
        "createdAt" : ISODate("2022-03-21T07:18:35.939Z"),
        "reputation" : "not_sufficient",
        "__v" : 0
}
> db.oAuthAccounts.find().pretty()
{
        "_id" : ObjectId("623826cbdd06415992f3afb6"),
        "hasJoinedAGroup" : true,
        "provider" : "github",
        "providerAccountId" : "2273858",
        "accessToken" : "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",
        "createdAt" : ISODate("2022-03-21T07:18:35.939Z"),
        "reputation" : "not_sufficient",
        "__v" : 0
```

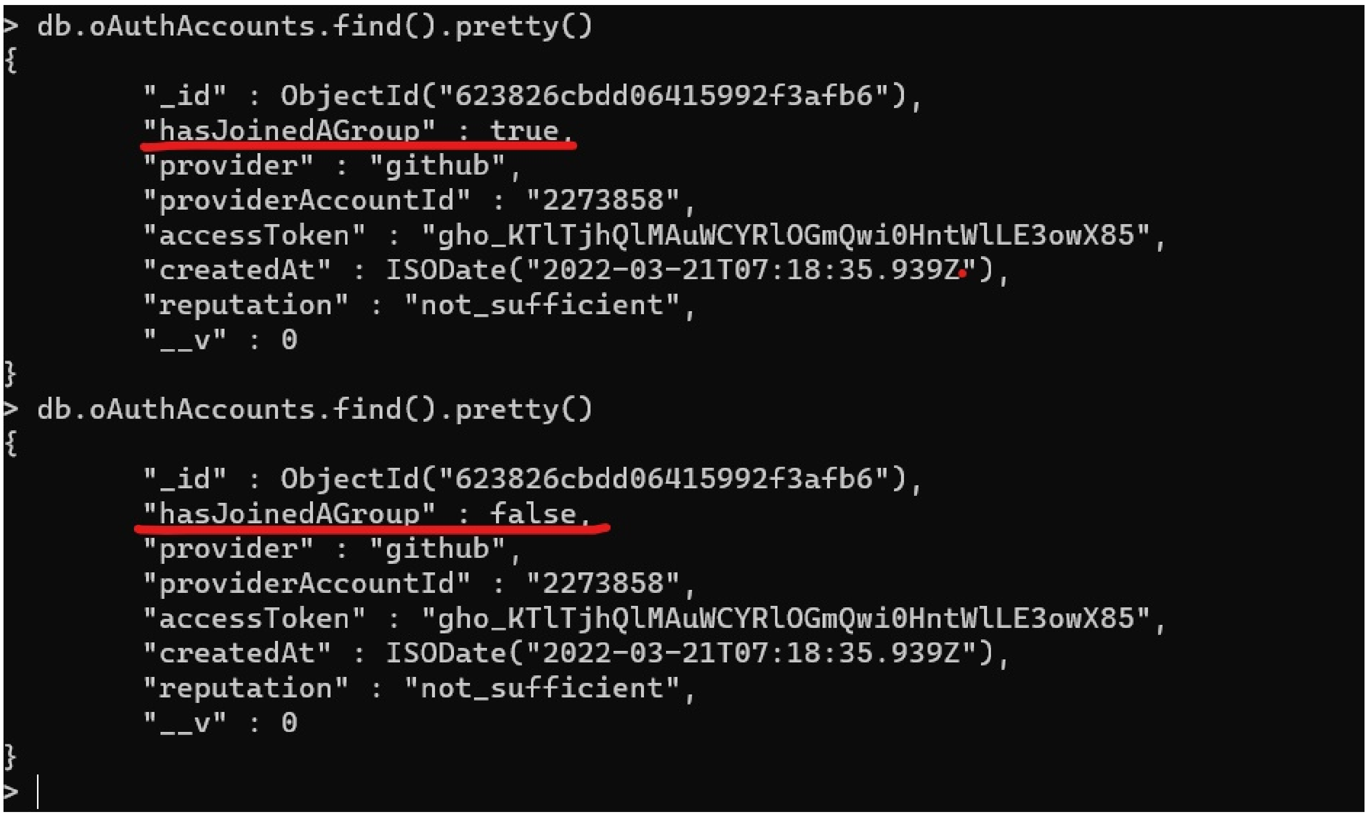
leave group:



The not_sufficient Github group has 1 members. Follow the steps below to join/leave it.

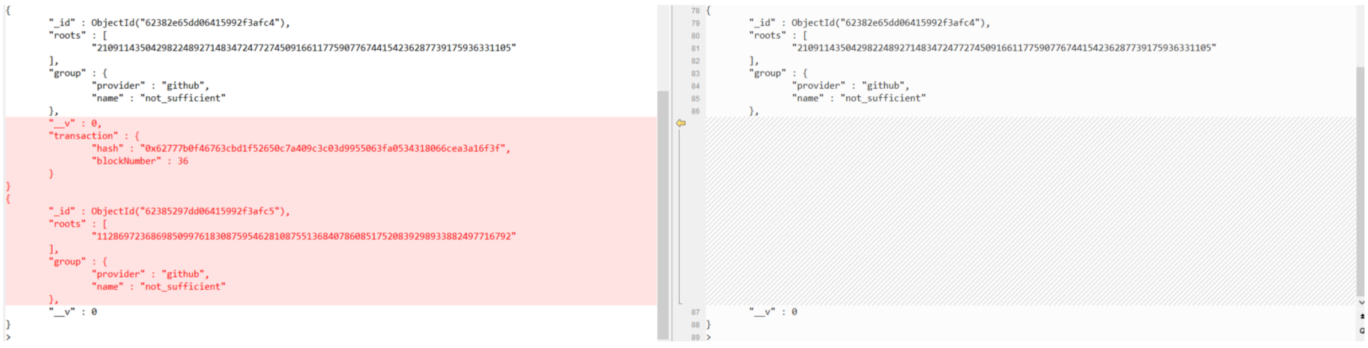
Step 1
Generate your Semaphore identity.

Step 2
Leave our Github Semaphore group.



```
> db.oAuthAccounts.find().pretty()
{
        "_id" : ObjectId("623826cbdd06415992f3afb6"),
        "hasJoinedAGroup" : true,
        "provider" : "github",
        "providerAccountId" : "2273858",
        "accessToken" : "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",
        "createdAt" : ISODate("2022-03-21T07:18:35.939Z"),
        "reputation" : "not_sufficient",
        "__v" : 0
}
> db.oAuthAccounts.find().pretty()
{
        "_id" : ObjectId("623826cbdd06415992f3afb6"),
        "hasJoinedAGroup" : false,
        "provider" : "github",
        "providerAccountId" : "2273858",
        "accessToken" : "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",
        "createdAt" : ISODate("2022-03-21T07:18:35.939Z"),
        "reputation" : "not_sufficient",
        "__v" : 0
}
> 
```

when a number leaves from a group, The Merkle Tree delete the hash of the number from the group's Merkle tree and update the the root of Merkle Tree. the change is as follows：



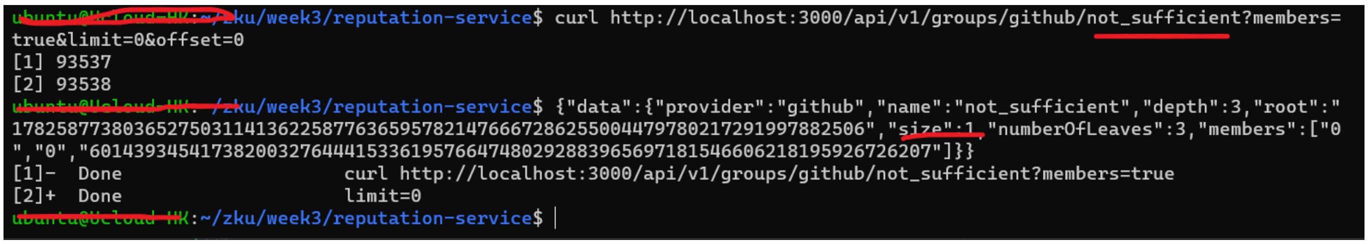
```
{
        "_id" : ObjectId("62382e65dd06415992f3afc4"),
        "roots" : [
                "21091143504298224892714834724772745091661177590776744154236287739175936331105"
        ],
        "group" : {
                "provider" : "github",
                "name" : "not_sufficient"
        },
        "__v" : 0,
        "transaction" : {
                "hash" : "0x62777b0f46763cbd1f52650c7a409c3c03d9955063fa0534318066cea3a16f3f",
                "blockNumber" : 36
        }
}
{
        "_id" : ObjectId("62385297dd06415992f3afc5"),
        "roots" : [
                "11286972368698509976183087595462810875513684078608517520839298933882497716792"
        ],
        "group" : {
                "provider" : "github",
                "name" : "not_sufficient"
        },
        "__v" : 0
}
>
```

```
{
        "_id" : ObjectId("62382e65dd06415992f3afc4"),
        "roots" : [
                "21091143504298224892714834724772745091661177590776744154236287739175936331105"
        ],
        "group" : {
                "provider" : "github",
                "name" : "not_sufficient"
        },
        "__v" : 0
}
>
```

InterRep contracts address in Kovan Testnet Network

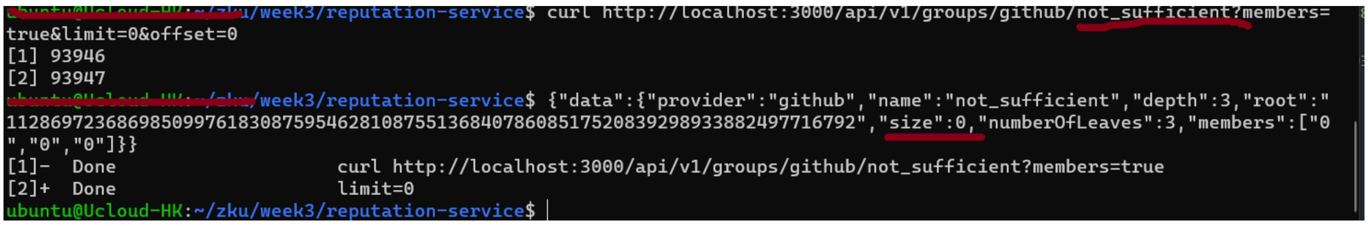
3. Use the public API (instead of calling the Kovan testnet, call your localhost) to query the status of your own identityCommitment in any of the social groups supported by InterRep before and after you leave the group. Take the screenshots of the responses and paste them to your assignment submission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

one user join the group:



```
[redacted]ku/week3/reputation-service$ curl http://localhost:3000/api/v1/groups/github/not_sufficient?members=
true&limit=0&offset=0
[1] 93537
[2] 93538
[redacted]ku/week3/reputation-service$ {"data":{"provider":"github","name":"not_sufficient","depth":3,"root":"
17825877380365275031141362258776365957821476667286255004479780217291997882506","size":1,"numberOfLeaves":3,"members":["0
","0","6014393454173820032764441533619576647480292883965697181546606218195926726207"]}}
[1]-  Done                    curl http://localhost:3000/api/v1/groups/github/not_sufficient?members=true
[2]+  Done                    limit=0
[redacted]:~/zku/week3/reputation-service$ 
```

one user leave the group:



```
[redacted]/week3/reputation-service$ curl http://localhost:3000/api/v1/groups/github/not_sufficient?members=
true&limit=0&offset=0
[1] 93946
[2] 93947
[redacted]/week3/reputation-service$ {"data":{"provider":"github","name":"not_sufficient","depth":3,"root":"
11286972368698509976183087595462810875513684078608517520839298933882497716792","size":0,"numberOfLeaves":3,"members":["0
","0","0"]}}
[1]-  Done                    curl http://localhost:3000/api/v1/groups/github/not_sufficient?members=true
[2]+  Done                    limit=0
ubuntu@Ucloud-HK:~/zku/week3/reputation-service$ 
```

Question 5: Thinking in ZK

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

Questions for InterRep:

1. Which is better to use for reputation, the account from web2 or the native account from web3 such as debank Web3 Social Ranking?
2. How to measure a user's reputation in your protocol?

reference

1. Assignment 3
2. zkDAO – Succinct, Private, Fair
3. zku.ONE – 2022 Mar-April Syllabus / Course Schedule