Assignment 3

Question 1: Dark Forest

Question 1-1:

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.

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.

Answer 1-1:

Link: https://github.com/amrosaeed/zku/blob/main/Q1/Triangle_Jump.circom

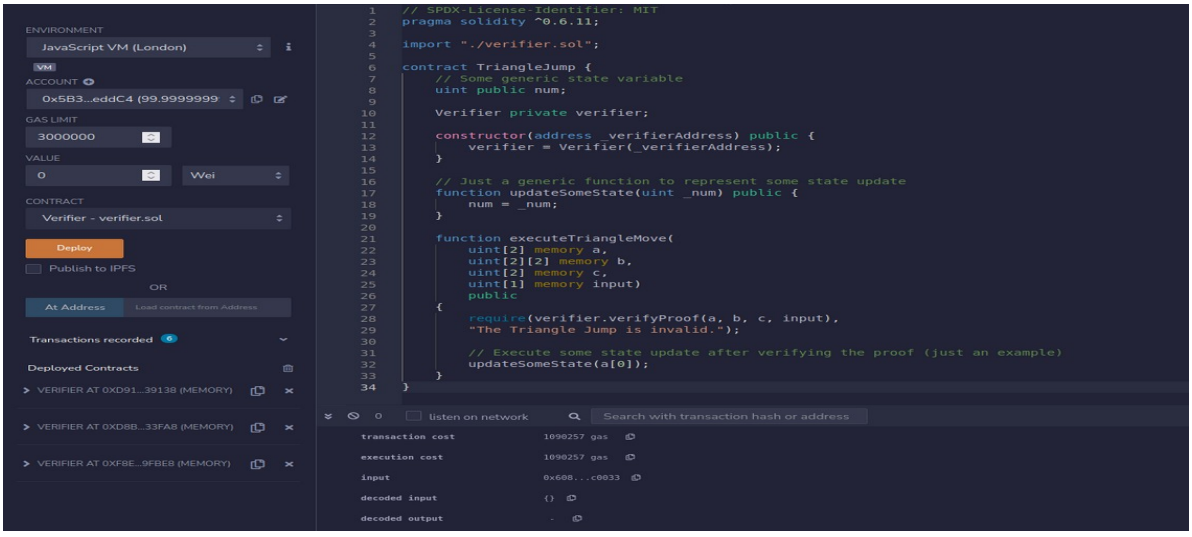
Question 1-2:

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

Answer 1-2:

Link: https://github.com/amrosaeed/zku/blob/main/Q1/verifier.sol

Link: https://github.com/amrosaeed/zku/blob/main/Q1/TriangleJump.sol



Question 2: Fairness in card games

Question 2-1:

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.

Answer 2-1:

In the case of Dark-Forest the location coordinates format makes its difficult to be mapped when they hashed out, however in the case of a deck of cards if we would to map cards to integers from 0 to 51 that gives us only 52 possible mappings, fortunately the deck of cards can be distinguished and mapped to 4 types of suites (S,T,H,K), where every suite can be mapped to integers from 0 to 12, In addition the 52 cards get mapped to nullifiers each, In that it would difficult to brute forced since we need combination of the card number, suite and a nullifier.

Commitment Design:

```
50      /*This circuit template CommitCard() Assigns Commitments to combination of Suites, Number and Nullifiers.*/
51
52 template CommitCard() {
53
54     signal input Suite;
55     signal input Number;
56     signal input Nullifier;
57
58
59     signal output Commitment;
60
61     /* Use template MiMCSponge(nInputs, nRounds, nOutputs) to hash (Suite, Number and NUllifier).*/
62
63     component mimc = MiMCSponge(3, 220, 1);
64     mimc.ins[0] <== Suite;
65     mimc.ins[1] <== Number;
66     mimc.ins[2] <== Nullifier;
67     mimc.k <== 0;
68
69     /* Commit output of MIMC hash.*/
70
71     Commitment <== mimc.outs[0];
72 }
```

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.

Answer 2-2:

Appropriate representation of cards as integers

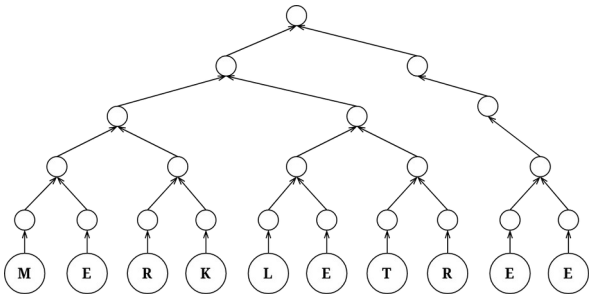
SUITE	INTEGER
TRIFLE	0
DIAMOND	1
HART	2
SPADE	3

2^2^13

CARD	INTEGER
2	0
3	1
4	2
5	3
6	4
7	5
8	6
9	7
10	8
J	9
D	10
K	11
ACE	12

2^26^52

NULLIFIER
0
1
2
3
4
5
6
...
...
...
...
...
51



2^78 Hash Commitments

Can the previous state be spoofed? The previous state consists of 2^78 Hash Commitments, it is 2^78 multiplied by duration of a single attack, if one billion attacks per 1 millisecond, it would take over 9000 years

What mechanism is needed in the contracts to verify this? A validation key `cardSuite_0001.zkey` called and outputs Solidity code in a file named `verifier.sol, The Verifier has a view function called verifyProof that returns TRUE if and only if the proof and the inputs are valid.`

Design a contract, necessary circuits, and verifiers to achieve this?

Link: https://github.com/amrosaeed/zku/blob/main/Q2/cardSuite.circom

Link: https://github.com/amrosaeed/zku/blob/main/Q2/verifier.sol

Link: https://github.com/amrosaeed/zku/blob/main/Q2/cardSuite.sol

Question 2-3:

[**Bonus**] How can a player reveal that it is a particular card (Say ace) without revealing which suit it belongs to (ace of diamonds etc.)

Answer 2-3:

Question 3: MACI and VDF

Question 3-1:

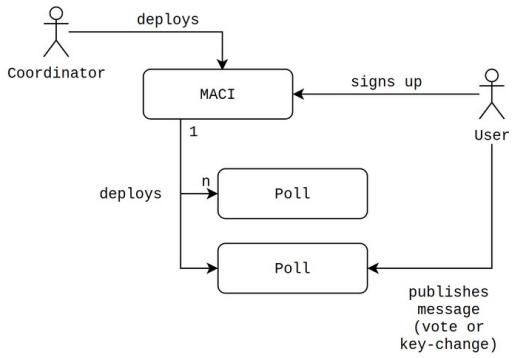
What problems in voting does MACI not solve? What are some potential solutions?

Coordinator can decrypt & censors messages

- First: the existence of a trusted coordinator that in theory can change the way a voter voted, he can decrypt messages and censor a message (n) and exclude it from the tally, so coordinator can prove to a briber how he altered votes
- Solution: A suggested upgrade to MACI is to use ElGamal re-randomisation for anonymity of voters. While all votes are encrypted, the coordinator would not be able to tell which user took which action.

User submits a message with an invalid signature

- Second: ease of use, coordinators have to interface with all deployed contracts and manage tallies for multiple polls, the coordinator have to verify the user's message's signature off-chain as if the user submits a message with an invalid signature and the coordinator does not do that, they will fail to generate a valid proof.
- Solution: MACI 1.0 support multiple polls within a single instance of MACI a coordinator tooling that allows users to generate proofs and query inputs from existing circuits through an easy-to-use API



Sybil attack on the message tree

- Third: As every user should be allowed to publish as many messages regard their vote options with no cost, in theory this render the coordinator to generate enough proofs to process them all a very high cost, and there is no way to enforce a limit on users without allowing a bribery to do so.
- Solution: I don't find more information about this particular issue.

One wallet – One vote

- Fourth: An attacker can create as many wallets as enough to launch a 51% attack, like in case of Interep

- Solution: Existence of a centralized authority of certificates is a must to assign 1 vote – 1 identity instead of 1 wallet – 1 vote.

however a lot of arguments that defends the UN-feasibility of the above attacks economically at this link :

https://hackmd.io/@OFccBlU5TNCiRhpIyT1m7g/SkXv-gO5r#Glossary

Question 3-2:

How can a pseudo-random dice roll be simulated using just Solidity?

Answer 3-2a:

Solidity is a DSL language for the EVM, In theory its always possible to front-run a VM generated pseudo-randomness as the VM itself runs on top of the OS and have no access to the underlying hardware, In addition as in case of solidity the EVM runs on multiple nodes and must reach consensus before committing to state, so it will always be vulnerable to both pre-date and post-date attacks.

The most agreed upon solution is to inherent the blockchain randomness and leverage it as a beacon, however all blockchains fails the test of randomness except for the bitcoin chain, and that is mainly due to its unique VM and consensus rules that render any type of malicious attack a loss, (In theory its both impossible to both prove or deny Bitcoin uniqueness as source of true randomness) but in practice it stood in the face of all front-running attacks even by the miners themselves contrary to the Ethereum chain.

Naive protocol solution:

Use of the most recent block hash.

Block.timestamp & Block.difficulty

```
A block.timestamp is assigned by the miner whenever he confirms the transaction. No
player of our Lottery contract is able to control it. Let’s take a look at this
piece of code for creating a random number.
```

```
function random() private view returns (uint8) {
  return uint8(uint256(keccak256(block.timestamp, block.difficulty))%251);
}
```

Question 3-2a:

What are the issues with this approach?

Answer 3-2a:

This piece of code first hashes a block timestamp and difficulty. Next, we convert the hash to an integer and divide it by 251 to get an integer between 0 and 250. However, the problem with this piece of code is that we shouldn't trust a miner for picking a winner.

Question 3-2b:

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

Answer 3-2b:

Allow both parties to choose a nullifier and hashing them together with the most recent block hash.

Question 3-2c:

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

Answer 3-2c:

The second way is fairer because parties don't have incentive to collude and as the pseudo random number depends on opponents sending inputs, it makes it harder to guess the "random" generated number on chain.

Question 3-2d:

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.

Answer 3-2d:

- parties can collude and with their inputs and front running everyone to the most recent block hash, they can guess the pseudo random generated number not in favor of everybody else.
- The last one-join attack, in that case the last user has an idea about the previous user's inputs

Question 3-3:

[**Bonus**] How would two players pick a random and mutually exclusive subsets of a set? For instance, in a poker game, how would two players decide on a hand through the exchange of messages on a blockchain?

The solution proposed: (a project to work on with ZKU Harmony)

The idea is to allow access to a protocol as a black box for time-stamping cryptographic primitives with respect to a global clock “Bitcoin chain” for authenticating informations, such as digital signatures, non-interactive zero-knowledge proofs and proofs of knowledge.

For such protocol to allow parties to have a bounded-drift view of a global clock “Bitcoin chain”

Proposed protocol solution: (To be continued)

The Clock functionality: G_{clock}, is to enable synchronous execution of smart contracts by parties in synchronized bounded by window rounds, it keeps track of round variable whose value can be retrieved by parties “smart contracts” via sending to it the pair: CLOCK-READ. This value increased for every honest party has sent to the clock a command CLOCK-UPDATE. Once any party has executed all its instructions for that round it instructs the clock to advance by sending a CLOCK-UPDATE command, and gets an idle mode where it simply reads the clock time in every activation until the round advances

The Random Oracle Functionality: $F_{RO,}$ querying the output SHA 256 hash function “like the block header of a block after 3 blocks deep in chain history; a hash chain” is assumed to be true random presuming the underlying randomness of the “Bitcoin chain itself even against the miners” as stated above the impossibility of proving otherwise theoretically as in practice it stand in face of the most important test “time” with value more than 1 trillion $ against all types of attacks “only in case of 3 blocks deep in chain history”

The Global Ledger Functionality: $G_{ledger,}$ the above two functionality only viable aganist post-dated attacks, however to ensure that any timed cryptographic primitives with this black box stand against pre-dated attacks, a write to the chain must take place as transaction.

System Architecture

Question 4: InterRep

Question 4-1

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

Answer 4-1:

From web2 – web3:
InterRep needs a centralized server for only one goal and that is to make a simple http API to check a Twitter account's reputation, they sacrifices some decentralization to gain more privacy, instead of querying using JSON-RPC thats open on every node in the network

From web3 – web2:
They used semaphore to hide identities "eth-accounts" attached to twitter handles back to the centralized sever.

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.

Answer 4-2:

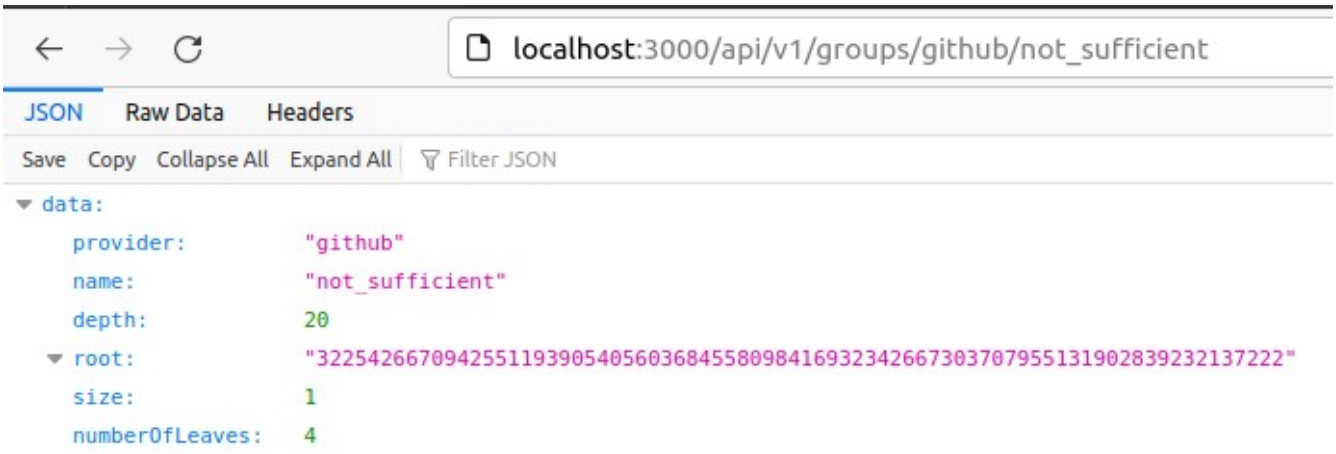
Hash of my commitment was deleted from the group tree

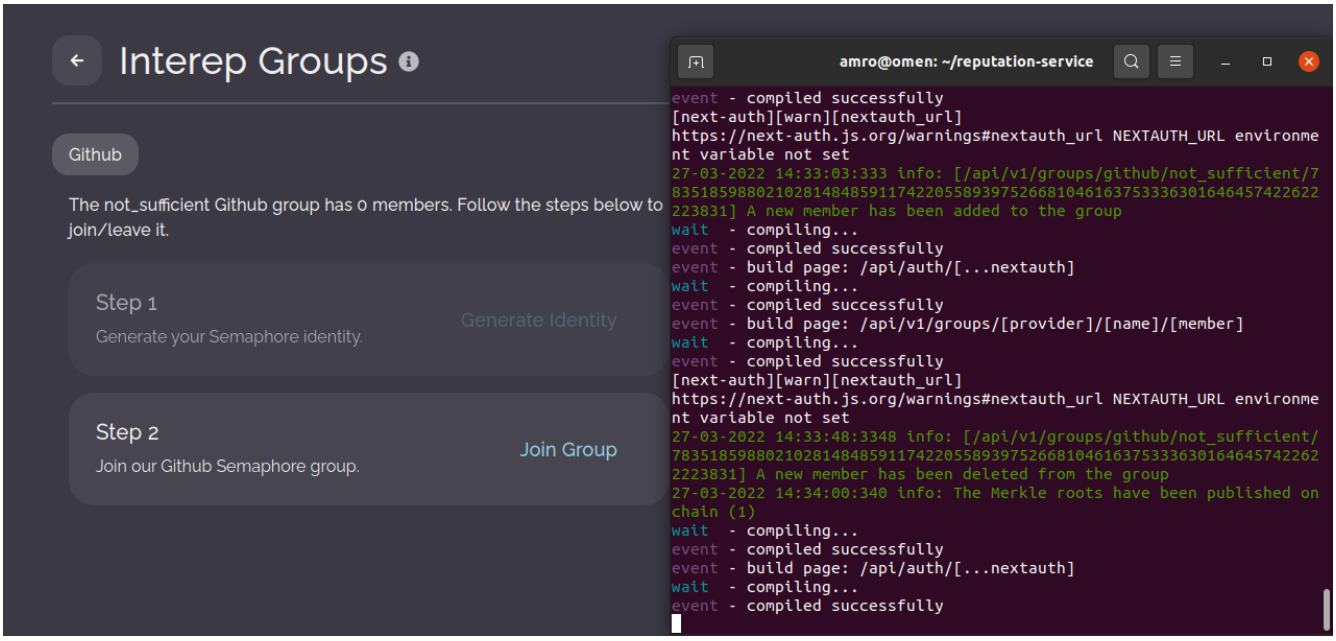
Question 4-3:

Use the public API (instead of calling the Kovan testnet, call your localhost) to query the status of your own identity Commitment 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.

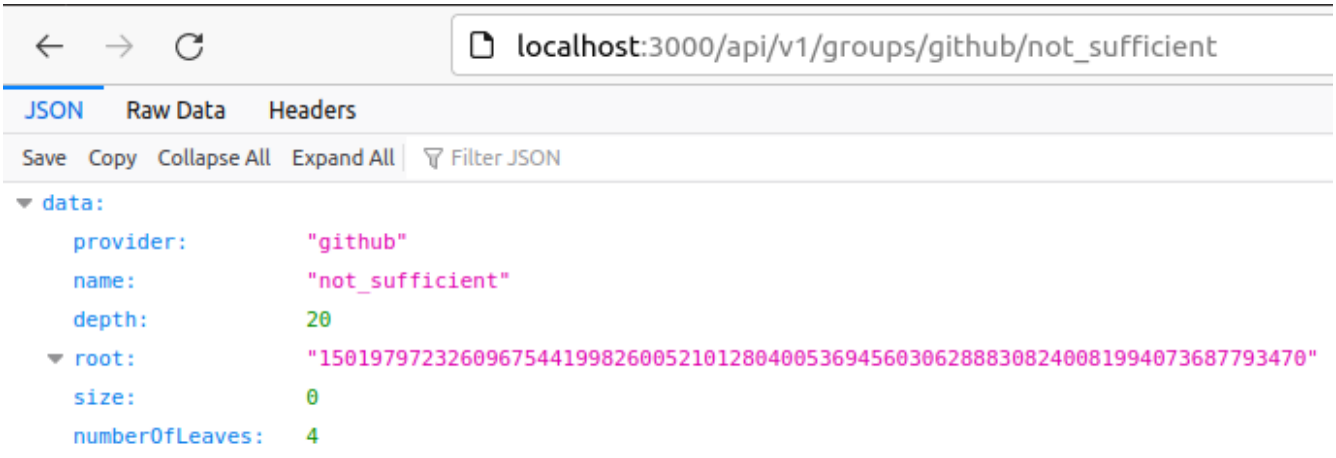
Answer 4-3:

I used Github's public api to call my localhost After joining the group





I used Github’s public api to call my localhost After leaving the group



Question 4-4:

[**Bonus**] Suggest a viable solution to make InterRep completely decentralized.

Answer 4-4:

For the centralized “Back-end” server, I would suggest deploying the server on Akash network a decentralized AWS alternative, using Akash’s DSL they can set the server deployment in any friendly borders and to be redeployed swiftly in case of hostile jurisdictions ,DSL also allows for multi-sig control over the deployment.

Example: https://github.com/amrosaeed/Akash-Hackathon

For the centralized “Front-end” I would suggest leveraging skynet’s Homescreen implementation & skyID, with that in place new Front-End can use https API to query web2 platforms, adding to that preventing against dev teams going rouge and trying to alter the front-end and limiting access to certain platforms, by using Homescreen and skyID every user would be able to store his/her own front end, even if the dev team update it, they will own their version of Front-End that will always query the existing smart contracts on the chain.

Example: https://github.com/amrosaeed/Terra-Challenge

Question 5: Thinking in ZK

Question 5-1:

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

Answer 5-1:

For InterRep team:

- Have you ever considered API3’s Airnode solution philosophy with allowing end-points “End-Users” to deploy there own serverless API nodes and push data to their identities on chain without the need of a collective centralized server to fetch data from them on their behalf, and what would be selfish-attack scenarios in that case?

For Dark-Forest team:

- Have you ever thought about integrating dark forest story in a mixed-reality type of game, with AR in a shared spatiality and temp-orality metaverse scheme around city scenes?

- Is the above suggested game experience/ecosystem, would it be viable in case of existence of both stateless node and native proof of location, or stateless node has no use and a minimum bar of light nodes would be viable querying the chain?