Assignment 3

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.

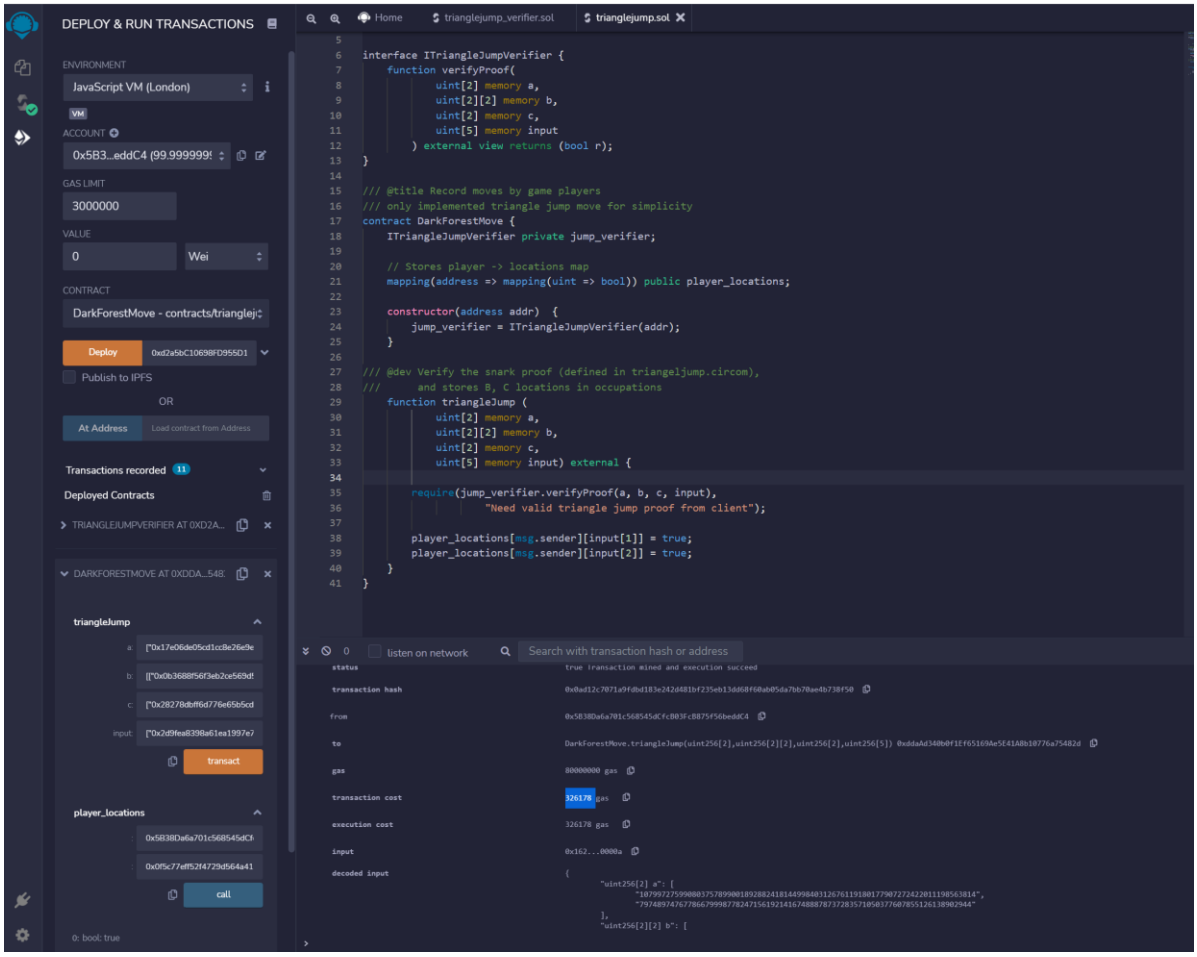
Consider a hypothetical move called the 'triangle jump', a player hops from planet A to B then to C and returns to A all in one move, such that A, B, and C lie on a triangle.

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.***

   [Answer]
   https://github.com/geesimon/zku/blob/main/week3/circuits/trianglejump.circom
   https://github.com/geesimon/zku/blob/main/week3/circuits/rangecheck.circom

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]
   https://github.com/geesimon/zku/blob/main/week3/contracts/trianglejump.sol



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 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] When player received a card from dealer, the card can be represented by a hash of 3 fields: (user generated random) nullifier, suite and number. Then the hash stored onchain as commitment for this card. Please refer to template:

CardCommitment in circuit:
https://github.com/geesimon/zku/blob/main/week3/circuits/card.circom

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] circuit:

   https://github.com/geesimon/zku/blob/main/week3/circuits/card.circom

   https://github.com/geesimon/zku/blob/main/week3/circuits/cardcommit.circom

   Contract (demo code, not full implementation)**:**
   https://github.com/geesimon/zku/blob/main/week3/contracts/cardgame.sol

   To prevent user from spoofing the state, we need to use circuit (template: CheckSuite) to verify user owns both cards and the two cards belong to same suite. In addition, the contract needs to record all commitments for the deal cards on chain and the status the played cards (by enum CardStatus) to avoid double-playing of same card.

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] see template:RevealNumber in
   https://github.com/geesimon/zku/blob/main/week3/circuits/card.circom

3. MACI and VDF

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

   [Answer] MACI can provide collusion resistance only if the coordinator is honest. Although a dishonest coordinator can neither censor nor tamper with its execution, it knows all actions taken by each voter. Current MACI implementation cannot prevent the dishonest coordinator from coordinating the collusion behavior. Ideally, we'd like a situation where the coordinator is responsible only for anti-collusion and doesn't know which user took what action.

One possible solution is use VDF to delay the user action proof generation until the end of the vote and use smart contract to verify and compute results. This eliminates the need of a centralized coordinator.

Another solution is re-randomization (ElGamal Encryption) to re-encrypt the voting message from being decrypted by coordinator. Coordinator only responsible for key management and can prove a voting message is encrypted by a user's key. The encrypted voting message and proof is published on chain for computing final voting result.

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

   [Answer] This pseudorandom can be easily implemented by using (pseudo) random source, like timestamp, blockhash. For example: dice_number = block.timestamp % 6.

   1. *What are the issues with this approach?*

   [Answer] The problem with this approach is the random result is generated by individual miner before committing to block chain and the miner thus can take this advantage to only commit the number that favorite his benefit.

   The random number is stored on chain and thus is public. This means the number should not be generated until after the entry into the (lottery) game has been closed. Otherwise, other players can take benefitable action based on earlier published numbers.

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

   [Answer]

   Phase 1: Participants generate random number

   1. The coordinator sets aside a reward for a random number. A reward is necessary to foster competition among participants.
   2. Each Participant generates their own secret random number N.
   3. Participants create their commitment by hashing their N with their address: hash = sha256(N, msg.sender).
   4. Participants send their hash to the contract.
   5. Submissions continue until sufficient participants have joined.

   Phase 2: Reveal round,

   1. Once the submission round has ended, each participant submits their random number N to the contract.
   2. The contract verifies that the sha256(N, msg.sender) matches the original submission. If a participant does not submit a valid N in time, his deposit is forfeit.

Phase 3: Final random number generation,

1. The Ns are all hashed together: sha256(N1, N2…Nn) to generate the resulting random number.
2. Valid participant participants receive the reward. (Reward / num of valid participants). This gives more incentive to be an honest participant since hide the secret can kick-out dishonest participants and thus win more rewards.

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

[Answer]

The second technique is fairer because compared with the simple hash of block.timestamp, these 3 phases approach makes the prediction harder (although not completely impossible, see description below) and not beneficial for dishonest participants.

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*.

[Answer]

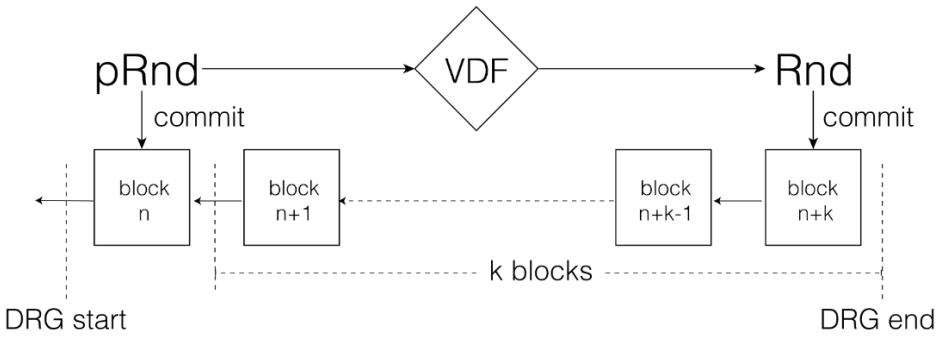
The above multi-party system is vulnerable to last-revealer-attack. In phase 2, the last revealer can still leverage all the already revealed numbers on chain and decide whether he wants to reveal his own number, thus benefit from avoiding the losing game. One workaround is to have a coordinator to be the last revealer, but this makes such coordinator a central point of trust.

To overcome this problem, we can introduce VDF in phase 3 (also illustrated in the below figure, borrowed from harmony web site),

1. Once all numbers are committed on chain and a smart contract calculates pRnd with hash of all random numbers: pRnd = Hash(N1…Nn).
2. A VDF is executed to generate the final random number: Rnd = VDF(pRnd, T) and its proof.
3. Each participant then validates VDF proof and make it the result committed to block chain.

The VDF is used to provably delay the revelation of final Rnd and prevent the last participant from biasing the randomness by choosing to submit the wrong random number. Because of the VDF, the last participant won't be able to know the actual final randomness before pRnd is calculated and committed to the blockchain. By the time Rnd is computed with the VDF, pRnd are already committed in a previous block so any participant cannot

manipulate it anymore. Therefore, the most a malicious participant can do is to either blindly reveal a random number or trying to delay the phase 3 by not revealing its random number. The former is the same as the honest behavior. The latter won't cause much damage as the timeout and competition mechanism will be triggered to forfeit this process or kick-out the dishonest participant.



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?***

[Answer]

We first design an exclusive random sets generator for poker cards using the following pseudo code,

```
Let CardSet = Shuffle(1..52); //CardSet stores random
Let Set1 = ();
Let Set2 = ();

for each i in [0..numOfElement) {
    Let n1 = random() % (52 - i * 2);
    Set1.push(CardSet[n1]);
    CardSet.remove(n1); //Remove the element from the CardSet

    Let n2 = random() % (52 – (i * 2 + 1));
    Set2.push(CardSet[n2]);
    CardSet.remove(n2);
}

Return (Set1, Set2);
```

Set1 and Set2 will be mutually exclusive 2 sets. To make this function complete random on chain, we can make the above function a VDF and process described above to avoid collusion and randomness prediction.

4. InterRep

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

   [Answer]

   Semephore is used for 3 purposes:

   1. *Create identity commitment to represent an on chain registered user.*
   2. *Manage on chain group and its membership. In InterRep, off chain group is mapped to Semaphore on chain group, thus can be managed by Semaphore.*
   3. *Generate zk-snark proof to verify a user belongs to a specific on chain group (reputation)*

      Centralized server is major used to support off chain group operations. The centralized (node.js) web server needs to get authorization from user to access traditional web services (Twitter, Github, Email, Telegram) and then retrieve user info stored in these services to compute reputation. MongoDB is also used to store (OAuth) access token, reputation and merkle tree of these off-chain user accounts.

      Since a lot of computation and storage are in done in centralized server, I guess this design is more gas efficient.

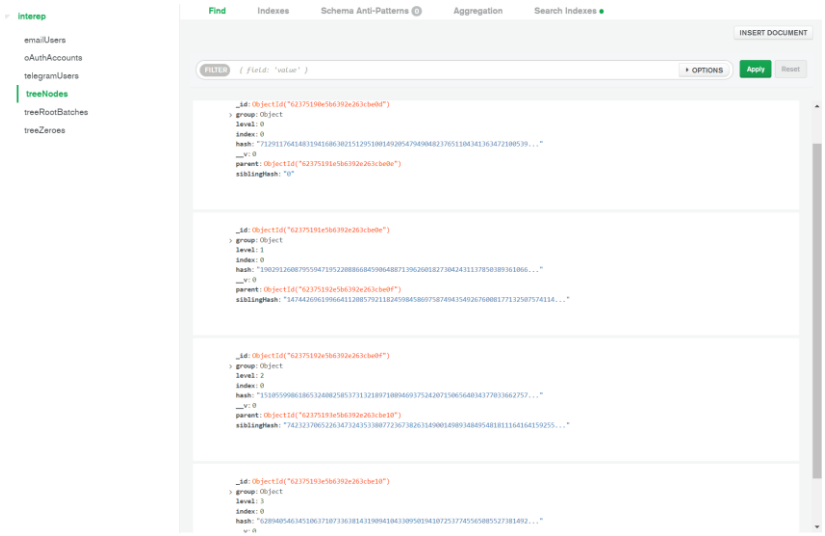
2. ***Clone the InterRep repos: contracts and reputation-service. Follow the instructions on the Github repos to start the development environment. Try to join one of the groups, and then leave the group. Explain what happens to the Merkle Tree in the MongoDB instance when you decide to leave a group.***

   [Answer]

   Join a group (assuming MERKLE_TREE_DEPTH = 3),

   - If this is the first user in the group, a leaf node and 3 ancestor nodes (MERKLE_TREE_DEPTH + 1) are created in treeNodes (collection). hash is computed using this user data, for empty sibling, the zero hashes represent each level of parent (stored in treeZereos) is used.

- Otherwise, a new node and related parent nodes are created, and their hashes are updated accordingly.
- Smart contract Interep::updateOffchainGroups is called to update the new merkle root of this group. Once done, the smart contract transaction is logged to treeRootBatches.

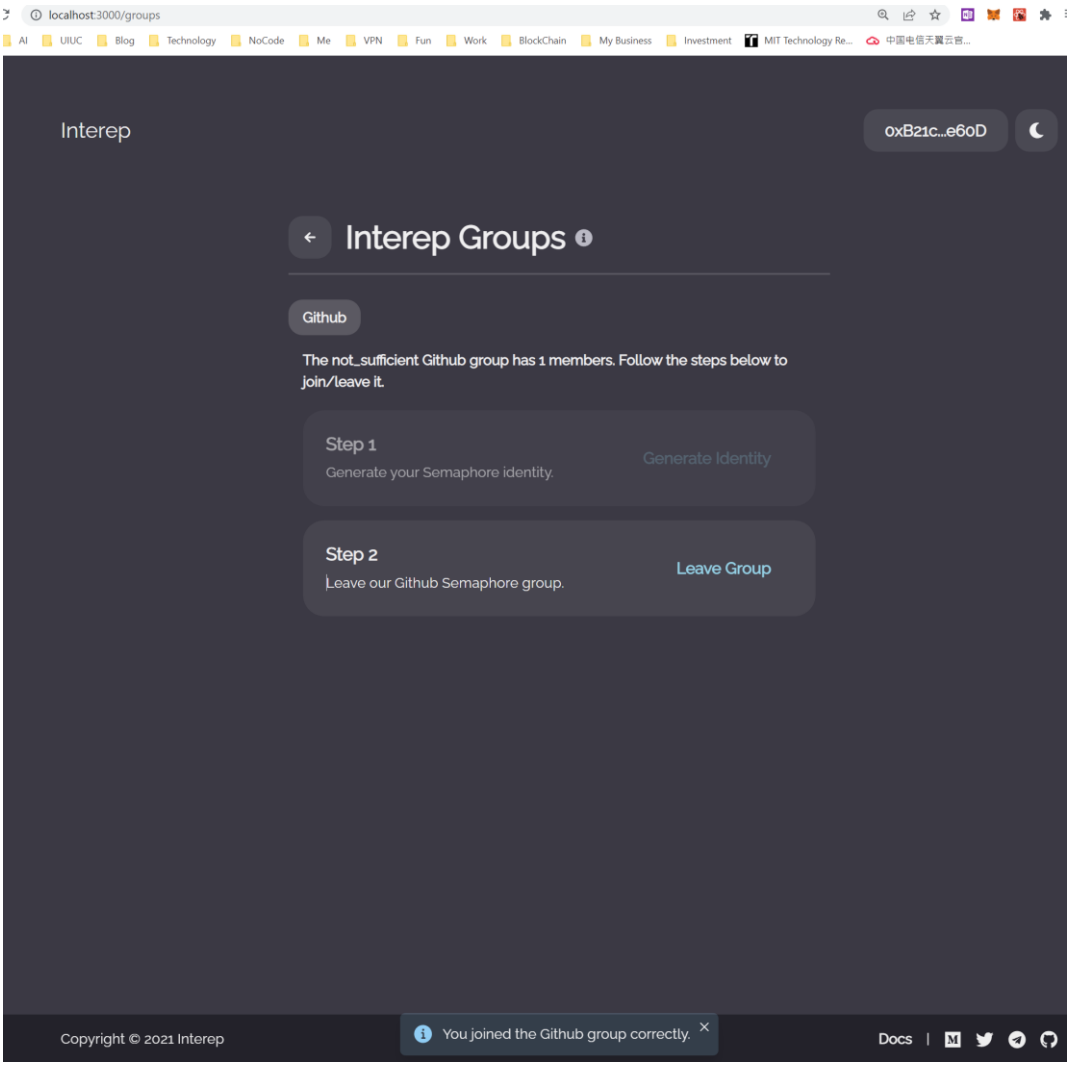


Leave a group

- The hash value of the old leaf is cleared with 0, and the hashes of its ancestor nodes are also updated accordingly.
- Smart contract Interep::updateOffchainGroups is called to update the new merkle root of this group. Once done, the smart contract transaction is logged to treeRootBatches.

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.***

   [Answer] Join a github group,



After joining this group,

```
(base) simon@home-win:~$ curl http://localhost:3000/api/v1/groups/github/not_sufficient/712911
7641483194168630215129510014920547949048237651104341363472100539593985
{"data":true}(base) simon@home-win:~$
```

After leaving this group,

```
(base) simon@home-win:~$ curl http://localhost:3000/api/v1/groups/github/not_sufficient/712911
7641483194168630215129510014920547949048237651104341363472100539593985
{"data":false}(base) simon@home-win:~$
```

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

[Answer] To make InterRep completely decentralized, I think we need to solve 2 major issues,

1. How to retrieve user info on traditional web services (github, twitter). A possible solution is to use Chainlink and make a data feed provider on this

decentralized platform. The provider then retrieves user data from traditional web service.

2. The data provider also needs use an access token to retrieve user info. This access token needs to be generated by each user in browser, then encrypted by the public key of the data provider and stored on chain.

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?***

[Answer]

For DarkForest,

1. It could take long time for user to figure out a valid coordinate and thus may make the game a little boring, can we design a better reward system to encourage more user engagement?
2. Is it possible to integrate NFT to this game so that user can exchange their resource (planet coordinate, resource in this planet) securely and confidentially?
3. In general, what major scenarios can zkp empower a game that is not possible before?

For InterRep,

1. Any real Dapp application that leverages InterRep? What could be the major difficulties when using InterRep in real Dapp?
2. Is it possible InterRep can be extended as a new decentralized credit system like the centralized ones broadly used today (Equifax, Experian, and TransUnion)? What could be the major challenges to implement such system?
3. Is it possible to make a reverse-InterRep, i.e., grant reputation to traditional services (Twitter, Github) based on users on chain behavior?