Final project

Protocol:

Phase 0 (t_0): Amir will start the block by choosing the number of choice k for voting, also initialise the _levels, _hasher, _verifier parameters for the zero knowledge proof in the constructor

Phase 1 $(t_0 \sim t_1)$: Sign up and deposit **(requirement a, c, d)**

- All sign up must be finished at the first 60 minutes → starting block ~ starting block + 300 blocks (assume each block generates every 12 seconds)
- Require all voters to deposit 1 ether at sign up phase. Each voter could keep depositing 1 ether each time as we can make sure the deposit is integer
- After t_1, no one is able to deposit anymore, meaning that the number of vote for every person is locked

Phase 2 $(t_1 \sim t_2)$: Commitment phase **(requirement e, f)**

- All sign up must be finished at the first 60 minutes → starting block ~ starting block + 300 blocks (assume each block generates every 12 seconds)
- We have to check the value of deposit_used_for_voting < deposit map of the voter, if we confirmed that the deposit used in voting is less than deposit, they can proceed
- We have to ensure the vote is unique as to indicate that the commited_vote haven't been revealed by others(even though the chances of $hash(choice + nonce)$ having duplicated values in the map is very low, we still have to prevent it in order to make the zero knowledge proof works). Hashing the vote with a nonce also prevent voters from front running attack.

- Before the function ends, it will record that the voter had used 1 ether for voting

Phase 3 $(t_2 \sim t_3)$: Reveal phase **(requirement h)**

- Voters will call the revealingVote() function with a dummy account, they won't use the original deposit account to prevent revealing their identity
- In the off-chain, we will have a zk-SNARK circuit to generate the proof that the vote knows a particular hashed value in the commited_vote map
    - We will provide the public secret: committed_vote and the private secret: the vote and nonce to the zk-SNARK. After the SNARK circuit verifies that the committed_vote matches the private secret, it will return proof_a, proof_b and proof_c
- In the on-chain(the solidity file), when the voters call the revealingVote function, they have to provide the proof_a, proof_b and proof_c
- However, notice that they might have unmatched choice in phase 3 when comparing to phase 2. But this is will be prevented because when the voter tries to prove what he voted for others, they have to also provide the value of the vote. If the vote don't matches the one in commitment phase, the SNARK circuit will return an error.

Phase 4 $(after\ t_3)$: Refund phase **(requirement g)**

- Everyone can refund their deposit if they have deposit into the contract before, it will be withdrawn based on the value in the deposit map.
- A reentrancy lock is crucial to prevent reentrancy attack because it only let one person getting in at a time
- Separating the deposit of different address is also important as when people withdrawn, they can't do an attack on other voters that made them could get their money back

Properties fulfilled:

Property a: Anyone on the Ethereum network can sign up in the protocol and depositing 1 ether at a time, they can sign up with multiple account which were splitted in the coin mixer in phase 0.

Property b: At the end of the protocol, everyone gets back all their ether

Property c: As we will check if the account have sufficient ether before committing, which ensure the account can only commit once.
Also, in the _nullifier of the SNARK, it will revert the transaction if it detected a committed vote being revealed already, which reinforce that the vote could only be revealed once.

Property d: After block.number > deployedBlock + 300, the contract won't accept any deposit, which ensures this property.

Property e: As we have reinforced that each account can only call committingVote once and revealingVote function once, we can be sure that no one can vote twice.

Property f: After deadline t2, the contract stops accepting new votes as there will be no new committing vote added to the contract.

Property g: In revealingVote function, as the voters are revealing their vote options and proof, the electedVote variable are kept being incremented. The rule for breaking tie is the first voting choice that has the highest vote, even there's a voting choice with same number of vote, but revealed later than the first highest vote, will loose.

Property h: No one will be able to tell other's vote because: 1. All people are using a temporary account and 2. They are feeding proof_a, proof_b and proof_c into the SNARK circuit, no one can get the value of the hashed commit(as it only tells if the hash exist or not in the circuit) and link back to the original voter

Property i: The verifier can access the blockchain and look for a particular transaction in commitVote function and ask for what the prover voted.

The prover needs to give the verifier the choice, nonce, proof_a, proof_b and proof_c. As in zero knowledge proof, only the voter knows the value of proof_a, proof_b and proof_c.

The verifier can reconstruct the hash value with the choice and nonce given by the prover. The verifier can test if the _commit already and see if it will generate proof_a, proof_b and proof_c. Then, if the proof_a, proof_b and proof_c matched what the provers provide and can prove that the secret exist in the nullifier, the prover can prover what he voted for.

Property k:

1. Constructor
    - **estimated Gas**:
        - Writing to storage (3 writes): 3 * 20,000 gas = 60,000 gas
        - Looping over _choice (assuming _choice is 3 for this example):
            - Each iteration writes to storage: 3 * 20,000 gas = 60,000 gas
            - Loop overhead (3 iterations): 2100 gas * 3 = 6,300 gas
    - **Total**: 60,000 + 60,000 + 6,300 = 126,300 gas
2. signUp
    - **Estimated Gas**:
        - require checks: Approximately 2100 gas each (2 checks) = 4,200 gas
        - Writing to storage: 20,000 gas
    - **Total**: 4,200 + 20,000 = 24,200 gas
3. committingVote
    - **Estimated Gas**:
        - require checks: Approximately 2100 gas each (2 checks) = 4,200 gas

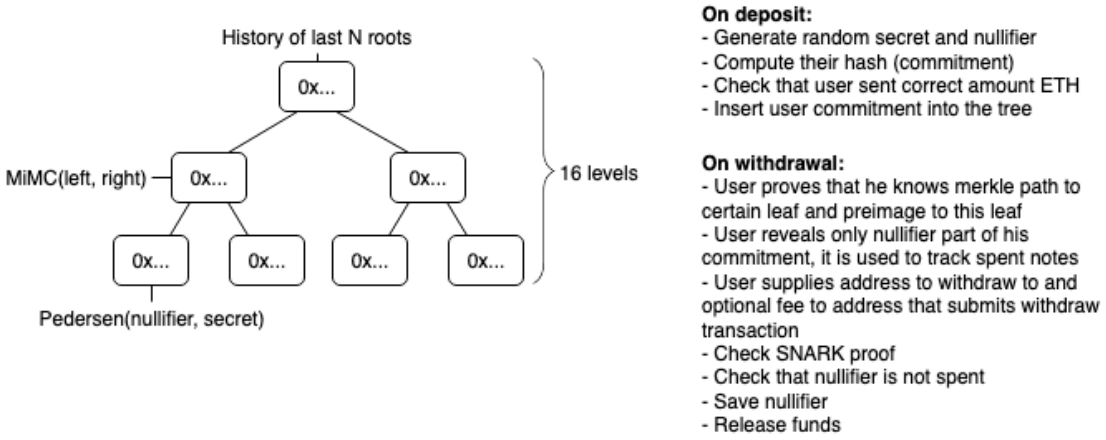
    - Checking storage (reading committed_vote): 2,100 gas
    - Writing to storage (2 writes): 20,000 gas * 2 = 40,000 gas
    - _commit function call: Variable, but typically around 20,000 gas
- **Total**: 4,200 + 2,100 + 40,000 + 20,000 = 66,300 gas

4. revealingVote

- **Estimated Gas**:
    - require checks: Approximately 2100 gas each (2 checks) = 4,200 gas
    - Calling _nullify: Variable, but typically around 50,000 gas

  For a specific tree depth, let's calculate the proofing gas cost.
  Assume tree_depth = 16:

  Calculate Circuit Constraints:



  From the library: Circuit Constraints = 28271 (1869 + 1325 * tree_depth)

  Assume there are 16 levels:
  Circuit Constraints=1869+1325×16=1869+21200=23069

  Circuit Constraints=1869+1325×16=1869+21200=23069

    - Writing to storage (voting_count, elected_vote): 2 writes, 20,000 gas and 5,000 gas (if updated) = 25,000 gas
- **Total**: 4,200 + 23069 + 25,000 = 52,260 gas

5. withdraw

- **Estimated Gas**:
  - require checks: Approximately 2100 gas each (2 checks) = 4,200 gas
  - Writing to storage (reentrancy lock): 2 * 20,000 gas = 40,000 gas
  - call to transfer ether: 9,000 gas
  - Resetting storage (deposit): 5,000 gas
- **Total**: 4,200 + 40,000 + 9,000 + 5,000 = 58,200 gas