Assignment 2

1. Privacy & ZK VMs

1. ***Explain in brief, how does the existing blockchain state transition to a new state? What is the advantage of using verification over re-execution?***

   [Answer] To make a change of state, one submits some input data as a transaction to the blockchain network. This transaction will be verified by a group of miners (PoW) or validators (PoS) to determine its validity. They do so by re-executing the State Transition Function using the user input data and comparing the results with each other. If most of them reach an agreement, the chain state will be altered, and the transaction will be permanently stored in the blockchain.

   The advantage of using verification over re-execution:

   1) *Privacy*. Users don't need to submit their input data to the public network for nodes to verify the validity of their transaction. As a result, both the input data and the STF is not exposed to the public.

   2) *Scalability*. The throughput of a blockchain network is notoriously slow. This is in part due to the computation overhead caused by the re-execution of each transaction in each node. If the time it takes to verify each transaction becomes shorter, the overall processing capability of the network will improve tremendously.

   3) *Computation Restriction.* If one tries to perform complex computation in a smart contract on chain, the gas fee will explode. Not to mention to re-execute this computation on each node. This limits the functionality of a smart contract in a blockchain. For many real-world use cases and more complicated computations, verification can circumvent this limitation.

   4) Storage Usage. As re-execution is the only way to restore the chain state, the transaction history of a blockchain must be stored in full across all the nodes and crease exponentially. Verification does not need to store all these histories and thus is more storage efficient.

2. ***Explain in brief what is a ZK VM (virtual machine) and how it works?***

[Answer] A ZK VM is a circuit (a representation of a program) that executes bytecode. It allows a prover to show that, given a set of inputs, as well as some bytecode, they have correctly executed the program code on said inputs.

1. ***Give examples of certain projects building Zk VMs (at-least 2-3 projects). Describe in brief, key differences in their VMs.***
   [Answer]
   1) *zkEVM*: a technology powers zkSync 2.0, is a virtual machine that executes solidity smart contracts in a way that is compatible with zero-knowledge-proof computation using ZK-SNARK.

   2) *Distaff*: a zero-knowledge virtual machine running a proprietary program syntax. For any program executed on Distaff VM, a STARK-based proof of execution is automatically generated. zCloak is based on this VM.

   3) *Cairo*: A Turing-complete language and VM making it possible for blockchain developers to harness the power of STARKs. It runs on StarkNet which is a permissionless decentralized ZK-Rollup operating as an L2 network over Ethereum, where any dApp can achieve unlimited scale for its computation, without compromising Ethereum's composability and security.

   4) *snarkvm:* based on ZK-SNARK proof technology and executes Leo Programming Language and running on Aleo blockchain. Aleo offers very good developer tool as Aleo Studio.

2. ***[Bonus] What are the advantages and disadvantages of some of the existing Zk VMs?***

   [Answer]

   Advantage:

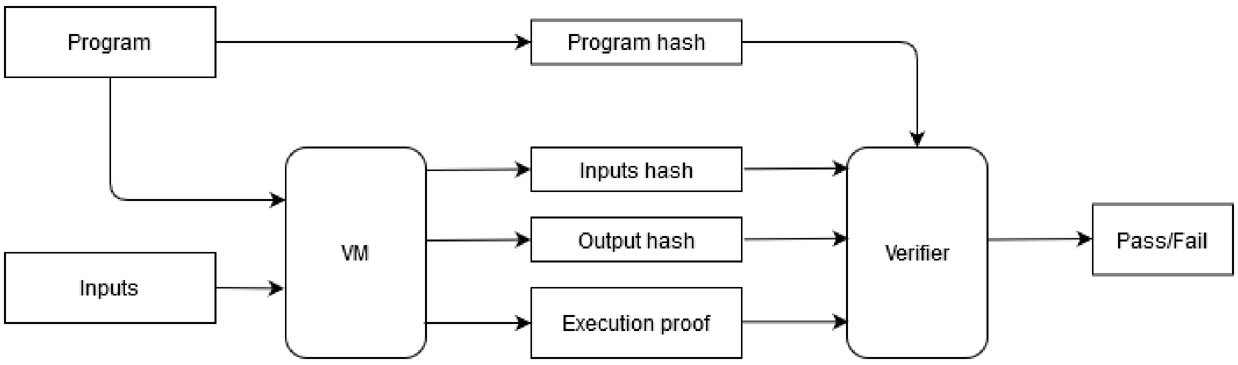
   1. The program, the inputs, and the outputs are fully private (they are never revealed to the verifier).
   2. A single verifier can verify any program executed on the VM (e.g. a verifier can be a single on-chain contract).
   3. A ZK VM abstracts away some tricky aspects of writing circuits. While developers should still understand the underlying system on which their code will run for security reasons, it is theoretically easier to write code for a VM than wiring a circuit by hand.
   4. Since the structure of the STARK is the same for all programs, batching of proofs using another STARK (or a SNARK) should be much easier to accomplish.

Disadvantage:

1. Vendor lock-in, the current selection of ZK VMs available is small, and some ZK VM authors have indicated that they may exercise a greater degree of control over their platforms than is usually seen in fully open-source projects.
2. Could be slower compared to dedicate circuit application, ZK VM could have poor performance if the circuit is small.

3. ***[Bonus] Explain in detail one of the Zk VM architectures using diagrams.***

[Answer]



In the above:

1. The VM does the following:
   a. Takes 2 inputs: a program and a set of inputs for the program,
   b. Executes the program with the given set of inputs,
   c. Outputs hash of the inputs, hash of the outputs generated by the program, and a STARK proof attesting to the correct execution of the program.
2. The verifier does the following:
   a. Takes the hash of the program, hash of the inputs, and hash of the outputs, and uses them to verify the STARK proof.

2. Semaphore

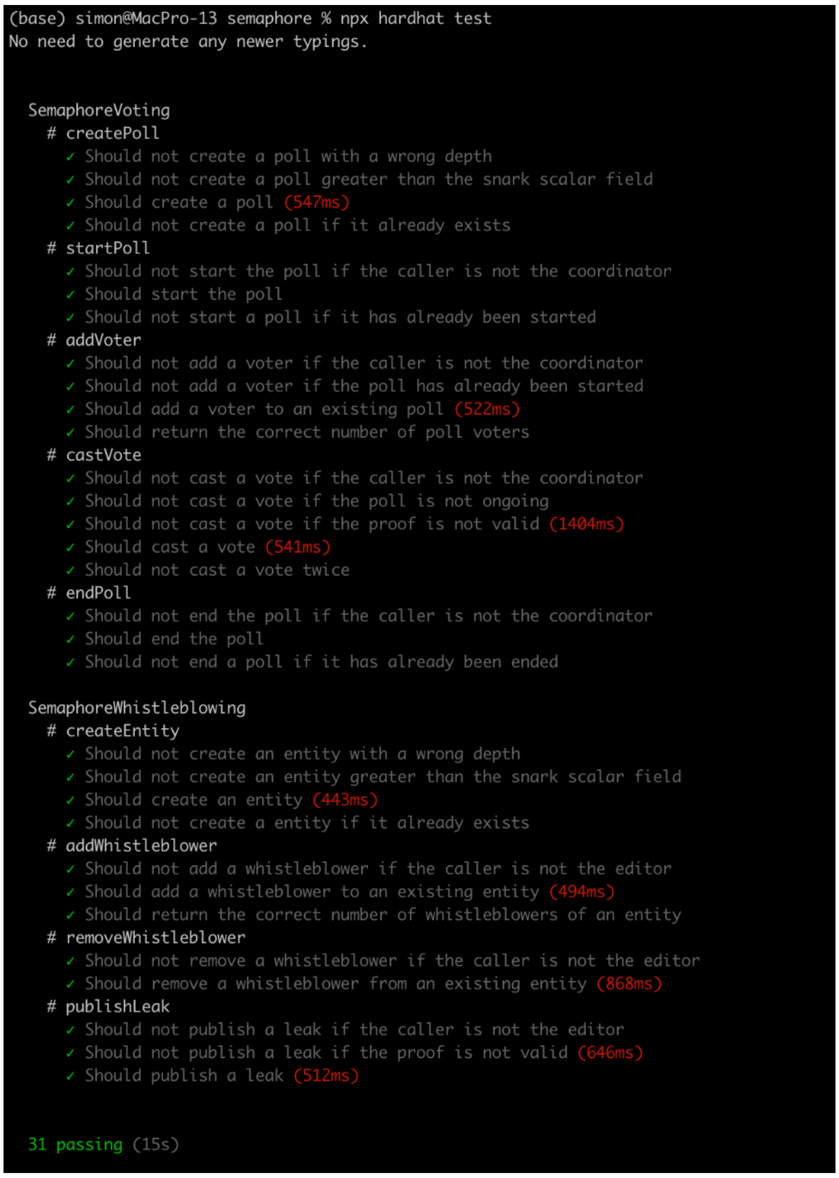
1. ***What is Semaphore? Explain in brief how it works? What applications can be developed using Semaphore (mention 3-4)?***

   [Answer] Semaphore is a zero-knowledge gadget which allows Ethereum users to prove their membership of a set without revealing their original identity. At the same time, it allows users to signal their endorsement of an arbitrary string and provides a simple built-in mechanism to prevent double-signalling or double-spending. It is designed to be a simple and generic privacy layer for Ethereum DApps.

This gadget comprises of smart contracts and zero-knowledge components which work in tandem. The Semaphore smart contract handles state, permissions, and proof verification onchain. The zero-knowledge components work offchain to allow users to generate proofs, which allow the smart contract to update its state if these proofs are valid.

Use cases include private voting, whistleblowing, mixers (Tornado Cash), and anonymous authentication.

2. ***Clone the semaphore repo (3bce72f).***
    1. *Run the tests and add a screenshot of all the test passing.*
    [Answer]



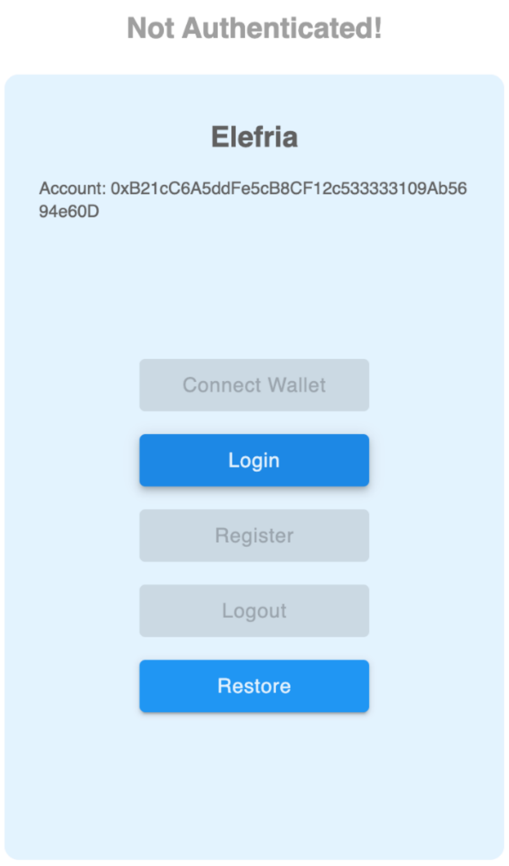
```
(base) simon@MacPro-13 semaphore % npx hardhat test
No need to generate any newer typings.


  SemaphoreVoting
    # createPoll
      ✓ Should not create a poll with a wrong depth
      ✓ Should not create a poll greater than the snark scalar field
      ✓ Should create a poll (547ms)
      ✓ Should not create a poll if it already exists
    # startPoll
      ✓ Should not start the poll if the caller is not the coordinator
      ✓ Should start the poll
      ✓ Should not start a poll if it has already been started
    # addVoter
      ✓ Should not add a voter if the caller is not the coordinator
      ✓ Should not add a voter if the poll has already been started
      ✓ Should add a voter to an existing poll (522ms)
      ✓ Should return the correct number of poll voters
    # castVote
      ✓ Should not cast a vote if the caller is not the coordinator
      ✓ Should not cast a vote if the poll is not ongoing
      ✓ Should not cast a vote if the proof is not valid (1404ms)
      ✓ Should cast a vote (541ms)
      ✓ Should not cast a vote twice
    # endPoll
      ✓ Should not end the poll if the caller is not the coordinator
      ✓ Should end the poll
      ✓ Should not end a poll if it has already been ended

  SemaphoreWhistleblowing
    # createEntity
      ✓ Should not create an entity with a wrong depth
      ✓ Should not create an entity greater than the snark scalar field
      ✓ Should create an entity (443ms)
      ✓ Should not create a entity if it already exists
    # addWhistleblower
      ✓ Should not add a whistleblower if the caller is not the editor
      ✓ Should add a whistleblower to an existing entity (494ms)
      ✓ Should return the correct number of whistleblowers of an entity
    # removeWhistleblower
      ✓ Should not remove a whistleblower if the caller is not the editor
      ✓ Should remove a whistleblower from an existing entity (868ms)
    # publishLeak
      ✓ Should not publish a leak if the caller is not the editor
      ✓ Should not publish a leak if the proof is not valid (646ms)
      ✓ Should publish a leak (512ms)


  31 passing (15s)
```

2. *Explain code in the sempahore.circom file (including public, private inputs).*
   [Answer] see
   https://github.com/geesimon/zku/blob/main/week2/semaphore.circom

3. *[Bonus] Create a frontend for the current semaphore version. You can use this as reference.*
   [Answer]

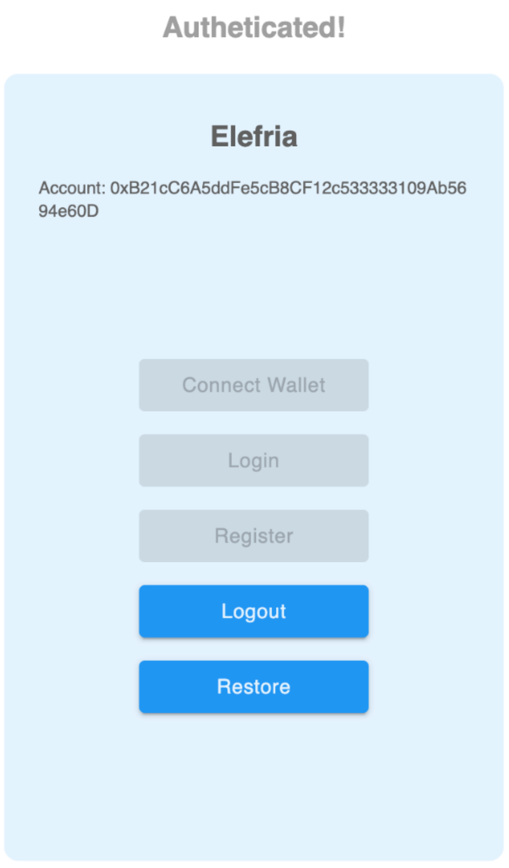
3. ***Use `Elefria` protocol on the `Harmony Testnet`, try to generate a ZK identity and authenticate yourself as a user.***

[ScreenShots]


Not Authenticated!

Elefria

Account: 0xB21cC6A5ddFe5cB8CF12c533333109Ab56
94e60D


Autheticated!

Elefria

Account: 0xB21cC6A5ddFe5cB8CF12c533333109Ab56
94e60D

.

1. What potential challenges are there to overcome in such an authentication system?

   [Answer]
   From the end user experience perspective,
   1. Such authentication seems quite slow compared with centralized authentication process (i.e., Facebook, Google). This could be due to the fact user needs to generate witness and proof each time when login.

2. Since the secret is stored in browser's local storage, user must authenticate twice (one to decrypt the secret, one to login to MetaMask), which is quite redundant.
3. Every time, user registers or logins, MetaMask will popup asking for fee (pay gas), I feel this could be a deal breaker for user adoption since no app/web applications charge this kind of fee today. We probably should transfer such gas consumption to the web3 developer to relief user concerns.

From Web3 developer perspective,

- When authentication is ZK, web3 developers need a new way to customize user experience base on their personal data and the solution is not quite there yet. Think of how to build a Amazon product recommendation in the ZK world.
- In case, user cleaned the browser, he probably cleaned all registered and personal tracking info. We need a better way to store user personal info (in IPFS?) to avoid such disaster.
- Since web3 app doesn't know which account is active, it creates big challenge to business to do customer based BI analysis and CRM in order to improve customer satisfaction.

2. *[Bonus]* What potential improvements can one make to simplify the Elefria authentication protocol?

[Answer] I think we could combine Connect Wallet, Register and Login processes in one process (Login), when user click on Login,

1. Check whether wallet is connected, if not, connect the wallet.
2. Check whether the user has already registered, if not, start the registration process.
3. Start the login process.
    - Also in this process, its unnecessary to ask user to enter password everytime in order to decrypt the secret. Instead, we could make a feature like, "login automatically in 15 days" to futhur simplify the login process.

3. Tornado Cash

1. ***Compare and contrast the circuits and contracts in the two repositories above (or consult this article), summarize the key improvements/upgrades from tornado-trees to tornado-nova in 100 words.***

[Answer]

1. Allow users to deposit & withdraw arbitrary amounts of ETH by generating 2 new merkle tree nodes for each transaction.
2. User can make shielded transfers/withdraw of deposited tokens through relayer.
3. Use Gnosis Chain (former xDAI) as a Layer2 so that users can benefit from cheaper fees, while still having fast transactions (a few minutes).

2. *Check out the tornado-trees repo*

1. *Look at the circuits/TreeUpdateArgsHasher.circom and contracts/TornadoTrees.sol. Explain the process to update the withdrawal tree (including public, private inputs to the circuit, arguments sent to the contract call, and the on-chain verification process).*

   [Answer]

   1. When user makes a withdrawal request on contract TornadoTrees, function registerWithdrawal(_instance, _nullifierHash) (through the tornadoProxy to shield the requester address) is called to queue the withdraw data (_instance: address of the recipient and _ nullifierHash: nullifier to prevent double spending). Note: the request has not inserted to the merkle tree yet, it is queued for later batch insertion. Use this approach to improve merkle tree update performance.

   2. updateDepositTree(_proof, _argsHash, _currentRoot, _newRoot, _pathIndices, _events) is called later to insert a full batch of queued withdrawals into a merkle tree and use snark proof (parameter: _proof) to verify records were inserted correctly. To do the verification, it concatenates all records (_events), merkle path to inserted batch (_pathIndices), updated merkle tree root (_newRoot) and current merkle tree root (_currentRoot) to build a hash on this blob data, then verify this hash is equal to _argsHash which is a public input of snark proof. The snark proof is generated by circuit BatchTreeUpdate.circom, in this circuit,
      1. Signal argsHash is the only public input.
      2. Signal oldRoot, newRoot, pathIndices, pathElements, hashes, instances and blocks are private inputs that need to be provided by end user. These signals map to the contact function updateDepositTree() parameters (_currentRoot, _newRoot, _pathIndices, _events.hash, _event.instance, _event.block) respectively and are used to recompute the hash of all elements using same logic in updateDepositTree mentioned above and make sure it equal to public input argsHash through circuit TreeUpdateArgsHasher.circom.

3. It also rebuilds the merkle tree root and verify the batch subtree was inserted correctly.

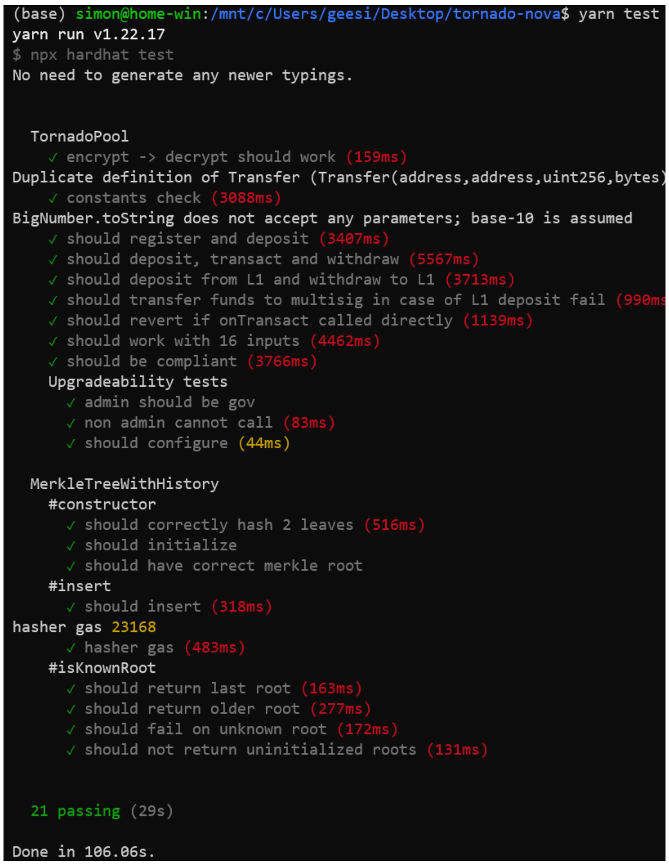
By doing this proof verification, the contact can make sure the withdrawal request is valid. The solidity verifyProof call only takes 1 input signal parameter (argsHash) instead of all public parameters (_currentRoot, _newRoot, _pathIndices, _events) to reduce gas consumption.

2. *Why do you think we use the SHA256 hash here instead of the Poseidon hash used elsewhere?*

   [Answer] SHA256 function is used in solidity function updateDepositTree() and is executed on chain to compute hash of data blob (described above). Since the data blob could be quite large, use SHA256 can reduce gas consumption significantly compared with Poseidon or other circuit friendly hash function. The drawback is the circuit becomes more complex. But since the circuit is executed offline, we think this is a good tradeoff for zk-Dapp design.

3. ***Clone/fork the tornado-nova repo***

   1. *Run the tests and add a screenshot of all the tests passing.*
      [ScreenShot]

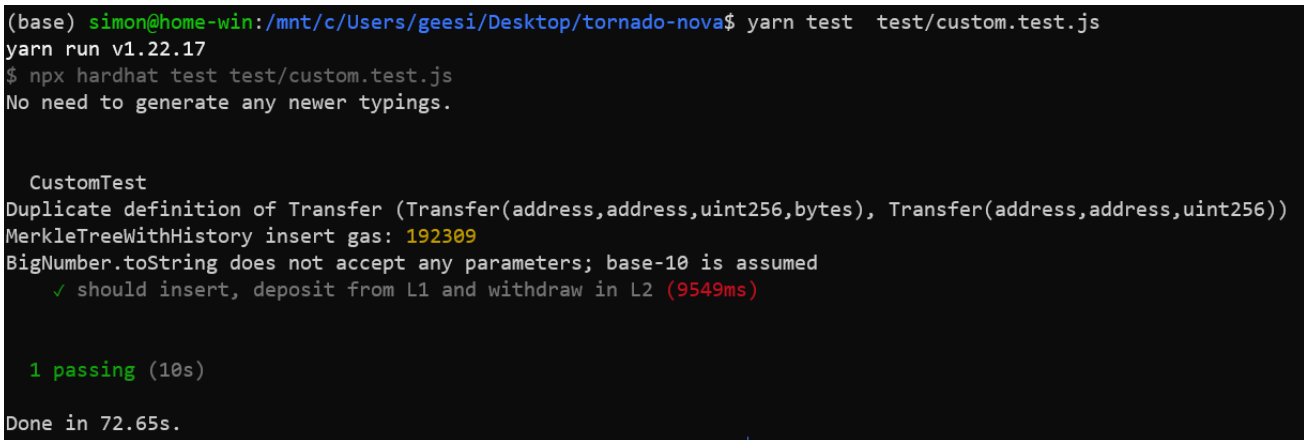


```
(base) simon@home-win:/mnt/c/Users/geesi/Desktop/tornado-nova$ yarn test
yarn run v1.22.17
$ npx hardhat test
No need to generate any newer typings.


  TornadoPool
    ✓ encrypt -> decrypt should work (159ms)
Duplicate definition of Transfer (Transfer(address,address,uint256,bytes)
    ✓ constants check (3088ms)
BigNumber.toString does not accept any parameters; base-10 is assumed
    ✓ should register and deposit (3407ms)
    ✓ should deposit, transact and withdraw (5567ms)
    ✓ should deposit from L1 and withdraw to L1 (3713ms)
    ✓ should transfer funds to multisig in case of L1 deposit fail (990ms
    ✓ should revert if onTransact called directly (1139ms)
    ✓ should work with 16 inputs (4462ms)
    ✓ should be compliant (3766ms)
    Upgradeability tests
      ✓ admin should be gov
      ✓ non admin cannot call (83ms)
      ✓ should configure (44ms)

  MerkleTreeWithHistory
    #constructor
      ✓ should correctly hash 2 leaves (516ms)
      ✓ should initialize
      ✓ should have correct merkle root
    #insert
      ✓ should insert (318ms)
hasher gas 23168
      ✓ hasher gas (483ms)
    #isKnownRoot
      ✓ should return last root (163ms)
      ✓ should return older root (277ms)
      ✓ should fail on unknown root (172ms)
      ✓ should not return uninitialized roots (131ms)


  21 passing (29s)

Done in 106.06s.
```

2. *Add a script named custom.test.js under test/ and write a test for all of the followings in a single it function*
    1. estimate and print gas needed to insert a pair of leaves to `MerkleTreeWithHistory`
    2. deposit 0.08 ETH in **L1**
    3. withdraw 0.05 ETH in **L2**
    4. assert recipient, omniBridge, and tornadoPool balances are correct

[Answer] See code: https://github.com/geesimon/zku/blob/main/week2/custom.test.js

Screenshot for test execution:



```
(base) simon@home-win:/mnt/c/Users/geesi/Desktop/tornado-nova$ yarn test  test/custom.test.js
yarn run v1.22.17
$ npx hardhat test test/custom.test.js
No need to generate any newer typings.


  CustomTest
Duplicate definition of Transfer (Transfer(address,address,uint256,bytes), Transfer(address,address,uint256))
MerkleTreeWithHistory insert gas: 192309
BigNumber.toString does not accept any parameters; base-10 is assumed
    ✓ should insert, deposit from L1 and withdraw in L2 (9549ms)


  1 passing (10s)

Done in 72.65s.
```

4. ***[Bonus] Read Proposal #11 of Tornado.cash governance, what is the purpose of the newly deployed L1Unwrapper contract?***

   [Answer] Currently, all fees occurred during withdrawals to L1 are subsidized by the xDai (Gnosis) team and this sponsorship will end soon. To make Tornado Cash Nova continue work, the proposal #11 is a new design to make user pays the L1 fee instead. A part of user's withdrawal will go directly to the a Omnibridge router that settles withdrawal transaction on L1.

4. Thinking In ZK

1. ***If you have a chance to meet with the people who built Tornado Cash & Semaphore, what questions would you ask them about their protocols?***

   [Answer]

   For both products,

1. It seems the client only support browser environment, any plan to provide cross app/device support? This is especially important for Tornado Cash to support mobile app.
2. Compared with zk-Snark, zk-Stark doesn’t need the trusted setup, any investigation to use zk-Stark or other ZKP technology instead?

Specific to Tornado Cash,

1. Besides ETH, any plan to support other crypto currencies (SOL, USDT etc)?
2. Is it possible to leverage this technology to support anonymous money transfer between bank accounts?
3. What are the key reasons to choose Gnosis over other blockchains (Harmony, Polygon) as the L2 solution?

2. ***[Bonus]** Regarding writing and maintaining circuits for each dapp separately, what are your thoughts about using just one circuit for all dapps? Is that even possible? What is likely to be a standard in the future for developing Zk dapps?*

   [Answer] This is what zkVM is designed to achieve, see detailed explanation about how zkVM works in question 1.1.

   I do like this idea and believe this will simplify zk-dapp development significantly. Today, design a solid circuit is not only difficult but also very constrained (the programming language used by circom is not tuning-complete while Cairo zkVM is), the trusted setup process could be expensive for dev-ops.

   I can image in the future, zk-dapp developer can simply write any business logic that needs take user personal data (for example: check what service level this user belongs to according to purchase history) in any language and compiled to zkVM bytecode, then distribute this dapp to user’s local device to take private input to compute the provable results, then send back to smart contract to take further actions.