Question 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?

the existing blockchain state transition to a new state(Taking ethereum as an example):

Suppose the existing blockchain state is state(n) and the new state is state(n+1).

1. from state(n) to state(n+1) by the execution of transactions.
2. finial state(n+1).

In summary, use math to describe(From ethereum yellow paper):

$$(1) \qquad \boldsymbol{\sigma}_{t+1} \equiv \Upsilon(\boldsymbol{\sigma}_t, T)$$

where Υ is the Ethereum state transition function. In Ethereum, Υ, together with $\boldsymbol{\sigma}$ are considerably more powerful then any existing comparable system; Υ allows components to carry out arbitrary computation, while $\boldsymbol{\sigma}$ allows components to store arbitrary state between transactions.

The advantage of using verification over re-execution is as follow:

- Fast verification
- Lower cost for verification
- Scalable
- Privacy

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

ZK VM is a virtual machine that executes smart contracts in a way that is compatible with zero-knowledge-proof computation.It is the key to building an EVM-compatible ZK Rollup while preserving the battle-tested code and knowledge gained after years of working with Solidity.

3. Give examples of certain projects building Zk VMs (at-least 2-3 projects). Describe in brief, key differences in their VMs.

Zk VM Project list:

- Hermez zkEVM
- AppliedZKP zkEVM
- zkSync EVM

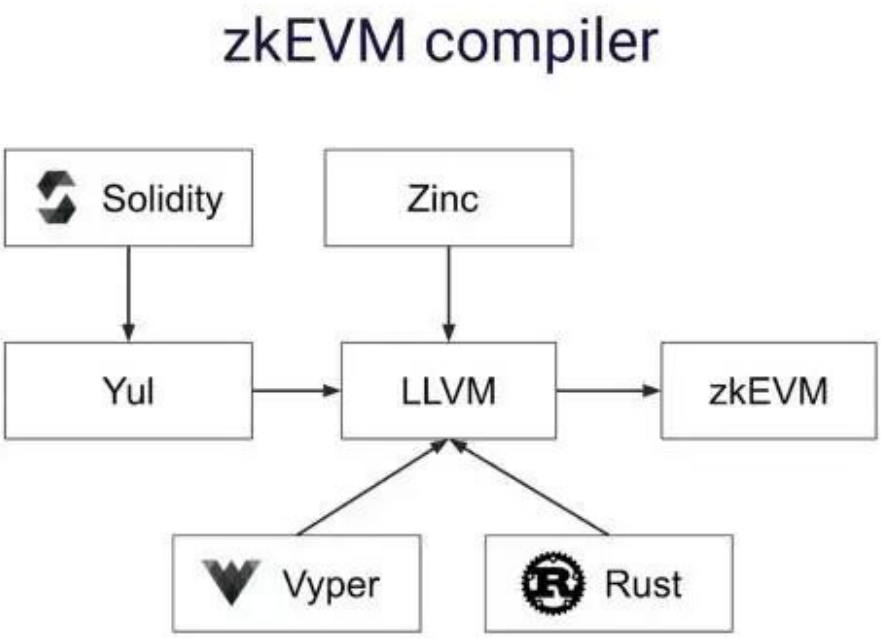
Both Hermez and AppliedZKP zkEVM use sololution A: Direct support for the existing set of EVM opcodes, which is fully compatible with the set of Solidity opcodes.

But zkSync EVM uses sololution B: Maintaining Solidity compatibility by designing a new virtual machine that is zero-knowledge proof-friendly and adapting EVM development tools.

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

For sololution A, since it fully supports the existing set of EVM opcodes and uses the same compiler as EVM, it is naturally fully compatible with the existing ecosystem and development tools, and also inherits the security model of Ethereum better. For sololution B, it is not bound by the original set of EVM opcodes, so it is more flexible to compile the code into a more zero-knowledge proof-friendly set of opcodes. It is also free from the hard and heavy work required to be compatible with all of the original set of EVM opcodes. Overall, sololution A is more compatible and safer, but with more work; sololution B is more flexible and less work, but extra effort in adaptation is required such as porting Smart Contracts.

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



The above picture show the context of zkSync EVM.

zkSync project architecture is a useful guide to learn zkysnc. But I do not have enough time to dive into the code.

Question 2. Semaphore

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

Semaphore is a zero-knowledge gadget which allows Ethereum users to prove their membership of a set which they had previously joined without revealing their original identity.

Consider Alice, Bob, and Charlie, who are users of an application built on Semaphore. They each register an identity into the application contract. Later, Alice can prove that she is part of the set of registered identities {Alice, Bob, Charlie}, without revealing that she is Alice. When she does so, she can broadcast an arbitrary string. Note that the contract ensures that she can only do so once within the context of an external nullifier. A simple way to understand an external nullifier is to think of it as a topic, towards which each user may only respond once. It serves different functions depending on the nature of the application which uses Semaphore.

Semaphore's Application Scenarios is as follow:

1. Private voting
2. Private whistleblowing
3. Private proof of asset
4. decentralized ID
5. Zero Knowledge Lottery

2. Clone the semaphore repo (3bce72f).

- Run the tests and add a screenshot of all the test passing.

The script for running test cases:

```
git clone https://github.com/appliedzkp/semaphore
cd semaphore
git checkout 3bce72febeba48454cb618a1f690045c04809900
yarn --frozen-lockfile
yarn compile
yarn test:prod
```

The screenshot is as follows:

```
× 1 problem (0 errors, 1 warning)

$ hardhat test
No need to generate any newer typings.


  SemaphoreVoting
    # createPoll
      ✓ Should create a poll (608ms)
      ✓ 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 (545ms)
      ✓ Should return the correct number of poll voters
    # castVote
      ✓ Should not cast a vote if the caller is not the coordinator (2927ms
      ✓ Should not cast a vote if the poll is not ongoing (2541ms)
      ✓ Should not cast a vote if the proof is not valid (4243ms)
      ✓ Should cast a vote (3230ms)
      ✓ Should not cast a vote twice (2542ms)
    # 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 create an entity (562ms)
      ✓ 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 (554ms)
      ✓ Should return the correct number of whistleblowers of an entity
    # removeWhistleblower
      ✓ Should not remove a whistleblower if the caller is not the editor (
      ✓ Should romove a whistleblower from an existing entity (998ms)
    # publishLeak
      ✓ Should not publish a leak if the caller is not the editor (2574ms)
      ✓ Should not publish a leak if the proof is not valid (3301ms)
      ✓ Should publish a leak (3321ms)


  27 passing (33s)

Done in 44.16s.
[redacted]:~/zku/week2/semaphore$
```

- Explain code in the sempahore.circom file (including public, private inputs).

1. public input: signalHash, externalNullifier, private inputs: identityNullifier, identityTrapdoor, treeSiblings, treePathIndices...
2. poseidon(identityNullifier,identityTrapdoor) => secret
3. poseidon(secret) => inclusionProof.leaf
4. poseidon(externalNullifier, identityNullifier) => nullifierHash
5. MerkleTreeInclusionProof(inclusionProof.leaf, treeSiblings, treePathIndices) => root
6. outputs: root(in step 5) and nullifierHash(in step 4)

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

Sorry, I do not have enough time to finish it.

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

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

The potential challenges are as follows:

1. how to handle many users's requests
2. how to reduce latency of the requests

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

Maybe we can let the user generates authToken to reduce steps and gas cost.

The Steps are as follows:

1. user generates authToken and submit zkid proof and authToken to smart contract
2. smart contract verfied zkid proof and store authToken
3. user send uuid and authToken to web3/web2 platform
4. web3/web2 platform send authToken to smart contract
5. smart contract respond uuid to web3/web2 platform
6. web3/web2 platform validate uuid
7. web3/web2 platform grant access to user

Question 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.

The key improvements/upgrades from tornado-trees to tornado-nova are as follows:

1. Suport L2(xdai) to reduce the costs of transactions
2. Contracts will be upgradable
3. Privacy pool with internal transactions
4. Support Arbitrary amounts for deposit

2. Check out the tornado-trees repo

Take a 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).

The circuit process: TreeUpdateArgsHasher(oldRoot, newRoot, pathIndices, hashes, instances, blocks) => argsHash

Because it updates tree in batch, so need to send raw data to verification process. The arguments sent to the contract call are as follows:

```
  /// @dev Insert a full batch of queued withdrawals into a merkle tree
  /// @param _proof A snark proof that elements were inserted correctly
  /// @param _argsHash A hash of snark inputs
  /// @param _currentRoot Current merkle tree root
  /// @param _newRoot Updated merkle tree root
  /// @param _pathIndices Merkle path to inserted batch
  /// @param _events A batch of inserted events (leaves)
 function updateWithdrawalTree(
    bytes calldata _proof,
    bytes32 _argsHash,
    bytes32 _currentRoot,
    bytes32 _newRoot,
    uint32 _pathIndices,
    TreeLeaf[CHUNK_SIZE] calldata _events
  ) public
```

The the on-chain verification process is described in the code note:

```
function updateWithdrawalTree(
        bytes calldata _proof,
        bytes32 _argsHash,
        bytes32 _currentRoot,
        bytes32 _newRoot,
        uint32 _pathIndices,
        TreeLeaf[CHUNK_SIZE] calldata _events
    ) public {
        uint256 offset = lastProcessedWithdrawalLeaf;
        require(_currentRoot == withdrawalRoot, "Proposed withdrawal root is
invalid");
        require(_pathIndices == offset >> CHUNK_TREE_HEIGHT, "Incorrect withdrawal
insert index");

        // pepare data for sha256
        bytes memory data = new bytes(BYTES_SIZE);
        assembly {
            mstore(add(data, 0x44), _pathIndices)
            mstore(add(data, 0x40), _newRoot)
```

```
            mstore(add(data, 0x20), _currentRoot)
        }
        for (uint256 i = 0; i < CHUNK_SIZE; i++) {
            (bytes32 hash, address instance, uint32 blockNumber) =
(_events[i].hash, _events[i].instance, _events[i].block);
            bytes32 leafHash = keccak256(abi.encode(instance, hash, blockNumber));

            bytes32 withdrawal = offset + i >= withdrawalsV1Length
                ? withdrawals[offset + i]
                : tornadoTreesV1.withdrawals(offset + i);
            // check leafhash
            require(leafHash == withdrawal, "Incorrect withdrawal");
            // push blockNumber, instance, hash to  data Array
            assembly {
                let itemOffset := add(data, mul(ITEM_SIZE, i))
                mstore(add(itemOffset, 0x7c), blockNumber)
                mstore(add(itemOffset, 0x78), instance)
                mstore(add(itemOffset, 0x64), hash)
            }
            if (offset + i >= withdrawalsV1Length) {
                delete withdrawals[offset + i];
            } else {
                emit WithdrawalData(instance, hash, blockNumber, offset + i);
            }
        }
        // compute argsHash by sha256
        uint256 argsHash = uint256(sha256(data)) % SNARK_FIELD;
        // check computed argsHash and  _argsHash
        require(argsHash == uint256(_argsHash), "Invalid args hash");
        // verify merkle proof
        require(treeUpdateVerifier.verifyProof(_proof, [argsHash]), "Invalid
withdrawal tree update proof");
        // verifyProof ok, update previousWithdrawalRoot
        previousWithdrawalRoot = _currentRoot;
        // verifyProof ok, update withdrawalRoot
        withdrawalRoot = _newRoot;
        // verifyProof ok, update  lastProcessedWithdrawalLeaf
        lastProcessedWithdrawalLeaf = offset + CHUNK_SIZE;
    }
```

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

In circuits/TreeUpdateArgsHasher.circom, we use the SHA256 hash to update ARgsHasher. The code is as follows:

```
template TreeUpdateArgsHasher(nLeaves) {
    signal input oldRoot;
    signal input newRoot;
    signal input pathIndices;
    signal input instances[nLeaves];
```

```
        signal input hashes[nLeaves];
        signal input blocks[nLeaves];
        signal output out;

        var header = 256 + 256 + 32;
        var bitsPerLeaf = 256 + 160 + 32;
        // using Sha256 to compute argsHash for verification in updateWithdrawalTree
        component hasher = Sha256(header + nLeaves * bitsPerLeaf);
        ...
    }
```

Because the output of TreeUpdateArgsHasher is used for merkle proof in the solidity(solidity do not support Poseidon hash). the related code is as follows:

```
    // use sha256 to process data(_currentRoot+_newRoot+_pathIndices)
    uint256 argsHash = uint256(sha256(data)) % SNARK_FIELD;
    require(argsHash == uint256(_argsHash), "Invalid args hash");
    require(treeUpdateVerifier.verifyProof(_proof, [argsHash]), "Invalid withdrawal
    tree update proof");
```

3. Clone/fork the tornado-nova repo

1. Run the tests and add a screenshot of all the tests passing.

```
[redacted]:~/zku/week2/tornado-nova$ yarn test
yarn run v1.22.5
$ npx hardhat test
No need to generate any newer typings.


  TornadoPool
    ✓ encrypt -> decrypt should work (197ms)
Duplicate definition of Transfer (Transfer(address,address,uint256,bytes), Transfer(address,address,uint256))
    ✓ constants check (993ms)
BigNumber.toString does not accept any parameters; base-10 is assumed
    ✓ should register and deposit (3113ms)
    ✓ should deposit, transact and withdraw (5733ms)
    ✓ should deposit from L1 and withdraw to L1 (3658ms)
    ✓ should transfer funds to multisig in case of L1 deposit fail (1118ms)
    ✓ should revert if onTransact called directly (990ms)
    ✓ should work with 16 inputs (4799ms)
    ✓ should be compliant (3572ms)
    Upgradeability tests
      ✓ admin should be gov
      ✓ non admin cannot call
      ✓ should configure

  MerkleTreeWithHistory
    #constructor
      ✓ should correctly hash 2 leaves (154ms)
      ✓ should initialize
      ✓ should have correct merkle root
    #insert
      ✓ should insert (356ms)
hasher gas 23168
      ✓ hasher gas (284ms)
    #isKnownRoot
      ✓ should return last root (92ms)
      ✓ should return older root (175ms)
      ✓ should fail on unknown root (105ms)
      ✓ should not return uninitialized roots (79ms)


  21 passing (26s)

Done in 28.63s.
```

2. Add a script named custom.test.js under test/ and write a test for all of the followings in a single it function

- estimate and print gas needed to insert a pair of leaves to MerkleTreeWithHistory
- deposit 0.08 ETH in L1
- withdraw 0.05 ETH in L2
- assert recipient, omniBridge, and tornadoPool balances are correct

The full code is here.

```
[redacted]zku/week2/tornado-nova$ yarn test
yarn run v1.22.5
$ npx hardhat test
No need to generate any newer typings.


  TornadoPoolCustom
insert gas 171309
Duplicate definition of Transfer (Transfer(address,address,uint256,bytes), Transfer(address,address,uint256))
BigNumber.toString does not accept any parameters; base-10 is assumed
    ✓ deposit 0.08 ETH in L1 withdraw 0.05 ETH in L2 (7573ms)

  TornadoPool
    ✓ encrypt -> decrypt should work
Duplicate definition of Transfer (Transfer(address,address,uint256,bytes), Transfer(address,address,uint256))
    ✓ constants check (649ms)
    ✓ should register and deposit (1871ms)
    ✓ should deposit, transact and withdraw (5733ms)
    ✓ should deposit from L1 and withdraw to L1 (3848ms)
    ✓ should transfer funds to multisig in case of L1 deposit fail (1192ms)
    ✓ should revert if onTransact called directly (1024ms)
    ✓ should work with 16 inputs (5147ms)
    ✓ should be compliant (3955ms)
    Upgradeability tests
      ✓ admin should be gov (39ms)
      ✓ non admin cannot call
      ✓ should configure (98ms)

  MerkleTreeWithHistory
    #constructor
      ✓ should correctly hash 2 leaves (223ms)
      ✓ should initialize
      ✓ should have correct merkle root
    #insert
      ✓ should insert (260ms)
hasher gas 23168
      ✓ hasher gas (248ms)
    #isKnownRoot
      ✓ should return last root (112ms)
      ✓ should return older root (252ms)
      ✓ should fail on unknown root (163ms)
      ✓ should not return uninitialized roots (105ms)


  22 passing (33s)
```

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

The purpose is to allow Tornado Cash Nova to continue running, even without the xDai team intervention.

Question 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?

questions for Tornado Cash:

1. can harmony be the L2 for Tornado Cash like xDai? and what will need to do for it ?
2. any plan to deploy Tornado Cash on harmony?
3. Tornado Cash is aimed to shield transaction, is there any plan to Expand the scope of Tornado Cash's application?

questions for Semaphore:

1. How to abstract protocols to meet many kinds of applications requirements?

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?

In theory using just one circuit for all dapps is possible , just like as an computer OS for many computer applicaions. But zk dapps often require much compute, use specific circuit for the specific dapp maybe is a better option because it is easy to develop, customized.

The standards in the future for developing Zk dapps:

1. the standard zk VM to supply environment for zk dapps
2. standard tools and standard libs for developing zk dapps

reference

1. zku.ONE – 2022 Mar-April Syllabus / Course Schedule
2. Assignment 2
3. Final Projects by ZKU Genesis Graduates