AbdulQadir#0432 qadir@xord.com

Question 1:

Part 1:

- **Gas Cost:**

  Comparison: Sha256 < Poseidon < MiMc < Pedersen

  Explanation: The gas cost to hash the two values in MiMc is 59840, Poseidon is 49858, and Sha256 is 23179. While Pedersen consumes greater gas than others, so it is inefficient in terms of gas consumption and may also consume infinite gas due to bad implementation

- **Capacity**

  Comparison: Sha256 < Pedersen ≈ Poseidon ≈ MiMc

  Explanation: Pedersen, Poseidon, and MiMc have no limit on the number of inputs. We can provide as many inputs as we can to the hashing function. The capacity increase in terms of the power of 2. While on the other hand, Sha256 only takes two inputs.

- **Poof Generation Efficiency**

  Comparison: Sha256 < Pedersn < MiMc < Poseidon

  Explanation: Prover time is usually taken as the proof of generation efficiency of the system.

Sha256 Prover Time	2.1ms
Pedersen Prover Time	4.7ms
Poseidon Prover Time	slower than MiMc
MiMc Prover Time	5.8ms

- **Proof Size**

  Comparison: Poseidon < MiMc < Pedersen < Sha256

  Explanation: Proof size depends on the number of wires in the system. Wires depend on the constraints of the circuit. An increase in the number of constraints results in a larger proof size. Proof size also depends on the circuit.zkey size. The above comparison is based on small to large sizes

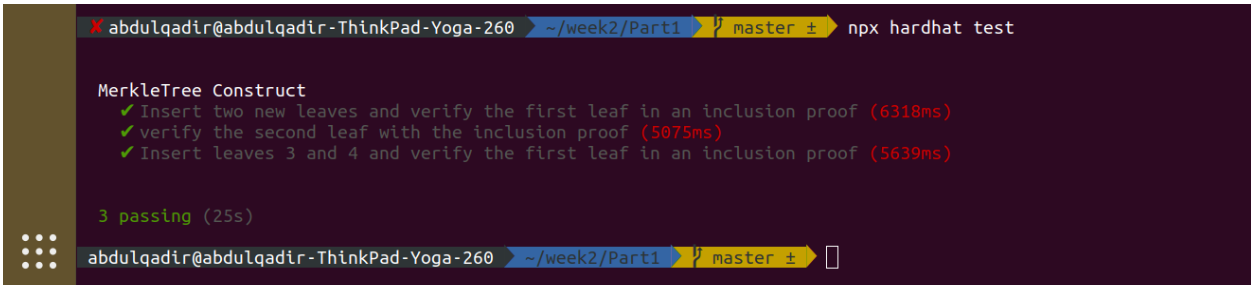
References:

➔ https://github.com/clearmatics/zeth/issues/4#issuecomment-483626241

➔ https://ethresear.ch/t/gas-and-circuit-constraint-benchmarks-of-binary-and-quinary-incremental-merkle-trees-using-the-poseidon-hash-function/7446/1

➔ https://medium.com/aztec-protocol/plonk-benchmarks-ii-5x-faster-than-groth16-on-pedersen-hashes-ea5285353db0

PART 2: [screenshot of all the tests passing of merkle tree]



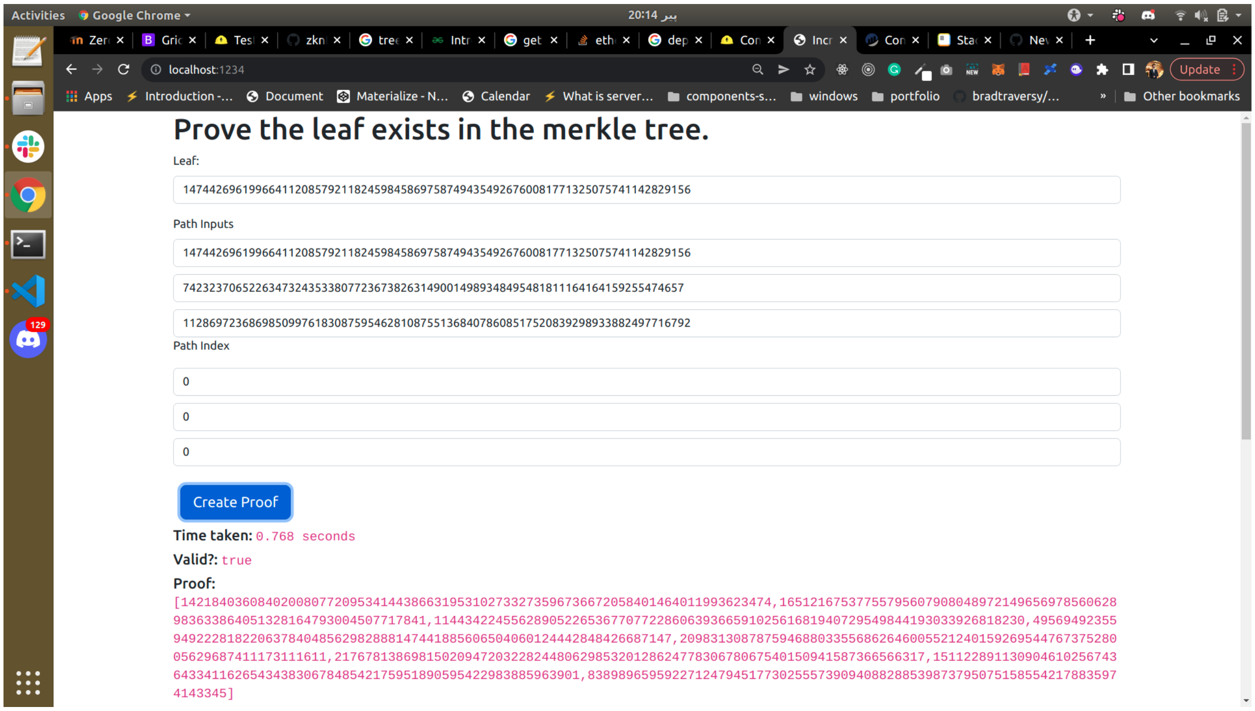
```
✘ abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week2/Part1  master ±  npx hardhat test


  MerkleTree Construct
    ✔ Insert two new leaves and verify the first leaf in an inclusion proof (6318ms)
    ✔ verify the second leaf with the inclusion proof (5075ms)
    ✔ Insert leaves 3 and 4 and verify the first leaf in an inclusion proof (5639ms)


  3 passing (25s)

abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week2/Part1  master ± 
```

Part 4: [Bonus]



Prove the leaf exists in the merkle tree.

Leaf:

14744269619966411208579211824598458697587494354926760081771325075741142829156

Path Inputs

14744269619966411208579211824598458697587494354926760081771325075741142829156

7423237065226347324353380772367382631490014989348495481811164164159255474657

11286972368698509976183087595462810875513684078608517520839298933882497716792

Path Index

0

0

0

Time taken: 0.768 seconds

Valid?: true

Proof:

```
[14218403608402008077209534144386631953102733273596736672058401464011993623474,16512167537755795607908048972149656978560628
983633864051328164793004507717841,11443422455628905226536770772286063936659102561681940729549844193033926818230,49569492355
94922281822063784048562982888147441885606504060124442848426687147,209831308787594688033556862646005521240159269544767375280
05629687411173111611,21767813869815020947203228244806298532012862477830678067540150941587366566317,151122891130904610256743
64334116265434383067848542175951890595422983885963901,838989659592271247945177302555739094088288539873795075158554217883597
4143345]
```

Question 2:

Part 1:

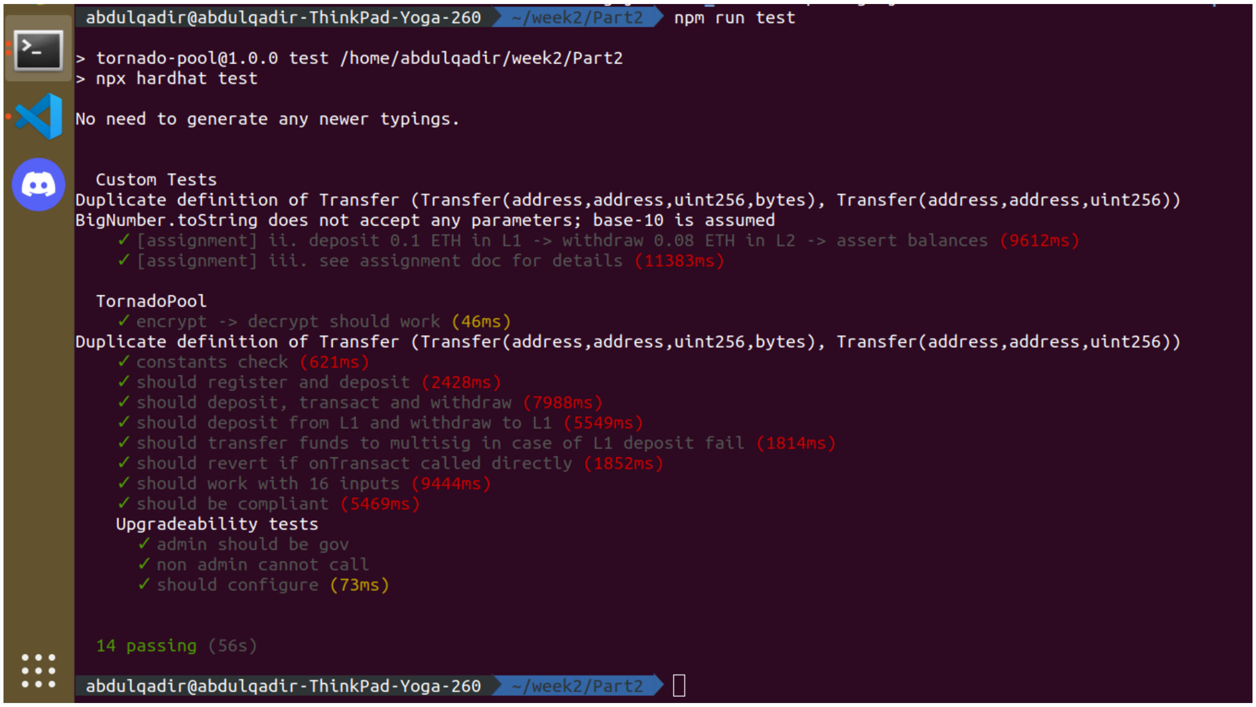
Tornado Cash Nova introduce the Arbitrary Amounts & Shielded transactions. In Tornado Classic user was only able to deposit and withdraw the fixed amount of eth. But in the nova version user can deposit and withdraw any amount he wants to. Users can also do the shielded transfers without revealing their identity, like ZCash.

Arbitrary Amounts and Shielded transactions become possible, using the Utxos modal in the tornado cash nova. When the user deposits funds in the tornado pool, he gets an Utxo for his deposits. Using Utxo, he can create a new Utxo to transfer to another account without revealing anything and withdraw using those Utxo.

Part 2:

Relayer is crucial to maintaining anonymity in the tornado cash while paying the gas fees. The relayer pays fees on behalf of the user while withdrawing. The fees and service fees are deducted from the withdrawal amount.

Part 3.1: [test cases passed screenshot]



```
abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week2/Part2  npm run test

> tornado-pool@1.0.0 test /home/abdulqadir/week2/Part2
> npx hardhat test

No need to generate any newer typings.


  Custom Tests
Duplicate definition of Transfer (Transfer(address,address,uint256,bytes), Transfer(address,address,uint256))
BigNumber.toString does not accept any parameters; base-10 is assumed
    ✓ [assignment] ii. deposit 0.1 ETH in L1 -> withdraw 0.08 ETH in L2 -> assert balances (9612ms)
    ✓ [assignment] iii. see assignment doc for details (11383ms)

  TornadoPool
    ✓ encrypt -> decrypt should work (46ms)
Duplicate definition of Transfer (Transfer(address,address,uint256,bytes), Transfer(address,address,uint256))
    ✓ constants check (621ms)
    ✓ should register and deposit (2428ms)
    ✓ should deposit, transact and withdraw (7988ms)
    ✓ should deposit from L1 and withdraw to L1 (5549ms)
    ✓ should transfer funds to multisig in case of L1 deposit fail (1814ms)
    ✓ should revert if onTransact called directly (1852ms)
    ✓ should work with 16 inputs (9444ms)
    ✓ should be compliant (5469ms)
    Upgradeability tests
      ✓ admin should be gov
      ✓ non admin cannot call
      ✓ should configure (73ms)


  14 passing (56s)

abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week2/Part2  
```

Question 3:

Part 1:

Semaphore is the zk tool that the users use to prove their identity without revealing it. Semaphore is used in privacy-enhancing applications that hide the user identity, such as login systems, anonymous DAOs, anonymous voting, and journalism. It is designed to be simple and can easily be used in the ethereum Dapps for adding the privacy layer. Simply semaphore is identity registration without revealing the identity.

Part 2:

Semaphore Contracts maintains the set of external nullifiers and prevents double-signaling to an external nullifier by the same identity commitment.

User,	External nullifier	Signal,	Accepted
Alice	123	getAmount()	Yes
Bob	456	getAmount()	Yes
Charlie	678	getAmount()	Yes
Bob	456	getAmount()	Terminated / No

External Nullifier is used and saved in the smart contract to ensure that the user does not double spent withdrawal.

Part 3:

- Attendance system
- Stock system (get data without revealing the identity or sell stocks without revealing identity).
- Semaphore based IMDb