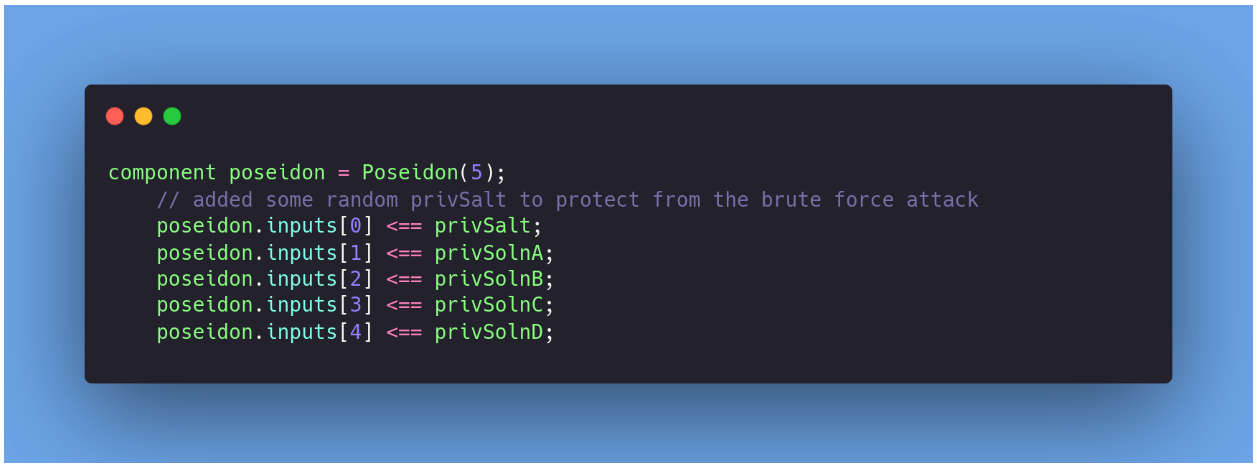
ZKU Week 3 Assignment

Email: qadir@xord.com
Discord Id: AbdulQadir#0432

Part 1 Circom in games

Question One:

We can quickly compute the proof for all the permutations in the hit and below circuit. But the author has used the random salt while hashing the private solution before comparing the guessed output with the secret solution. In this way, it is difficult to predict the salt and brute force the circuit to generate a valid proof.

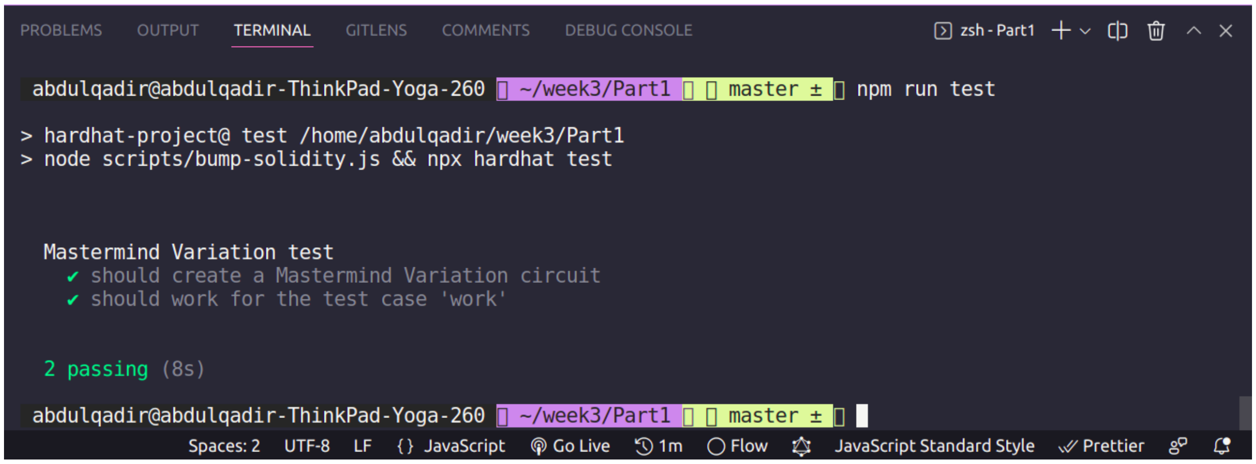


```
component poseidon = Poseidon(5);
    // added some random privSalt to protect from the brute force attack
    poseidon.inputs[0] <== privSalt;
    poseidon.inputs[1] <== privSolnA;
    poseidon.inputs[2] <== privSolnB;
    poseidon.inputs[3] <== privSolnC;
    poseidon.inputs[4] <== privSolnD;
```

Question Two:

[Code]()

Question Three:



```
abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week3/Part1   master ±  npm run test

> hardhat-project@ test /home/abdulqadir/week3/Part1
> node scripts/bump-solidity.js && npx hardhat test



  Mastermind Variation test
    ✔ should create a Mastermind Variation circuit
    ✔ should work for the test case 'work'


  2 passing (8s)

abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week3/Part1   master ± 
```

Question Four:

Game: Zk-Chessle https://jackli.gg/chessle/

- How it is played:

  Chessle is a wordle variant. The game consists of a single-player The player has to guess the starting three opening sequences, for both white and black. The player has six chances to guess the correct sequence, and after each turn, the player is given hints using the color that whether he has picked the right move or not. The green color indicates the right move on the correct sequence position. The yellow indicates the right move but on the wrong sequence position, and the Gray shows the bad move.

- How it can benefit from ZK:

  Chessle can benefit from the zero-knowledge by verifying the players' sequences that will be sealed and committed on-chain so no centralized judge is needed while no player can cheat. Also, the solution every day will be to remain private.

- How it can be implemented with ZK:

  Cheesle can be implemented with the ZK in such a way that the verification part of the guessed sequence of the player will be made through Zk. No third party will be involved in the verification process, and the solution will remain private.

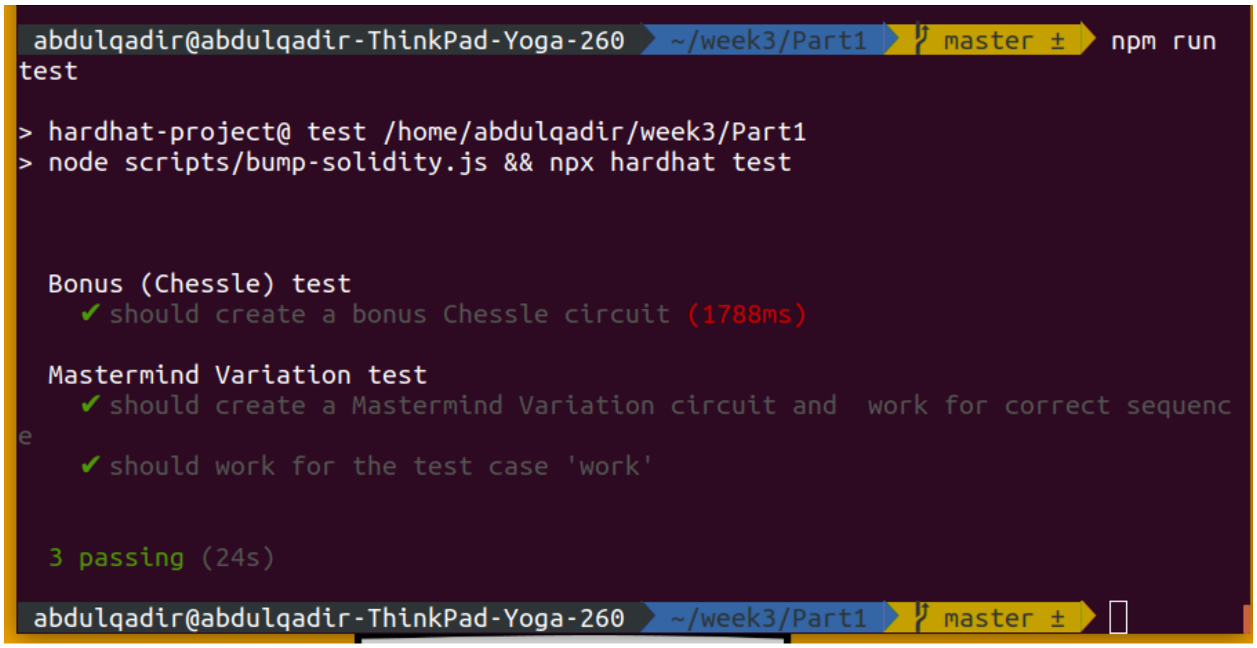
- How it is protected from brute-force attacks:

  In Chess, after both players move, 400 possible board setups exist. After the second pair of turns, there are 197,742 possible games, and after three moves, 121 million. It is difficult to brute force the solution in the cheesle, but the most common moves at the start. Hashing the solution with the random private salt will stop the brute force attack.

Question Five [bonus]:

code

Question Six [bonus]:



```
 abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week3/Part1  master ±  npm run
test

> hardhat-project@ test /home/abdulqadir/week3/Part1
> node scripts/bump-solidity.js && npx hardhat test



  Bonus (Chessle) test
    ✔ should create a bonus Chessle circuit (1788ms)

  Mastermind Variation test
    ✔ should create a Mastermind Variation circuit and  work for correct sequenc
e
    ✔ should work for the test case 'work'


  3 passing (24s)

 abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week3/Part1  master ± 
```

Part 2 Anti-collusion and Fairness

Question One:

Part 1:

It is easy to bribe voters to vote for a particular party in the on-chain voting system. MACI solves the problem of bribery and allows the user to have an on-chain voting process with significantly increased collusion resistance.

In MACI, if Alice is casting a vote and Eve is trying to bribe her to give her vote to party B. Alice has no way to tell Eve he has voted for Party B for two reasons.

- In MACI, Alice's vote is encrypted, so she can not prove whom she has voted for.
- Also, she can claim she has lost his private key and generate a new keyPair to vote again.

Part 2:

Several collusions are solved by MACI using the power of Zk-Snarks some of them arefollowsllow

- Collusion Resistance
- Privacy
- Receipt-freeness
- Uncensorability
- Unforgeability
- Non-repudiation
- Correct-Execution

Part 3:

MACI is a huge step toward solving the collision problem, but still, it does not solve all kinds of collusion. Hardware wallets or trusted hardware that prevents the fundamental changes and where attackers can be a part of the trusted hardware

Question Two:

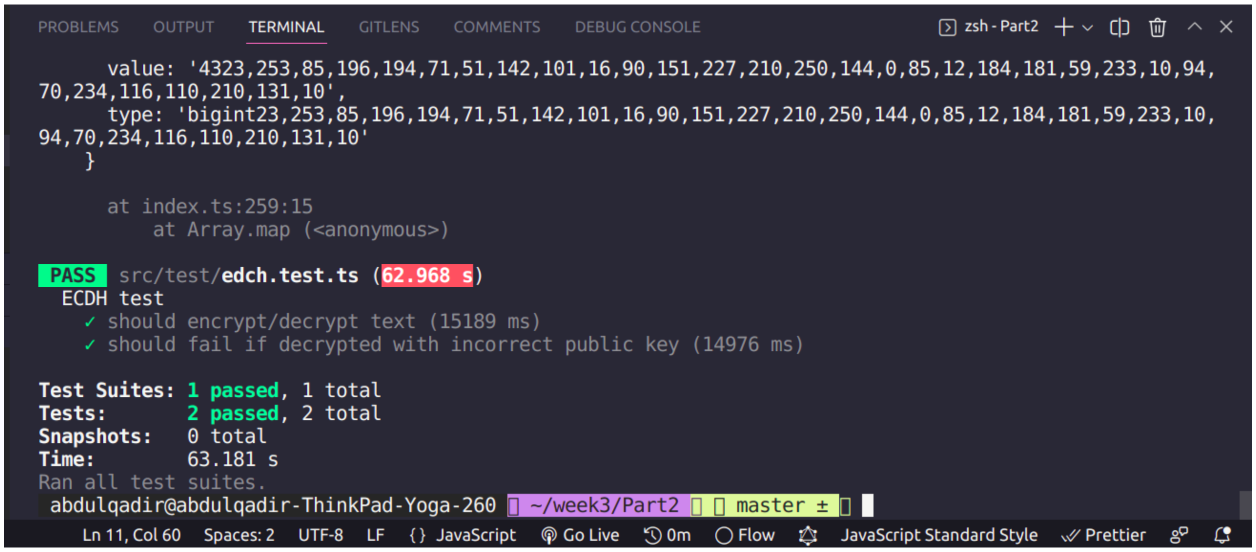
Part 1:

[Installation part]

Part 2:

code

Part 3:



```
      value: '4323,253,85,196,194,71,51,142,101,16,90,151,227,210,250,144,0,85,12,184,181,59,233,10,94,
70,234,116,110,210,131,10',
      type: 'bigint23,253,85,196,194,71,51,142,101,16,90,151,227,210,250,144,0,85,12,184,181,59,233,10,
94,70,234,116,110,210,131,10'
    }

      at index.ts:259:15
          at Array.map (<anonymous>)

 PASS  src/test/edch.test.ts (62.968 s)
  ECDH test
    ✓ should encrypt/decrypt text (15189 ms)
    ✓ should fail if decrypted with incorrect public key (14976 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        63.181 s
Ran all test suites.
 abdulqadir@abdulqadir-ThinkPad-Yoga-260  ~/week3/Part2   master ±
```

Question Three:

VDF is a function that requires running a given number of sequential steps to evaluate. But after, it is easy for anyone to verify whether it is correct. VDF is used in decentralized applications to generate public randomness in a trustless environment

Function f(x): Unique output for every input
Delay: Can be evaluated in time **T**
Verifiable: Correctness of function can be verified effectively

The function **f(x)** can be evaluated in **T** in any machine in a sequential amount of steps. Machine with many processors will take the same number of steps as it is sequential.

Question Four:

Part 1:

We can do this using a center relayer that collects all the user bids and maintains them in the Merkle tree on-chain—returning users the Utxo of their deposit. This way, we can solve both centralized server dependency issues and submit the bidding amount at the time of the bid.

Part 2:

We can introduce the Utxo modal in which the user submits his amount and get the Utxo. And at the time of the paying, the user will use his Utxo to pay for the bid amount he placed before in the auction and will get the remaining amount in the form of the Utxo return, or we can also utilize the smart contract to auto-pay back him and others user bids amount

Part 3 Time to start thinking about your final project!

Question One:

- **Relayer:** An center relayer that collects the user bids and sends a transaction on behalf of the user to add the amount to the pool and generate its Utxo's
  - **Framework:** Node.js, ethersJs

- **Smart Contracts:**
  - **Tools/Farmeworks:** solidity, hardhat

- **Frontend:** An frontend web application where the user can place the bid and interact with the relayer
  - **Framework:** React, ethersJs, material Ui

Question Two:

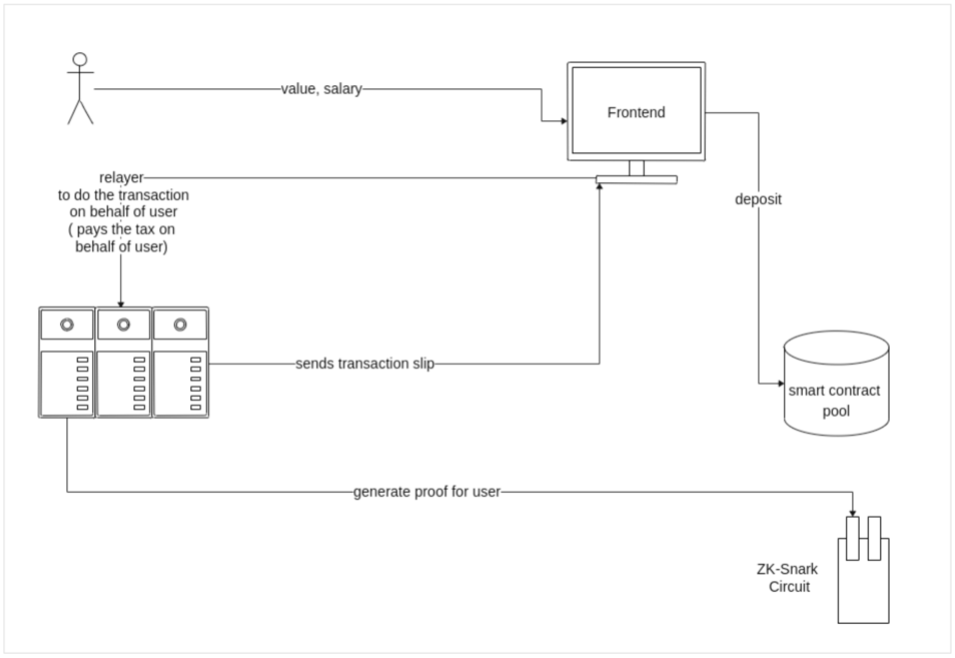
Idea 1: Tax on Salary - FBR

The problem tackled:

While paying salary tax to FBR causes the problem of revealing salary information and incentives. Also, if we make a decentralized system to pay the salary tax, it comes with the issue of showing the salary to all the blockchain. Everyone can know my salary if I pay my salary on-chain. Also, people can calculate it based on the amount of tax I pay to the FBR. So there should be a way to pay the tax to the FBR on-chain without revealing the salary information and proving that you have paid valid tax on your paycheck without revealing your paid tax and salary to other systems and blockchain.

A design overview:

A user will interact with the frontend application, which interacts with the relayer to generate the proof of private salary and the private amount of tax to be paid. Also, the relayer will send the transaction on the user's behalf and give back proof to the user.



Obstacles anticipated:

The problem faced in a zero-knowledge-based taxation system is the stealth transaction part. First, the tax paid by the actor should be stealthy. Then, we can relayer to do the transaction on behalf of the user.

<u>Idea 2: Zk-MultiSig (Signature Privacy)</u>

The problem tackled:

In a Multi-Sig wallet transaction, one can determine who has sent the transaction and the wallets that sign the transaction. Anyone can detect the wallet that has signed the transaction. Multi-Sig transactions reveal the addresses' identity, who owns it, and who signed it.

A Design Overview:

A person will create the proof that he is part of the Multi-Sig Wallet with a transaction he wants to execute that will be saved in the blockchain. Users can also add evidence of the transaction signing. Potential components can be:

- Frontend (interact with a circuit to create a proof of transactions)
- Relayer (A relayer will do the transaction on behalf of the user)
- Circuits (generate proofs)
- Smart Contracts (record the transaction)

Obstacles anticipated:

The Idea still needs to refine, and the architecture can solve the problem of privacy and identity revealing in the Multi-Sig wallet. Then, we can add the relayer or something that will get the proofs and do the transaction on behalf of the user.

<u>Idea 3: ZK-Quiz</u>

The problem tackled:

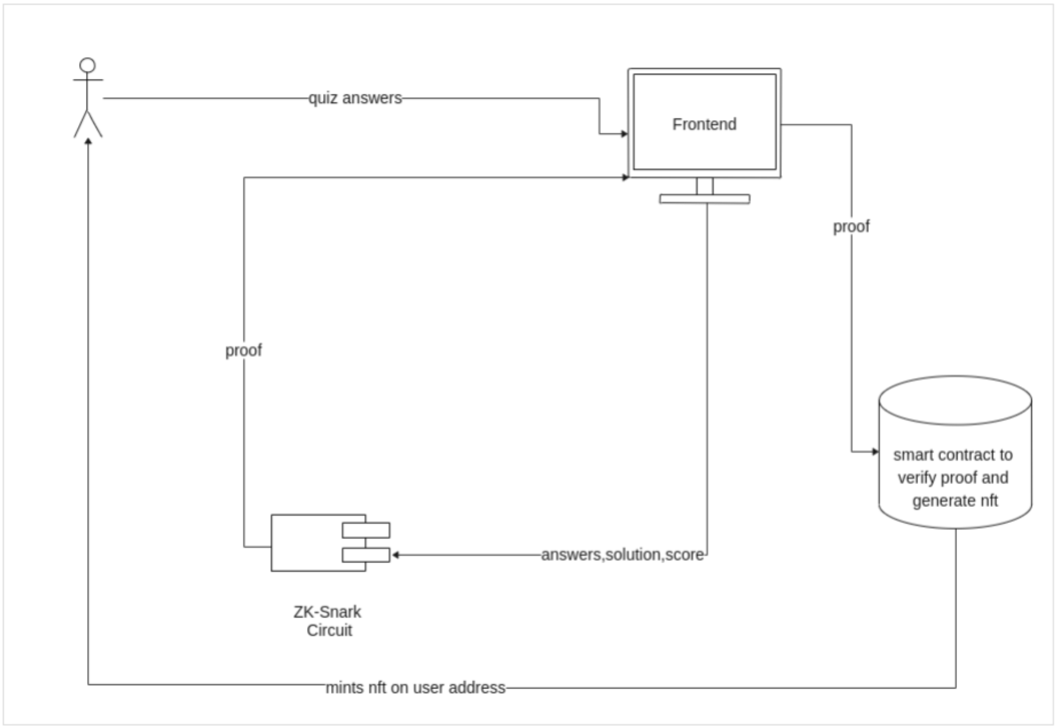
In the Quiz application, Solutions submitted by the user can be verified wrong by the party and come with the problem of leaking the solution. Other than this, a user can also bribe the checker to mark his answers right. Quiz Application causes the pain of bribery, verification, and privacy.

A design overview:

In Zk-Quiz user will submit the guessed answers on the frontend application that calculates the user's score based on the solution, and then it will generate the proof of the answers, solutions, and score on the frontend. The proof will then be sent to the smart contract to verify and mint the NFT of the result, aka score on user address.

There are multiple components of the frontend part of the ZK-Quiz, which can be:

- Create Quiz
- Attempt Quiz
- NFT Display


quiz answers
Frontend
proof
proof
smart contract to verify proof and generate nft
answers,solution,score
ZK-Snark Circuit
mints nft on user address

Obstacles anticipated:

The Obstacles anticipated in the Zk-Quiz will be around the circuit part to create a generalized circuit for all types of quizzes. Users can create any quiz with an arbitrary number of questions and answers. Other than this, the application's frontend part will also be one of the problematic parts.

References:

https://www.wazeesupperclub.com/what-are-different-chess-strategies/

https://medium.com/privacy-scaling-explorations/release-announcement-maci-1-0-c032bddd2157