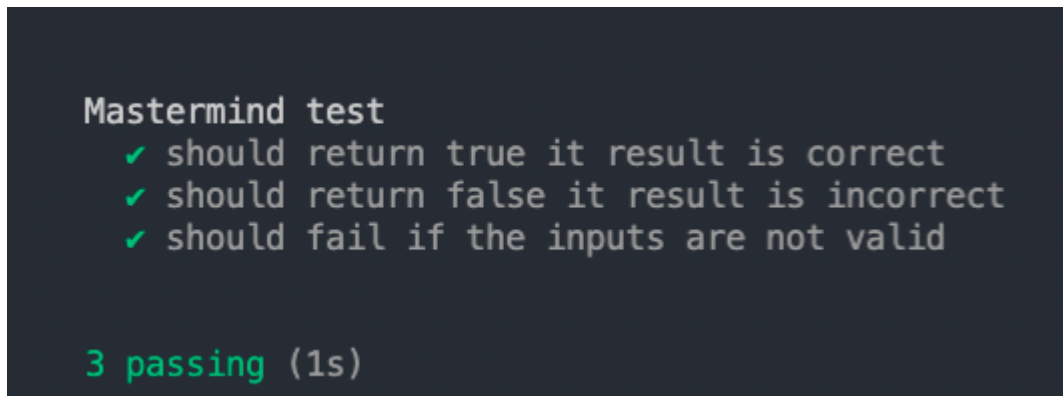


ZKU assignment - week 3:

Part1:

1. To prevent a brute force attack, the authors added a privSalt signal, which is a private input used solely to compute the hash of the solution (lines 92-98), making the combinations of the five inputs of the poseidon hash function too high to execute a brute force attack easily.
2. -----



```
Mastermind test
✓ should return true it result is correct
✓ should return false it result is incorrect
✓ should fail if the inputs are not valid

3 passing (1s)
```

- 3.
4. A game like hangman would benefit from zk because it wouldn't need a layer of trust between the game master, who knows the secret word, and the player. It would also benefit from zk because it would be more easy to deploy on a blockchain, since it wouldn't need to upload the secret word to the blockchain, but only the proofs of game interactions. We could implement hangman by creating a circuit that checks for the length of a string (which would be the first piece of information that the player needs from the game master) and another circuit that checks for the presence of a certain character in certain positions of the secret word. The game will end when the secret word is found out or when a certain amount of wrong guesses are submitted. This is also a feature that gives resistance from a brute force attack, because the attacker could only submit a few guesses before losing the game therefore invalidating the effort done in the attack.

Part 2:

1. MACI works by keeping an encrypted registry of addresses of voters and. In the timeframe of voting, users can perform two actions, submit a vote, or change their key. Since all of these actions are submitted using zero-knowledge and encrypted with a personal salt value, each user can proof its transaction, but it's never possible from an external viewer, who doesn't know salt value, to check what has been submitted.
This implementation solves the problem of bribing voters, infact, voters can submit to bribers a proof that they sent a certain vote from a certain address, but bribers cannot verify that this address was on the registry at the moment of submission, thus disincentivizing bribes. Also, it disincentivizes selling or renting addresses, because it could lead to the stealing of the amount staked for the vote and the malicious user could not be sure that his new address is still valid when he sends his transaction.

MACI, however, does not solve problems regarding trust in the role of the operator, who manages the registry and decrypts transaction data.

```
> template@1.0.0 test
> jest

PASS src/test/edch.test.ts (15.626 s)
  ECDH test
    ✓ should encrypt/decrypt text (4176 ms)
    ✓ should fail if decrypted with incorrect public key (4138 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        15.677 s, estimated 16 s
```

- 2.
3. Since VDFs are functions that need to be calculated in steps, require the output of the previous step for the next one and each step does not give away information of the result until the last computation, running parallel processors is not an efficient way to find the solution more quickly.
4. We could create a dashboard where users that participated at the auction could submit a proof that they bid at least X. Then other participants that bid higher could submit proofs that they bid at least $Y > X$. The process ends when no user has bid more than what has been asserted.
We could make users deposit a bid fee of a standard amount when a bid is made, alongside with the proof of the bid value. Then, users could claim their deposit at the end of the auction if the bid is found out to be valid.

Part 3:

1. First I would need a smart contract that manages the bidding: it would take in input the proof of a certain bid and store it on the blockchain. Then a smart contract that manages the thread for claiming the auction. For the smart contract side I would use hardhat for building, testing and deploying. For the frontend I would need a page for managing the auction, and a page for managing the claim. For this I would use React with a Chakra UI.
2. a. One project that I could tackle for my final assignment is the creation of a dapp that allows users to mint NFTs that attest their membership in communities locked by certain parameters, without exposing their wallet or other info that they wouldn't want to make public when interacting with that community. For example, some users would want to participate in communities that are locked by the possession of an NFT, but without giving away the information of their personal wallet (the tokens they hold, the value of their wallet, the protocols they interact with,...). This could be done by giving a wallet some kind of proof that it has the same owner as an account that satisfies the requirements to join a specific community.

b. Another idea could be a platform to make a survey to which users have to answer truthfully. A user who creates a survey could ask for some data obtainable from the wallet and stake some amount in the dapp to pay users that take the survey. Users then could fill the form and submit it to compute a proof of their answers that then would be sent to the survey taker, and unlock the prize. I would also like to implement a way to allow users to maintain their privacy when answering questions.

c. Lastly, I could build a variation of [stratego](#) that takes advantage of zk uniquenesses to provide a new layer of challenge for the players. Stratego is a tabletop game where each of two players fight on a 10x10 grid with some pawns that represent units of an army. The uniqueness brought by a zk setup could be some kind of fog of war, that is that users can only see squares on the grid close to their units. Another interesting improvement could be that we could assign different movement properties to different units, without giving away what the units are (in stratego, units identities are kept secret until combat), for example, some units could move as a rook in chess, or some others could only move each other turn,... Also the combat would be improved because only one player (the player who attacks) would be needed to reveal the information about their unit.