Cryptography

Section -1

ZoKrates Program Explanation and Procedure:

This ZoKrates program is designed to take four private inputs, compute their SHA-256 hash, and check whether the resulting hash matches the expected values provided as additional inputs. Here's a clear explanation of the code and how to work with it:

Step 1: Overview of the Program

- **Imports**: The program uses a SHA-256 hashing function by importing sha256packed from ZoKrates' library.

```
import "hashes/sha256/512bitPacked" as sha256packed;
```

- **Main Function**: The main function takes six arguments:
  - **a, b, c, d**: These are private inputs that will be used as the data to hash.
  - **h0, h1**: These are the expected hash values for the inputs (a, b, c, d).

```
def main(private field a, private field b, private field c, private field d, field h0, field h1)
```

- **Hash Calculation**: Inside the function, the sha256packed function takes the four inputs [a, b, c, d] and calculates the 256-bit SHA-256 hash. Since the hash is 256 bits long, it's split into two parts of 128 bits each: h[0] and h[1].

```
field[2] h = sha256packed([a, b, c, d]);
```

- **Assertions**: The program then checks if the calculated hash (h[0], h[1]) matches the expected hash values (h0, h1). If they match, the program completes successfully. If not, it will throw an error.

```
assert(h[0] == h0);
assert(h[1] == h1);
```

Step 2: Inputs and Outputs

- **Inputs**:
    1. a, b, c, d: The data values to be hashed.
    2. h0, h1: These are the expected hash values. After running the hash function on a, b, c, d, the program checks if the output matches these.
- **Outputs**:
    - The program does not explicitly return a value but verifies whether the computed hash matches the expected result. If it matches, the program runs successfully.

Step 3: Running the Program

To work with this program, you will follow these steps:

1. **Set up ZoKrates**:
    - First, make sure you have ZoKrates installed and configured on your machine.
2. **Compile the Program**:
    - Use the following command to compile the program in bash or terminal:

```
zokrates compile -i <your_program.zok>
```

3. **Compute Witness**:
    - Now, you need to compute the witness by providing the required inputs (a, b, c, d, h0, and h1). This is the command you will use:

```
zokrates compute-witness -a <a> <b> <c> <d> <h0> <h1>
```

    - Replace <a>, <b>, etc., with actual numeric values. For example:

```
zokrates compute-witness -a 337 113569 1 2 12345 67890
```

4. **Generate Proof**:
    - After computing the witness, generate a proof:

```
zokrates generate-proof
```

5. **Verify the Proof**:
   - To verify that the computed hash matches the expected hash, run:

     ```
     zokrates verify
     ```

Step 4: Example Use Case

Let’s say you want to verify that the SHA-256 hash of a=337, b=113569, c=1, and d=2 matches the hash represented by h0=12345 and h1=67890. You would run the program by passing these values to compute-witness. If the hash doesn’t match, the program will fail the assert check and give an error.

Step 5: Common Errors

- **Mismatched Inputs**: If you don't pass the correct number of inputs to the compute-witness command, you’ll get an error saying the program takes more or fewer arguments than expected.

For instance, if your program expects six inputs but you only pass two, you’ll get an error like:

```
Program takes 6 inputs but was passed 2 values
```

- **Assertion Failures**: If the computed hash doesn’t match the expected h0 and h1, the program will throw an assertion failure.

Step 6: Modifying the Program

If you want to change what data is being hashed or add more inputs, you can modify the main function to accept more arguments and pass those into the sha256packed function. You can also update the assert statements if you want to check different hash values.