ZKPlus Writeup

Smart conttrac analyze

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

pragma abicoder v2;

library AdvancedZKPLibrary {
    struct Proof {
        uint256 a;
        uint256 b;
        uint256 c;
    }

    function hash(Proof memory proof) internal pure returns (bytes32) {
        return keccak256(abi.encodePacked(proof.a, proof.b, proof.c));
    }

    function verifyProof(Proof memory proof, address from, address to, uint256
amount) internal view returns (bool) {
        uint256 challenge = uint256(keccak256(abi.encodePacked(from, to,
amount))) % 2**128;
        uint256 sum = proof.a + proof.b;

        uint256 hashed_sender_relation =
uint256(keccak256(abi.encodePacked(msg.sender, proof.a))) % 2**128;

        return (sum == amount)
            && (challenge * proof.a % 2**128 == proof.c)
            && (proof.a * proof.b % 2**128 == proof.c)
            && (hashed_sender_relation == proof.b);
    }
}
contract AdvancedZKPCTF {
    using AdvancedZKPLibrary for AdvancedZKPLibrary.Proof;

    uint256 private constant TOKEN_SUPPLY = 100000000000000000000000000;
    address owner;
    mapping(address => uint256) private balances;
```

```
    mapping(bytes32 => bool) private usedProofs;

    constructor()  {
        balances[msg.sender] = TOKEN_SUPPLY;
        owner = msg.sender;
    }

    function transfer(address to, uint256 amount, AdvancedZKPLibrary.Proof
memory proof) public {
        require(amount <= balances[owner], "Insufficient offer");
        require(!usedProofs[proof.hash()], "Proof already used");
        require(AdvancedZKPLibrary.verifyProof(proof, msg.sender, to, amount),
"Invalid proof");

        usedProofs[proof.hash()] = true;
        balances[owner] -= amount;
        balances[to] += amount;
    }

    function balanceOf(address account) public view returns (uint256) {
        return balances[account];
    }

    function isSolved() public view returns (bool) {
        return balances[owner] == 0;
    }
}
```

The codes' logic are simple. We must make the balance of contract's owner become 0. Noticed that the transfer function can transfer the owner's balance to any address. The required conditions said the transfer amount <= balances[owner] and cannot pass the used proofs. So we should construct the proof struct that satisified 4 conditions:

```
a + b == 100000000000000000000000000,
a * b % 2**128 == c,
challenge * a % 2**128 == c
hashed_sender_relation == b
```

That means we should find the "to" address and controllable "from" address, int a, b, c to satisified the constrait conditions.

Idea

Of course, we can count the 5 values by force trial, but since it is a contrait solver problem, we can use Z3 solver to solve the values.

POC

```
from z3 import *
from web3 import Web3
# constants
amount = 100000000000000000000000000
msg_sender = "0x8a49383......52D5B4" # use the address you
can controll
solver = Solver()
# variable
a, b, c = Ints('a b c')
to = Int('to')
# constraint conditions
proof_conditions = And(
    a + b == amount,
    a * b % 2**128 == c
    b == Web3.to_int(Web3.solidity_keccak(['address',
  'uint256'], [msg_sender,a])) %
  2**128
)
solver.add(proof_conditions)
solver.add(to >= 0)
solver.add(to <= 2**160 - 1)
   found = False
  # solve
  while solver.check() == sat and not found:
      model = solver.model()
      to_value = model[to].as_long()
      to_address = "0x%040x" % to_value
      challenge = Web3.to_int(Web3.solidity_keccak(['address', 'address',
  'uint256'], [msg_sender, Web3.to_checksum_address(to_address) , amount])) %
  2**128

      if challenge * a % 2**128 == c:
          found = True
          print("found solve:")
          print("to:", to_address)
          print("challenge:", challenge)
          print(f"a: {model[a]}")
          print(f"b: {model[b]}")
```

```
            print(f"c: {model[c]}")
        else:
            solver.add(to != to_value)
    if not found:
        print("No solve")
```

Suggest

1. Run the code with 5 or 6 v100 GPU instances can improve the solving time!
2. use Overflow to expand the range of values for a and b