Project

This project is divided into 3 phases:

1. RNG phase
    a. Commit phase
    b. Reveal phase
    c. Reveal private keys
    d. Reconstruction phase
2. Determine random function
3. Casino Deposit Phase
4. Bet phase
5. Reveal PQR phase
6. Report phase
7. Withdraw Money

Step 1: RNG Phase: RSA Cryptosystem + Secret Sharing + Goldwasser-micali cryptosystem

At the start of the day, both casino and authority generate a pair of public and private keys using goldwasser-micali cryptosystem and RSA cryptosystem to get (n_c, x_c), (p_c, q_c), (e_c, d_c), (n_a, x_a) and (p_a, q_a) and (e_a, d_a) respectively. They should make sure that they suffices all the requirement of both cryptosystem as they will be checked in the SC. For example, casino should make sure that $lcm(p_c - 1, q_c - 1) | e_c d_c - 1$ to make sure that it can prove that both rsa and goldwasser-micali keys are generated from the start and not modified in the middle. Then, they should provide a $n_{secretsharing} > max(n_c, n_a)$ so that when players encrypt their values, the next larger prime number generated using $n_{secretsharing}$ is ensured to be larger than each encrypted bits.

Then, after key generation, casino deploy the SC and submit the public keys to the SC so that (n_c, n_a, x) and (e_c, e_a) are visible to all nodes on the network.

Commit

For each public player, since they can choose not to reveal in the reveal phase, we need to use secret sharing where the casino and authority can reconstruct their $Enc_{goldwasser}(r)$ in case they don't reveal. Since in the reveal phase, players will reveal in different time, and hence attackers may choose not to reveal after seeing others' revealed values. So, we need to grant the SC the ability to reconstruct players' values but at the same time prevent any of the casino and authority to know even the encrypted bit of the players.

1. Each player chooses a r
2. They produce $Enc_{goldwasser}(r) = [Enc(b_1), Enc(b_2), \ldots, Enc(b_{16})]$ using (n, x) of the goldwasser-micali cryptosystem
3. Then, they choose a p for secret sharing purpose. Since each $Enc(b_i)$ is guaranteed to be smaller than n, we can generate p that is the nearest prime to n, or at least larger than all $Enc(b_i)$. Then all the calculations in this step are $mod\ p$ (It is the player's responsibility to have a valid p. If the casino can't reconstruct $Enc_{goldwasser}(r)$ using the p they provided, their value will not be cumulated in to the encrypted R on the SC).
    a. For each $Enc(b_i)$ they generate a polynomial g of degree 1 so that $g(0) = Enc(b_i)$
    b. They generate $s_i = (g(1), g(2))$, so they can have $s = [s_1, s_2, ..., s_{16}]$
    c. For each s_i, they calculate $s_i' = (g(1)^{e_c}, g(2)^{e_a})$ ($mod\ n_c$ and $mod\ n_a$ respectively)
    d. Finally, they will have $s' = [s_1', s_2', ..., s_{16}']$ and $h(s)$
4. Then, the final product they will commit to the SC by calling `publicCommitRNG` is $h(s), s', p$

In this phase, for casino and authority, they can just commit using simple commitment scheme as we require them to reveal all the commited hashed values before proceeding to the bet phase.

So they need to:

1. For each time they play they choose a r
2. Then they produce $Enc_{goldwasser}(r_i) = [Enc(b_1), Enc(b_2), \ldots, Enc(b_{16})]$ using (n, x) published before.
3. They commit $h(Enc_{goldwasser}(r_i))$ to the SC by calling `nonPublicCommitRNG`

For the hash function h used in the commit phase, it should produce the same result as this function in solidity:

```
// Helper function to encrypt values
function sha256Encoder(uint256[2][16] memory listToEncode) pure
    return sha256(abi.encode(listToEncode));
}
```

Moreover, we only allow each public players to commit once, and only after the times that casino and authority committed is equal to the times that they signed up in the sign up phase can we start the next phase.

After commit phase, the casino should make sure that the authority has committed `t` times and start the next phase by calling `startNextPhase`. Also, the above Enc function should be provided by the casino so that every player can generate the value that they should commit directly.

Reveal

This phase is again divided into public reveal and nonPublic reveal.

For public players:

1. They submit s and their value k without deposit to reveal

2. Then the SC will check if the hash of this s indeed match the $h(s)$ they committed before

3. If they match, the SC will record their value of k and also that they have revealed honestly in this phase (at least temporarily)

In the later reconstruction phase, the casino and authority will reconstruct their value of s using s', only if the decrypted s matches with $h(s)$, we can be sure that this player did not try to commit s' and s that are produced by different r, and try to tamper with the encrypted R that will be cumulated in the reconstruction step by inspecting the s that are already revealed by other players.

In addition, since the revealed value is s, even if the players don't reveal, we ensure that this decision is not made after they see other players' $Enc_{goldwasser}(r)$.

For non-public players, they simply reveal $Enc(r)$ for the SC to check whether their hashes match with the one they committed.

Only after casino and authority revealed all their committed values and after a specific amount of time can the casino start the next phase.

Reveal Private Keys

In this phase, the casino and authority should reveal their private keys (d_c, d_a) to the SC by calling `revealPrivateKeys` function. Only after both the keys are revealed can we start the reconstruction phase.

Reconstruction

In this phase, the casino should simply call the `reconstruct` function for each public players that have committed before by passing the address of this player into the function.

1. The SC will decrypt each element in the s' by $Dec_{rsa}(s'_i) = (g(1)^{d_c e_c}, g(2)^{d_a e_a})$ using the private keys (d_c, d_a) revealed in the previous phase to get $s_i = (g(1), g(2))$

2. Then, SC can get s

3. SC will check $h(s_{reconstructed}) = h(s_{committed})$

a. If true

   i. SC reconstructs $Enc_{goldwasser}(r)$ using s and cumulates this value into the $Enc_{goldwasser}(R)$, which is the XOR results of all players' $Enc_{goldwasser}(r)$

   ii. SC records the value k submitted by the player in the reveal phase and allows them to bet without depositing (so they may have a try without loss).

      1. if this k is already taken up by other players → automatically increase 1

   iii. SC records that this player played honestly in the RNG

b. If false

   i. SC disallows this player to bet in the subsequent phases

   ii. SC records that this player played dishonestly in the RNG

4. Casino can start depositing money

Step 2: Determine random function

In this step, casino only need to confirm with authority that they should use the same function that produces the same result as the `myRand()` function on the SC. The `myRand()` function looks like this:

```
int myRand(unsigned int seed) // RAND_MAX assumed to be 32767
{
   myNext = seed;
   myNext = myNext * 1103515245 + 12345;
   return (unsigned int)(myNext/65536) % 32768;
}

x = myRand(decryptedR+k); // x = 29969 when decryptedR is 5257 a
```

where `x` is the value that is used to determine the results of the bettors

Step 3: Casino Deposit Phase

In this phase, the casino should deposit this time a large amount of money into the SC. The amount is set to 90 ether so it is enough for 9000 bets. If casino did not deposit enough money into the SC, the bet phase can not start. It has no incentive to not deposit money.

Step 4: Bet Phase

Flow

1. Casino first gets the `encryptedR` stored on the SC and decrypt it using (p, q) to get `decryptedR` . It then waits for the bettors to submit their value of `k`
2. Bettors submits their value of `k` and deposits 0.01 ether into the SC by calling the `bet` function:
    a. We only accept those `k` which can fit in `uint256` variable type
    b. We don't allow casino to bet as they know `decryptedR` and can always submit those `k` that wins
    c. We don't allow the same `k` to be used twice. If the bettor submits an used `k` , their deposits will be returned
    d. Then, the bet
        i. this `k` is used in `KUsed`
        ii. who betted this `k` in `KBettedBy`
        iii. this bettor has betted another `k` in `bettorBet` of this bettor
        iv. the amount of times that this bettor betted by increasing `timesBettorBetted` of this bettor by 1
3. Casino calculates `x` by passing the sum of `decryptedR` and `k` to the `myRand` function off-the-chain.
4. Then, casino checks whether `x` is even or odd.
    a. If `x` is even → bettor wins → `result` = true

b. If `x` is odd → bettor loses → `result` = false

5. The casino submits `k` and `result` to the `winOrLose` function, the function will record:
   a. the result of this `k` is announced in `KAnnounced`
   b. the result of this `k` in `KResult`
6. If `result` is true, the SC will pay 0.02 ether to the bettor that betted this `k`

Logic

In the bet phase, for each bet that the bettor submitted, the SC will record it in `bettorBet`
.

Since the k and result are all submitted by the casino, it can cheat in the following ways:

1. It uses different values for k or R to generate x
2. After it discovers that the bettor has won on this bet, it can choose not to announce, so that it won't lose

However, they can be checked if the bettor reports in the report phase.

Step 5: Reveal PQR Phase

After the bet phase, the casino should reveal the `encryptedR` stored on the SC along the way. In this phase, casino should submit the private key (p, q) to the `revealR` function.

Then, this function will check

1. whether this private key is indeed associated with the public key on the SC by:
   a. calculate the jacobi symbol of x with p and q
   b. only if $jacobi(x,p) == -1 \;\; and \;\; jacobi(x,p) == -1$, `x` is indeed not a quadratic residue `mod p` and `mod q` → private key (p, q) is the right private key
   c. and only if $lcm(p_c - 1, q_c - 1) | e_c d_c - 1$, casino submitted the right key d_c in the previous phase and has not cheated
   d. and only if $p * q = n_c$:
      i. the function will store (p, q) on the SC

ii. it calculates the uses the private key to decrypt the `encryptedR` stored along the way and stores `decryptedR` on the SC

iii. it starts the report phase

We don't need to submit the actual value of `R` since the SC can calculates `decryptedR` itself providing the private keys. This `decryptedR` is guaranteed to be honest since it is modified and cumulated by casino, (public), and authority players' submitted values in and only in the RNG phase. It is also guaranteed to be random if the RNG phase of this protocol can really ensure that casino, (public), and authority players cannot tamper with `R`.

If the casino did not reveal the private keys within the end of this phase, all bettors who have betted in the bet phase can withdraw $remainingBalance/numberOfBettor$ ether by calling `bettorWithdraw` function.

Also, authority must call this function and provide (p_a, q_a) to check that it did not cheat by submitting the wrong key in the previous phase. If the authority does not call this function after the report phase, it will be regarded as cheated and can be detected.

Step 6: Report Phase

In this phase, every bettor who has betted before can call the `report` function to check if the casino cheated in their bets.

For every bet each bettor has betted, the `report` function checks whether casino cheated by:

1. if casino did not announce for this bet → returns twice the bet to the bettor
2. if casino has announced for this bet:
    a. calculate `calculatedX` using `decryptedR` and the value `k` of this bet by passing this two values to the `myRand` function. It is a function that reduplicates the calculation of `x` on the chain, which was done by casino off-the-chain in the bet phase.

```
function myRand(uint256 seed) view public returns (uint25
    require(block.timestamp > reportStart && block.timest
    seed = seed * 1103515245 + 12345;
```

```
    return (seed/65536) % 32768;
}
uint256 calculatedX = myRand(decryptedR + bets[i]); // whe
```

b. if this `calcualtedX` is odd and the casino reported that this bet has won → pay twice the bettor and set `casinoCheated` to true

c. if this `calcualtedX` is even and the casino reported that this bet has lost → pay twice the bettor and set `casinoCheated` to true

The `calculatedX` is guaranteed to be the honest result of each bet since:

1. `decryptedR` is the value that SC calculated by decrypting the `encryptedR`, which was guaranteed to be generated by casino, (public), and authority players in the RNG phase, using the private key provided by the casino (which was proved to be associated with the public key the casino announced at the deployment of this SC). So `decryptedR` is guaranteed to be random and honest.
2. `bets[i]` is the value of `k` which the bettor themselves provided

It hence can report the if the casino cheated using the method raised in bet phase:

1. Casino uses different values for `k` or `R` to generate `x` or it directly reports the wrong result
    a. The SC itself calculates the honest result, if this result matches the result submitted by the casino, then the casino did not cheat (we don't care if the casino uses different k and r as long as the result is honest).
2. Casino does not announce to prevent losing
    a. If the casino did not announce, the bettor can get twice their bets in the report phase if they report.

Step 7: Withdraw Money

At the end of the day, if the `casinoCheated` is false → there is no report indicating that the casino has cheated → the casino can withdraw all the remaining balance in the SC by calling `casinoWithdraw` function.

If `casinoCheated` is true → casino has cheated for at least one bet → casino cannot withdraw money and all bettors who have betted before can withdraw $remainingBalance/numberOfBettor$ ether by calling `bettorWithdraw` function.