Protocol

1 RNG

Random number r generation (Requirements 1, a, b, c, d, e, f, g)

The casino will generate key-pair in Goldwasser-Micali cryptosystem, publish $pk = (x, N)$ to the blockchain and keep $sk = (p, q)$ secret. During a period of time the casino will accept RNC from players, $r_i' = Enc_{gm}(r_i)$, where r_i is the random int16 given by the player and Enc_{gm} is the encryption function in Goldwasser-Micali cryptosystem using casino pk. So that only the casino can decrypt the r_i' and get r_i. This process will be recorded in the blockchain.

```
// RNC phase, player can encrypt their r_i by the gm encryption system.
bytes32 verEnc_r = 0x0000000000000000000000000000000000000000000000000000000000000001; // the multiple of all enc(ri) until a time
mapping(address => bytes32) private enced_ris; // player to their enc(ri)
address[] public RNCcontributedPlayers; // list of contributed players that is not controlled by the authorities
bytes32[] public authorities_enced_ris; // list of enc(ri) given by the authorities
uint private t; // the number required by the authorities
function contributeRNC(bytes32 enced_ri) external{
    require(!isCasinoOpened, "RNG finished");
    require(enced_ris[msg.sender]!=bytes32(0), "You have already contributed.");
    if(msg.sender != authorities){ // this player is not controlled by the authorities
        require(block.timestamp <= RNCTime, "The RNC time for player not controlled by authorities is over.");
        enced_ris[msg.sender] = enced_ri;
        RNCcontributedPlayers.push(msg.sender);
        t = calT();
    }
    else if (msg.sender == authorities){ // this player is controlled by the authorities
        if(block.timestamp > RNCTime){ // normal RNC time is over, only allow authorities add enc_ri if t is not enough
            require(authorities_enced_ris.length < t, "The authorities have already controlled t players.");
        }
        authorities_enced_ris.push(enced_ri);
        t = calT();
    }
    verEnc_r = bytes32(uint256(verEnc_r) * uint256(enced_ri));
}
function calT() private view returns(uint t){
    uint n = RNCcontributedPlayers.length + authorities_enced_ris.length;
    uint value;
    if (n % 2 == 0) value = n / 2;
    else value = n / 2 + 1;
    return value;
}
```

After the RNC period, the casino will use all the r_i to generate a random number r by XOR all the r_i together, i.e. $r = r_1 \oplus r_2 \oplus \cdots \oplus r_n$. The casino will not publish r to the blockchain and keep it secret but will publich $Enc_{gm}(r_1 \oplus r_2 \oplus \cdots \oplus r_n) = Enc_{gm}(r_1) \times Enc_{gm}(r_2) \times \cdots \times Enc_{gm}(r_n)$.

```
// the casino is able to decrypt r because they have sk=(p,q), they keep r secret
bytes32 public enc_r; // the pulish enc_r used for the day
bool private isEnc_rPublished;
uint private betLimit; // bet limit for requirement i
function pulishEnc_r(bytes32 _enc_r) external payable onlyAfter(RNCTime) {
    require(msg.sender == casino, "Only casino can public Enc(r) and open casino.");
    require(msg.value >= 100 ether, "Must deposits a huge amount of money in the smart contract");
    require(!isEnc_rPublished, "Casino have already opened.");
    require(authorities_enced_ris.length >= t, "The authorities have not control at least t players yet.");
    enc_r = _enc_r;
    openTime = block.timestamp;
    closeTime = openTime + OPEN_PERIOD;
    reveal_r_Time = closeTime + REVEAL_R_DEADLINE;
    reportTime = reveal_r_Time + REPORT_PERIOD;
    isEnc_rPublished = true;
```

```
16      isCasinoOpened = true;
17      betLimit = (msg.value - 1 ether)/(0.01 ether); // -1 ether as 1 ether will be given to RNG
            players
18      emit BettingPeriod(openTime, closeTime);
19  }
```

Requirements explaination:

1. r is only visible to the casino and no one else has any information about it. Also, in order to ensure that the authorites have control at least $t = \lceil \frac{n}{2} + 1 \rceil$ players, the casino will only open the casino after the authorities address have contributed at least t number of $Enc_{gm}(r_i)$ to the casino, it is the assumption that the authorities will provide enought $Enc_{gm}(r_i)$ to the casino from authorites address.

a. The RNG is open to everyone for participations as all of them can send $Enc_{gm}(r_i)$ to the casino during `RNGTime`. During the RNC period, everyone can send $Enc_{gm}(r_i)$ to the casino by calling contributeRNC() function. And the player controlled by the authorities can send $Enc_{gm}(r_i)$ to the casino through authorities address by calling contributeRNC() function too.

b. r is only visible to the casino because it is encrypted by the casino's pk of GM cryptosystem. No one else can decrypt $Enc_{gm}(r)$ without the casino's sk.

c. RNG is tamper-proof

- The casino cannot tamper with the result even if it colludes with all the players who are not controlled by the authorites becasuse $Enc_{gm}(r) = Enc_{gm}(r_1) \times Enc_{gm}(r_2) \times \cdots \times Enc_{gm}(r_n)$ is publiched, any $enc_{gm}(r_i)$ given by any player will change the result during RNC period.
- The authorites cannot tamper with the result even if they collude with all the players who are not controlled by the casino becasuse $r = r_1 \oplus r_2 \oplus \cdots \oplus r_n$, as r depends on all the r_i and at least one of them is controlled by the casino.
- No one else can tamper with the result as it is all encrypted on the blockchain and front running is no use.

d. No one can predict the result of the RNG as any r_i is unpredictable and will be XOR with other r_i. Before the casino release their r and $sk = (p, q)$, no one can decrypt $Enc_{gm}(r)$ without casino GM sk even if the authorites collude with all non-casino players to predict r.

e. The value r is generated uniformly at random as it is the XOR of all the r_i which are given by the players that they generate uniformly at random.

f. The RNG will never fail as long as there are players participating in the RNG by calling `contributeRNC()` and the authorites will contributed at least t player as they required, which is the assumption that they will. No other further actions are required for any players.

g. As the player contributed the to RNG is recorded in the smart contract, after the RNG process, RNG players can call `getIncentive()` to get a extra amount of money (1 ether evenly divided to all players) to incentivize them to participate in the RNG.

2 Fixing Deterministic pseudo-random number generator function (Requirement 2, 4)

With the same seed, the result of the `rand()` function will be the same according to the C++ random number generator implementation. As the casino will use the same rand mechanism to generate x and $r + k$ is the seed for the function. So, same k provided by bettor will have the same outcome. In order to prevent bettors using the same k to win again, each k value can be used only once.

```
1  mapping(uint256 => address) public k_To_bettor; // the same k can only be used for one bettor, and
       cant reuse again.
2  require(k_To_bettor[_k]==address(0), "This value k is used, choose another one.");
```

A sample pure function to generate x from r and k as seed is shown below.

```
// This function will not be recorded on the blockchain, only use locally.
// This part is not real implemented.
// seed = _r + _k
function rand(uint16 _r, uint _k) public pure returns(uint256 x){
    uint256 seed = uint256(_r) + _k;
    uint256 random = uint256(keccak256(abi.encodePacked(seed)));
    return random;
}
```

After the casino generate x from their local machine (not on blockchain as `rand()` is a pure function), so that neither r nor $r+k$ nor x will be disclosed. Then the casino can use the following function to announce the result of the bet.

```
// This function is called by the casino to announce a player is won or not.
mapping(uint256 => bool) public isThisKWin;
mapping(uint256 => bool) public announced; // casino have announced this k value result
function announceResult(uint _k, bool isXEven) public onlyOpened() onlyAfter(openTime){
    require(msg.sender == casino, "Only casino can announce bet result");
    require(!announced[_k], "The result of this bet is announced already");
    require(k_To_bettor[_k]!=address(0), "Bettor not exist");
    require(!isCasinoCashOuted, "You have already cash outed and cant announce result anymore");
    isThisKWin[_k] = isXEven;
    announced[_k] = true;
    if(!isXEven) betNotPaid--; // casino win, bet need to pay -1 as the casino no need to pay this
        bet
}
```

After r is disclosed in later step, bettors can generate the same x again using the same function and seed.

3 Casino deposits a huge amount of money (Requirements 3, i)

In order to ensure that the deposit put by the casino is enough to compensated twice the money they lost in case the casino cheats, we have to assume that the casino cheat in all the bets. So, given a initial deposit of d ether, the casino will only number of bets is $\frac{d-1ether}{0.01ether}$ (minus one as 1 ehter will pay for the RNG) as the casino will lose 0.01 ether in each bet.

```
// casino add deposit to allow more bets
function addDeposit() external payable onlyOpened() onlyAfter(openTime){
    require(msg.sender == casino, "Only casino can add deposit");
    require(msg.value > 0, "You must add deposit");
    betLimit += msg.value/0.01 ether;
}
```

However, the casino can add deposit to the contract to allow more bets using the above function. The value added can increase the number of bets by $\frac{d}{0.01ether}$.

4 r announcement (Requirements 5, 6, h)

At the end of the day, the casino can announce the value of r and the secret key used in the GM encryption system $sk = (p, q)$ to the contract.

```
// this function is called by the casino to disclose r used for the day
bool public isDisclosed;
bool public cheatInR;
uint16 public r;
function discloseR(uint16 _r, uint256 _p, uint256 _q) public onlyOpened() onlyAfter(closeTime)
    onlyBefore(reveal_r_Time){
    require(msg.sender == casino, "Only casino can disclosed r and open casino.");
    require(!isDisclosed, "You have already disclosed r.");
    p = _p;
    r = _r;
    q = _q;
    isDisclosed = true;
    // secret key not match with public key
    if(p*q != N) cheatInR = true;
    // the r is not from all player r_i, that is enc_r1 * enc_r2 * ... * enc_rn
    if (enc_r != verEnc_r) cheatInR = true;
    // decryption not equal
```

```
17      if (!testDecryption()) cheatInR = true;
18
19      reveal_r_Time = block.timestamp;
20      reportTime = reveal_r_Time+ REPORT_PERIOD;
21      emit ReportPeriod(reveal_r_Time, reportTime);
22  }
23  function testDecryption() private view returns(bool same){
24      // this is a function to check whether Dec(Enc(r)) = r using sk=(p,q)
25      // but I dont know how to implement the decryption system of GM :(
26      return true;
27  }
```

After that, The authorities and every bettor can verify that:

i. r was really generated by the RNG process in Step-1. The contract will check that:

   (a) Check whether the secret key is $n = pq$ (line 12-13).

   (b) Check whether $Enc_{gm}(r)$ is generated using all of the players contributions, that is $Enc_{gm}(r) = Enc_{gm}(r_1) \times Enc_{gm}(r_2) \times \cdots \times Enc_{gm}(r_n)$ (lines 14-15).

   (c) Check whether decryption of $Enc_{gm}(r)$ is really r, that is $r = Dec_{gm}(Enc_{gm}(r)) = Dec_{gm}(Enc_{gm}(r_1 \oplus r_2 \oplus \cdots \oplus r_n))$ (lines 16-17).

   If any of the above check fails, the casino is cheating in r and will set `bool cheatInR = true`.

   Also, if the casino doesn't disclose r before the deadline, the following function can be called and the casino will be considered cheating in r.

```
1  // this function can be called by anyone if the casino fails to disclose r before the deadline
2  function noRDisclose() public onlyOpened() onlyAfter(reveal_r_Time){
3      require(!isDisclosed, "The casino have already disclosed r.");
4      require(!cheatInR, "Already know casino cheated in r value, you can call reportCheating()
           before deadline");
5      cheatInR = true;
6      reveal_r_Time = block.timestamp;
7      reportTime = reveal_r_Time+ REPORT_PERIOD;
8      emit ReportPeriod(reveal_r_Time, reportTime);
9  }
```

ii. The casino did not cheat in any of the bets (i.e. using other value of r to generate x or announce odd number when x is even). After the casino discloses r, each bettor can check whether x generated by the function is really what the casino claims. If the player finds that the casino is cheating, he can report the cheating to the contract by calling the following function and get compensation of twice the money they lost.

```
1  // for bettor to report casino cheating in a specify k
2  // return true if report successful, false if not
3  mapping(uint => bool) public isSuccessReport;
4  function reportCheating(uint _k) external onlyOpened() onlyAfter(reveal_r_Time) onlyBefore(
       reportTime) returns(bool isSuccess){
5      require(k_To_bettor[_k]==msg.sender, "You are not the bettor of this k value");
6      require(isProcessed[_k], "You must call claimResult() first.");
7      require(!isThisKWin[_k], "You didn't lose any money.");
8      require(!isSuccessReport[_k], "You have already reported and compensated.");
9      if(cheatInR){ // the casino cheated in disclosing not the valid r
10         isSuccessReport[_k] = true;
11         (bool sent, bytes memory __) = payable(msg.sender).call{value: 2*BET_VALUE}("");
12         require(sent, "fallback function exc fail");
13         return true;
14     }
15     // check whether different r is used in the bet
16     x = rand(r, _k);
17     if(x%2 == 0){ // x is indeed even but casino claim it is odd before
18         isSuccessReport[_k] = true;
19         (bool sent, bytes memory __) = payable(msg.sender).call{value: 2*BET_VALUE}("");
20         require(sent, "fallback function exc fail");
21         return true;
22     }
23     // no cheating for the casino
24     return false;
25 }
```

Notice that only bet that is lose by the bettor can be reported as cheating. If the bettor win the bet, he will not be able to report cheating as he didn't lose any money(even winning).

5 Casino withdrawl (Requirements 7)

```
// casino withdraw money from contract
bool isCasinoCashOuted;
function casinoWithdrawl() public onlyOpened() onlyAfter(reportTime){
    require(msg.sender == casino, "Only casino can withdraw its deposit");
    require(!isCasinoCashOuted, "You have already withdrawed");
    isCasinoCashOuted = true;
    // will not withdrawl all balance.
    (bool sent, bytes memory __) = payable(casino).call{value: address(this).balance - betNotPaid
        *2*BET_VALUE}("");
    require(sent, "fallback function exc fail");
}
```

Instead of withdrawing all the balance in the contract, the casino will only withdraw the balance that really belong to them. As there are chance that won bettor have not claim their result yet.

Also, if the casino haven't announced the result of some bets when try to withdraw, all those unannounced bets will be considered won by the bettor and the casino will lose money in these bets (bettor can later call `claimResult()` after the CASINO_RESPOND_TIME).

The above two number of bets will be recorded in `betNotPaid`, `betNotPaid` increase and decrease can be found in `bet()`, `announceResult()` and `claimResult()`.

```
// betNotPaid is used for recording the bet that have not paid (for casino cash out withdrawl use)
// it count for bet that is:
// 1. Bet not yet announced result.
// 2. Bet that is bettor won but not yet claimed.
uint betNotPaid;
```

The casino will withdraw the rest of the balance in the contract and can no longer call `announceResult()`.

6 No vulnerabilities (Requirements j)

6.1 Flow control

Functions can only be called during the corresponding period of time by using `modifier` and `require`.

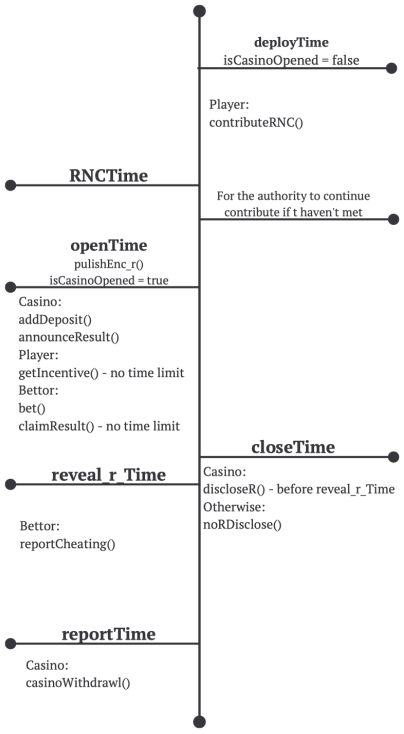

Figure 1: Flow of the protocol

6.2 Front running

As the protocol is not order sensitive, front running have no effect on the result.

6.3 Reenterancy

Each ether call will be following by a state change, so that reenterancy is not possible.

```
require(!paid);
paid = true;
(bool sent, bytes memory __) = payable(msg.sender).call{value:}("");
require(sent, "fallback function exc fail");
```

6.4 Pull over push

Each bettor will have to call `claimResult()` to get their money and the casino will have to call `casinoWithdrawl()` to get their money. No one is pushing money to other address but only their only address.

6.5 Deadline of not disclosing r or result of bet

There will be punlishment if the casino doesn't disclose r or result of bet before the deadline. So that the casino cant intentionally not disclose r or result of bet.

6.6 Withdrawl Check

The casino will only withdraw the money that really belong to them as stated in the section 5 Casino withdrawl.

6.7 No looping

There is no looping in the sol code so that the EVM will not throw an exception due to infinite loop.

7 Appendix

7.1 Full code

```
pragma solidity >=0.7.0 <0.9.0;

contract Casino{
    // phase constant
    uint private constant RNC_PERIOD = 2 hours;
    uint private constant OPEN_PERIOD = 12 hours;
    uint private constant REVEAL_R_DEADLINE = 1 hours;
    uint private constant REPORT_PERIOD = 2 hours; // must be longer than CASINO_RESPOND_TIME,
        otherwise casino can cash out without announce result for bet
    uint private constant BET_VALUE = 0.01 ether;
    uint private constant CASINO_RESPOND_TIME = 1 hours; // casino must announce the bet within 1
        hour after the betting.

    // casino address and the actual time for each phase
    address public casino;
    address public authorities = 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2;
    uint public deployTime;
    uint public RNCTime;
    uint public openTime;
    uint public closeTime;
    uint public reveal_r_Time;
    uint public reportTime;
    bool public isCasinoOpened;

    // Modifier that make sure correct function called in correct phase
    event RNCPeriod(uint startingTime, uint endingTime);
    event BettingPeriod(uint startingTime, uint endingTime);
    event ReportPeriod(uint startingTime, uint endingTime);

    // The function has been called too early.
    // Try again at `time`.
    error TooEarly(uint time);
    // The function has been called too late.
    // It cannot be called after `time`.
    error TooLate(uint time);
    // The function has been called too early.
    // It cannot be called when casino have not opened, i.e. after announce Enc_r.
    error CasinoNotOpen();

    modifier onlyBefore(uint time) {
        if (block.timestamp >= time) revert TooLate(time);
        _;
    }
    modifier onlyAfter(uint time) {
        if (block.timestamp <= time) revert TooEarly(time);
        _;
    }
    modifier onlyOpened(){
        if (!isCasinoOpened) revert CasinoNotOpen();
        _;
    }

    // Assume the casino generate those pair of key using Goldwasser-Micali system.
    // This pair of key is just for demo purpose.
    // uint p = 13;
    // uint q = 17;
    // uint x = 7;
    // uint N = 221;
    uint256 public p;
    uint256 public q;
    uint256 public x;
    uint256 public N;

    constructor(uint256 _x, uint256 _N){
        x = _x;
        N = _N;
        casino = msg.sender;
```

```
66          deployTime = block.timestamp;
67          RNCTime = deployTime + RNC_PERIOD;
68          isCasinoOpened = false;
69          emit RNCPeriod(deployTime, RNCTime);
70      }
71
72      // RNC phase, player can encrypt their r_i by the gm encryption system.
73      bytes32 verEnc_r = 0x0000000000000000000000000000000000000000000000000000000000000001; // the
            multiple of all enc(ri) until a time
74      mapping(address => bytes32) private enced_ris; // player to their enc(ri)
75      address[] public RNCcontributedPlayers; // list of contributed players that is not controlled
            by the authorities
76      bytes32[] public authorities_enced_ris; // list of enc(ri) given by the authorities
77      uint private t; // the number required by the authorities
78      function contributeRNC(bytes32 enced_ri) external{
79          require(!isCasinoOpened, "RNG finished");
80          require(enced_ris[msg.sender]!=bytes32(0), "You have already contributed.");
81          if(msg.sender != authorities){ // this player is not controlled by the authorities
82              require(block.timestamp <= RNCTime, "The RNC time for player not controlled by
                    authorities is over.");
83              enced_ris[msg.sender] = enced_ri;
84              RNCcontributedPlayers.push(msg.sender);
85              t = calT();
86          }
87          else if (msg.sender == authorities){ // this player is controlled by the authorities
88              if(block.timestamp > RNCTime){ // normal RNC time is over, only allow authorities add
                    enc_ri if t is not enough
89                  require(authorities_enced_ris.length < t, "The authorities have already controlled
                        t players.");
90              }
91              authorities_enced_ris.push(enced_ri);
92              t = calT();
93          }
94          verEnc_r = bytes32(uint256(verEnc_r) * uint256(enced_ri));
95      }
96      function calT() private view returns(uint t){
97          uint n = RNCcontributedPlayers.length + authorities_enced_ris.length;
98          uint value;
99          if (n % 2 == 0) value = n / 2;
100         else value = n / 2 + 1;
101         return value;
102     }
103
104     // the casino is able to decrypt r because they have sk=(p,q), they keep r secret
105     bytes32 public enc_r; // the pulish enc_r used for the day
106     bool private isEnc_rPublished;
107     uint private betLimit; // bet limit for requirement i
108     function pulishEnc_r(bytes32 _enc_r) external payable onlyAfter(RNCTime) {
109         require(msg.sender == casino, "Only casino can public Enc(r) and open casino.");
110         require(msg.value >= 100 ether, "Must deposits a huge amount of money in the smart
                contract");
111         require(!isEnc_rPublished, "Casino have already opened.");
112         require(authorities_enced_ris.length >= t, "The authorities have not control at least t
                players yet.");
113         enc_r = _enc_r;
114         openTime = block.timestamp;
115         closeTime = openTime + OPEN_PERIOD;
116         reveal_r_Time = closeTime + REVEAL_R_DEADLINE;
117         reportTime = reveal_r_Time + REPORT_PERIOD;
118         isEnc_rPublished = true;
119         isCasinoOpened = true;
120         betLimit = (msg.value - 1 ether)/(0.01 ether); // -1 ether as 1 ether will be given to RNG
                players
121         emit BettingPeriod(openTime, closeTime);
122     }
123
124     // after that the casino is opened and bettor can bet.
125
126     // casino add deposit to allow more bets (for requirement i)
127     function addDeposit() external payable onlyOpened() onlyAfter(openTime){
128         require(msg.sender == casino, "Only casino can add deposit");
```

```
        require(msg.value > 0, "You must add deposit");
        betLimit += msg.value/0.01 ether;
    }

    // 1 ETH will evenly distributed to RNG players
    mapping (address => bool) public gotIncentivized;
    function getIncentive() public onlyOpened(){
        require(enced_ris[msg.sender]!=bytes32(0), "You didnt contributed to the RNC");
        require(!gotIncentivized[msg.sender], "You have already claim the incentivized.");
        uint reward = 1 ether;
        if(msg.sender != authorities){ // not controlled players get their own incentive
            reward /= (RNCcontributedPlayers.length + authorities_enced_ris.length);
            gotIncentivized[msg.sender] = true;
            (bool sent, bytes memory __) = payable(msg.sender).call{value: reward}("");
            require(sent, "fallback function exc fail");
        }
        else if (msg.sender == authorities){ // authorities will get the incentive for all its
            controlled players
            reward /= (RNCcontributedPlayers.length + authorities_enced_ris.length);
            reward *= authorities_enced_ris.length;
            gotIncentivized[msg.sender] = true;
            (bool sent, bytes memory __) = payable(msg.sender).call{value: reward}("");
            require(sent, "fallback function exc fail");
        }
    }

    // The betting phase, bettor call this function.
    uint private numBet;
    mapping(uint256 => address) public k_To_bettor; // the same k can only be used for one bettor,
         and cant reuse again.
    mapping(uint256 => uint) public betTime; // the betting time of this bet
    // betNotPaid is used for recording the bet that have not paid (for casino cash out withdrawl
        use)
    // it count for bet that is:
    // 1. Bet not yet announced result.
    // 2. Bet that is bettor won but not yet claimed.
    uint betNotPaid;
    function bet(uint256 _k) external payable onlyOpened() onlyAfter(openTime) onlyBefore(
        closeTime){
        require(msg.value == BET_VALUE, "You must place a bet.");
        require(numBet < betLimit, "current betting number is no enought to ensure compensation.")
            ;
        require(k_To_bettor[_k]==address(0), "This value k is used, choose another one.");
        k_To_bettor[_k] = msg.sender;
        betTime[_k] = block.timestamp;
        betNotPaid++; // consider this bet have not paid in current moment
    }

    // This function will not be recorded on the blockchain, only use locally.
    // This part is not real implemented.
    // seed = _r + _k
    function rand(uint16 _r, uint _k) public pure returns(uint256 x){
        uint256 seed = uint256(_r) + _k;
        uint256 random = uint256(keccak256(abi.encodePacked(seed)));
        return random;
    }

    // This function is called by the casino to announce a player is won or not.
    mapping(uint256 => bool) public isThisKWin;
    mapping(uint256 => bool) public announced; // casino have announced this k value result
    function announceResult(uint _k, bool isXEven) public onlyOpened() onlyAfter(openTime){
        require(msg.sender == casino, "Only casino can announce bet result");
        require(!announced[_k], "The result of this bet is announced already");
        require(k_To_bettor[_k]!=address(0), "Bettor not exist");
        require(!isCasinoCashOuted, "You have already cash outed and cant announce result anymore"
            );
        isThisKWin[_k] = isXEven;
        announced[_k] = true;
        if(!isXEven) betNotPaid--; // casino win, bet need to pay -1 as the casino no need to pay
            this bet
    }
```

```
193
194     // This function is called by the bettor to know the result of a bet and claim this bet reward
            if any.
195     // return true if win, false if lose
196     mapping(uint => bool) public isProcessed;
197     function claimResult(uint _k) external onlyOpened() onlyAfter(openTime) returns(bool win){
198         require(!isProcessed[_k], "The bet have already processed.");
199         require(k_To_bettor[_k]==msg.sender, "You are not the bettor of this k value");
200         if (!announced[_k]){ // the casino have not process the bet yet
201             if(block.timestamp > betTime[_k] + CASINO_RESPOND_TIME){ // the casino fails to
                    respond the bet within time
202                 isProcessed[_k] = true;
203                 isThisKWin[_k] = true;
204                 announced[_k] = true;
205                 (bool sent, bytes memory __) = payable(msg.sender).call{value: 2*BET_VALUE}("");
206                 require(sent, "fallback function exc fail");
207                 betNotPaid--;
208                 return true;
209             }
210             else{
211                 require(false, "The casino have not announce you bet result but they still have
                        time to do it.");
212             }
213         }
214         else{ // the casino have processed the bet
215             if(isThisKWin[_k]){ // this bet is won
216                 isProcessed[_k] = true;
217                 (bool sent, bytes memory __) = payable(msg.sender).call{value: 2*BET_VALUE}("");
218                 require(sent, "fallback function exc fail");
219                 betNotPaid--;
220                 return true;
221             }
222             else{ // this bet lose
223                 isProcessed[_k] = true;
224                 return false;
225             }
226         }
227     }
228
229     // this function is called by the casino to disclose r used for the day
230     bool public isDisclosed;
231     bool public cheatInR;
232     uint16 public r;
233     function discloseR(uint16 _r, uint256 _p, uint256 _q) public onlyOpened() onlyAfter(closeTime)
            onlyBefore(reveal_r_Time){
234         require(msg.sender == casino, "Only casino can disclosed r and open casino.");
235         require(!isDisclosed, "You have already disclosed r.");
236         p = _p;
237         r = _r;
238         q = _q;
239         isDisclosed = true;
240         // secret key not match with public key
241         if(p*q != N) cheatInR = true;
242         // the r is not from all player r_i, that is enc_r1 * enc_r2 * ... * enc_rn
243         if (enc_r != verEnc_r) cheatInR = true;
244         // decryption not equal
245         if (!testDecryption()) cheatInR = true;
246
247         reveal_r_Time = block.timestamp;
248         reportTime = reveal_r_Time+ REPORT_PERIOD;
249         emit ReportPeriod(reveal_r_Time, reportTime);
250     }
251     function testDecryption() private view returns(bool same){
252         // this is a function to check whether Dec(Enc(r)) = r using sk=(p,q)
253         // but I dont know how to implement the decryption system of GM :(
254         return true;
255     }
256
257     // this function can be called by anyone if the casino fails to disclose r before the deadline
258     function noRDisclose() public onlyOpened() onlyAfter(reveal_r_Time){
259         require(!isDisclosed, "The casino have already disclosed r.");
```

```
260         require(!cheatInR, "Already know casino cheated in r value, you can call reportCheating()
                before deadline");
261         cheatInR = true;
262         reveal_r_Time = block.timestamp;
263         reportTime = reveal_r_Time+ REPORT_PERIOD;
264         emit ReportPeriod(reveal_r_Time, reportTime);
265     }
266
267     // for bettor to report casino cheating in a specify k
268     // return true if report successful, false if not
269     mapping(uint => bool) public isSuccessReport;
270     function reportCheating(uint _k) external onlyOpened() onlyAfter(reveal_r_Time) onlyBefore(
            reportTime) returns(bool isSuccess){
271         require(k_To_bettor[_k]==msg.sender, "You are not the bettor of this k value");
272         require(isProcessed[_k], "You must call claimResult() first.");
273         require(!isThisKWin[_k], "You didn't lose any money.");
274         require(!isSuccessReport[_k], "You have already reported and compensated.");
275         if(cheatInR){ // the casino cheated in disclosing not the valid r
276             isSuccessReport[_k] = true;
277             (bool sent, bytes memory __) = payable(msg.sender).call{value: 2*BET_VALUE}("");
278             require(sent, "fallback function exc fail");
279             return true;
280         }
281         // check whether different r is used in the bet
282         x = rand(r, _k);
283         if(x%2 == 0){ // x is indeed even but casino claim it is odd before
284             isSuccessReport[_k] = true;
285             (bool sent, bytes memory __) = payable(msg.sender).call{value: 2*BET_VALUE}("");
286             require(sent, "fallback function exc fail");
287             return true;
288         }
289         // no cheating for the casino
290         return false;
291     }
292
293     // casino withdraw money from contract
294     bool isCasinoCashOuted;
295     function casinoWithdrawl() public onlyOpened() onlyAfter(reportTime){
296         require(msg.sender == casino, "Only casino can withdraw its deposit");
297         require(!isCasinoCashOuted, "You have already withdrawed");
298         isCasinoCashOuted = true;
299         // will not withdrawl all balance.
300         (bool sent, bytes memory __) = payable(casino).call{value: address(this).balance -
                betNotPaid*2*BET_VALUE}("");
301         require(sent, "fallback function exc fail");
302     }
303
304
305 }
```