# COMP4901W - Project

ZENG, Lingqi

## 1 Brief Outline

**RNG Phase**

1. (8 a.m.) The casino deploys the contract, puts its deposits into the contract.
2. (before 8:20 a.m.) The players contribute to the generation of the random number $r$. They should first commit the hash values to the contract, so that the casino won't know the result of the RNG before the player all have contributed.
3. (8:20 a.m. - 8:30 a.m.) Then, the players send their encrypted random numbers to the contract.
4. (8:30 a.m. - 9 a.m.) The casino computes the random number $r$.

**Betting Phase**

1. (9 a.m. - 8:50 p.m.) The bettors bet and gain/lose money.
2. (8:50 p.m. - 9 p.m.) The casino announces the value of $r$ at the end of the day.
3. (9 p.m. - 11 p.m.) Players can verify the validity of $r$, and the casino didn't cheat in any of the bets. If the casino cheat, the bettors split the deposit of the casino.
4. (after 11 p.m.) The casino gets back its deposit, if it didn't cheat.

## 2 The RNG Protocol, a Brief Introduction

We use Goldwasser-Micali Cryptosystem to implement the RNG protocol. Detailed implementation for verification is in the following sections.

**Homomorphic Property**

The GM system has a homomorphic property:

$$Enc(m_1)Enc(m_2) = Enc(m_1 \oplus m_2) \bmod n \tag{2.1}$$

**Easy Cases Introduced During the Lectures**

i Easy Case 1: If $n$ is a prime, $n = p$, by Fermat's Little Theorem,

$$x = y^2 \iff x^{(p-1)/2} \equiv 1(\mathrm{mod} p) \tag{2.2}$$

ii If $n = p \cdot q$, and we know primes $p$, $q$,

$$\exists y, x = y^2 \bmod n$$

It is easy to check,

$$x = y^2 \bmod p$$
$$x = y^2 \bmod q$$

The protocol is as follows:

**Key Generation**

i The casino chooses two large primes $p$ and $q$ and computes $n = pq$.

ii The casino finds $x \in \{0, 1, \ldots, n-1\}$ such that $x$ is not a quadratic residue modulo $p$ nor a quadratic residue modulo $q$.

iii The casino publishes $n$ and $x$ as the public key, and uses $p$ and $q$ as the private key.

**Encryption**

i The players want to send message $m = m_1 m_2 \ldots m_k$ to the casino. The message $m$ is a binary string of length $k$, which is the binary expression of the random number the player choose.

ii For each $m_i$, the player chooses a random $m_i \in \{0, 1, \ldots, n-1\}$ and computes $c_i = y_i^2 x^{m_i} \bmod n$.

iii A property is that if $m_i = 0$, then $c_i$ is a quadratic residue modulo $n$, otherwise $c_i$ is not a quadratic residue modulo $n$.

iv The player sends $c_i$ to the casino.

**Decryption**

i The casino receives $c_i$ from the player.

$$c = c_1 \cdot c_2 \cdot \cdots \cdot c_k.$$

For every bits $c_j$ of $c$, the casino checks whether it is a quadratic residue modulo $n$, and decrypts it bit-by-bit. If $c_j$ is a quadratic residue modulo $n$, then $m_j = 0$, otherwise $m_j = 1$.

ii Using the homomorphic property of Eq. (2.1) of the GM cryptosystem, we have:
For player $j$,

$$c_j = c_{j1} \cdot c_{j2} \cdot \cdots \cdot c_{jk}.$$

The casino calculate the multiplication of the $c_j$, i.e.

$$c = c_1 \cdot c_2 \cdot \cdots \cdot c_k.$$

$$c = Enc(m_1) \cdot Enc(m_2) \cdot \cdots \cdot Enc(m_k) = Enc(m_1 \oplus m_2 \oplus \cdots \oplus m_k) = Enc(r) \bmod n \tag{2.3}$$

iii The r is the generated random seed.

# 3 Code Explanation and Workflow

## 3.1 Variables and Mappings

The contract uses several variables and mappings to store information about the betting process, players, and bettors.

- `START_TIME`: The start time of the bet, immutable(uint256).
- `DEPOSIT`: The bettor puts down a deposit of 1 Schweizerlandish schilling, which is 0.01 ETH in our case, immutable.
- `PLAYER_REWARD`: The reward for players who contribute to generating the random number (uint256).
- `CASINO_ADDRESS`: The address of the casino, immutable (address).
- `AUTHORITY_ADDRESS`: The address of the authority, immutable (address).
- `encrypted_r` and `r`: The encrypted and actual random number (uint128[] and uint16, respectively).
- `deposit_of_casino`: The amount of money the casino deposits in the smart contract (uint256).
- `number_of_bettors`: The total number of bettors who placed bets (uint256).
- `required_deposit`: The required deposit for the casino (uint256).
- `n_players`: The number of players (uint256).

- `casino_cheated`: A boolean variable indicating if the casino cheated.
- `CASINO_PK_X` and `CASINO_PK_N`: The public key of the casino for the GM crypto system, immutable (uint128).
- `prime_p` and `prime_q`: The secret prime numbers used in the GM crypto system (uint128).
- `bettor_k`: A mapping that stores the random number k of each bettor.
- `player_random`: A mapping that stores the random number of each player.
- `player_encrypted_random`: A mapping that stores the encrypted random number of each player.
- `player_hashed_random`: A mapping that stores the hashed random number of each player.
- `bettor_balance`: A mapping that stores the balance of each bettor.
- `bettor_win`: A mapping that stores if each bettor won or lost.
- `bettor_can_pick_up`: A mapping that stores if each bettor can pick up their money.
- `player_can_pick_up`: A mapping that stores if each player can pick up their money.
- `player_list`: A mapping that stores the address of each player.
- `bettor_list`: A mapping that stores the address of each bettor.
- `betting_blocknum`: A mapping that stores the block number of each betting event.

**An Explanation**

My idea is that, for tracing the players and the bettors, since the authority is participated, and only provided one public key, I used index number to trace them. Mapping using address is not that practival, since here an address would map to several account.

In the following, each time a function designed for players or bettors to call, when calling the function an index checking would be performed. The checking would compare the address that index maps to with the address of the message sender. If they are not the same, the function would not be executed.

I used several mappings to store the state, to avoid the problem of calling a function redundantly, etc.

Besides, I also created several events, for announcing the register of a player, the result of a bet, waiting for casino to reveal bet result, the end of the bet, and the casino being found cheating.

## 3.2 Constructor

The constructor requires the caller to be a real account instead of a contract. It takes in the deposit of the casino, initialized the values of the immutable variables, and modify the value of the required deposit for the casino. the initial value the casino puts down should be larger than the reward for the players who contribute to generating the random number, which is set to 1000 gwei here.

## 3.3 RNG

- `register_player`: Registers a player by adding their address to the `player_list` array and emitting an event to announce the registration. The `AUTHORITY_ADDRESS` is also added to the array.
- `player_commit_hashed_rng`: Allows a player to commit their hashed random number by providing the value and their index in the `player_list` array. The function checks the time, verifies the player's index, and checks their previous contribution. If valid, the hashed random number is stored.
- `player_commit_encrypted_rng`: Allows a player to commit their encrypted random number along with their index and nonce. The function verifies the time, player's index, authority status, previous contribution, and hash value using `verify_player_hash`. If all checks pass, the encrypted random number is stored, and the `player_can_pick_up` flag is set.
- `verify_player_hash`: Takes the encrypted random number, nonce, and player's index as input. It checks if the committed hash value matches the computed hash value using the current parameters. Returns `true` if they match.

**An Explanation**

A commit-reveal scheme is used here, so that the casino can't compute the result before all players have decided the choices. This is to avoid the case that the players are colluded with the casino, and generated some value which affect the random seed. Since the bettor can also be the player, the casino may cheat in this way.

Anyway we hope to avoid this situation. Hence, a commit-reveal scheme is used. The players first commit the hashed values of the encrypted random number they chose, together with the nonce and their index. Then after certain time, they are asked to reveal the encrypted random number. Only if they successfully commit the encrypted random number can they get the reward later. If they fail to reveal, they won't get their reward. Since we assume they are rational, they tend to choose to reveal if they can't gain more by cheating. It is possible that the player stands for the casino choose not to reveal to affect the generated random seed, but in this situation, the authority can also have some player not revealing the choices as a countermeasure. As long as there are some players controlled by the authority (and of course it is the case!), the casino can't tamper with the result here.

What about the authority, can they tamper with the result? In my opinion, they can't. We assume there are at least one player controlled by the casino. Since the random seed r is generated by the G-M system, and an XOR operation is used, as long as there are one submission generated uniformly random, the result would be uniformly random.

The generation of the random number does not depend on the on-chain information such as block number and block timestamp, other people including the miners can't tamper with the result.

The players and bettors should remember their own index. The index is for them to locate their address, and other properties used in mappings.

## 3.4 Deposit Functions

- `casino_deposit`: Allows the casino to deposit funds into the contract. It is used for the casino to add deposit when its initial deposit is not enough. The function checks the time and only allows the casino to call it. The deposited value is added to `deposit_of_casino`.

- `bettor_bet`: Allows a bettor to place a bet by sending the required deposit amount. The function checks the time, deposit amount, casino's deposit, and bettor's previous bet status. The casino should have enough deposit to pay for cheating in all the bets. If the deposit is not enough, you should stop accepting new bets unless the casino increases it. If all conditions are met, the bettor's information is stored, and an event is emitted.

- `check_win`: Allows the casino to announce the result of the bet. The function checks the casino's identity, bettor's balance, and betting block number. If conditions are met, the bettor's balance is updated, and an event is emitted.

**An Explanation**

It is required that the casino should announce the result of the bet within a few blocks. Hence, a limit of 10 blocks is used. If the time exceed the limit, the bettor is considered as winning the bet.

The bettor tells the casino to reveal the result by emitting an event. The casino reply also by emitting an event. The money is not transferred, but the balance of the account of the bettor is modified.

The bettor may later take the value himself.

## 3.5 Two Unimplemented Rand Functions

Unimplemented but will be used in later section.

## 3.6 Deposit Functions

- `announce_r`: It is for the casino to reveal the r, and the private keys. It checks the message sender, time, and in the end emitting an event announcing the reveal of r and the end of the bet.

## 3.7 Functions for Decryption

- `bitwise_qr_check`: Takes a `uint128` value `c` as input and checks if `c` is a quadratic residue modulo `prime_p` and `prime_q`. It returns 1 if `c` is a quadratic residue modulo both primes, and 0 otherwise. It is for computing the value of the plaintext.

- `decrypt_m`: Takes an array of 16 `uint128` values `encrypted_m` as input and returns a `uint16` value. The length should be 16, since the original random value is of type `uint16`. It then iterates over each element of the array and applies the `bitwise_qr_check` function to each element. The function sets the corresponding bit of the `result` variable to 1 if the `bitwise_qr_check` function returns 1 for the element; otherwise, the bit remains 0. The function returns the `result` variable, which represents the plaintext of the players.

- `reveal_all_m`: A public function that allows someone to reveal the random values of all players that were used for generating the random seed. The caller is not checked. The function contains many loop and may be very costly. The function iterates over each player's encrypted random number and decrypts it using the `decrypt_m` function. The resulting plaintext value is then stored in the `player_random` mapping.

## 3.8 Functions for Verification

- `claim_cheated`: We assume the player himself would claim the cheating of the casino. The function takes in an index amd checking it is the corresponding player calling the function. It also checks the time. It verifies if the decrypted r using the revealed r value, and the r calculated using the submitted values are the same.

- `plaintext_r` is to calculate the plaintext of the random seed r using the players' choices.

## 3.9 Functions for Refunding

- `bettor_get_money`: Allows a bettor to retrieve their money if the conditions are met. The bettor should call the function himself. Checks include ensuring the current time is before the casino's money retrieval deadline, verifying the caller is the bettor, and confirming the availability of the bettor's money. If the casino has cheated, the deposits of the casino excluding the players' reward are splitted among the bettors.

- `player_get_money`: Enables a player to retrieve their money if the conditions are met. The player should call the function himself. Checks include ensuring the current time is before the casino's money retrieval deadline, verifying the caller is the player, and confirming the player has contributed. Transfers the player's share of the reward.

- `casino_get_money`: Allows the casino to retrieve its money after the deadline. Checks the current time and verifies the caller is the casino. Transfers the contract's balance to the casino.